



Sviluppo di applicazioni per dispositivi mobili

Simone Malesardi

Matricole 978989

Professore Sergio Mascetti

A.A. 2020/2021

Ultimo aggiornamento January 22, 2022

Indice

1	Introduzione al Mobile Computing - Mobile Computing	1
1.1	Caratteristiche dei dispositivi mobili	1
1.1.1	Unità di calcolo	2
1.1.2	Comunicazione di rete	3
1.1.3	Display	3
1.1.4	Altre periferiche per l'interazione utente	4
1.1.5	Sensori	5
1.1.5.1	Sensori inerziali	5
1.1.5.2	Sensori di acquisizione immagini (fotocamere)	6
1.1.5.3	Sensori virtuali	7
1.2	Mobile computing	8
2	Sistemi operativi	9
2.1	Introduzione ai SO per dispositivi mobili	9
2.2	Gestione della memoria	10
2.2.1	Gestione della memoria nei dispositivi tradizionali	10
2.2.2	Gestione della memoria nei dispositivi mobili	10
2.2.3	Ciclo di vita concettuale	10
2.2.4	Ciclo di vita in Android	11
2.2.5	SO tradizionale	12
2.3	Astrazione hardware e software	13
2.3.1	Frammentazione software	14
2.4	Gestione della sincronizzazione	15
2.4.1	Errori e soluzioni	16
2.4.1.1	Errori	16
2.4.1.2	Soluzioni	17
2.5	Gestione del background	17
2.5.1	Background in iOS	18
2.5.2	Background in Android	18

INDICE

3 Reti e architetture	19
3.1 Le reti cellulari	19
3.2 Le architetture per dispositivi mobili	20
3.2.1 Connection-oriented vs connection-less	21
3.2.2 Architettura three-tier	21
3.2.3 La definizione del protocollo di comunicazione	23
3.2.3.1 Esempio di comunicazione	23
3.3 La codifica dell'informazione	24
3.3.1 Serializzazione e deserializzazione	26
3.3.2 Codifica di dati binari	26
3.3.3 Protocol Buffer	26
3.4 Le notifiche push	27
3.4.1 Componenti delle notifiche	28
3.4.2 Il protocollo	28
3.4.2.1 La fase di setup	28
3.4.2.2 Fase di invio	29
3.4.3 Attacchi	30
3.4.4 Notifiche push e locali	30
4 Calcolo e uso della posizione	31
4.1 Introduzione all'uso dell'informazione di posizione	31
4.1.1 Accuratezza e precisione	32
4.1.2 Calcolo della posizione con trilaterazione	33
4.2 Calcolo della posizione outdoor	34
4.2.1 Global Navigation Satellite System (GNSS)	34
4.2.2 Rete cellulare	36
4.2.3 WiFi positioning	36
4.2.3.1 Problemi	36
4.3 Gestione dei dati di posizione	37
5 Analisi	39
5.1 L'analisi delle applicazioni per dispositivi mobili	39
5.2 Progettazione della user experience	40
5.2.1 Content Prioritization	40
6 Progettazione	43
6.1 Progettazione delle interfacce	43
6.1.1 Responsive design	43
6.1.1.1 I layout "liquidi" (o "fluidi")	44
6.1.1.2 Ri-definizione delle schermate	44
6.1.1.3 Densità di pixel	44

INDICE

6.1.2	Navigazione	45
6.2	Progettare la struttura del codice	46
6.2.1	I pattern di progettazione	46
6.2.2	MVC: Model View Controller	46
6.2.3	Pattern Observer	48
6.2.4	Vantaggi e svantaggi di MVC	50
7	Sviluppo	52
7.1	Strumenti e tecniche di sviluppo per dispositivi mobili	52
7.1.1	Sviluppo cross-platform	52
7.1.2	Codice in comune	53
7.1.3	Web app/web site	53
7.1.4	Hybrid App	54
7.1.4.1	Accesso alle funzionalità di SO	55
7.1.4.2	Accesso alle funzionalità del SO in Apache Cordova . .	55
7.1.4.3	I plugin	55
7.1.5	App ibride vs web app	56
7.2	Conversione del codice	56
7.2.1	Xamarin	56
7.2.2	Altre piattaforme di conversione	57
7.2.3	La scelta della tecnologia	57
8	Testing e debugging	58
8.1	Approccio alla scrittura del codice	58
8.2	Testing	59
8.2.1	Pro e contro dell'automazione del testing	61
8.3	Debugging	61
8.3.1	Logging	62
8.3.2	Errori comuni	62
9	Posizionamento indoor - MobiDEV	63
9.1	Concetti preliminari	63
9.1.1	Calcolo della posizione indoor vs outdoor	63
9.1.2	Applicazioni principali	64
9.1.3	Paradigmi di navigazione	64
9.1.4	Caratteristiche dei sistemi di posizionamento indoor	64
9.2	Tecniche di posizionamento	65
9.3	Tecniche basate su immagini	65
9.3.1	Pro e contro dei marcatori espliciti	66
9.3.2	I marcatori impliciti	66
9.3.2.1	Pro e contro	66

INDICE

9.3.2.2	Tecniche marker-less	67
9.3.3	Uso di tecniche di visione per posizione outdoor	67
9.4	Calcolo della posizione basato su segnale radio	67
9.4.1	Posizionamento per prossimità	68
9.4.2	Multilaterazione	68
9.4.2.1	Calcolo della distanza con RSSI	69
9.4.3	Fingerprinting (o "scene analysis")	69
9.4.4	Le tecniche a confronto	72
9.5	Sensori inerziali: calcolo dello spostamento	72
9.5.1	Gestione dell'errore	73
9.5.2	Calcolare un passo	73
9.5.3	Tecniche visuo-inerziali	74
9.6	Tecniche ibride	74
9.6.1	Tecniche ibride e calibrazione crowdsourced	74
10	Tecniche di aggregazione di dati soggetti a rumore	75
10.1	Stima della posizione con dati soggetti a rumore	75
10.1.1	Modello bayesiano	75
10.1.2	Modello Markoviano	76
10.1.3	Modello di transizione di stato (o modello di spostamento)	76
10.1.4	Modello percettivo	77
10.2	Calcolo della posizione adottando il modello Markoviano	78
10.2.1	Approccio basato su griglia	79
10.3	Particle filter (Sequential Monte Carlo)	80
10.3.1	Funzionamento	81
10.3.1.1	Quanta incertezza abbiamo?	82
10.3.1.2	Quante particelle servono?	83
10.3.1.3	Implementazione	83
11	Augmented Reality	84
11.1	Registration	85
11.1.1	Registration con segnale radio	85
11.1.2	Registrazione visuale	86
11.1.2.1	Uso dei marcatori esplicativi	86
11.1.2.2	Uso dei marcatori impliciti	89
11.1.2.3	Registrazione visuale senza marcatori (markerless)	90
11.1.3	Il sistema di riferimento	91
11.2	Il problema del tracking nei sistemi AR	91
11.2.1	Registration ripetuta	92
11.2.2	Tecniche visuo-inerziali	92

11.2.2.1 Optical flow	93
11.2.3 Registration e tracking combinate	94
11.2.3.1 Esempio SLAM	94
12 Augmented Reality - display e interaction	95
12.1 Display visuale	95
12.1.1 Visualizzazione realistica	96
12.1.2 Problema dell'occlusione	97
12.1.3 Riconoscimento della scena	97
12.2 Display audio	98
12.3 Interazione	99
12.3.1 Interazione tramite schermo	100
12.3.2 Interagire tramite proxy fisici	100
13 Augmented Reality in pratica	101
13.1 Panoramica	101
13.1.1 Macro funzionalità	101
13.1.2 Comprendere la scena	102
13.2 Alcune funzionalità di ARKit 4	103
13.2.1 AR multiuser	103
13.2.2 Scene geometry	103
14 Activity recognition - Gabriele Civitarese	104
14.1 Introduzione al riconoscimento di attività	104
14.1.1 Riconoscimento	105
14.2 Acquisizione dei dati grezzi di movimento dai dispositivi mobili	106
14.3 Tecniche di machine learning per il riconoscimento di attività	106
14.3.1 Apprendimento supervisionato	107
14.3.1.1 Training set	108
14.4 Pre-processing	109
14.4.1 Features	111
14.4.1.1 Generazione del feature vector	111
14.4.1.2 Feature scaling	111
14.5 Riconoscimento di attività: classificazione	111
14.5.1 Algoritmi di machine learning "noti" in AR	112
14.5.2 Modello di riconoscimento	114
14.5.3 Uso del modello	115
14.5.4 Il risultato del classificatore	115
14.6 Validazione	116
14.6.1 Validazione "naive"	116
14.6.2 Cross-validation	117

INDICE

14.6.3	Leave one subject out cross validation	117
14.6.4	True/false positives/negatives	118
14.6.5	Matrice di confusione	119
14.6.6	Tecniche di tuning	119
14.6.7	Recap. activity recognition con supervised learning	120
14.7	Approcci semi-supervisionati per il riconoscimento di attività	120
14.7.1	Tecniche di apprendimento semi-supervisionato	120
14.8	Use Case: SmartWheels	122
14.8.1	Riconoscere attività vs UF	122
14.8.2	Architettura del sistema	123
14.8.2.1	Acquisizione del dataset	124
15	Introduzione a Swift	125
15.1	Swift	125
15.2	Programmazione procedurale in Swift	125
15.2.1	Tuple	126
15.2.2	Valori opzionali	127
15.2.3	Operatori	129
15.2.4	Gestione dell'overflow	130
15.2.5	Operatori di range	130
15.2.6	Collezioni	131
15.2.7	Selezione con costrutto guard	131
15.3	Funzioni e closures	131
15.3.1	I parametri	132
15.3.2	Ordinamento di un Array	133
15.3.3	Closures	134

Introduzione al Mobile Computing - Mobile Computing

———— Lezione 1 - 30 settembre 2020 ———

1.1 Caratteristiche dei dispositivi mobili

Dalla fine degli anni '90 si sono iniziati a diffondere dispositivi adatti ad essere usati in mobilità, in particolare due tipologie di dispositivi si evolvono in parallelo:

- PDA
- cellulari

Dal 2007, i due dispositivi confluiscono in un unico device, comunemente chiamato smartphone. Oggi invece abbiamo diversi dispositivi quali smartphone, tablet, dispositivi indossabili (es: smartwatch) ecc.

I dispositivi devono essere utilizzati in mobilità. Per rendere questo possibile è stato necessario progettare appositamente diverse componenti hardware e software.

I dispositivi mobili devono:

- essere **compatti** e **leggeri**
- poter funzionare senza essere collegati alla rete elettrica e quindi dotati di **batteria**, ma l'energia immagazzinata nelle batterie è una risorsa limitata e per ottimizzarla sono stati introdotti diversi accorgimenti hardware e software
- disporre di una **comunicazione wireless**
- avere **nuove forme di interazione con l'utente**: devono cambiare gli strumenti di interfaccia e interazione con l'utente

1.1.1 Unità di calcolo

Tra dispositivi mobili e tradizionali non ci sono differenze nelle unità di calcolo ed entrambi i dispositivi implementano l'architettura di Von Neumann con una unità di calcolo (CPU) collegata tramite un bus alla memoria principale che memorizza i programmi e i dati da essi utilizzati.

Dal punto di vista implementativo ci sono in realtà molte differenze significative. In primo luogo, una caratteristica tradizionalmente associata ai dispositivi mobili è quella di avere minori capacità di calcolo rispetto ai dispositivi tradizionali. Questo vuol dire in particolar modo un processore meno performante e minore quantità di memoria principale. Questa differenza era molto marcata nei primi dispositivi mobili, mentre ora si sta assottigliando, almeno per alcuni dispositivi, come i tablet e gli smartphone di fascia alta che possono avere capacità computazionali analoghe a quelle di alcuni dispositivi tradizionali.

Le unità di calcolo sono:

- processori (CPU): sono ottimizzati per garantire minori consumi. Su alcuni dispositivi come ad esempio quelli indossabili (smartwatch), le performance sono minori. Nella CPU ci sono i core, unità di calcolo indipendenti che possono eseguire in parallelo. Una CPU con un core esegue un'operazione, una CPU con più core esegue più operazioni in parallelo
- coprocessori: sono delle unità di calcolo come le CPU, ma non sono general purpose. Se abbiamo una serie di conti che vanno ripetuti spesso, si può alleggerire il carico di lavoro sulla CPU eseguendo questo lavoro su un processore apposito. La GPU è un esempio di coprocessore

Prendiamo ad esempio l'iPhone 11 pro. Il sistema dispone di due componenti che si occupano delle funzioni di calcolo: A13 ed M13. A13 è un SOC (system on a chip), cioè un'unica componente elettronica che implementa più funzionalità hardware. M13 invece è un coprocessore per la gestione dei dati di movimento.

L'A13 include una CPU e vari coprocessori:

- GPU
- Neural engine: coprocessore dedicato per compiti di machine learning
- Image coprocessor: viene utilizzato per la gestione delle immagini
- Secure Enclave coprocessor: viene usato per le chiavi crittografiche

1.1.2 Comunicazione di rete

Un altro aspetto importante nei dispositivi mobili è quello sulla comunicazione di rete. I dispositivi mobili sono progettati per avere una connettività quasi permanente. Permettono di comunicare sia su rete telefonica (rete cellulare) attraverso voce e dati oppure su rete locale.

Ci sono varie tecnologie e protocolli di rete per supportare la trasmissione di voce e/o dati tra il dispositivo mobile e l'operatore telefonico come ad esempio GSM, GPRS, EDGE, UMTS, LTE. La trasmissione avviene grazie alla comunicazione con infrastrutture sparse nel territorio (antenne). La distanza massima tra dispositivo mobile e l'antenna, varia in funzione dello specifico protocollo adottato, ma è di circa 1 km.

Ci sono diverse tecnologie di rete locale:

- WiFi: per trasmissione dati in una rete locale. La distanza massima di trasmissione è di 100 metri.
- Bluetooth: permette la trasmissione di dati a corto raggio (10 metri). Viene usato sia per far comunicare dispositivi dello stesso utente che dispositivi di utenti diversi (es: app immuni che usa bluetooth per far capire quando due persone sono vicine). Un esempio di questa tecnologia sono i beacon, dispositivi appositi che continuano a trasmettere segnali bluetooth che contengono un identificativo. I beacon vengono posizionati in una posizione nota
- NFC: trasmissione dati a cortissimo raggio (10 cm) per effettuare ad esempio pagamenti
- UWB: ultra wide band (introdotta recentemente su iPhone 11). Permette sia lo scambio di informazioni tra dispositivi sia il calcolo della distanza tra i dispositivi stessi

I dispositivi mobili dispongono di un'altra antenna GNSS che permette di ricevere informazioni da trasmittitori satellitari. Questa antenna permette solamente di ricevere informazioni e non di trasmetterle. Questa antenna viene anche erroneamente chiamata "antenna GPS": GPS è una particolare tecnologia.

1.1.3 Display

Le periferiche utilizzate per le operazioni di I/O (Input e Output) verso l'utente sono differenti rispetto a quelle disponibili nei dispositivi tradizionali.

Nei dispositivi tradizionali ci sono strumenti di input come tastiera, mouse, trackpad ecc. che non sono solitamente disponibili nei dispositivi mobili. Nei computer il display

è una periferica di output, nei dispositivi mobili invece è una periferica di I/O.
Le caratteristiche principali di un display variano tra diversi dispositivi:

- risoluzione (numero totale di pixel)
- densità dei pixel (misura in DPI)
- dimensione fisica
- aspect ratio (rapporto tra i due lati)

Esistono due tecnologie di schermo touch:

- resistivo: alla pressione due strati che conducono elettricità si vanno a toccare, permettendo di calcolare dove è avvenuto il tocco. È una tecnologia più economica
- capacitivo: un flusso di elettroni che scorre sulla superficie dello schermo. Dobbiamo toccare con un oggetto capacitivo come il dito e non importa quanto forte premiamo, infatti con i guanti non funziona il touch. Possono gestire il multi-touch

1.1.4 Altre periferiche per l'interazione utente

Oltre allo schermo, esistono varie altre periferiche per l'interazione con l'utente che permettono in particolare due forme di interazione:

- **aptica/motoria** che si basa su:
 - vibrazione: tramite periferica di output
 - riconoscimento del movimento con sensori inerziali

Ad esempio possiamo immaginarcì un videogioco in cui l'utente comanda lo sterzo di una automobile ruotando il dispositivo. Nel caso in cui l'utente esce di strada l'applicazione potrebbe, oltre al feedback visivo mostrato a schermo, dare un feedback aptico sotto forma di vibrazione.

- **audio**: ottenuta grazie all'altoparlante (periferica di output) e al microfono (periferica di input). Se il sistema è in grado di riconoscere il parlato, l'interfaccia si può definire "vocale". L'interfaccia audio è necessaria nei dispositivi che non sono dotati di schermo (es: gli smart-ring) ed è anche utile per l'utilizzo di tutti i dispositivi mobili in particolari contesti, ad esempio durante la guida

1.1.5 Sensori

Ci sono anche dei sensori, strumenti che misurano una quantità fisica e la convertono in un segnale.

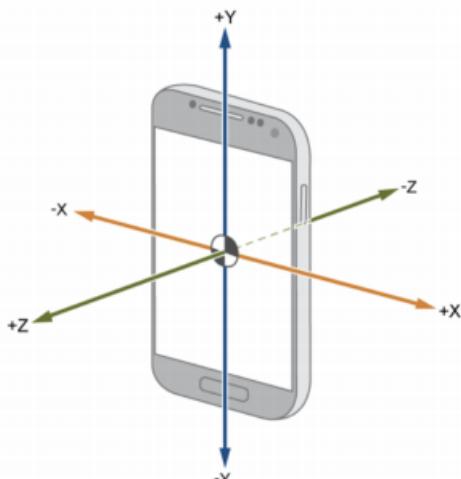
Nei dispositivi mobili i sensori prendono la forma di periferiche. Degli esempi ne sono i sensori:

- di prossimità: tipicamente posizionato in prossimità dell'altoparlante nel punto che viene avvicinato all'orecchio durante una telefonata. Questo permette di capire quando l'utente avvicina il telefono all'orecchio, così da disattivare lo schermo touch per evitare che l'utente interagisca inavvertitamente con il display durante una telefonata
- di luminosità: per adattare la luminosità del display alla luce ambientale
- biometrici: permettono di misurare delle grandezze biologiche dell'utente come ad esempio le impronte digitali o il battito cardiaco
- inerziali: permettono di capire il movimento e lo stato dei dispositivi nello spazio

1.1.5.1 Sensori inerziali

I sensori inerziali più nuovi sono:

- accelerometro: misura l'accelerazione del dispositivo nello spazio rispetto ai tre assi del dispositivo stesso. L'accelerazione di gravità fa parte della misurazione. Gli assi sono tipicamente orientati come in figura e ogni misurazione è composta



da una tripla di valori, uno per ciascun asse. Se tengo il device fermo e perfettamente in verticale, vedrò un asse che misura circa 1g ($9.8m/s^2$) e gli altri circa 0g. Quando il dispositivo viene spostato, gli accelerometri misurano la combinazione della forza di gravità e dell'accelerazione dovuta effettivamente allo spostamento del dispositivo. Si può usare per conoscere la posizione del dispositivo rispetto a terra. Un esempio di utilizzo di accelerometro è la rotazione dello schermo

- giroscopio: misura quanto rapidamente stiamo ruotando il device sui tre assi. Ogni lettura di valori, come per l'accelerometro, è una tripla di numeri e i valori di velocità vengono campionati nel tempo. Il giroscopio non misura l'accelerazione

quindi non è influenzato dall'accelerazione di gravità. Se un dispositivo si trova in stato di quiete si avrà una lettura di valori prossimi allo zero su tutti e tre gli assi. Una cosa che non si riesce a fare con il giroscopio è darci una posizione

- magnetometro: è utile per capire dove sono orientato, rispetto al Nord magnetico terrestre. Misura i valori sui tre assi. Può avere una precisione molto bassa in particolari ambienti urbani. Non calcola la velocità di rotazione come il giroscopio, ma se usati in combinazione riusciamo a fornire un punto di orientamento assoluto (il giroscopio misura solo le rotazioni, ma non ha un orientamento di riferimento)

Ogni misurazione è soggetta ad errore. Il device non conosce l'errore ma possiamo misurarlo con strumenti esterni.

Una fonte d'errore nei sistemi inerziali è che vengono campionati i valori ad intervalli, ciò significa che i valori misurati dai sensori non sono acquisiti in modo costante.

Anche la misurazione istantanea può essere soggetta ad errore.

Spesso vogliamo usare i sensori inerziali per calcolare lo spostamento.

Usando i dati di accelerometri e giroscopi posso calcolare lo spostamento di un device con 6 gradi di libertà:

- 3 gradi di libertà per lo spostamento
- 3 gradi di libertà per l'orientamento

Il problema è che l'errore si accumula nel tempo e più tempo passa, più il calcolo dello spostamento è soggetto ad un errore maggiore. Questo effetto si chiama drift.

1.1.5.2 Sensori di acquisizione immagini (fotocamere)

Molti dispositivi mobili, in particolare smartphone e tablet, sono dotati di una o più videocamere per l'acquisizione di immagini e video. Sono spesso utilizzate in molte applicazioni e servono per raccogliere informazioni dall'ambiente circostante e fornire altre funzionalità. Oltre che per scattare foto e registrare video possono essere usate in combinazione con le tecniche di visione artificiale.

Le applicazioni di queste tecniche sono numerose e includono:

- augmented-reality, per riconoscere il contesto attorno all'utente e mostrare elementi virtuali all'interno del mondo reale
- riconoscimento di oggetti
- sicurezza, ad esempio riconoscere il volto di una persona che sta cercando di sbloccare lo smartphone

- calcolo dello spostamento, usando una combinazione di sensori inerziali e dati della camera

Una caratteristica delle fotocamere è che sono in grado di calcolare le depth images. I pixel più chiari rappresentano oggetti più vicini.

Nella fotografia sono importanti quando ad esempio dobbiamo mettere a fuoco un soggetto, ma sono utili anche per comprendere l'ambiente.

Nei dispositivi mobili le immagini di profondità possono essere calcolate con quattro approcci principali:

- stereo triangulation: come fa il cervello a calcolare la distanza dagli oggetti? Usando gli occhi, sfrutta le due immagini per vedere le differenze tra i due occhi e per calcolare la distanza.
La stessa cosa succede con le telecamere. Prendiamo due foto scattate dalle due fotocamere, le confrontiamo tra di loro e calcoliamo la distanza
- motion parallax: usiamo una sola camera per far finta di averne due. Prendo una foto in un certo istante, cerco di capire come si è spostato il device e dopo poco prendo un'altra immagine.
Mi aspetto che quello che sto fotografando non si sia spostato di molto.
- structured light: siamo in una stanza buia, dobbiamo calcolare la distanza tra una sorgente e un muro. Lanciamo due fasci di luce con un angolo noto da una sorgente. Faccio poi una foto al muro. Se il muro è molto vicino, i punti dei fasci sul muro saranno vicini. Se il muro è lontano, i due punti saranno lontani tra di loro.
- lidar: il funzionamento è simile a quello del radar. Il radar trasmette delle onde radio e in base a come ritornano posso calcolare quanto sono lontani gli oggetti. Nel lidar, invece, non vengono trasmesse onde radio, ma un laser

1.1.5.3 Sensori virtuali

Oltre ai sensori fisici i sistemi operativi dei dispositivi mobili rendono disponibili dei sensori virtuali, componenti software che simulano dei sensori fisici utilizzando i dati provenienti da altri sensori.

Un sensore virtuale è il contapassi, che non è un sensore vero e proprio, ma il SO acquisisce i dati dai sensori inerziali, li analizza, usa un algoritmo e rende disponibile al programmatore i passi contati.

1.2 Mobile computing

I dispositivi mobili hanno introdotto una serie di problemi e opportunità che hanno richiesto uno sforzo tra tutte le discipline informatiche:

- molti protocolli di **comunicazione di rete** sono stati studiati appositamente per i dispositivi mobili
- i **sistemi operativi** sono stati profondamente modificati per rispondere alle esigenze dei dispositivi mobili
- sono state introdotte nuove tecniche di **data management**, per trattare i dati dei dispositivi mobili, per esempio richiedendo lo sviluppo di **basi di dati** apposite che adottano nuovi **algoritmi e strutture dati** per trattare i dati spaziali e temporali generati dai dispositivi mobili
- nuove soluzioni sono state necessarie per adattare i **sistemi distribuiti** alle esigenze dei dispositivi mobili
- i dispositivi mobili hanno accentuato problemi relativi ai **linguaggi di programmazione** (es: la programmazione multi-piattaforma) ma al contempo hanno dato un impulso alla creazione di nuovi e moderni linguaggi
- i dispositivi mobili hanno richiesto di ri-progettare i paradigmi di **interazione uomo-macchina** e hanno evidenziato come gli aspetti di user-experience siano fondamentali per il successo commerciale delle applicazioni
- nuovi problemi e opportunità sono emersi nell'ambito della **sicurezza** (es., sistemi di riconoscimento biometrico applicati su larga scala), con un forte impatto anche su questioni di **privacy**
- i dispositivi mobili forniscono un nuovo contesto applicativo per gli ambiti della visione **artificiale**, del **machine learning**, della **grafica** e del **gaming**

Sistemi operativi

———— Lezione 2 - 2 ottobre 2020 ———

I sistemi operativi per dispositivi mobili condividono con i SO per i dispositivi tradizionali moltissimi aspetti concettuali.

2.1 Introduzione ai SO per dispositivi mobili

L'obiettivo dei SO per dispositivi mobili sono:

- ottimizzare l'uso delle risorse
- semplificare l'accesso alle risorse da parte del programmatore

Una peculiarità dei SO è che le risorse sono particolarmente limitate, non solo le risorse computazionali, ma anche altre risorse come ad esempio la gestione energetica. Un secondo aspetto sono i vincoli real-time, ad esempio nei dispositivi mobili quando si riceve una chiamata, il SO deve far mostrare all'utente la schermata apposita.

I SO operativi più diffusi sul mercato sono due:

- iOS: SO per i dispositivi Apple. Basato su Darwin, un SO open source sul quale sono basati molti dispositivi di Apple tra cui macOS, iOS, iPadOS, watchOS, tvOS. Supporta architetture ARM ed è basato su due linguaggi: Objective C, estensione di C++ e SWIFT
- Android: piattaforma sviluppata da Google attraverso la Open Handset Alliance. Basato su kernel Linux, supporta architetture ARM e x86. È basato su due linguaggi: Java e Kotlin.

2.2 Gestione della memoria

2.2.1 Gestione della memoria nei dispositivi tradizionali

Nei dispositivi tradizionali abbiamo una memoria virtuale, ogni processo sa che ha uno spazio di memoria a lui dedicato e può leggere e scrivere in quello spazio. Se i processi scrivessero dove vogliono, si scriverebbero uno sull'altro.

La memoria è organizzata in pagine, in blocchi. Quando il SO si accorge che la memoria finisce, prende alcune pagine e le sposta in memoria secondaria (swap). In questo modo libera la memoria principale. Lo swap è un'operazione costosa in termini computazionali, perché il tempo di lettura/scrittura di dati da memoria secondaria è diversi ordini di grandezza maggiore rispetto a quello da memoria principale.

2.2.2 Gestione della memoria nei dispositivi mobili

Nei dispositivi mobili esiste il concetto di memoria virtuale, esiste il concetto della paginazione ma non esiste lo swapping, ovvero i dati sulla memoria principale non vengono mai spostati sulla memoria secondaria.

Quando il sistema operativo termina la memoria principale, uno o più processi vengono terminati per liberare spazio.

Ad esempio stiamo scrivendo una mail e ci arriva una chiamata. Il SO avvia la schermata per gestire la chiamata, ma supponiamo che la memoria finisce. Il SO termina il client di posta, in questo modo però andiamo a perdere le informazioni perché una volta finita la chiamata il SO fa ripartire automaticamente il client di posta ma ciò che stavamo scrivendo è andato perso.

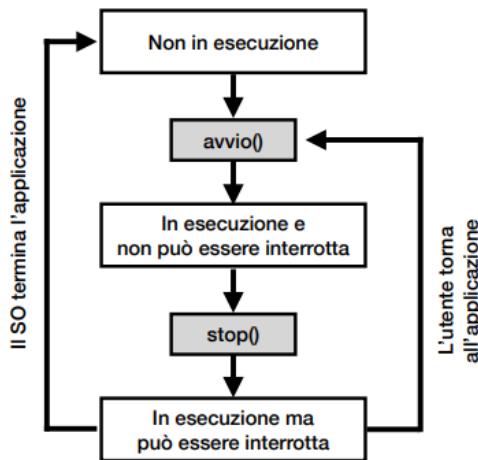
La soluzione è che SO e applicazioni collaborano per nascondere all'utente la terminazione del processo. È definito un ciclo di vita dell'applicazione organizzato in eventi. Il SO richiama questi eventi e ad ogni evento l'applicazione può eseguire un codice definito dal programmatore.

2.2.3 Ciclo di vita concettuale

Per gestire questa forma di collaborazione tra SO e applicazioni, ogni SO definisce un ciclo di vita delle applicazioni che specifica quali metodi vengono richiamati, dal SO, in quali situazioni. In questo modo il programmatore può implementare i metodi che servono in una specifica applicazione. La figura mostra uno schema astratto e semplificato di un ciclo di vita di un'applicazione. Nella realtà il ciclo di vita delle applicazioni è un po' più complicato, per permettere al programmatore di avere maggiore controllo e per evitare di svolgere operazioni non necessarie. Nella figura

i riquadri bianchi rappresentano lo stato dell'applicazione, mentre i riquadri grigi rappresentano i metodi che il SO operativo richiama quando l'applicazione cambia di stato. Gli stati sono tre:

- l'applicazione non è in esecuzione (nel senso che non esiste un processo)
- l'applicazione è in esecuzione ma non può essere interrotta dal SO per liberare memoria, per esempio perché sta eseguendo in foreground
- l'applicazione è in esecuzione e può essere interrotta dal SO per liberare memoria (per esempio perché non è visibile e non sta eseguendo codice in background)



2.2.4 Ciclo di vita in Android

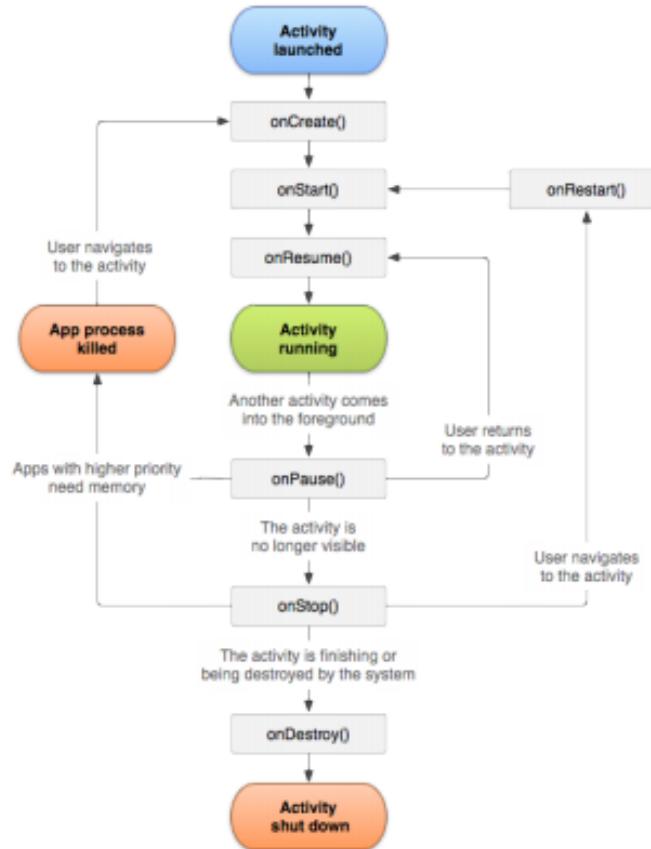
Il procedimento avviene in un modo più complicato.

Quando scriviamo un'applicazione è possibile scrivere dei metodi che gestiscono l'applicazione quando cambia di stato. In questo modo quando riceviamo una chiamata mentre stiamo scrivendo una mail, la chiamata viene messa in foreground e la mail in background. Quando l'applicazione viene messa in background viene eseguito il metodo che ha implementato il programmatore, ad esempio vengono salvati sul file il destinatario, il testo e l'oggetto della mail. Una volta terminata la chiamata, viene riaperta la mail e viene eseguito un altro metodo di cambio stato che va a leggere il file salvato in precedenza e la mail viene ripristinata.

In Android il ciclo di vita viene associato ad un'Activity, un oggetto che rappresenta una schermata dell'applicazione. Quando l'Activity viene avviata, vengono richiamati in ordine tre metodi:

- onCreate()
- onStart()
- onResume()

Sono questi i metodi che devono eventualmente ripristinare lo stato dell'applicazione.



2.2.5 SO tradizionale

Quando il SO necessita di memoria principale, sposta le pagine relative al processo (solo in parte o tutte) su disco. Queste pagine possono includere sia dati che effettivamente devono essere memorizzati (come il testo della mail), sia altri che magari non sono necessariamente da memorizzare, come ad esempio il codice dell'applicazione,

le informazioni sul layout grafico (es. gli oggetti che rappresentano i pulsanti e gli altri oggetti di interfaccia), dati multimediali che non devono essere salvati perché già disponibili su disco (es. un'immagine).

Può succedere che siano scritti su disco molti più dati di quelli che siano necessari. Questo avviene perché il SO non ha modo di sapere quali dati sia necessario salvare e quali no, in quanto questa sia un'informazione specifica alla logica dell'applicazione.

Il vantaggio di questa soluzione è che il programmatore non si preoccupa di scrivere il codice per salvare i dati dell'applicazione.

Nei dispositivi mobili il SO delega al programmatore l'onere di scegliere quali dati salvare e di ripristinare lo stato dell'applicazione quando questa viene fatta (ri)partire. Abbiamo diversi svantaggi:

- il programmatore deve scrivere del codice in più per salvare e ripristinare i dati
- potrebbero esserci comportamenti indesiderati, come una perdita di dati, nel caso in cui il codice scritto dal programmatore contenga degli errori
- l'applicazione può memorizzare dati in memoria persistente anche quando non viene terminata, svolgendo così operazioni non necessarie

Tra i vantaggi invece:

- l'applicazione riesce a memorizzare solo i dati strettamente necessari
- il SO può terminare dei processi senza dover salvare prima dei dati, perché questi sono già stati memorizzati

2.3 Astrazione hardware e software

Una delle funzioni principali offerte dai SO (sia per dispositivi tradizionali che mobili) è quella di permettere al programmatore di scrivere codice che possa funzionare su dispositivi differenti. Questa funzione si chiama *astrazione hardware*.

Il problema dell'astrazione è più complesso rispetto ai dispositivi tradizionali ed è causato dalla frammentazione:

- hardware: ci sono molti dispositivi differenti, spesso con caratteristiche molto diverse tra loro. Un utente dovrebbe verificare la disponibilità di una periferica prima di utilizzarla e prevedere un comportamento dell'app nel caso in cui non ci sia
- software

2.3.1 Frammentazione software

I processi interagiscono con il SO tramite chiamate dette API di SO. Queste API cambiano nel tempo quando cambia la versione del SO. Due versioni hanno alcuni metodi che possono rimanere invariati, altri che cambiano.

Le API vengono introdotte al fine di:

- supportare nuove periferiche HW
- supportare nuovi servizi
- risolvere problemi delle versioni precedenti

La soluzione al problema è che il programmatore, quando sviluppa un'app, deve specificare quali versioni del SO sono supportate, indicando la più vecchia e la più recente da supportare. L'ultima versione supportata tipicamente è l'ultima rilasciata, mentre per valutare la prima dobbiamo considerare:

- raggiungere il più utenti possibili
- maggiore è il numero di versioni che l'app deve supportare, maggiore potrebbe essere lo sforzo per realizzarla



Quando facciamo un nuovo progetto in Android, dobbiamo dire qual'è la minima versione di API che vogliamo supportare. Se sviluppo con l'ultima versione, devo gestire solo le API 29 e so che va fatto in un'unica maniera, però vado a raggiungere l'8.2% degli utenti. Se scelgo la versione 4.0, diventa veramente complicato gestire il codice di tutte e quante le versioni, per questo scelgo un compromesso, come ad esempio la versione 5.0.

Il problema è particolarmente sentito su Android perché alcuni dispositivi non permettono l'aggiornamento del SO.

Su iOS il problema esiste ma è meno marcato. Ciò non vuol dire che il problema della frammentazione non ci sia.

2.4 Gestione della sincronizzazione

I SO moderni sono multi-task, gestiscono più processi in esecuzione in contemporanea. Se abbiamo una CPU single core, la CPU alterna i processi facendo sembrare che i processi siano eseguiti in parallelo. Ora la maggior parte delle CPU sono multi-core ed eseguono i processi in parallelo. L'esecuzione in parallelo non avviene solo tra singoli processi, ma all'interno del singolo processo posso avere più thread che possono essere eseguiti in parallelo. In questo modo abbiamo dei problemi di sincronizzazione e dobbiamo assicurare che i thread collaborino e che non rimangano in attesa di qualcosa che non succeda mai (deadlock).

Nella programmazione dei dispositivi mobili e in generale per tutti i sistemi che hanno un'interfaccia grafica, la programmazione concorrente deve essere considerata in un contesto di programmazione ad eventi. I programmi hanno eventi che sono asincroni rispetto al codice, ad esempio il tap dell'utente sullo schermo. Non sappiamo quando accadrà l'evento, ma quando accade verrà eseguito un metodo.

Nei dispositivi mobili in ogni processo esiste un solo thread, *main thread*, che può modificare l'interfaccia grafica e gestire gli eventi generati dall'interfaccia utente. Gli altri thread secondari sono chiamati *background thread* e non possono modificare l'interfaccia utente altrimenti il sistema genera un'eccezione a runtime. Il main thread non deve rimanere bloccato, altrimenti non riuscirebbe a gestire l'interfaccia grafica.

Abbiamo ad esempio un'app che, al tocco su un pulsante, fa una comunicazione di rete. L'utente tocca il pulsante, il SO genera un evento che viene gestito dal thread principale e l'app effettua una comunicazione di rete. La chiamata è bloccante perché il thread rimane in attesa della risposta. Nessun altro thread può modificare la GUI perché il thread è in attesa.

Dobbiamo far sì che il main thread vada a generare un altro thread (thread secondario) che gestisca le operazioni che richiedono I/O e la computazione CPU bound. Quando il thread secondario termina non può aggiornare l'interfaccia grafica e deve far intervenire il main thread in quanto solo lui possa aggiornare. In questo caso abbiamo un problema di sincronizzazione. Questo può avvenire in vari modi:

- costrutti di sincronizzazione di basso livello: come wait, notify o i semafori. Sono complessi da utilizzare, soggetti ad errore e difficili da testare.
- costrutti di sincronizzazione di alto livello: introdotti per semplificare la gestione del codice nei casi tipici della gestione degli eventi, come ad esempio un thread secondario che può richiedere al thread principale di eseguire del codice.
- chiamate asincrone: in alcuni casi si scrivono metodi non bloccanti che al loro interno eseguono il codice bloccante.

2.4.1 Errori e soluzioni

2.4.1.1 Errori

Algoritmo 1. Esempio di errore: il main thread fa una chiamata bloccante

```

1 //Codice eseguito quando l'utente preme sul pulsante
2 result = sendRequest("myServer.com");
3 myTextField.setText(result);

```

Algoritmo 2. Esempio di errore: un thread secondario prova a modificare l'interfaccia grafica

```

1 //Codice eseguito quando l'utente preme sul pulsante
2 Crea un thread secondario che faccia questo {
3     result = sendRequest("myServer.com");
4     myTextField.setText(result);
5 }

```

Algoritmo 3. Esempio di errore: il main thread prova ad utilizzare il risultato della chiamata di rete prima che questo sia disponibile

```

1 //Codice eseguito quando l'utente preme sul pulsante
2 Crea un thread secondario che faccia questo {
3     result = sendRequest("myServer.com");
4 }
5 myTextField.setText(result);

```

2.4.1.2 Soluzioni

Algoritmo 4. Soluzione: uso di costrutti di sincronizzazione di alto livello

```

1 //Codice eseguito quando l'utente preme sul pulsante
2 Crea un thread secondario che faccia questo {
3     result = sendRequest("myServer.com");
4     esegui questo codice nel thread principale {
5         myTextField.setText(result);
6     }
7 }
```

Algoritmo 5. Soluzione: uso delle chiamate asincrone

```

1 //Definiamo prima questa chiamata
2 func manageResult(result) {
3     myTextField.setText(result);
4 }
```

```

1 //Codice eseguito quando l'utente preme sul pulsante
2 sendRequestAsync("myServer.com", manageResult);
```

2.5 Gestione del background

Nei dispositivi tradizionali, quando scriviamo app non ci poniamo il problema se il codice viene eseguito in foreground o in background. Il programmatore non deve fare nulla per fare funzionare il codice in background e non c'è differenza tra codice che viene eseguito in background e in foreground.

Nei dispositivi mobili c'è una differenza. Anche i dispositivi mobili permettono il multitasking, oltre al processo in foreground possono essere in esecuzione anche processi in background.

I processi in background hanno un forte impatto sulle performance delle app in foreground e sulla durata della batteria. Per questo i SO per dispositivi mobili pongono delle limitazioni all'esecuzione in background. L'ideale sarebbe non fare eseguire del codice in background ma questo non è possibile perché ci sono sistemi di messaggistica che devono poter ricevere messaggi anche quando l'app non è in foreground. Un altro esempio ne è un sistema che deve riprodurre musica anche in background.

In tutti i casi in cui un'applicazione necessita di ricevere dati in modo asincrono via rete (es: sistema di messaggistica), viene adottato un protocollo di comunicazione

chiamato *push notification*, che permette all'applicazione di non dover eseguire codice in background.

In altri casi come l'app di musica, non è possibile evitare di eseguire il codice in background.

2.5.1 Background in iOS

Apple ha da sempre avuto un approccio molto restrittivo rispetto al background, le applicazioni non possono eseguire codice arbitrario in background se non in risposta ad alcuni specifici eventi.

Ad esempio abbiamo un'applicazione che ha bisogno di conoscere la posizione dell'utente anche in background, può registrarsi per essere notificata dal SO quando la posizione cambia. Il SO ogni volta che osserva un cambio di posizione, notifica l'app che ha poi un tempo ridotto per gestire l'evento e se non lo gestisce entro questo tempo, il SO termina il processo.

2.5.2 Background in Android

Ci sono due categorie di processi in background:

- quelli che hanno effetti immediati sull'utente (es: la musica, app che mostrano informazioni saltuariamente all'utente, come le app di navigazione) possono eseguire codice arbitrario anche in background
- quelli che non hanno effetti immediati (es: compatimento di un database) possono eseguire codice solo attraverso uno strumento (SW) di pianificazione delle attività: il SO eseguirà quelle attività quando possibile (es: quando disponibile una connessione WiFi, quando la batteria è carica, ecc.)

Reti e architetture

———— Lezione 3 - 6 ottobre 2020 ——

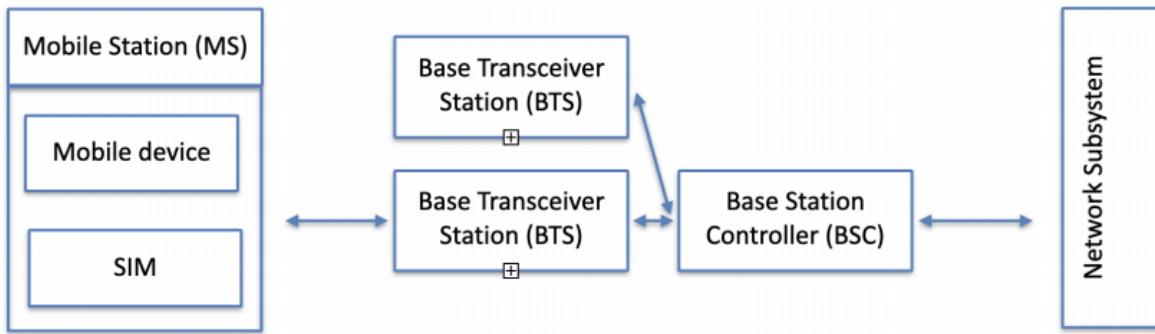
3.1 Le reti cellulari

Le reti cellulari permettono ad un dispositivo mobile di trasmettere voce e dati attraverso un'infrastruttura distribuita nel territorio composta da:

- Antenne: sono sparse nel territorio ed ognuna di queste ha una copertura limitata. Per permettere ai dispositivi di avere una copertura sufficiente (una copertura geografica), ci sono molte antenne
- Varie componenti di elaborazione delle informazioni

Nascono negli anni '80 e si sono rapidamente evolute. Esempi di tecnologie: GSM, GPRS, EDGE, LTE.

Queste tecnologie sono organizzate in generazioni (es. 1G, 2G, 3G, ecc.) dove ogni generazione stabilisce delle performance di riferimento. Le diverse tecnologie condividono una stessa architettura.



La *SIM* è una componente HW che ha lo scopo di identificare e autenticare l'utente per l'accesso alla rete. Le *Mobile Station (MS)* includono due componenti: la SIM e il device. La SIM contribuisce a funzioni di sicurezza in quanto contiene chiavi

crittografiche utilizzate per realizzare la comunicazione di rete.

La mobile station comunica via radio con una componente chiamata *Base Transceiver Station (BTS)* che include una o più antenne per la comunicazione e un hw di gestione dell'informazione per gestire il segnale in ingresso e in uscita. Ogni BTS è collegata al *Base Station Controller (BSC)* che coordina varie BTS e a sua volta comunica con l'infrastruttura di rete chiamata *Network Subsystem*.

La potenza del segnale diminuisce all'aumentare della distanza tra la MS e l'antenna. La distanza massima di comunicazione dipende dal tipo di tecnologia, ma in linea di massima entro la quale si può comunicare è anche di diversi chilometri. Per erogare il servizio su una scala geografica, sono sparse nel territorio le BTS. Ciascuna BTS definisce una cella, cioè una regione geografica dove il segnale di quella BTS è più forte, rispetto al segnale delle BTS circostanti. Man mano che ci allontaniamo dall'antenna entriamo in un'altra cella perché il segnale di quest'altra antenna è più forte della precedente.

Ogni BTS comunica con varie MS tramite onde radio, su intervalli di frequenze che sono specificati dal protocollo utilizzato. Ogni intervallo è suddiviso in un numero finito di portanti che sono a loro volta suddivise in uplink e downlink (comunicazione verso la BTS). Ad esempio GSM ha 248 portanti nell'intervallo attorno ai 900MHz.

Ogni portante è suddivisa in canali usando tecniche di FDM (frequency division multiplexing) e TDM (time division multiplexing).

Ogni MS nel momento in cui comunica con la BTS necessita di un canale in uplink e uno in downlink. Il numero massimo di MS che possono comunicare con una stessa BTS è limitato. In zone ad alta densità di popolazione è necessario avere celle più piccole, per suddividere la popolazione tra più BTS. In questo caso una procedura, chiamata session handover, permette il passaggio di controllo di una mobile station ad un'altra e il dispositivo non si accorge di essere passato da un'antenna all'altra. L'IP del telefono rimane invariato.

3.2 Le architetture per dispositivi mobili

Molto spesso le applicazioni di dispositivi mobili fanno parte di un sistema distribuito e comunicano con un server ed altri dispositivi mobili.

Ci sono fattori che influenzano molto la progettazione della app:

- la connessione potrebbe andare persa. Nei dispositivi mobili è anche normale che l'accesso ad Internet avvenga tramite diversi tipi di connessione che cambiano nel tempo
- il tipo di connessione cambia. In alcuni casi grazie al session handover è possibile

mantenere la stessa connessione anche quando un dispositivo si sposta all'interno di una rete cellulare, ma la stessa funzionalità non è disponibile quando un dispositivo si sposta tra reti diverse e dunque riceve un nuovo indirizzo IP. Ad esempio un utente è connesso alla rete WiFi di casa, quando esce si connette alla rete cellulare e quando arriva a lavoro si connette ad un'altra rete WiFi che gli assegnerà un nuovo IP

3.2.1 Connection-oriented vs connection-less

I due fattori portano a sconsigliare l'utilizzo di protocolli connection-oriented dove si ha una connessione prolungata tra le due entità che comunicano. Non è consigliabile perché il dispositivo perde la connessione e siccome ho una connessione aperta tra client server, se il client cambia l'indirizzo IP, la connessione va persa.

Nei protocolli connection-less invece non viene mantenuta una connessione prolungata tra due componenti. Queste comunicazioni sono più adatte ai dispositivi mobili.

In un'architettura client-server, il server espone delle API (chiamate) che i client possono sfruttare per portare a termine il loro compito.

3.2.2 Architettura three-tier

Lo schema architetturale three-tier è comunemente utilizzato per erogare servizi web. Il browser fa una richiesta HTTP al web server che legge e scrive i dati su un DB. Una volta che il web server termina la scrittura e la lettura dei dati, esegue lo script lato server, recupera il file necessario ed esegue il codice PHP che può richiedere di interagire con una base di dati. Il web server al browser ritorna la pagina html, il codice javascript che esegue lato client e dei css.

PHP è lo scripting lato server e non viene eseguito lato client, javascript invece è lo scripting lato client e non viene eseguito lato server.

L'architettura per i dispositivi mobili è molto simile. L'applicazione comunica con un web service tramite HTTP. Il web spesso comunica con un DB che memorizza l'informazione in modo persistente. Una delle differenze principali è che l'applicazione client solitamente riceve dal web service solo i dati richiesti, in quanto il comportamento dell'applicazione, incluse le informazioni su come formattare i dati ricevuti, fanno già parte dell'applicazione stessa. I dati possono essere scambiati in un qualunque formato, incluso il testo semplice. Tuttavia, per dati complessi esistono vari formati, tra cui XML e JSON.

I modelli presentano due componenti diversi:

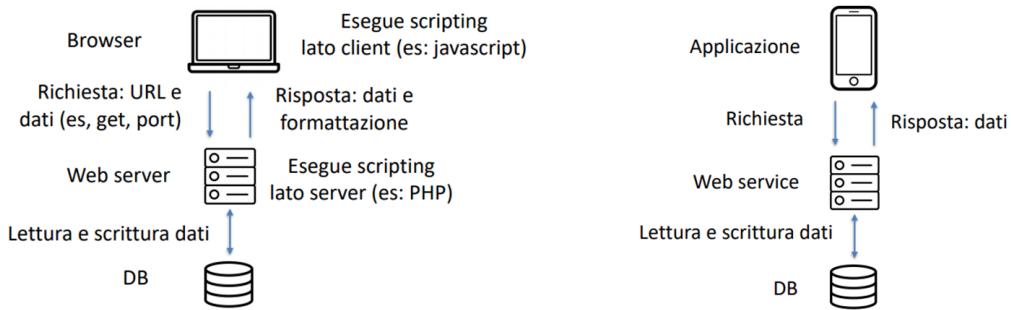


Figure 3.1: Schema del modello three-tier applicato al web (sinistra) e alle applicazioni (destra)

- **web server:** un server che fornisce informazioni (generalmente HTML+CSS+JS) finalizzate ad essere mostrate (tramite un browser) all'utente
- **web service:** un server fornisce informazioni finalizzate ad essere ricevute da un'applicazione (anche mobile)

Quando facciamo un sistema vero, vogliamo far sì che un sistema sia utilizzabile sia su app che su web e vorremmo evitare di scrivere due volte il codice. Quello che si fa è usare un web server che manda tutti i contenuti al browser e che gli permettono di avere lo stesso comportamento dell'app mobile.

Invece di avere web service e web server, abbiamo un web service che interagisce con l'applicazione e poi il web server che passa semplicemente al browser la web app.

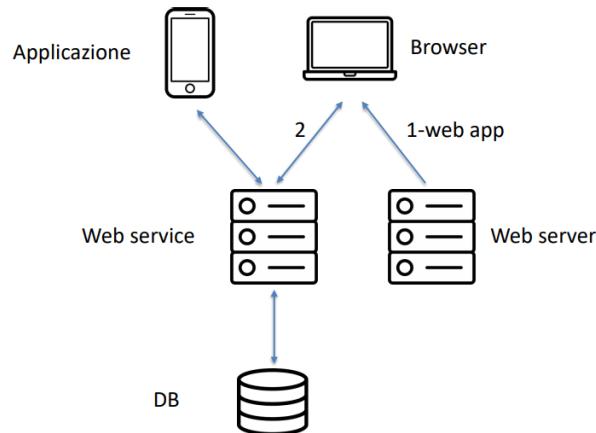


Figure 3.2: Architettura per supportare contemporaneamente l'accesso da browser e da applicazioni

3.2.3 La definizione del protocollo di comunicazione

Progettare un protocollo di comunicazione non è facile. Vogliamo definire API sul server con le quali i client possono interagire: scambiare i dati con il server e ottimizzare il volume di dati scambiati, tenendo conto dei vincoli dei dispositivi mobili e permettendo una buona esperienza utente.

Il protocollo è di tipo connection-less. Le chiamate avvengono con connessioni diverse, ma solitamente sono logicamente legate le une alle altre. Bisogna definire in quale ordine avvengono, in quali situazioni, quali dati si devono scambiare, ecc.

3.2.3.1 Esempio di comunicazione

Pensiamo ad un social. Ogni utente ha un profilo (con nome e foto) e uno stato (online o offline). Vogliamo che un utente possa scaricare la lista degli amici.

Il modo più semplice per implementare questa cosa è richiedere al server l'elenco dei contatti. È una soluzione semplice, ma ogni volta che l'utente vuole accedere alla lista, deve riscaricare tutti i contatti (con la foto, il nome e lo stato).

Un'altra soluzione sarebbe di andare a scaricare in locale gli amici, in modo tale che una seconda volta scarico solamente quelli che precedentemente non avevo ancora scaricato (ad esempio un nuovo amico). Qui ho un altro problema, perché se un amico aggiorna la foto profilo, io in locale ho quella vecchia.

La soluzione corretta e ottimizzata è quella di utilizzare un contatore. Per ogni utente il server gestisce un contatore delle foto di profilo caricate (un numero di versione). Quando un utente scarica l'elenco degli amici il server gli manda, per ciascuno, il numero di versione della foto (è un dato di dimensione irrilevante rispetto ad un'immagine). Per ogni amico verifico se ho già salvato in locale la foto giusta e nel caso la mostro, altrimenti chiedo la foto al server e poi la salvo in locale per la prossima volta.

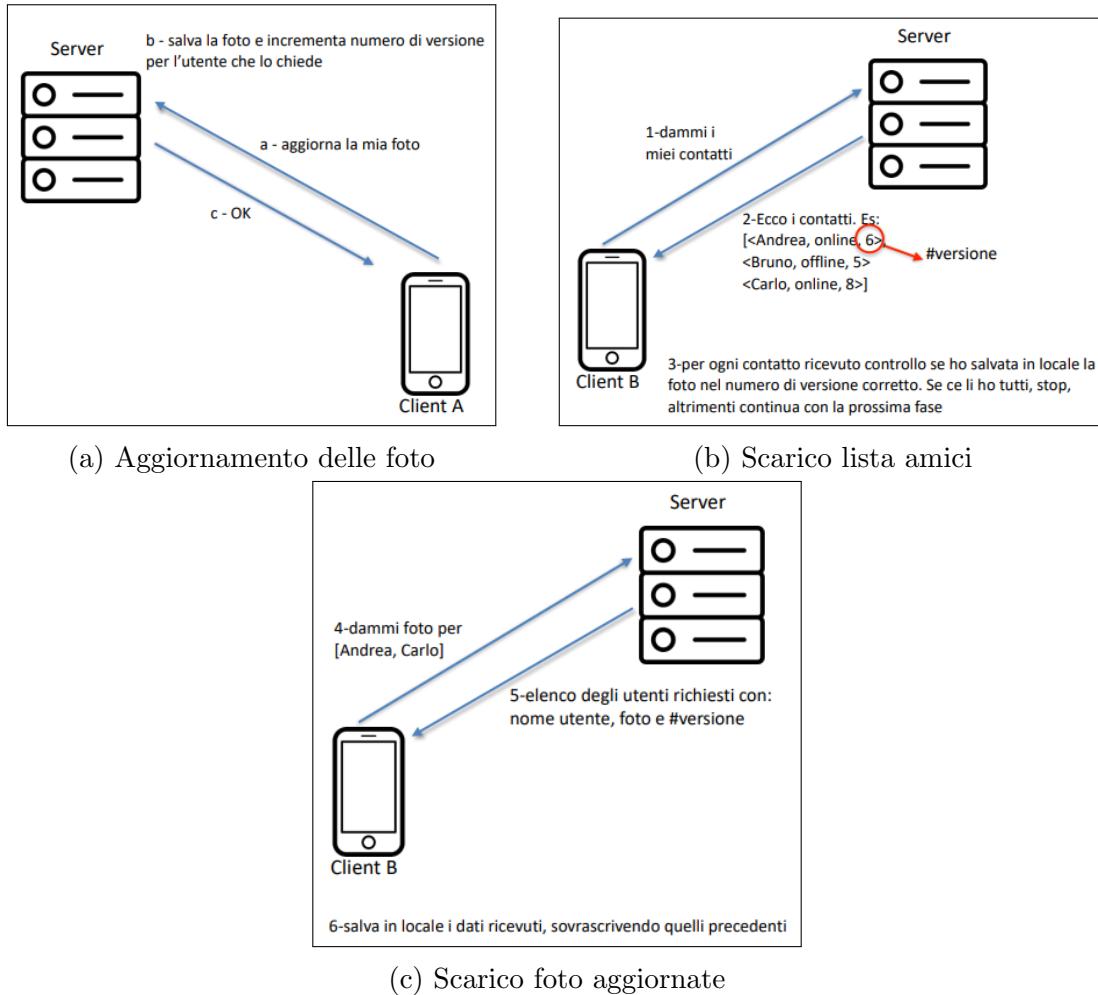


Figure 3.3: Trasmissione delle immagini profilo in un social network

3.3 La codifica dell'informazione

In una richiesta HTTP è possibile trasmettere informazioni in testo semplice.

Prendiamo ad esempio un servizio che prende una parola in input e risponde con un sinonimo. L'output è sempre una stringa e se non viene trovata nessuna corrispondenza, ritorna la stringa vuota.

Prendiamo ad esempio, invece, un servizio che data una parola in input risponde con una lista sinonimi. In questo caso potremmo ad esempio usare una codifica di parole separate da virgolette.

Diventa peggiore se data una parola in input, risponde con una lista di sinonimi e una

di contrari. Dobbiamo rappresentare due liste di parole e potremmo inventarci un modo per separare le due liste, per esempio usare il punto e virgola per separare.

Esistono degli strumenti che ci permettono di codificare l'informazione come ad esempio XML e JSON.

XML è un linguaggio di markup che permette di definire formati dei dati che sono “human and machine readable”.

Un documento XML è ben formattato se rispetta alcune regole (ad esempio se tutti i tag sono aperti e poi chiusi). XML presenta alcuni limiti:

- tende ad essere verboso
- non facilmente leggibile da umani
- gli schemi XML sono abbastanza difficili da definire e spesso vengono ignorati

Un esempio di risposta XML per il servizio dictionary:

```

1   <dictionary entry="nice">
2       <synonyms>
3           <word>pleasant</word>
4           <word>agreeable</word>
5           <word>enjoyable</word>
6       </synonyms>
7       <antonyms>
8           <word>ugly</word>
9           <word>unpleasant</word>
10      </antonyms>
11  </dictionary>
```

JSON concettualmente svolge lo stesso lavoro di XML, cerca di ridurre la verbosità e di migliorare la lettura.

Un esempio di risposta JSON per il servizio dictionary:

```

1  {
2      "entry": "nice",
3      "synonyms": ["pleasant", "agreeable", "enjoyable"],
4      "antonyms": ["ugly", "unpleasant"]
5 }
```

XML definisce delle entità, JSON definisce degli oggetti. Sono due rappresentazioni diverse, ma concettualmente sono analoghi, infatti rappresentano i dati di un elemento che vogliamo descrivere.

Un'entità in XML o un oggetto JSON mi definisce uno stato nel mondo reale ma non il comportamento, nei linguaggi di programmazione invece un oggetto definisce sia lo stato che il comportamento ad esso associato. Questa distinzione è molto importante in quanto esiste una forte correlazione tra le strutture dati dei programmi e la codifica in XML o JSON.

3.3.1 Serializzazione e deserializzazione

In memoria principale lavoriamo con oggetti dei linguaggi di programmazione ma questi oggetti non possono transitare in rete. Quello che ci serve è andare a convertirli nella propria rappresentazione XML o JSON. Questa operazione si chiama *serializzazione* (o *marshalling*). Il risultato è una stringa (internamente codificata in XML o JSON). L'entità che riceve la comunicazione di rete può leggere questa stringa e riconvertirla in struttura dati tramite un'operazione che si chiama *deserializzazione* o *unmarshalling*. Uno dei vantaggi nell'uso di XML o JSON è che le operazioni di serializzazione e deserializzazione sono supportate, in quasi tutti i linguaggi di programmazione, da apposite librerie. Un esempio di queste librerie è GSON.

GSON è lo strumento di conversione, JSON è il formato di rappresentazione.

3.3.2 Codifica di dati binari

Sia in XML che in JSON vengono rappresentati i dati singoli nella loro codifica testuale. Non c'è nessun problema per stringhe, interi, float, boolean, ecc, ma si pone un problema per i dati binari ad esempio le immagini. Il modo più comune è la codifica Base64 che permette di rimappare la codifica binaria dell'informazione in alcune informazioni testuali.

Definisco una conversione tra 64 simboli (caratteri ASCII) e un valore numerico: A=1, B=2, ecc. Scelgo dei caratteri ASCII che non mi introducano confusione nelle rappresentazioni XML o JSON (es: non uso "<", né ","). Un simbolo rappresenta dunque 6 bit ($2^6 = 64$). Dunque per rappresentare 3 byte in binario mi servono 4 simboli ($3*8 = 4*6$). Per convertire da binario a Base64 suddivido l'input in gruppi di 3 byte, codifico ciascun gruppo con 4 simboli Base64. Si usa la Base64 perché è una codifica testuale che usa meno caratteri per rappresentare e perché permette di non utilizzare caratteri strani.

3.3.3 Protocol Buffer

Protocol Buffer è il protocollo definito da Google per serializzare le informazioni. Risolve lo stesso problema di XML e JSON definendo non solo il formato di scambio, ma anche gli strumenti SW di (de)serializzazione.

Protocol Buffer è un protocollo:

- multi-piattaforma, multi-linguaggio
- ottimizzato in termini di dimensione del messaggio da scambiare
- che risolve diversi problemi tipici ad esempio la gestione della versione dei protocolli

Lezione 4 - 7 ottobre 2020

3.4 Le notifiche push

Nei protocolli di comunicazione abbiamo visto che è preferibile fare in modo che sia un'applicazione per dispositivi mobili a contattare un server e non viceversa. Ci sono però dei casi in cui non è possibile, se c'è un evento che si genera sul server, è il server stesso che deve contattare il client.

Ho diverse soluzioni:

- mantenere una comunicazione aperta rispetto al server, ad esempio via TCP.
Questa soluzione non va bene, è meglio evitare approcci connection-oriented
- il client rimane in attesa di connessione, esempio con un server socket. Anche questa soluzione non va bene, il client non dovrebbe rimanere in ascolto di connessioni da parte del server
- il client fa polling: "hai qualche messaggio per me?" La soluzione non va bene in quanto sia costosa, dato che il client continua ad interrogare il server senza ricevere, nella maggior parte dei casi, nessun aggiornamento
- BOSH: permette al client di fare una richiesta HTTP al server e risponde subito se ha qualcosa. Se non ha niente, tiene la richiesta HTTP in attesa fino a quando non ha nulla da comunicare. Può succedere che il client si disconnetta o che il client cambi IP. La richiesta fallisce e il client ne fa una nuova. Anche qui la soluzione richiede che il client sia sempre in esecuzione

C'è una soluzione ottimale, rispetto a quelle precedenti, che è quella delle *notifiche push*. Si tiene un solo servizio attivo a livello di SO per ricevere comunicazioni asincrone. Questa soluzione fa in modo che sia il SO a rimanere in attesa di comunicazioni da un server esterno, dunque con una sola connessione, offrendo poi il servizio alle altre applicazioni.

3.4.1 Componenti delle notifiche

Ci sono diverse componenti coinvolte:

- l'**utente** che usa il dispositivo mobile
- **SO** del device
- **app** che deve ricevere la notifica
- **push server**, un servizio esterno che comunica con il SO (sviluppato da Apple e Google). Offre il servizio di push notification
- **app server**, un server che vuole inviare comunicazioni asincrone all'app

3.4.2 Il protocollo

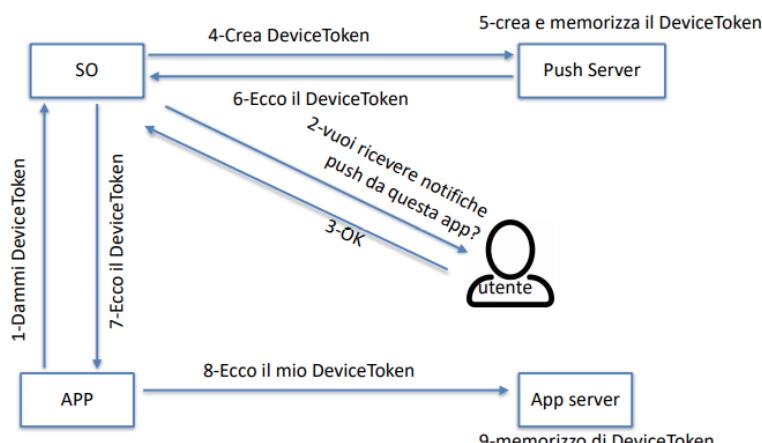
Il protocollo alla base delle notifiche push è suddiviso in due fasi: setup e invio.

È necessario introdurre un elemento fondamentale del protocollo, il *device token* che identifica la coppia data dal dispositivo e dell'applicazione (< *applicazione, device* >) che vuole ricevere le notifiche push. Il device token è creato dal push server usando tecniche crittografiche che garantiscono che il device token non possa essere creato da altri e che, una volta creato, non possa essere modificato.

3.4.2.1 La fase di setup

La fase di setup viene iniziata dall'app. Voglio permettere all'app server di mandare delle notifiche. Lo scopo è quindi quello di fare arrivare il device token all'app server.

Il procedimento è il seguente.



L'app fa una chiamata di API al SO che chiede all'utente se vuole autorizzare l'applicazione ad inviare le notifiche push. Se l'utente non autorizza, il procedimento finisce, se invece autorizza il SO chiede al Push Server di creare il DeviceToken. Il Push Server lo crea, lo memorizza e lo manda al SO. Il SO risponde all'app dando il DeviceToken.

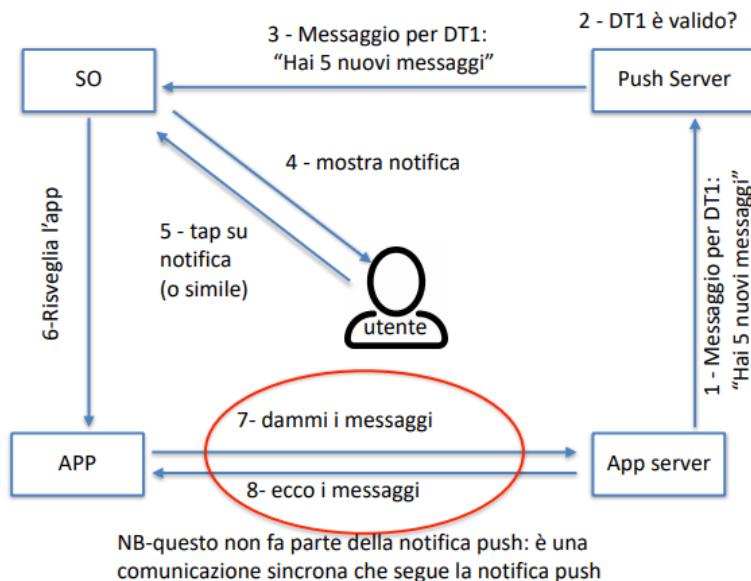
La richiesta "1. Dammi DeviceToken" è una chiamata bloccante oppure asincrona. L'app riceve il DeviceToken e lo comunica all'App server.

3.4.2.2 Fase di invio

La fase di invio delle notifiche, invece, viene eseguita ogni volta che l'app server vuole mandare una comunicazione asincrona all'applicazione.

Lo scopo è quindi di fare arrivare, tramite l'app server, all'app un messaggio con un piccolo payload. Il payload è il testo da mostrare ed eventuali altre informazioni come ad esempio "Hai 5 nuovi messaggi".

Il procedimento è il seguente.



L'app server contatta il push server dicendo che ha un nuovo messaggio e allega due informazioni: il device token memorizzato in fase di setup e i payload. Il messaggio tra app server e push server è un messaggio tra due computer in rete. Il push server verifica che il device token sia valido e se è valido contatta il SO in modo asincrono, per esempio tramite l'implementazione di un protocollo simile a BOSH. Se il push server non riesce a contattare il SO, memorizza l'informazione della richiesta (ad esempio quando

il telefono è spento o non ha connessione). Il SO mostra la notifica all'utente con anche il testo contenuto nel payload. L'utente fa il tap sulla notifica e il SO risveglia l'app.

3.4.3 Attacchi

Il protocollo alla base delle notifiche push fornisce garanzie di sicurezza più elevate rispetto ad altri servizi (ad esempio l'email). Il push server può agire attivamente per prevenire gli attacchi. Alcuni esempi di attacchi che si possono prevenire sono:

- **spamming**: un'app server non può inviare notifiche push ad un client che non lo ha autorizzato, perché non dispone del device token
- **impersonificazione**: un'app server non può far finta di essere un altro app server (es: telegram non può mandare una notifica push facendo finta di essere whatsapp). Questo aiuta a prevenire gli attacchi di tipo phishing (truffe)
- **messaggi indesiderati**: se un app server inizia a mandare troppi messaggi o messaggi indesiderati all'utente, l'utente ha la possibilità di revocare il device token
- **flooding**: il push server può bloccare le richieste, se ad esempio un app server (anche autorizzato) prova ad inviarne un numero eccessivo allo stesso device

3.4.4 Notifiche push e locali

I SO rendono disponibili due tipi di notifiche: push e locali. È facile fare confusione perché sono presentate in modo simile all'utente, ma sono completamente differenti da un punto di vista architettonale. Le notifiche locali vengono generate dall'app stessa mediante delle chiamate al SO, di solito per dare informazioni all'utente quando l'app è in background (es: app di navigazione), oppure per mandare reminder all'utente ad orari prestabiliti.

Calcolo e uso della posizione

4.1 Introduzione all'uso dell'informazione di posizione

La posizione ci indica dove si trova un dispositivo in un ambiente di riferimento che può essere nel mondo, in un edificio, ecc.

Possiamo avere 2 dimensioni, dove la posizione è considerata rispetto ad un piano, oppure 3 dimensioni, dove la posizione è considerata rispetto ad un piano ed un'altezza.
Possiamo rappresentare la posizione attraverso la:

- rappresentazione geografica: la posizione nel mondo
- rappresentazione in coordinate locali: la posizione ad es. su una mappa
- rappresentazione semantica: es. indirizzo

Nella rappresentazione geografica ci sono due sistemi di riferimento:

- latitudine: viene preso come punto di riferimento l'equatore
- longitudine: viene preso come riferimento il meridiano di Greenwich

L'informazione di posizione non riporta il tempo, ma l'informazione temporale è fondamentale da associare alla posizione in molti contesti, ad esempio se vogliamo descrivere una traiettoria, cioè una lista di elementi <timestamp, position>.

Il calcolo della posizione (e del tempo) non è esatto. In termini matematici potremmo definire una funzione, di distribuzione di probabilità, che associa ad ogni punto dello spazio la probabilità che il device si trovi lì. In termini pratici, si modella un'area dove probabilmente si trova il device. Questa approssimazione però non ci dice dove si trova il device all'interno dell'area.

4.1.1 Accuratezza e precisione

Una delle caratteristiche più rilevante di un sistema di calcolo della posizione è quanto la posizione calcolata differisce da quella reale.

Supponiamo che l'utente sia fermo e che siano date varie misurazioni della sua posizione. Con *precisione* si intende quanto le misurazioni sono vicine l'una all'altra. Con *accuratezza* si intende quanto le misurazioni siano vicine alla posizione vera.

L'errore di posizionamento viene calcolato in questo modo: indico che la posizione calcola ha un $x\%$ di probabilità di probabilità di essere più vicina di y metri alla posizione corretta.

Il livello di precisione non sempre è lo stesso, dipende dal tipo di applicazione e dal tipo di utente. Possono avere una precisione a livello di:

- regione (fino a 200km): meteo, news, etc...
- area metropolitana (fino a 20km): local news, traffico, ecc.
- quartiere-città (fino a 2km): gestione flotte
- isolato (50-100m): gestione emergenze, pubblicità geo-referenziata, informazioni sui POI (Point Of Interest)
- dintorni dell'utente (5-50m): navigazione outdoor, taxi, car sharing
- prossimità dell'utente (1-5m): navigazione indoor

Oltre alla posizione siamo anche interessati a conoscere l'orientamento, che può assumere diversi significati:

- come è orientato, sui tre assi, il device
- in quale direzione (2D) è direzionato l'utente (heading). Heading lo posso misurare con i sistemi inerziali
- in quale direzione (2D) si sta muovendo l'utente (course)

Ad esempio quando ci spostiamo con il navigatore e non sa dove stiamo guardando, finché non iniziamo a muoverci non riesce a darci le indicazioni giuste.

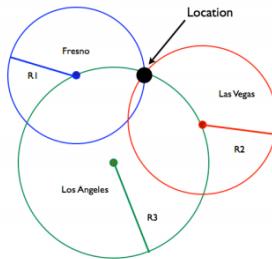
Possiamo capire l'orientamento attraverso:

- heading: uso i sensori inerziali per capire dove sta puntando il device
- course: vedo in quale direzione si sta spostando l'utente. Posso calcolarlo solo quando ho una traccia di spostamento dell'utente

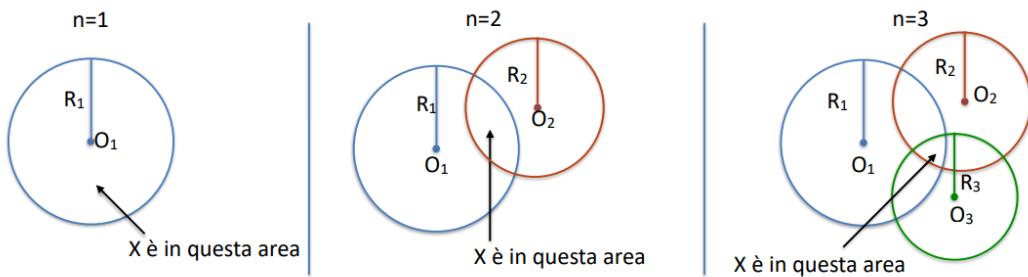
4.1.2 Calcolo della posizione con trilaterazione

La trilaterazione assume che io debba calcolare la posizione di un oggetto sapendo la distanza da altri oggetti e conoscendo la loro posizione. Ci sono diverse varianti:

- numero degli oggetti dai quali conosco la distanza
- in che termini conosco la distanza tra X e O_i ?
 - **distanza esatta**: se conosco la distanza esatta di X da 3 oggetti, posso sapere qual è la posizione di X.

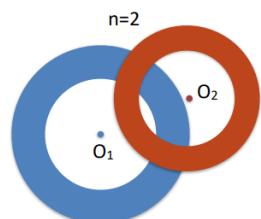


- **upper bound** della distanza: "la distanza è al più...". Molte tecniche forniscono una distanza massima del device da una posizione nota. Se ho un solo oggetto so che la posizione sarà all'interno della circonferenza. Se ho due o più oggetti, la X starà nell'intersezione.



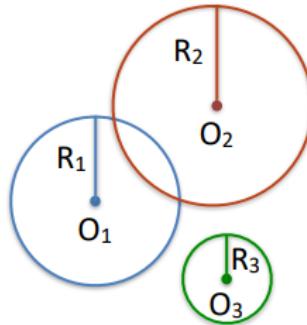
- **range di distanza**: "la distanza è compresa tra ...".

Possiamo definire il problema in modo più generale considerando che per ogni oggetto O_i conosco una distanza minima e massima rispetto ad x



In tutti i casi abbiamo una stima della distanza, ma la stima può essere errata. Il sistema di calcolo della posizione deve essere in grado di gestire anche questi casi, ad esempio provando ad identificare la stima errata sulla base delle altre.

$n=3$



Dove sta X?

In questo esempio posso stimare che la posizione di X sia nell'intersezione tra R1 ed R2 ma nel punto più basso, perché devo anche considerare R3.

4.2 Calcolo della posizione outdoor

La posizione outdoor si calcola con 3 soluzioni:

- Global Navigation Satellite System (GNSS)
- Rete Cellulare
- WiFi Network based

Queste tecniche da sole non funzionano, devono essere combinate con altre tecniche ibride.

4.2.1 Global Navigation Satellite System (GNSS)

Ci sono una serie di satelliti che viene messa in orbita e in ogni istante possiamo sapere la loro posizione. Hanno a bordo un orologio atomico che gli permette di sapere l'ora in modo preciso. Mentre i satelliti ruotano, inviano il proprio id insieme all'informazione temporale.

Il dispositivo mobile riceve l'informazione e riesce a calcolare la distanza da uno o più satelliti. Sfruttando il tempo di propagazione del segnale posso conoscere la distanza

dal satellite (la cui posizione può essere calcolata), quindi si usa una tecnica simile alla trilaterazione per calcolare la posizione.

Abbiamo un problema. Il device non ha un orologio atomico, quindi deve essere sincronizzato e richiede di ricevere il segnale da almeno 4 satelliti. Posso calcolare così la mia posizione nello spazio e la mia posizione precisa. Possiamo avere anche problemi legati a:

- interferenze: in particolare atmosfera (condizioni meteo) e edifici
- posizione dei satelliti potrebbe non essere esattamente quella prevista

Ci sono diversi sistemi in uso:

- GPS (USA): è in grado di fornire una posizione maggiore ma per motivi di sicurezza, la precisione sotto ai 3 metri viene usata in ambiti militari
- GLONASS (Russia)
- Galileo (Europa): fornisce una posizione sotto al metro

Ci sono due modi per diminuire i problemi del segnale satellitare:

- Differential GPS (D-GPS): è un sistema che permette di correggere (almeno in parte) l'errore dovuto alle interferenze dell'atmosfera e ad una non precisa posizione dei satelliti rispetto a quanto previsto. Consiste nell'avere stazioni terrestri collocate in posizioni note che misurano la potenza del segnale dei satelliti. Un ricevitore in posizione fissa a terra calcola la distanza tra il segnale atteso e quello che effettivamente riceve. Il ricevitore poi comunica questa informazione ai dispositivi mobili che dunque possono compensare l'errore
- Assisted GPS (A-GPS): il calcolo iniziale della posizione può essere molto lungo (anche qualche minuto) perché il device deve capire quali satelliti sono in vista. A-GPS risolve questo problema:
 - ogni antenna della rete cellulare mantiene un elenco dei satelliti in vista
 - quando il device deve usare GPS, richiede, tramite un apposito servizio, quali sono i satelliti in vista all'antenna più vicina. Saranno gli stessi in vista al device (l'antenna non può essere troppo lontana)

Questo riduce notevolmente il tempo di calcolo della posizione che in genere è di pochi secondi

4.2.2 Rete cellulare

Un altro modo per calcolare la posizione è attraverso la rete cellulare. Ogni device è connesso ad un'antenna di cui sono note la posizione e l'area della relativa cella. Il problema è che l'approssimazione è molto alta e dipende dalla dimensione della cella. Una prima soluzione è il protocollo GSM, che prevede l'uso di una tecnica per stimare la distanza tra il device e l'antenna. Lo scopo all'interno di GSM è quello di evitare collisioni, c'è quindi una tecnica che permette di stimare la distanza tra il device e l'antenna.

Oltre al protocollo GSM, possiamo sviluppare tecniche per stimare la distanza tra un device mobile e un'antenna dove usiamo il ritardo id propagazione del segnale per stimare la distanza tra un device mobile e un'antenna di rete cellulare. La precisione è nell'ordine di 50-125 metri.

4.2.3 WiFi positioning

Ci serve conoscere la distanza tra il device e gli Access Point e la posizione di questi.

Il protocollo 802.11 (protocollo di base alle reti WiFi) prevede che gli access point comunichino il proprio ID, anche ai device che non sono connessi/autenticati a quella rete. Possiamo sapere quali sono i device che sono nelle vicinanze e la loro relativa potenza del segnale. Minore è la potenza del segnale, significa che sono lontano, maggiore è la potenza del segnale e più sono vicini.

4.2.3.1 Problemi

La potenza del segnale è approssimativamente proporzionale alla distanza perché ci potrebbero essere delle interferenze, infatti se un device e un AP sono vicini, ma c'è qualcosa che crea interferenza, la potenza del segnale sarà bassa.

Un altro problema è che non sappiamo dove sono posizionati gli AP, per questo usiamo un approccio basato su crowdsourcing, un modo di raccogliere i dati dagli utenti che usano il servizio. Il crowdsourcing può richiedere o meno un'azione esplicita da parte degli utenti. Quando l'utente ha il GPS attivo, calcola la propria posizione con una buona precisione e comunica ad un location service (server) la propria posizione e gli access point nelle vicinanze. Il location service riceve le informazioni da più utenti e stima la posizione degli access point. Quando un utente non ha il GPS attivo:

- invia al location service l'elenco degli AP nelle vicinanze (e la potenza del segnale)
- il location service calcola la posizione dell'utente e gliela comunica

Ogni volta che il device vuole calcolare la posizione, guarda quello che ha e lo comunica al location service che risponde con una posizione. Questa è una tecnica ibrida perché è basata su due tecnologie, GPS e WiFi.

Chi calcola la posizione usando il sistema satellitare? Device o server?

Il device ha l'antenna e può calcolarlo in autonomia.

Per il calcolo della posizione WiFi non ho in locale la conoscenza delle posizioni degli AP e quindi ho la necessità di comunicare con un location service.

———— Lezione 5 - 9 ottobre 2020 ———

4.3 Gestione dei dati di posizione

Quando il device mobile ha calcolato la posizione, la può usare localmente oppure la può inviare ad un server.

In entrambi i casi si possono adottare tecniche per gestire il dato di posizione. Le più comuni sono:

- Geocoding e reverse-geocoding: entrambi vengono svolti dal server, geocoding converte l'indirizzo geografico in coordinate spaziali, reverse geocoding è l'operazione inversa
- Geofencing: viene svolta in locale dal device, si definisce un'area e il device tiene monitorata la posizione dell'utente quando si entra e si esce dall'area (ad esempio attraverso una notifica)
- calcolo delle distanze: esiste una formula che calcola la distanza dalla terra (calcolo in linea d'aria)
- calcolo delle distanze su rete stradale: non abbiamo un calcolo in linea d'aria ma si modella la rete stradale come un grafo dove:
 - i nodi sono le intersezioni
 - gli archi sono i segmenti di strada

Ogni arco è etichettato con la distanza geografica tra i due nodi oppure con il tempo necessario per andare da un nodo all'altro calcolato come:

- tempo: distanza / limite di velocità (non molto affidabile, dipende dal traffico e altri fattori)
- tempo di percorrenza medio di altri utenti

Spesso i dati vengono memorizzati lato server dove è necessario definire delle tecniche di trattamento apposite.

Ad esempio molti DBMS utilizzano dati spazio temporali ed offrono delle estensioni a DB classici per gestirli.

Ci sono tre query comunemente adottate nei servizi e nel DBMS con estensioni spaziali:

- Range query: si dà in input una zona geografica (che può essere un rettangolo oppure un cerchio) e la query ritorna tutti i punti che appartengono a quella zona indicata. Es: dammi tutte le automobili che sono nel posteggio
- Nearest Neighbors (NN): ho un insieme di punti e do in input uno di questi punti. La NN mi ritorna l'oggetto più vicino ad un punto o ad un altro oggetto. Es: dammi le tre stazioni di servizio più vicine a me
- Reverse-NN: dato un insieme di punti P e un punto q di P, ritorna tutti i punti di P che hanno q come punto più vicino. Es: dammi tutti gli utenti che hanno me come loro utente più vicino o dammi tutte le case che hanno quel supermercato come il più vicino

I DB implementano delle strutture dati chiamati indici che permettono di rendere le operazioni più veloci.

Per esempio una range query su un'area “piccola”, rispetto alla distribuzione dei punti, ha una complessità logaritmica nel numero dei punti.

Analisi

5.1 L'analisi delle applicazioni per dispositivi mobili

Rispetto ad un'analisi del software nei dispositivi tradizionali, nei dispositivi mobili l'utente agisce in un contesto diverso.

Le applicazioni si usano per due motivi:

- per risolvere i problemi: l'utente spesso usa le app per risolvere un problema della vita quotidiana, come ad esempio chiamare qualcuno.
- sono pronti all'uso, sono quasi sempre accesi

Rispetto ai dispositivi tradizionali, le sessioni d'uso sono mediamente molto brevi, infatti gli utenti vogliono risolvere il loro problema il più velocemente possibile.

L'esperienza dell'utente è un aspetto importante nell'analisi dell'applicazione ed è necessario pensare a come l'utente interagisce con essa. Per questo bisogna eliminare, o minimizzare:

- tempi di apprendimento
- tempi di set/up e registrazione al servizio
- tempi di utilizzo

WhatsApp è un esempio tra le prime applicazioni di messaggistica che ha avuto la meglio sulle app concorrenti perché non richiede la registrazione e perché aggiunge automaticamente i contatti tra quelli già presenti in rubrica.

Uno dei primi giochi che ha basato il suo successo su sessioni di gioco brevissime è Angry Birds, dove una partita può durare alcuni secondi e le istruzioni fornite al primo livello sono presentate in un video di meno di 3 secondi.

5.2 Progettazione della user experience

La user experience (UX) è l'interazione che c'è tra uomo macchina, ciò che una persona prova quando usa un prodotto. Include vari aspetti: esperienza nell'uso, utilità, semplicità d'uso La UX estende:

- il concetto di usabilità: è semplice da usare?
- il concetto di user interface (UI), una componente della UX

Un'app deve:

- avere le funzionalità richieste
- funzionare come previsto
- essere usabile dall'utente
- dare soddisfazione all'utente

Tutte queste componenti contribuiscono a formare la UX.

La cosa più importante è che dobbiamo pensare all'utente. L'utente vuole risolvere il problema, non vuole sapere la struttura dati o le chiamate al server effettuate a livello programmatico. Un aspetto a cui bisogna prestare attenzione è il fatto che le icone e il testo debbano essere scelti con attenzione.

Correggere un errore in fase di analisi costa poco, invece correggerlo in fase di testing, il costo è più elevato in quanto si debba tornare nella fase di analisi e rifare tutte le fasi successive (analisi, implementazione ecc.).

5.2.1 Content Prioritization

All'utente bisogna presentare i contenuti e le funzionalità più importanti. L'utente deve poter accedere a contenuti e funzionalità aggiuntive in un secondo momento, ad esempio attraverso un menù.

Dobbiamo conoscere gli utenti, è necessario sapere a cosa sono interessati. Deve essere effettuata un'analisi iniziale con interviste, questionari, ecc. dove vengono coinvolti gli attori e gli altri stakeholder. Dobbiamo capire:

- il prodotto serve veramente? Quanto valore ha? Come sarà utilizzato?
- quali funzionalità sono più importanti?
- quali problemi possono emergere?

Dobbiamo ad esempio realizzare un'applicazione per la didattica per bambini. In una fase di analisi parlo con i bambini, gli insegnanti, i genitori, ecc. Poi devo pormi delle domande: useranno l'app in classe o a casa? Da soli o con un adulto?

Dopo che l'app è stata pubblicata posso raccogliere dati di utilizzo come ad esempio quanto spesso l'app viene usata, quali schermate sono mostrate, quali funzionalità sono usate, ecc. Esistono vari strumenti, ad esempio Google analytics for mobile, che permettono di svolgere un'analisi data driven che influenza le successive attività di sviluppo.

Ci sono diversi fattori da rispettare:

- la semplicità dell'interfaccia grafica: le interfacce grafiche devono essere il più semplici possibili. Negli schermi dei dispositivi mobili lo spazio dedicato ad elementi grafici non indispensabili toglie spazio a quelli indispensabili. Un esempio di interfaccia grafica confusionaria era quella di Virgilio, quella di Google invece è minimale.
- integrità estetica: l'aspetto estetico dell'applicazione deve riflettere la natura dell'applicazione stessa. Un'applicazione per la produttività deve avere un aspetto serio, semplice, lineare e non frivolo, mentre un'applicazione ludica deve avere un maggiore spazio alla grafica ricercata, divertente e appassionante.
- consistenza: l'app dovrebbe funzionare e ricordare altre app che l'utente ha già usato. Se abbiamo delle funzionalità nuove, dobbiamo spiegarle all'utente.
- affordance: caratteristica di un oggetto o di un ambiente di "suggerire" a un individuo la possibilità di compiere un'azione. Attraverso il proprio aspetto l'interfaccia deve invitare l'utente a interagire intuitivamente con essa, sfruttando l'esperienza pregressa (accumulata nel mondo reale o nell'uso di altre app), ad esempio "scroll to refresh"
- metafore: i riferimenti al mondo reale aiutano a capire meglio le funzionalità di un'applicazione
- personalizzazione: cambiando un modo di fare una certa azione si crea un interesse da parte dell'utente. L'interfaccia standard è semplice da usare e il comportamento è consistente con il sistema operativo, ma è poco attraente. L'interfaccia personalizzata crea divertimento, "wow effect", ma l'utilizzo è immediato? Il comportamento è consistente? È compatibile con diverse versioni OS?
- minimizzare l'input: bisogna permettere all'utente di fare meno fatica possibile. Inserire testo su dispositivi mobili richiede uno sforzo per molti utenti, per questo dobbiamo mettere l'utente nelle condizioni di dover scrivere meno testo possibile.

- minimizzare lo sforzo: gli oggetti di interfaccia non devono essere troppo piccoli e devono essere sufficientemente distanziati. Per posizionare gli oggetti di interfaccia dovete pensare a come gli utenti tengono lo smartphone durante l'uso.
- prima impressione: è fondamentale nelle applicazioni l'importanza della prima impressione. Dobbiamo fare in modo che non appena l'app venga aperta, sia possibile da parte dell'utente utilizzare subito il servizio. Non dobbiamo chiedere all'utente di registrarsi o loggarsi se non veramente indispensabile e dobbiamo ritardare quanto più possibile la richiesta di permessi
- strumenti di interfaccia nativi: usare strumenti di interfaccia tipici delle app native e non del web

Progettazione

———— Lezione 6 - 12 ottobre 2020 ——

6.1 Progettazione delle interfacce

Mentre per i dispositivi tradizionali abbiamo come dispositivi di input tastiere, mouse, microfono ecc. per i dispositivi mobili abbiamo touch screen, pulsanti fisici e altri sensori come accelerometro, giroscopi.

Il touch screen permette l'interazione molto più sofisticata del mouse, infatti è possibile definire gesture più complesse. Ci sono dei casi in cui introdurre dei gesti può risaltare a pieno le funzionalità di uno schermo touch come in un'applicazione per i non vedenti.

Nei dispositivi tradizionali chi progetta le applicazioni, progetta una sola interfaccia, nei dispositivi mobili invece le dimensioni variano e bisogna progettare un'interfaccia diversa.

L'orientamento può cambiare con l'orientamento del dispositivo e quindi la GUI si deve adattare.

Ci sono diversi problemi:

- la risoluzione: se la definizione aumenta (cioè aumenta il numero di pixel per pollice) gli oggetti di interfaccia potrebbero essere troppo piccoli
- la grandezza di uno schermo: su schermi con risoluzioni diverse gli oggetti potrebbero avere posizioni poche estetiche (es: occupare solo la parte sinistra dello schermo) o non funzionali (es: elementi che non sono visibili)

6.1.1 Responsive design

L'idea del responsive design è di progettare interfacce per schermi con risoluzione diverse. Il responsive design nasce per il web, ma molte soluzioni sono state adottate per

le applicazioni per dispositivi mobili. Il responsive design fa affidamento su funzionalità (di tool, librerie, ecc.) che lo rendono possibile.

Gli strumenti principali della progettazione responsive sono:

- i layout "liquidi"
- Ri-definizione delle schermate
- densità di pixel

6.1.1.1 I layout "liquidi" (o "fluidi")

Non vogliamo definire delle dimensioni assolute nei layout, ma vogliamo usare delle dimensioni relative ad altri elementi, come alla dimensione dello schermo, allo spazio reso disponibile dal "contenitore" oppure allo spazio richiesto dal contenuto. Degli esempi ne sono:

- `android:layout_width:"match_parent"`
`android:layout_height:"wrap_content"`
- bootstrap: è una delle librerie più usate nello sviluppo web. Divide lo schermo in 12 colonne, quando specifichiamo la dimensione di un elemento, possiamo dire quanto deve essere largo in colonne. Ad esempio un pulsante può essere largo 3 colonne, significa che deve essere largo 1/4 della schermata e in questo modo si definisce la dimensione di un elemento in termini relativi, non assoluti

6.1.1.2 Ri-definizione delle schermate

I layout liquidi solitamente permettono di scrivere poco codice e sono un'ottima soluzione quando riescono a garantire una buona UX. A volte però non sono sufficienti perché su schermi diversi potrebbe essere necessario progettare interfacce utente totalmente differenti, a volte infatti si riprogetta l'intero schema di navigazione.

Non voglio rilasciare app diverse per ogni dispositivo, quindi tipicamente si usano strumenti che ci permettono di implementare interfacce diverse in funzione di vari parametri dello schermo.

6.1.1.3 Densità di pixel

Gli informatici misurano le dimensioni degli oggetti sullo schermo in pixel, ma il problema è che maggiore è la definizione di uno schermo, minore la dimensione degli oggetti. La soluzione è definire una dimensione che non cambia al variare delle densità di pixel sullo schermo: density independent pixel (dip o dp). Il dp è un'unità di misura della lunghezza in mm che equivale ad un pixel con una densità di 160 dpi. Se la

risoluzione raddoppia $1\text{dp} = 2\text{dp}$. I dpi misurano il numero di pixel che ci sono in un inch: 2,54 cm. 160 dpi significa che in un pollice ci sono 160 pixel

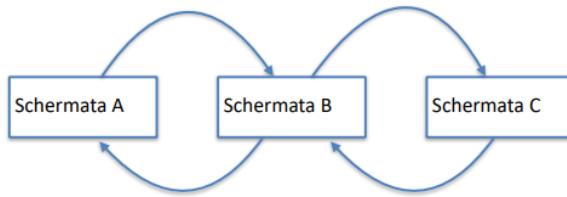
6.1.2 Navigazione

Nelle applicazioni per dispositivi mobili in genere ogni schermata mostra un solo task e l'applicazione deve permettere di navigare tra diversi task. Su dispositivi con schermi più grandi si possono progettare più task in un'unica schermata.

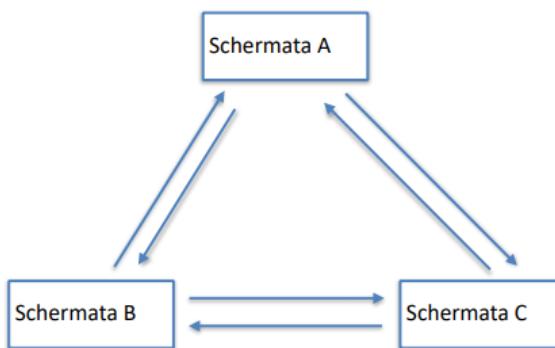
Lo schema di navigazione definisce come l'utente si muoverà tra le varie schermate. Non è necessario definire il dettaglio grafico di ogni schermata, basta indicare la funzionalità. Le funzionalità più importanti/frequenti devono essere presentate il prima possibile all'utente. La prima schermata mostrata all'utente può variare: primo avvio / avvii successivi.

La navigazione si può organizzare in due modi diversi:

- sequenziale: realizzata con una barra di navigazione in cui da una schermata mi sposto ad un'altra in sequenza (avanti o indietro)



- per task: permette di saltare da una schermata ad un'altra in modo diretto



Una volta completato lo schema di navigazione bisogna progettare le singole schermate. A volte ci si rende conto che una schermata va divisa in due o più, oppure che due schermate si possono accoppare. In questo modo si torna a modificare lo schema di

navigazione. Se ci rendiamo di aver commesso un errore, basta poco per correggerlo, se invece lo individuiamo in fase di implementazione, il costo sarà più elevato.

Gli strumenti di interfaccia sono forniti dal framework come ad esempio pulsanti, caselle di testo, ma è anche possibile creare nuovi oggetti.

6.2 Progettare la struttura del codice

Un progetto complesso è meglio suddividerlo in più componenti logiche, grazie alle quali è più semplice:

- risolvere il problema adottando la stessa struttura già usata da altri in casi simili in precedenza
- distribuire il lavoro tra vari membri del team di lavoro
- valutare la correttezza delle singole componenti
- correggere eventuali errori
- mantenere il codice nel tempo: aggiornare le componenti ed il codice che dobbiamo aggiornare è più contenuto

Queste componenti logiche in alcuni casi possono corrispondere con componenti tecniche (es: una classe o un package) ma questo non è necessario.

6.2.1 I pattern di progettazione

La progettazione della struttura del codice spetta al programmatore ed è specifica di un singolo progetto. I pattern di progettazione suggeriscono una possibile soluzione a problemi specifici nel codice o alla progettazione di massima della struttura del codice.

6.2.2 MVC: Model View Controller

Nasce prima che nascono i dispositivi mobili e nasce per la progettazione di applicazioni con interfaccia grafica. È parte integrante dei framework ed è utilizzato in quasi tutte le applicazioni per dispositivi mobili.

L'idea è quella di dividere l'applicazione in tre componenti che vanno tenute nel modo più separato possibile:

- Model: organizza e struttura i dati dell'applicazione. Definisce le operazioni sui dati ai quali le altre componenti possono accedere. Spesso si occupa di:
 - creare oggetti dalla memoria persistente

- salvare gli oggetti nella memoria persistente
- serializzare e de-serializzare i dati

Ad esempio in un social network avremo un model che contiene il profilo

- View: presenta il contenuto all'utente. Le classi della view sono spesso fornite all'interno del framework dell'ambiente di sviluppo. Si possono usare degli oggetti già creati o possono essere personalizzati
- Controller: agisce come intermediario tra la View e la Model. Gestisce il ciclo di vita di un'applicazione e gestisce gli eventi

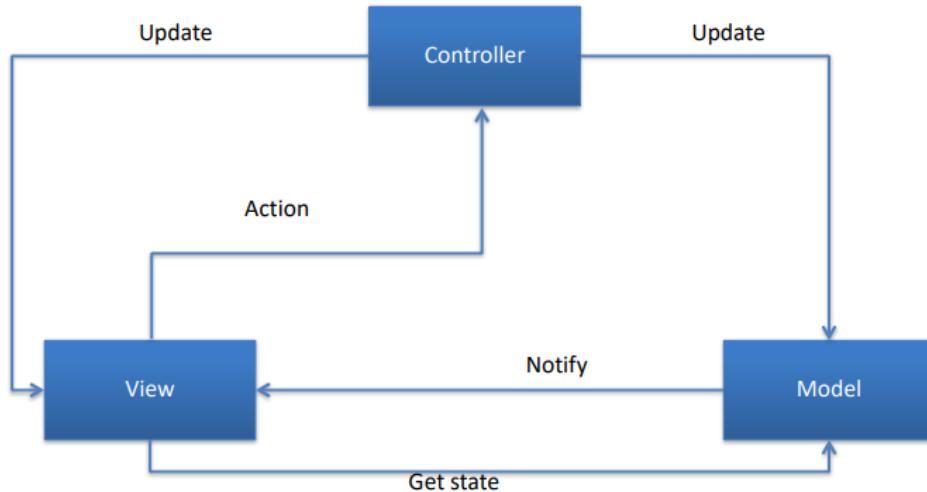


Figure 6.1: Struttura originale di MVC

La View notifica le azioni al Controller che può aggiornare o la View o la Model. Il Controller fa partire la richiesta. La Model quando riceve delle modifiche, può notificare la View che può leggere lo stato per aggiornarsi di conseguenza.

Per i dispositivi mobili viene usato il Model View Presenter (MVP).

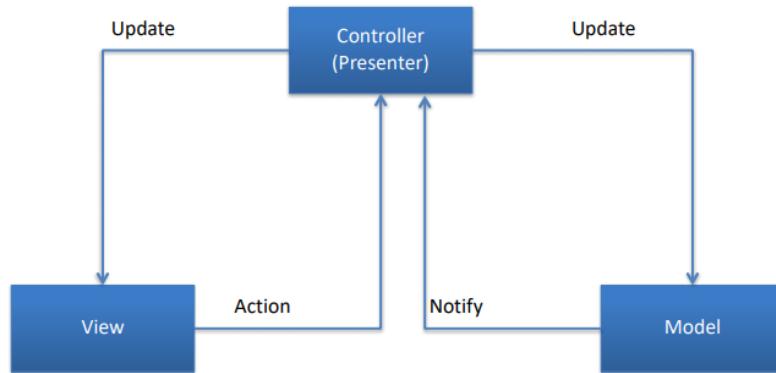


Figure 6.2: Interpretazione Apple di MVP

La differenza con MVC è che Model e View non comunicano direttamente. Model notifica il Controller. La View notifica le azioni al Controller che può aggiornare View o Model. Se aggiorna la Model, questa può notificare al Controller che nel caso notifica la View.

Uno dei pattern per separare le componenti è Observer.

6.2.3 Pattern Observer

Ci sono oggetti "observer" che devono essere avvisati quando lo stato di un altro "subject" cambia. Lo stato può essere, ad esempio il valore di una o più variabili o un evento.

Si implementa con tre componenti:

- una classe che rappresenta il subject
- una o più classi che rappresentano l'observer
- un'interfaccia (o classe astratta) implementata da tutti gli observer. Descrive quali metodi il subject può chiamare sugli observer

Il subject mantiene una lista di observer che si sono registrati, ha dunque un metodo che permette agli observer di registrarsi

¹ `public void addObserver(ObserverInterface observer)`

Un Observer deve avere un riferimento al subject e si registra solitamente con una chiamata tipo

¹ `subject.addObserver(this)`

La chiamata è sintatticamente corretta perché Observer implementa ObserverInterface. La fase di registrazione è rappresentata in figura 6.3a.

Nella fase di notifica, ObserverInterface definisce quali metodi il subject può chiamare, ad esempio:

```

1  public void onClick(View view)
2  public void newLocation(Location l)

```

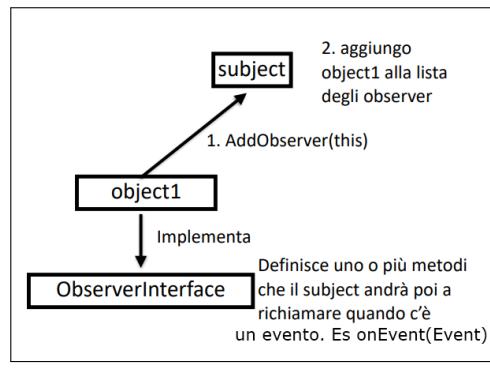
Quando il subject ha un cambio di stato (es: un evento) richiama il metodo su tutti gli observer registrati. Es ("observers" è la lista di observer):

```

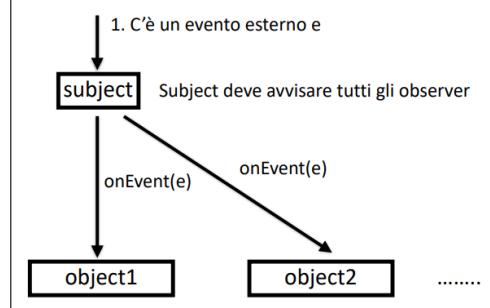
1  for (ObserverInterface o: observers) {
2      o.onClick(View view);
3 }

```

La fase di notifica è rappresentata in figura 6.3b.



(a) Registrazione



(b) Notifica

Figure 6.3: Il pattern observer

Il pattern observer si può implementare con tanti observer o con un solo observer.

Un modo per implementare il pattern observer con MVC è rappresentato in figura 6.4. Il controller si registra come observer con la view e la view si registra come observer con il model. Richiamami quando noti un evento.

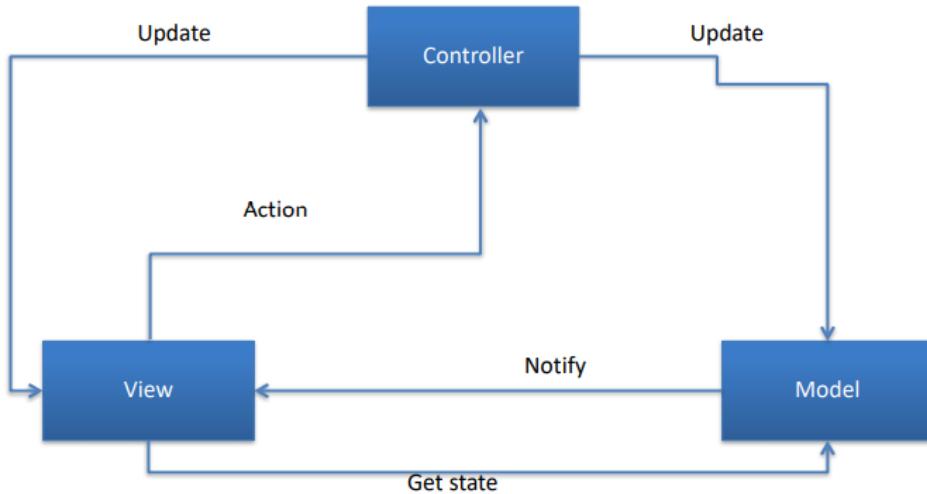


Figure 6.4: Patter observer e MVC

6.2.4 Vantaggi e svantaggi di MVC

MVC è uno strumento molto utile per guidare la progettazione, ci aiuta a strutturare il codice in modo tale che possiamo trovare la struttura in ogni applicazione. Migliora il codice a livello di:

- leggibilità
- riusabilità
- mantenibilità
- testabilità

MVC ha dei limiti: non sempre si riesce ad avere una separazione netta tra le tre componenti.

Sia in Android che in iOS si usano delle componenti combinate di MVC, componenti che mostrano l'informazione all'utente e gestiscono gli eventi:

- ViewController in iOS
- Activity in Android

MVC non si adatta a soluzioni complesse come ad esempio una situazione in cui ci sono più schermate. L'idea è quella di avere un controller generale (detto coordinating controller) per l'applicazione e vari view-controller (detti mediating controller) per le singole schermate.

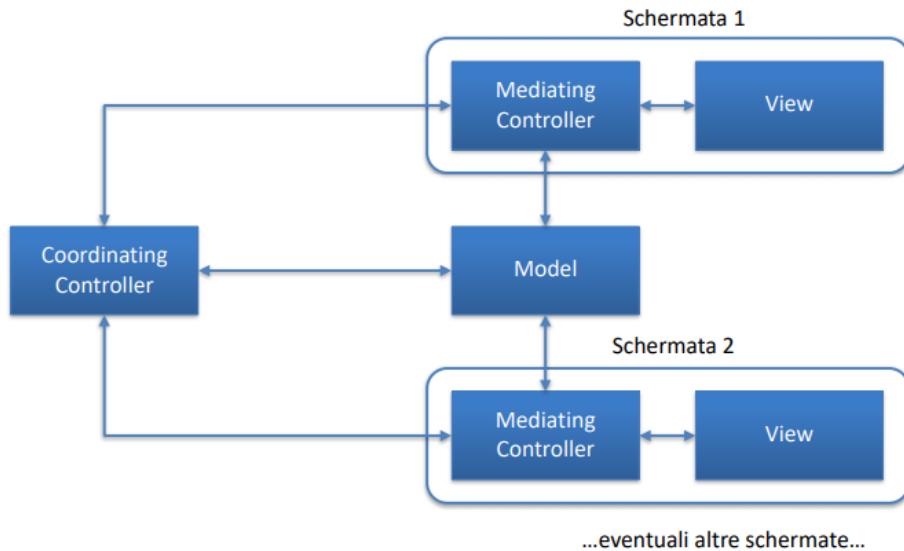


Figure 6.5: Mediating controller

Uno dei principali motivi dell'utilizzo del MVC è la riusabilità del model. Idealmente il model può essere utilizzato sia da applicazioni per dispositivi mobili che per dispositivi tradizionali.

Le classi del Model sono riutilizzabili tra applicazioni che condividono le stesse strutture dati, ad esempio in Java il Collection Framework.

Gli oggetti View sono riutilizzabili tra tutte le applicazioni che condividono gli stessi strumenti di interfaccia, anche se le funzionalità delle applicazioni sono completamente diverse.

Il Controller è il più complicato da riutilizzare, è specifico all'app e alla piattaforma e difficilmente può essere condiviso tra client e server o tra due applicazioni diverse. Anche se il controller è più difficilmente riutilizzabile, esistono alcuni problemi comuni che sono indipendenti dalle applicazioni. Esistono delle classi definite nei framework per risolvere questi problemi.

Gli strumenti di sviluppo per applicazioni mobili sono pensati per uno sviluppo basato su MVC. Gli ambienti di sviluppo permettono di creare in modo statico gli elementi di View. Permettono anche di creare il legame tra view e controller.

Sviluppo

———— Lezione 7 - 14 ottobre 2020 ———

7.1 Strumenti e tecniche di sviluppo per dispositivi mobili

Le applicazioni si scrivono in linguaggi nativi differenti:

- Android in Java o Kotlin
- iOS in Objective C o SWIFT

Quando si sviluppa un'applicazione dobbiamo scegliere per chi sviluppare. e quindi ci poniamo domande come "ci sono più dispositivi Android o iOS?" In realtà è vero che ci sono più download per app Android, ma se le app sono in vendita, il download è maggiore sulla piattaforma iOS rispetto ad Android.

In genere sviluppare su una sola piattaforma non è una soluzione accettabile, tranne che in casi particolari come prototipi o situazione di distribuzione controllata, come ad esempio un'app richiesta da un cliente che fa utilizzare agli operai lo stesso device aziendale.

7.1.1 Sviluppo cross-platform

La soluzione più ovvia è sviluppare ogni applicazione due volte, per Android e iOS. I linguaggi di programmazione sono differenti e le chiamate di SO sono spesso simili concettualmente, ma tecnicamente differenti. In questo caso raddoppiano i costi di:

- implementazione
- testing
- manutenzione

Da questo problema nasce l'esigenza di sviluppare applicazioni cross-platform: sviluppare l'applicazione, o una parte di essa, una sola volta per entrambe le piattaforme.

Ci sono diverse soluzioni per lo sviluppo cross-platform:

- codice in comune
- web app
- hybrid app
- conversione del codice

7.1.2 Codice in comune

In questa soluzione viene scritto il codice C/C++ che poi viene eseguito su iOS perché Objective C estende C/C++, mentre su Android esiste un'estensione JNI che permette di eseguire codice C/C++ all'interno del codice Java.

I vantaggi sono:

- il codice viene scritto una sola volta
- le performance sono elevate
- posso usare le librerie esistenti di C/C++

Lo svantaggio è che non si può interagire con il SO: ho bisogno di una parte di codice nel linguaggio della piattaforma che fa da tramite tra il codice C/C++ e il SO. Questo approccio può essere utile per creare librerie di basso livello, come ad esempio librerie crittografiche, ma non è adatto per creare applicazioni complete

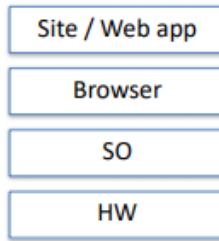
7.1.3 Web app/web site

Al posto di creare un'app creiamo un sito web, con l'utilizzo di tecnologie come HTML 5, CSS 3, Javascript. L'utente accede tramite browser, tipicamente è creare un'applicazione single-page e responsive.

La differenza tra web app e web site è:

- il sito è più orientato al contenuto, ha tante pagine ed ha uno script lato client ma non indispensabile
- web app è finalizzata all'interazione con l'utente, è organizzata in una single page e lo script lato client è necessario

Avere un'app single page non significa che c'è una sola schermata ma viene scaricata l'app con una singola chiamata al web server. Tramite script lato client l'utente può



navigare tra più schermate senza dover ricontattare il web server. La web app può anche comunicare con dei web service per scambiare dati (in modo analogo a quanto fanno le app native).

Web app è una buona soluzione. La stessa web app può funzionare anche su dispositivi tradizionali. Tra i problemi principali ci sono:

- user experience: la UX non è la stessa che su un app
- accesso al SO: da browser non si può accedere a diverse funzioni del SO, come ad esempio il background
- performance minori
- connessione necessaria per caricare la pagina: se si è offline non è possibile scaricare la pagina
- l'app realizzata non è disponibile sullo store

7.1.4 Hybrid App

Il programmatore scrive una web app (HTML, CSS, Javascript). La web app viene eseguita all'interno di un'app nativa che contiene una web view, un componente grafico nativo che contiene contenuto web. L'app nativa può essere generata in automatico.



Apache Cordova è un framework molto diffuso per realizzare hybrid app. È open source.

Permette di scrivere applicazioni con tecnologie web. Permette di accedere ad alcune API del device.

PhoneGap è un framework commerciale di Adobe. Fornisce un'interfaccia grafica per la gestione di Apache Cordova e alcuni strumenti avanzati di sviluppo e testing.

7.1.4.1 Accesso alle funzionalità di SO

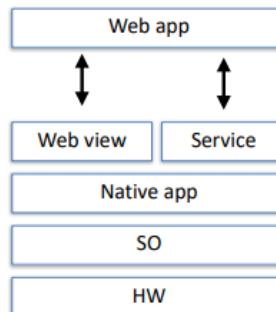
In tutte le tecnologie di sviluppo cross-platform si pone un problema: come faccio a permettere l'accesso alle stesse funzionalità di SO alle quale si può accedere da codice nativo?

iOS e Android hanno funzionalità comune ma vi si accede in modo differente.

7.1.4.2 Accesso alle funzionalità del SO in Apache Cordova

In realtà l'app nativa non contiene solo la web view, ma anche altre funzionalità (Service), implementate in codice nativo, che è possibile richiamare dal codice JavaScript.

Questa cosa non può essere fatta nella web app perchè manca la componente di Service.



7.1.4.3 I plugin

In Apache Cordova esistono dei plugin che vengono distribuiti assieme al sistema. Possono poi essere sviluppati dei plugin aggiuntivi, ma si deve scrivere codice nativo in Android e iOS.

In teoria, i plugin permettono di accedere a qualunque funzionalità del SO. In pratica aggiungono un livello di complessità ed obbligano il programmatore a scrivere codice nativo. In alcuni casi l'integrazione è complessa se non impossibile.

7.1.5 App ibride vs web app

Un'app ibrida è di fatto una web app, infatti un problema che permane nelle app ibride è quello relativo alla user experience. L'accesso al SO viene in parte risolto con i plugin. Il problema delle performance minori permane. Non è necessario avere la connessione per caricare la pagina e l'app realizzata è disponibile sullo store.

7.2 Conversione del codice

Per risolvere il problema delle performance e della user experience, vogliamo fare in modo che il codice esegua come se fosse scritto in nativo. L'idea è di scrivere il codice una volta sola in un unico linguaggio e poi viene usato uno strumento automatico che converte questo codice in codice nativo per varie piattaforme.

Questo approccio è adottato da Xamarin.

7.2.1 Xamarin

Il programmatore scrive codice in C#, ed ha accesso, tramite librerie di Xamarin, ad una chiamata in C# per ogni API disponibile in Android e iOS. Xamarin prevede due livelli di astrazione:

- livello 1: ciascuna API di SO (Android e iOS) è ri-mappata in una corrispettiva API in C#. È possibile scrivere applicazioni cross-platform scrivendo una sola volta il codice che implementa la logica dell'app. Ogni volta che il codice interagisce con il SO, bisogna distinguere due casi, in base al SO. Il codice viene convertito come se fosse codice nativo. L'utente finale che vede un'app non distingue la differenza tra un pulsante creato in Xamarin e uno creato in codice nativo.

Rimane il problema che stiamo scrivendo buona parte del codice due volte

- livello 2: se a livello 1 ho una chiamata per creare un pulsante in Android e un'altra chiamata per creare un pulsante in iOS, posso astrarre il concetto di pulsante e creare una chiamata che verifica la piattaforma e poi crea il pulsante corretto oppure posso applicare lo stesso approccio a tutte le principali API. Il risultato è che ho delle API di più alto livello che sono condivise per iOS e per Android

I vantaggi di Xamarin sono che il programmatore deve conoscere un unico linguaggio di programmazione, le performance elevate e la UX che è uguale a quella nativa. Come svantaggio si ha che tutti gli strumenti di sviluppo cross-platform aggiungono un livello di astrazione in un ambiente tecnologico e può rendere l'applicazione inutilizzabile.

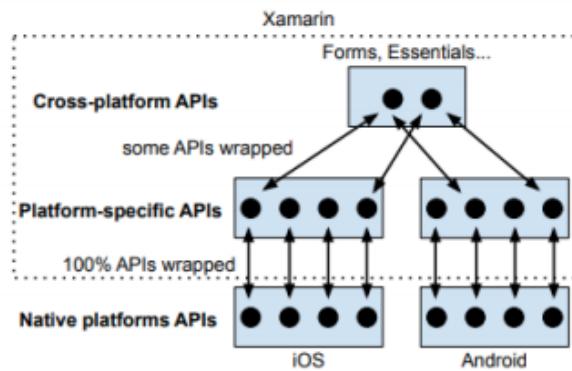


Figure 7.1: Architettura Xamarin

7.2.2 Altre piattaforme di conversione

Unity è un ambiente di sviluppo per giochi multi piattaforma. Si può scrivere codice in C# o in Javascript e viene convertito per essere usato nativamente sulle varie piattaforme.

L'altra tecnologia è Flutter e le applicazioni sono scritte nel linguaggio di programmazione Dart. Il codice viene compilato in codice eseguibile per iOS e Android. Flutter crea un contenitore dentro il quale mostra i suoi oggetti, che non sono quelli nativi.

Un'altra tecnologia è React native che è una via di mezzo tra un sistema Hybrid e un sistema a conversione del codice. Definiamo una GUI in HTML, ma non la mostra in una web view, ma la converte nel corrispettivo oggetto nativo. L'obiettivo è risolvere i problemi di UX, mediante l'uso di oggetti grafici nativi e di prestazioni, eseguendo codice nativo. Un problema di React native è che è basato su uno stack di librerie molto complesso e in rapida evoluzione. Basta cambiare una di queste componenti che tutto smette di funzionare. Questi non sono problemi di codice, ma sistematici.

7.2.3 La scelta della tecnologia

La scelta di quale tecnologia di sviluppo adottare è complessa e richiede di considerare sia aspetti tecnici che aspetti non tecnici.

- Su quali piattaforme dovrà eseguire l'app?
- Quali linguaggi di programmazione conosce il team di sviluppatori?
- L'app dovrà eseguire anche su dispositivi tradizionali o su web?
- L'app deve fare uso di funzionalità evolute di accesso alle risorse HW o SW?

Testing e debugging

— Lezione 8 - 21 ottobre 2020 —

8.1 Approccio alla scrittura del codice

La scrittura di un programma di un sistema complesso passa attraverso una serie di errori.

Uno degli errori più comuni è scrivere tutto o buona parte del codice e poi verificare se l'effetto finale del software realizzato è quello desiderato. Il problema è che non funziona mai al primo tentativo e correggere gli errori diventa molto complicato, perché non si sa dove possono essere.

L'idea è quella dividere il processo di sviluppo in tanti piccoli passi, ognuno dei quali può essere provato per controllarne il funzionamento. Più difficile è il codice che si scrive, più i passi devono essere piccoli. Anche una sola riga di codice, se ad esempio si sta usando una tecnologia che non si conosce bene.

Il problema è che, a volte, abbiamo delle righe di codice che non producono un effetto visibile, come la gestione degli eventi. Per questo bisogna scrivere del codice per verificare lo stato di un'applicazione, come l'utilizzo di messaggi di log.

Il tempo di sviluppo è in buona parte dedicato a correggere gli errori soprattutto quando lo sviluppatore è inesperto che scrive tutto il codice insieme. Ridurre il numero di errori e il tempo necessario per correggerli è l'obiettivo principale se si vuole sviluppare velocemente.

Dividere il codice in piccole parti verificabili è difficile. Prima di iniziare a sviluppare una parte di codice bisogna pensare a come suddividerla in parti, sfruttando l'utilizzo di carta e penna, o di algoritmi in pseudo codice. Un errore comune è che viene provata l'app senza sapere cosa aspettarsi. Prima di fare una prova bisogna pensare a cosa ci si aspetta che sia il risultato.

Diverse parti del codice vengono adattate da esempi trovati online. Questi sono i casi nei quali è più facile commettere errori, perché magari non conosciamo esattamente le operazioni che fa il codice.

Due strategie di sviluppo sono:

- top-down: iniziare a scrivere le funzioni principali per poi scrivere i metodi di dettaglio che sono necessari per le funzioni principali
- bottom-up: iniziare a scrivere i metodi di dettaglio per poi comporli nelle funzioni principali

Prendiamo ad esempio un algoritmo di ordinamento di un array composto da due funzioni:

- la funzione principale di ordinamento
- la funzione che scambia il valore di due elementi dell'array

Nell'approccio top-down prima viene scritta la funzione principale di ordinamento, poi quella che scambia i due valori

Nell'approccio bottom-up prima viene scritta la funzione che scambia i due valori, poi quella di ordinamento.

Nella programmazione tipicamente si usa una combinazione di approccio bottom-up e top-down.

8.2 Testing

Il testing di un'app ha lo scopo di valutare diversi aspetti:

- funzionalità: l'applicazione fa quello che dovrebbe?
- usabilità: l'utente riesce ad usare l'app come previsto?
- performance: l'applicazione ha le performance desiderate?
- sicurezza: l'app è soggetta ad attacchi di sicurezza o privacy?

Se emergono problemi durante il testing significa che l'operazione è efficace. Se non emergono problemi in fase di test l'applicazione è corretta oppure non sono stati svolti i test corretti.

Il testing sui dispositivi mobili ha diverse peculiarità:

- i dispositivi sono diversi dal punto di vista hw e sw

- diversi contesti di utilizzo: c'è connessione ad internet? potrebbe essere molto rallentata? i server a cui l'app si connette funzionano o sono molto lenti? ecc.
- interrupt che avvengono durante l'uso
- il codice su dispositivi mobili è basato su eventi. La sequenza con cui avvengono gli eventi spesso non è deterministica e di fatto si pongono dei problemi di concorrenza

Il fatto che l'applicazione funzioni una volta non significa che sia corretta perché potrebbe presentare degli errori in futuro, anche se eseguita nello stesso modo sullo stesso HW, SW, nello stesso contesto e con gli stessi interrupt.

Per effettuare un test di un'applicazione concorrente, dobbiamo provare tutte le possibili combinazioni di ordine di chiamate. Questo però è un problema, perché provare tutte le combinazioni per ogni device in commercio, è impossibile.

Al posto che svolgere tutti i test per tutti i possibili casi, ciascun test si svolge solo in alcuni casi. In questo modo si hanno meno casi, ma è possibile che qualche caso problematico sia tralasciato.

Linee guida e best practice da utilizzare:

- scegliere i dispositivi più diffusi e caratteristici, tipicamente almeno 2 smartphone e 2 tablet
- svolgere i test con la versione minima supportata del SO e uno con la versione più recente
- svolgere i test in assenza di problemi e in presenza dei problemi più comuni come la mancanza o il rallentamento della connessione
- verificare il funzionamento di tutto il codice
- provare l'app su dispositivi diversi
- provare l'app in situazioni particolari come assenza o rallentamento della connessione Internet

Fare il test di un'applicazione è un'operazione lunga, soggetta ad errori che deve essere fatta ogni volta che si mette mano al codice. L'idea è quella di automatizzare il test.

Alcuni parti possono essere automatizzate, scrivendo del codice che verifica la correttezza di altre parti del codice.

Questo codice può, ad esempio, creare istanze delle classi del model o generare degli eventi su oggetti di interfaccia.

Esistono diverse librerie che supportano i test automatizzati. I test automatizzati hanno una chiamata "assert" che indica cosa ci si aspetta sia vero. Se tutti gli assert sono

verificati, il test ha successo, altrimenti fallisce.

Un esempio ne è il testing del model: viene creata un'istanza della classe che si vuole testare, si richiama il metodo e si verifica, attraverso un assert, che il risultato del metodo sia quello che atteso.

Un altro esempio è il testing della view-controller: ci sono librerie che permettono di scrivere del codice al cui interno si possono scatenare degli eventi sugli oggetti interfaccia, ovvero simulare un'azione che farebbe un utente e si verifica che il risultato sia quello atteso.

8.2.1 Pro e contro dell'automazione del testing

Il vantaggio del testing è che dopo aver scritto il codice di test, il test avviene molto velocemente. La scrittura di test, però, è onerosa ed è soggetto ad errori: può segnalare errori che non ci sono o ignorare errori che ci sono.

In generale il test automatizzato è molto vantaggioso per progetti che devono essere mantenuti nel tempo, cioè quasi tutti i progetti commerciali, mentre nei prototipi a volte potrebbe essere controproducente.

Si può verificare il comportamento delle singole componenti attraverso unit testing.

8.3 Debugging

Il debugging funziona meglio se combinato con una procedura di scrittura passo passo. Il debugging è la procedura attraverso la quale si identifica e risolve un bug, cioè un problema che causa un malfunzionamento.

Si divide in tre passi principali:

- riprodurre il malfunzionamento: per algoritmi deterministici è semplice, lo stesso input produce sempre lo stesso output deterministico, ma in realtà in casi pratici non è così. Se si effettua un test manuale (non automatizzato) bisogna trovare una serie di passaggi che generino sempre l'errore: se i passaggi sono lunghi da riprodurre, è conveniente modificare temporaneamente il codice per velocizzare il test
- trovare il bug: vengono usati due strumenti principali: logging e debugging. È possibile eseguire un'applicazione in modalità debugging: l'esecuzione si ferma in alcuni punti definiti dal programmatore
- risolvere il bug: ci sono varie strategie per trovare e risolvere gli errori:
 - "piccoli passi"

- tecnica "wolf fence": ricerca dicotomica
- "torna indietro e prova": cancello delle righe commentando fino a quando non arrivo ad una riga in cui il codice funziona. Pian piano si decommenta
- "semplifica il codice": se il bug si verifica in una riga di codice, ma è complicata. Si può riscrivere la riga di codice in più righe e andare a verificarle una dopo l'altra

8.3.1 Logging

Una parte del codice dell'applicazione può essere finalizzata a stampare messaggi per il programmatore, così che possa capire meglio cosa accade nel codice. I log, a volte, vengono inseriti ancora prima di avere un bug, perché servono anche da documentazione del codice. In Android abbiamo accesso a tutti i log del sistema e il problema è che il nostro messaggio vada perso tra tutti gli altri. È possibile filtrare i messaggi di log per:

- processo: spesso selezionare il processo giusto non basta, perché ci sono tanti messaggi di log non scritti da noi, ma generati da altre componenti del nostro processo
- importanza: vengono mostrati tutti i messaggi più importanti del livello scelto, ma a volte non basta
- tag: per ogni messaggio si usa un TAG che permette di categorizzare il messaggio e un testo

8.3.2 Errori comuni

- il messaggio d'errore non viene letto. In molti casi si può trovare il messaggio lungo e non sempre è semplice capirlo
- testare l'app senza sapere cosa aspettarsi: prima di provare l'app bisogna avere chiaro cosa ci si aspetta che l'app faccia
- l'app non fa ciò che dovrebbe: ad esempio va in crash
- fare sempre le stesse operazioni durante le prove dell'applicazione, ma si rischia di testare le stesse parti del codice

Posizionamento indoor - MobiDEV

———— Lezione 1 - 1 marzo 2021 ———

9.1 Concetti preliminari

9.1.1 Calcolo della posizione indoor vs outdoor

La maggior parte delle app richiede una maggior precisione, l'altezza infatti è fondamentale: sono al primo o al quinto piano di un edificio? Negli ambienti outdoor l'altezza non è così fondamentale.

Le tecniche che utilizziamo in outdoor non possono essere utilizzate in indoor:

- GPS: linea di comunicazione tra la sorgente (satellite) e il dispositivo
- le interferenze possono rendere inutilizzabile il sistema segnale, come il segnale radio che viene disturbato da muri o da altri ostacoli

L'area di copertura in indoor è ridotta. Nel maggior parte dei casi lo spostamento avviene a piedi ed il vantaggio spostandosi lentamente è che si ha una minore rapidità di spostamento degli utenti, lo svantaggio è che siamo soggetti a rumori come a cambiamenti di direzione improvvisi.

Attualmente l'indoor positioning è ancor poco comune.

Chi è il soggetto da localizzare?

Possiamo voler monitorare la posizione di persone, robot o oggetti come ad esempio il carrello delle emergenze in un ospedale.

Soggetti diversi implicano tecnologie diverse:

- per le persone abbiamo tecniche di posizionamento che sfruttano i dispositivi mobili degli utenti
- per i robot e gli oggetti possiamo prevedere un hardware apposito

Chi effettua il calcolo della posizione?

Ci sono due tipi di tecniche:

- tecniche attive: il calcolo della posizione viene effettuato dal dispositivo in movimento (le tecniche trattate nel corso)
- tecniche passive: l'hardware distribuito nell'ambiente capisce in che posizione sono gli utenti

9.1.2 Applicazioni principali

- navigazione: non è una tecnologia molto utilizzata. Sapere dove muovermi all'interno di un aeroporto sarebbe comodo, come ad esempio "guidami fino al gate 15"
- location based services: vengono utilizzati per l'outdoor, non ancora per l'indoor
- home automation: es. un sistema che in base alla stanza nel quale mi sposto, accende le luci
- context detection: es. il riconoscimento precoce di malattie degenerative tramite un'analisi del comportamento degli utenti
- social networking: es. friend finder
- emergenze: es. in caso di incendio i vigili del fuoco sanno dove sono le persone da salvare

9.1.3 Paradigmi di navigazione

- Navigazione allocentrica: il sistema fornisce le indicazioni di navigazione rispetto all'ambiente come ad esempio la vista dall'alto con la mappa.
- Navigazione egocentrica: il sistema fornisce le indicazioni di navigazione rispetto all'utente, come ad esempio una freccia che indica dove andare

L'interpretazione da parte dell'utente delle informazioni allocentriche richiede all'utente di conoscere com'è orientato e non sempre questa navigazione è possibile quando c'è ad esempio del fumo o della nebbia.

9.1.4 Caratteristiche dei sistemi di posizionamento indoor

Quali caratteristiche deve avere il sistema per adattarsi alle necessità dell'utente?

- il sistema indoor deve essere preciso nel calcolo della posizione

- area di copertura (stanza, edificio, globale)
- il costo (di installazione, di manutenzione, ecc.)
- infrastruttura (nessuna, trasmettitori sparsi nell'ambiente, ecc.)
- maturità della tecnologia (prototipo, prodotto, ecc.)
- tipo di informazione di posizione che viene prodotto (posizione 2D/3D, orientamento, ecc)
- privacy (il sistema viene a sapere dove si trova l'utente?)
- frequenza di update (quando l'utente fa qualcosa, automatico ogni 4 secondi)
- azione richiesta all'utente (nessuna, inquadrare il marker, ecc.)
- interfaccia (grafica, vocale, ecc.)
- robustezza della misura (nelle giornate affollate, ecc.)
- robustezza della sistema (i beacon si possono rubare, i marcatori si staccano, ecc.)
- scalabilità (non scalabile, scalabile aumentando l'infrastruttura, scalabile diminuendo l'accuratezza, ecc.)
- impatto sull'ambiente (i marcatori visivi rovinano l'aspetto estetico, ecc.)

9.2 Tecniche di posizionamento

Ci sono 4 tecniche di posizionamento indoor che verranno approfondite:

- immagini
- segnali radio
- sistemi inerziali
- soluzioni ibride

9.3 Tecniche basate su immagini

Ci sono diverse tecniche che permettono di calcolare la posizione usando le immagini tra cui i marcatori visivi. I marcatori visivi sono oggetti facilmente riconoscibili tramite tecniche di computer vision. Distinguiamo marcatori:

- esplicativi: progettati per essere facilmente riconoscibili come i QR-code

- impliciti: creati per altri scopi, ma facilmente riconoscibili con tecniche di computer vision come i cartelli stradali

I marcatori visivi possono essere installati nell'ambiente. Quando un utente ad esempio inquadra il marcatore, so che l'utente si trova lì vicino. Ciascun marcatore contiene la propria posizione oppure un identificatore che il device sa associare alla posizione.

9.3.1 Pro e contro dei marcatori esplicativi

Vantaggi	Svantaggi
<ul style="list-style-type: none"> • tecnologia semplice e stabile • i marcatori sono economici 	<ul style="list-style-type: none"> • la precisione è legata alla distanza alla quale il device riesce a riconoscere il marcatore • non permettono di calcolare l'angolo • è richiesta un'azione esplicita all'utente

9.3.2 I marcatori impliciti

Se so la dimensione in centimetri del marcatore, dove viene appeso e con che angolo, quando il marcatore viene inquadrato, posso usare tecniche di geometria per capire dove si trova l'utente rispetto al marcatore.

Però ho un problema: ho il sistema dove ho cartelli facilmente riconoscibili e sviluppo una tecnica che riconosca i cartelli ma ci potrebbero essere due cartelli uguali. Bisogna combinare la tecnica con altre tecniche ad esempio usando i segnali radio. La tecnica funziona da sola nel caso in cui i marcatori impliciti siano tutti diversi.

9.3.2.1 Pro e contro

Vantaggi	Svantaggi
Non devo installare altri segnali nell'ambiente (in alcuni ambienti potrebbe essere impossibile, ad esempio nei musei)	se i marcatori non sono univoci la tecnica da sola non permette di calcolare la posizione e deve essere combinata con altre tecniche

9.3.2.2 Tecniche marker-less

Per risolvere i problemi posso usare tecniche marker-less, non vengono scelti marcatori da me programmatore ma il sistema estrae da solo delle informazioni visive che permettono poi all'utente di capire dove si trova. Questa tecnica opera a due fasi:

- setup: viene costruita una mappa 3D dell'ambiente, il sistema identifica diversi punti di riferimento con caratteristiche geometriche peculiari
- calcolo posizione: quando l'utente è nell'ambiente, vengono identificati alcuni punti di riferimento e poi vengono confrontati con quelli ottenuti in fase di setup, per risalire alla posizione

Ho però un problema: durante la fase di setup ho un dispositivo che mentre si muove, deve creare una mappa dell'ambiente e deve sapere dove sono rispetto all'ambiente.

Durante la fasi di calcolo della posizione, dal device mobile si cerca di trovare gli stessi punti di riferimento identificati in fase di setup e di stimare la posizione della camera rispetto ad essi.

Ho un problema: due zone possono avere le stesse caratteristiche, ad esempio possono esserci due stanze che presentano le stesse feature su piani diversi di un edificio.

———— Lezione 2 - 4 marzo 2021 ————

9.3.3 Uso di tecniche di visione per posizione outdoor

Dalle immagini catturate dalla macchina google (street view) possiamo estrarre dei punti di riferimento (feature points). La posizione da cui sono scattate le foto stradali è nota con buona precisione, quindi si riesce a conoscere la posizione dei feature points. Quando l'utente inquadra attorno a sé, si conosce più o meno la posizione, e dalle immagini dell'utente si identificano le feature che corrispondono a quelle identificate dalle immagini stradali. Poi si fa un match con i feature points della macchina google in modo tale che si possa calcolare la posizione dell'utente rispetto alla macchina (dato che so come si trova la macchina rispetto ai feature points). Si calcola la posizione (molto precisa) e l'orientamento dell'utente.

Una funzione simile è messa a disposizione per le librerie AR da Apple (ARKit).

9.4 Calcolo della posizione basato su segnale radio

I dispositivi mobili al momento sono dotati di hardware per questi segnali radio:

- WiFi: abbiamo un problema applicativo, con le tecnologie attuali non è possibile da programma accedere alle antenne WiFi nelle vicinanze, il SO operativo conosce le informazioni ma le API dei dispositivi iOS non consentono al programmatore di accedere.
- bluetooth: dispositivi detti beacon da acquistare ed installare specificamente per il calcolo della posizione
- Near Field Communication (NFC)
- Ultra Wide-Band (UWB): ancora poco diffuso nei dispositivi mobili

RSSI indica la potenza del segnale ricevuto dal dispositivo mobile.

Ci sono 3 diverse tecniche basate su segnale radio:

- prossimità
- multilaterazione
- fingerprinting

9.4.1 Posizionamento per prossimità

Il dispositivo mobile percepisce il segnale radio inviato da un'antenna con un certo identificativo. Se conosciamo l'id dell'antenna e la distanza massima con la quale si può percepire il segnale da quell'antenna, allora riusciamo a scoprire la posizione di un device.

L'accuratezza con Wifi e bluetooth è bassa mentre con NFC è alta ma il calcolo della posizione avviene solo quando l'utente avvicina il device ad un sensore NFC installato nell'ambiente.

Ci sono due limiti parlando di prossimità:

- si considera una sola antenna
- si considera solo il fatto di percepire il segnale di quell'antenna, non si prova a stimare la distanza dall'antenna stessa

9.4.2 Multilaterazione

Posso sapere che sono vicino ad un'antenna senza sapere la distanza oppure posso provare a stimare la distanza usando RSSI ma mi aspetto che la potenza decresca con l'aumentare della distanza.

9.4.2.1 Calcolo della distanza con RSSI

Conosciamo la potenza del segnale inviato, le caratteristiche dell'antenna, vogliamo calcolare la distanza ma abbiamo un altro fattore da prendere in considerazione che non conosciamo, il path lost factor che dipende dall'ambiente (è un fattore di attenuazione del segnale).

In ambienti aperti assume un valore di 2, in ambienti indoor tra 4 e 6 e in particolari indoor ha un valore minore di 2.

Stimando con RSSI si ottengono errori troppo grandi.

Una tecnica per risolvere il problema è utilizzare un WiFi Round Trip Time.

Google ha proposto una soluzione basata su RTT. RTT è la misura che rappresenta quanto ci mette un segnale ad andare e tornare da un'antenna.

9.4.3 Fingerprinting (o "scene analysis")

La soluzione più adottata per risolvere i problemi della multilaterazione è quella del fingerprinting. Il valore di un RSSI in un punto è più o meno sempre quello, ma se metto più antenne, è difficile che la combinazione di questi valori si ripeta in un altro punto.

Data una posizione, il fingerprint è l'insieme di coppie <ID, RSSI_ID> dove ID è l'identificatore di un'antenna e RSSI_ID il suo valore di RSSI misurato nella data posizione.

Un addetto si sposta all'interno dell'edificio segnalando al sistema la posizione corrente (es: toccando con il dito sulla mappa) e poi misura la potenza del segnale. Questa operazione viene ripetuta in tutto l'ambiente, ad esempio ogni metro e prende le misure per tutte le antenne del fingerprint.

Il risultato prende il nome di radio map.

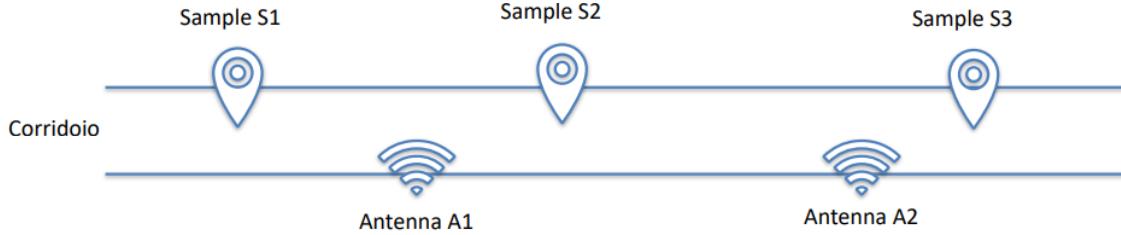
La radio map contiene un insieme di coppie <posizione, fingerprint>, quindi <posizione, [<ID, RSSI_ID>]>.

Quando un utente si trova nell'ambiente e vuole calcolare la posizione, possiamo prendere il fingerprint nella posizione corrente.

Il modo più semplice per trovare la posizione dell'utente è:

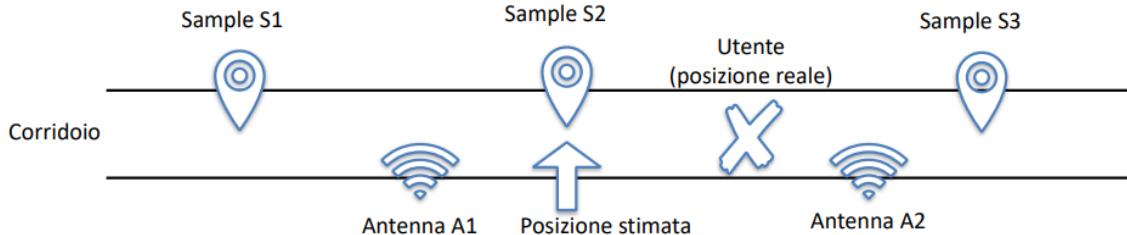
- considerare tutti fingerprint in tutte le posizioni della radio-map
- calcolare il fingerprint dell'utente
- calcolare la distanza dal fingerprint in radio map

- vedere a quale campione l'utente è più vicino e assumo che quel campione sia la sua posizione



Ad esempio ho un corridoio con due antenne e riesco a percepirlne sempre il segnale. Un incaricato misura i fingerprint in 3 posizioni campione e misura su ognuna la potenza del segnale. La radio map è l'insieme composto dai seguenti elementi:

- $\langle S1, [A1, 0,5], [A2, 0,01] \rangle$
- $\langle S2, [A1, 0,4], [A2, 0,6] \rangle$
- $\langle S3, [A1, 0,1], [A2, 0,8] \rangle$



Il fingerprint calcolato dell'utente è $[A1, 0,3], [A2, 0,7]$. Si procede con il calcolo delle distanze da fingerprint in radio map:

- S1: $|0,3-0,5| + |0,7-0,01| = 0,2 + 0,699 = 0,899$
- S2: $|0,3-0,4| + |0,7-0,6| = 0,1 + 0,1 = 0,2$
- S3: $|0,3-0,1| + |0,7-0,8| = 0,2 + 0,1 = 0,3$

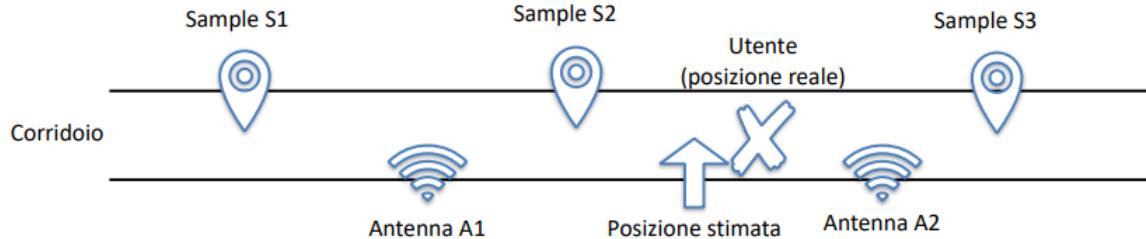
La distanza è minore rispetto ad S2 e quindi assumo che la posizione dell'utente sia quella di S2.

Se tutto va bene trovo effettivamente il fingerprint vicino alla mia posizione. Se in fase di calibrazione ho collezionato pochi fingerprint l'errore potrebbe essere ampio, ad esempio se in un corridoio ho un fingerprint ogni 10 metri, l'errore potrebbe essere anche di 5 metri.

Se invece, per via dei dati inesatti, scelgo un fingerprint che non è il più vicino alla mia posizione, l'errore potrebbe essere anche più grande.

Per risolvere questo problema, calcolo la stessa distanza di prima ma al posto di prendere il più vicino, considero i due campioni più vicini e assumo che la posizione dell'utente sia tra quei campioni.

Prendiamo sempre come esempio il corridoio.



Si procede con il calcolo delle distanze da fingerprint in radio map, come nel caso precedente:

- S1: $|0,3-0,5| + |0,7-0,01| = 0,2 + 0,699 = 0,899$
- S2: $|0,3-0,4| + |0,7-0,6| = 0,1 + 0,1 = 0,2$
- S3: $|0,3-0,1| + |0,7-0,8| = 0,2 + 0,1 = 0,3$

Successivamente stimo che la posizione dell'utente sia tra S2 ed S3 ad una distanza da S2 di $0,2 / (0,2+0,3)$ rispetto alla distanza totale tra S2 ed S3.

Abbiamo diversi problemi: la tecnica non è robustissima, abbiamo i dati che sono approssimativi, l'ambiente può cambiare (una porta che si apre o chiude influenza il radiomap), il numero di persone nell'ambiente influenza la radiomap, il tipo di antenna del device, l'umidità.

Bisogna avere quindi un sistema più robusto che riesca a gestire tutti questi fattori, quindi si usano tecniche probabilistiche o basate su machine learning (es: reti neurali).

C'è un altro problema, la fase di setup è molto onerosa. Ogni misurazione può durare anche decine di secondi. Ci sono vari modi per risolvere il problema:

- interpolazione: l'incaricato parte da un certo punto, cammina con velocità regolare, il sistema misura la potenza del segnale e l'incaricato arriva al punto di destinazione. Il sistema sa interpolare i punti, avendo il punto di inizio e quello di fine e la velocità che è costante, l'applicazione calcola i vari fingerprint lungo il percorso
- robot: un sistema basato su ruote. In questo modo è più facile misurare lo spostamento. I robot esplorano l'ambiente, sappiamo quanto si spostano, in base a quanto girano le ruote

- crowdsourcing: non faccio fare la lettura all'incaricato ma faccio in modo che i dati vengano inseriti dagli utenti finali

9.4.4 Le tecniche a confronto

Tecnica	Segnale	Errore ODG	Complessità	Costo (setup, hardware)	Note
Prossimità	WiFi	100m	Molto bassa	Molto basso	Errore troppo elevato
	BT	10m	Molto bassa	Medio	Richiede acquisto e manutenzione beacon
	NFC	cm	Molto bassa	Medio	Richiede azione esplicita Calcolo sporadico della posizione
Multilaterazione	WiFi	10m	Bassa	Molto basso	API non consentono accesso a RSSI
	BT	10m	Bassa	Medio	Errore non significativamente migliore a tecnica basata su prossimità
	WiFi RTT	m	Bassa	Alto (se bisogna cambiare gli access)	Disponibilità hardware
Fingerprinting	WiFi	m	Alta	Alto	API non consentono accesso a RSSI
	BT	m	Alta	Molto alto	Richiede acquisto e manutenzione beacon

- prossimità: hanno una minor precisione ma una maggior semplicità, hanno un costo ridotto
- multilaterazione: compromesso tra precisione e semplicità ma non aumentano in modo significativo la precisione
- fingerprinting: costo alto e complessità alta

9.5 Sensori inerziali: calcolo dello spostamento

Ci sono tecniche che permettono di calcolare la posizione come tecniche di dead reckoning utilizzate in diversi ambiti come in quello militare. Viene utilizzato un sommersibile che quando emerge, viene calcolata la posizione con GPS. Durante l'immersione, il segnale sotto una certa profondità non capisce più la posizione, per questo vengono sfruttati gli accelerometri che permettono di misurare l'accelerazione istantanea, dunque otteniamo lo spostamento.

Si può usare in dispositivi mobili?

A bordo del sommersibile ci saranno accelerometri sicuramente più sofisticati rispetto ai telefoni. I sensori inerziali sono soggetti a meno rumore rispetto ad un telefono tenuto in mano mentre si cammina.

Le tecniche inerziali vanno combinate con altre tecniche perché da sole non calcolano la posizione ma calcolano solo lo spostamento. Un modo per rendere più robuste le tecniche è di combinare le tecniche inerziali con quelle visive.

Se non conosco l'orientamento dell'utente, non serve a nulla sapere lo spostamento a meno che non usiamo filtri appositi.

9.5.1 Gestione dell'errore

Il sistema è soggetto a molte forme di errore:

- potrebbe sbagliare a contare i passi o la loro lunghezza
- potrebbe sbagliare a calcolare la direzione

A differenza di tutte le altre tecniche, l'approssimazione nel calcolo della posizione cresce linearmente con il tempo.

Ad esempio ho un fix di posizione e orientamento a $t=0s$ (errore zero). A $t=2s$ mi aspetto un errore di posizione e orientamento contenuto: non posso aver sbagliato di tanto, in così poco tempo. A $t=4s$ l'errore è dato dall'errore a $t=2$ più l'errore accumulato tra $t=2$ e $t=4$. Sarà sempre contenuto, ma generalmente maggiore dell'errore in $t=2$. A $t=60s$ ho accumulato l'errore in tutti gli istanti precedenti, dunque generalmente avrò un errore ben più grande rispetto a $t=2s$.

Questo tipo di errore viene anche chiamato drift.

9.5.2 Calcolare un passo

In teoria è possibile calcolare quando l'utente fa un passo processando i dati di accelerometri e giroscopi. In termini pratici però sia iOS che Android rendono disponibile un sensore virtuale che calcola questa informazione (usando i dati di accelerometri e giroscopi), dunque è molto semplice ottenerla.

Il limite di queste tecniche è che contano i passi, ma non si conosce la lunghezza del passo.

Esistono diversi lavori finalizzati a calcolare la lunghezza del passo:

- molti assumono che il device mobile sia solidale con il centro di massa dell'utente, come ad esempio lo smartphone legato alla cintura
- alcuni considerano il caso in cui il device sia tenuto in mano
- altri considerano che una persona faccia passi lunghi sempre uguali, in questo modo se conto i passi e so dove si sposta, posso imparare la lunghezza dei passi stessi da usare in futuro

9.5.3 Tecniche visuo-inerziali

Usando tecniche di computer vision è possibile stimare come si sta spostando-ruotando la camera. Le tecniche visuo-inerziali combinano i dati dei sensori inerziali con la stima dello spostamento-rotazione ottenuta attraverso l'analisi del flusso video.

Questo permette di ottenere una stima dello spostamento-rotazione più robusta rispetto al solo uso dei sensori inerziali.

La tecnica è adottata dalle librerie esistenti di augmented reality.

9.6 Tecniche ibride

Combinano più soluzioni per cercare di avere una soluzione più affidabile.

Un esempio di tecnica ibrida ne è il segnale radio usato in combinazione con la computer vision.

Se ho due stanze identiche in piani diversi, con la tecnica computer vision risulta impossibile identificare la posizione esatta di un utente, per questo viene combinata con il segnale radio.

Un'altra tecnica è l'utilizzo dei marcatori visivi con il calcolo della posizione. Non puoi chiedere ad un utente di spostarsi inquadrando sempre un marcatore, quindi uso i marcatori visivi.

Mentre l'utente si sposta uso le tecniche di spostamento, il problema è che dopo un po' la posizione non sarà più calcolata perfettamente, in quanto le tecniche di spostamento accumulino l'errore.

Per azzerare l'errore, l'utente inquadra un marcatore visivo in modo tale che venga fatto anche un fix di posizione.

9.6.1 Tecniche ibride e calibrazione crowdsourced

In questo caso uso più tecniche di posizionamento. Quando una tecnica mi fornisce una posizione con buona approssimazione, ad esempio un utente inquadra un marcatore, oppure utente passa vicino ad un beacon, utilizzo questa informazione per calibrare le altre tecniche. Ad esempio tengo traccia della potenza delle antenne WiFi in quella precisa posizione. Questo procedimento è analogo a quanto avviene in outdoor.

Tecniche di aggregazione di dati soggetti a rumore

———— Lezione 3 - 8 marzo 2021 ——

10.1 Stima della posizione con dati soggetti a rumore

In diversi contesti si ha il problema che devo stimare un valore o lo stato di un sistema come ad esempio l'angolo di pitch di un aereo (angolo che assume la pala rispetto al piano di rotazione del rotore). Si ha anche il problema che il calcolo deve essere effettuato sulla base di varie misurazioni soggette a rumore, come ad esempio accelerometri installati in varie parti dell'aereo.

Applichiamo il problema all'indoor positioning, andiamo quindi a stimare la posizione di un utente. Ci sono alcuni spostamenti che non sono possibili, ci facciamo aiutare dalla planimetria, ad esempio un utente non può passare attraverso i muri oppure non può salire su un piano superiore se non ci sono scale o ascensori.

10.1.1 Modello bayesiano

Abbiamo una serie di osservazioni $z_1 \dots z_n$ che rappresentano le osservazioni nel tempo. Vogliamo calcolare $B(x_t)$, dove B sta per belief, che rappresenta la certezza che la posizione al tempo t sia x . Abbiamo quindi che

$$B(x_t) = P(x_t | z_1 \dots z_n)$$

Nel modello, la $B(x_t)$ dipende da tutte le osservazioni possibili. Siccome è una distribuzione di probabilità, la somma per tutte le x di $B(x_t)$ è 1.

Il modello bayesiano è generico ma non ci indica come calcolare la soluzione.

10.1.2 Modello Markoviano

Nel modello bayesiano è difficile calcolare una soluzione perchè la valutazione dipende da tutta la storia delle osservazioni.

C'è un modello meno espressivo che permette di rappresentare una sottoclassificazione di problemi, il modello markoviano. È più semplice dal punto di vista formale e di calcolo, ma ha un'espressività tale da permettere di modellare la maggior parte dei problemi di interesse pratico in questo campo. Assumiamo che la posizione al tempo t (cioè $B(x_t)$ per tutte le x_t) dipenda solo dal valore di belief al tempo $t-1$, cioè $B(x_{t-1})$ e dal valore di z_t .

In pratica non mi interessano tutte le misurazioni ma l'ultima osservazione al tempo $t-1$.

10.1.3 Modello di transizione di stato (o modello di spostamento)

Il modello di transizione di stato mi permette di capire dove io posso essere al tempo t data la mia posizione al tempo $t-1$.

Ero da qualche parte in $t-1$, dove potrei essere al tempo t ?

$B^-(x_t)$ rappresenta la probabilità di dove si trovi l'utente al tempo t tenendo in considerazione che so dov'era al tempo $t-1$.

$B^-(x_t)$ dipende da $B(x_{t-1})$ e dalla probabilità $P(x_t | x_{t-1})$ che ci sia stato uno spostamento tra x_{t-1} e x_t .

Esempi:

- la distanza tra x_{t-1} e x_t è 20 metri e la distanza tra $t-1$ e t è di 5 secondi. Considerando che l'utente sia a piedi, è improbabile che l'utente abbia percorso quella distanza in così poco tempo, quindi il valore della probabilità $P(x_t | x_{t-1})$ sarà basso ma non zero. Può essere che l'utente stesse correndo
- al tempo $t-1$ gli unici valori di x_{t-1} per i quali $B(x_{t-1})$ è non-zero sono al primo piano. Al tempo t considero una x_t al secondo piano e non ci sono né scale né ascensori lì vicino. La distanza temporale tra $t-1$ e t è di 5 secondi. $B^-(x_t)$ sarà zero perché è impossibile che un utente passi da un piano all'altro in così poco tempo non avendo scale o ascensore
- consideriamo uno spazio dimensionale e tre posizioni nel tempo $t-1$: x^a, x^b, x^c . Con $B(x^a) = 1/2$, con $B(x^b) = 1/6$, con $B(x^c) = 2/6$.
 - $P(x_t | x^a) = 0$, la probabilità che un utente si muova da x^a ad x_t è 0 perchè la distanza è troppa

– $P(x_t | x^b) = 0,1$, è possibile ma improbabile, l'utente deve essere andato molto velocemente

– $P(x_t | x^c) = 0,5$ è probabile nel mio modello

$$B^-(x_t) = \frac{1}{2} * 0 + \frac{1}{6} * 0,1 + \frac{2}{6} * 0,5$$



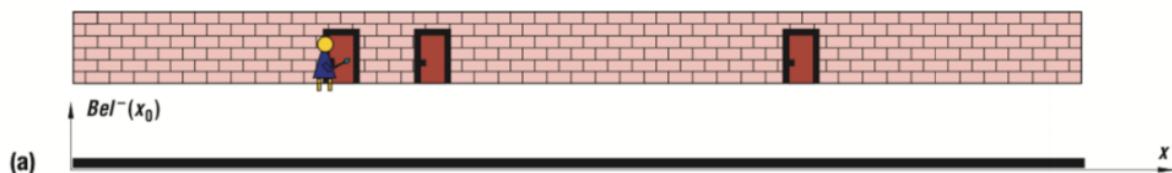
10.1.4 Modello percettivo

Prende in considerazione l'osservazione che abbiamo fatto. Vogliamo stimare quanto è la probabilità che in x_t percepiamo z_t . Non stiamo stimando il punto ma la probabilità di percepire quella certa osservazione.

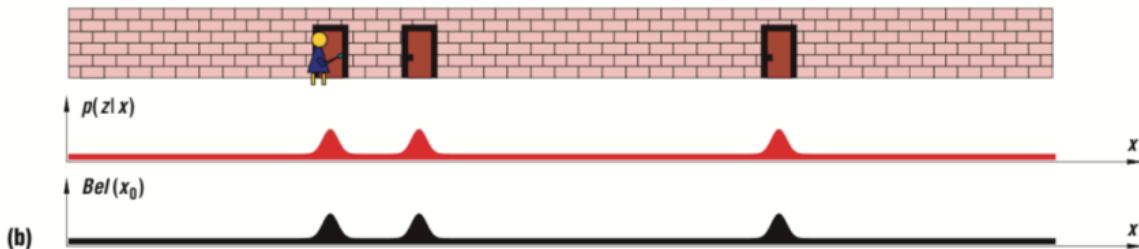
Esempio: z_t è un alto valore di RSSI rispetto ad un beacon bluetooth:

- x_t^1 è una posizione a 100 metri dal beacon, quindi $P(z_t | x_t^1) = 0$, questo perché il beacon non può essere percepito a 100 metri di distanza
- x_t^2 è una posizione a 1m dal beacon, quindi $P(z_t | x_t^2)$ sarà un valore più alto, certamente non zero

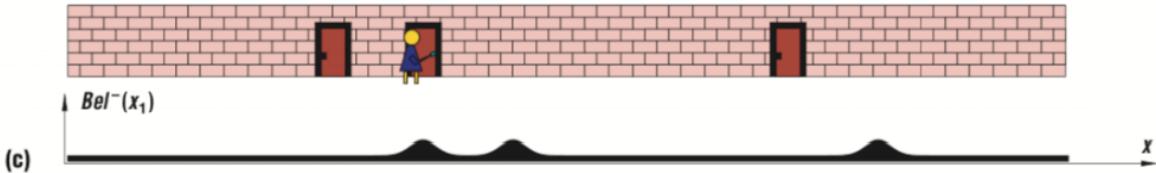
Esempio del sensore e della porta: abbiamo uno spazio mono-dimensionale, c'è un sensore che ti avvisa quando sei vicino ad una porta, ma non sai a quale porta. Le osservazioni z_t sono del tipo "sei vicino ad una porta". In qualche modo il sistema sa anche come si sta spostando l'utente.



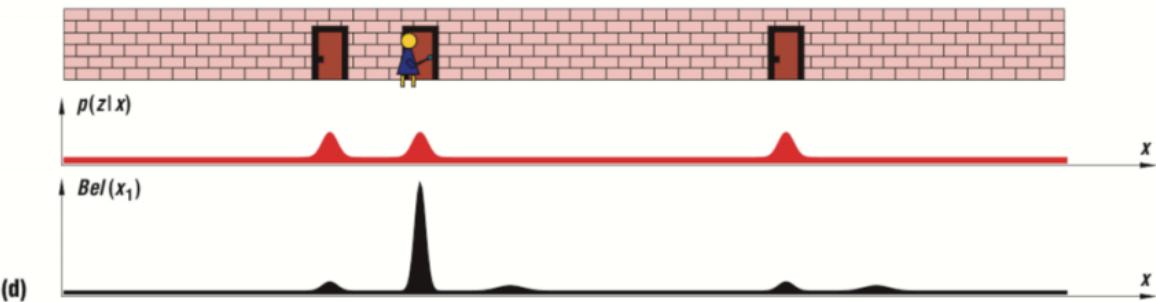
Il Bel- mi indica il Belief prima di effettuare l'osservazione. In questo caso, non ho nessuna storia dell'osservazione quindi l'individuo potrebbe essere in qualsiasi parte della stanza: la distribuzione è uniforme.



Alla prima osservazione sappiamo che abbiamo una probabilità più alta di essere vicino ad una porta. Il sensore può sbagliare, dunque la probabilità lontano da una porta è bassa, ma non zero. In questo caso $B(x_0)$ coincide con $p(z | x)$ perché la distribuzione precedente $B_{-}(x_0)$ è uniforme



Il sistema poi percepisce che l'utente si è spostato e confronta i picchi con quelli precedenti. I picchi sono meno marcati perché anche nell'osservazione dello spostamento ci possono essere degli errori.



Quando viene fatta la seconda osservazione, $p(z | x)$ non cambia rispetto a prima. $B(x_1)$ viene calcolato partendo da una distribuzione precedente non uniforme, otteniamo un picco alto dove un picco su $p(z | x)$ coincide con un picco $B^{-1}(x_1)$.

10.2 Calcolo della posizione adottando il modello Markoviano

Allargando lo spazio il costo computazionale aumenta.

Per applicare il modello markoviano dobbiamo specificare come calcolare:

- il modello di transizione di stato, dipende da molti fattori come ad esempio il modo di spostamento dell'utente, la mappa dell'ambiente, l'adozione di sistemi inerziali per il calcolo dello spostamento
- il modello percettivo, dipende principalmente da quali tecnologie di posizionamento stiamo adottando

Esistono vari approcci per calcolare il modello di transizione e percettivo:

- filtro di Kalman

- sistemi discreti che rappresentano lo spazio come una griglia o un grafo
- particle filter

Una delle difficoltà, dal punto di vista computazionale, deriva dal fatto che lo spazio sia una dimensione continua.

Le soluzioni basate sui filtri di Kalman si adattano al dominio continuo, forniscono una soluzione esatta e necessitano di alcune assunzioni sul problema stesso che spesso non valgono per il calcolo della posizione.

10.2.1 Approccio basato su griglia

Discretizzo (trasformo le dimensioni continue dello spazio in discontinue) lo spazio attraverso una griglia che copre l'intera area.

Immaginiamo di avere una griglia 3D, oppure un insieme di griglie 2D, una per piano. Ogni elemento della griglia è una cella.

All'inizio $B(x)$ è uniforme per tutte le celle x , ciò significa che non so dove si possa trovare l'utente.

Per ogni coppia di celle x_1, x_2 e per una distanza temporale Δt , posso calcolare la probabilità $p(x_{1t} | x_{2t-1})$ di essere in x_1 al tempo t , condizionata dal fatto che ero in x_2 al tempo $t-1$.

Questo calcolo può tener conto di:

- distanza spaziale e temporale
- un'osservazione di spostamento, ad esempio l'utente è rimasto fermo, oppure si è spostato per circa x metri in una direzione, ecc.)
- della struttura dell'ambiente, ad esempio l'utente non può attraversare i muri

Dato $B^-(x_t)$, per ottenere $B^+(x_t)$ mi serve calcolare $P(z_t | x_t)$. Per ogni cella x posso valutare la probabilità di fare l'osservazione z_t .

Ovviamente devo tener conto della tipologia di osservazione (segnaletica radio, riconoscimento di un marcitore, ecc.) Posso farlo perché ho un numero finito di celle. Dato $B^+(x_t)$ è semplice normalizzare e ottenere $B(x)$.

Assumiamo che per ogni coppia di x_1, x_2 il calcolo di $P(x_{1t} | x_{2t-1})$ richieda tempo costante e che per ogni cella x , il calcolo di $P(z_t | x_t)$ richiede tempo costante. Allora ottengo, dato n il numero di celle:

- Spostamento $O(n^2)$
- Percezione: $O(n)$

Abbiamo una complessità totale: $O(n^2)$

La complessità computazionale potrebbe essere maggiore, se ho complessità non costante per calcolare $P(x_{1t} | x_{2t-1})$ e $P(z_t | x_t)$

Fino ad ora abbiamo assunto di voler tenere traccia della posizione dell'utente, dunque abbiamo 3 dimensioni. Se volessimo tener traccia dell'orientamento, potremmo considerare l'orientamento come unico valore (la direzione in cui sta puntando l'utente, mentre si muove sul piano) anche se in alcuni contesti ci serve un orientamento a 3 dimensioni (i 3 angoli con i quali è orientato il dispositivo). Possiamo discretizzare la rotazione e avere altre 3 dimensioni:

- in questo caso dobbiamo avere una cella per ogni combinazione di posizione (3 dimensioni) e rotazione (altre 3 dimensioni)
- il numero di celle cresce esponenzialmente con il numero di dimensioni

In termini pratici vogliamo che le celle siano piccole, perché devono rappresentare la posizione/orientamento e se fossero troppo grandi non sarebbero precise.

Questa tecnica può andare bene in alcuni casi, caratterizzati da almeno alcuni di questi fattori:

- sono interessato solo a poche dimensioni
- ambienti di piccole dimensioni
- non ho bisogno di grande precisione, dunque posso usare celle più grandi
- gli aggiornamenti sono discontinui e ho elevata potenza di calcolo

10.3 Particle filter (Sequential Monte Carlo)

È una tecnica utile per calcolare la posizione secondo il modello Markoviano che calcola una soluzione approssimata, come per le griglie con la differenza che permette di gestire meglio il problema della discretizzazione dello spazio.

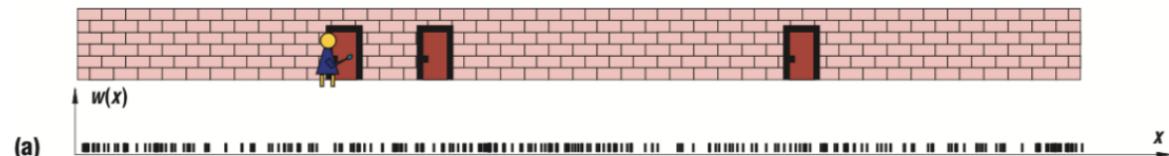
Facciamo tante ipotesi, ciascuna associata ad un peso (probabilità di quell'ipotesi).

Ogni coppia <ipotesi, peso> è chiamata particella.

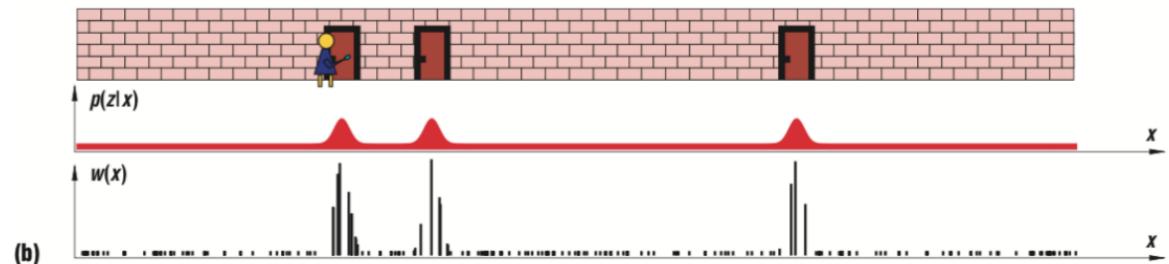
È più facile vedere le particelle come punti nello spazio.

La particella non rappresenta solo la posizione ma potrebbe anche includere anche l'orientamento.

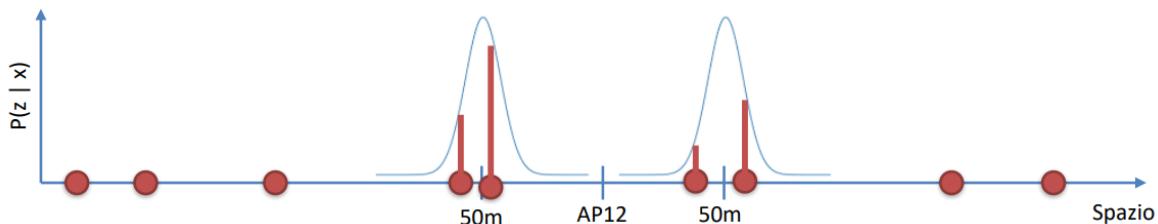
10.3.1 Funzionamento



All'inizio non sappiamo dove sia l'utente, le particelle vengono disposte a caso ed hanno tutte un peso uguale.



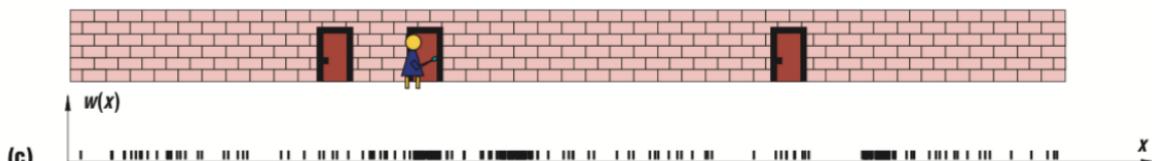
Quando arriva un'osservazione, cambiamo il peso di ogni particella e lo rendiamo proporzionale alla funzione di probabilità.



Ad esempio, abbiamo 9 particelle sparse nello spazio e percepisco l'antenna WiFi con AP12.

Data la potenza del segnale stimo di essere all'incirca a 50 metri di distanza dall'antenna, ma sappiamo che ci potremmo sbagliare.

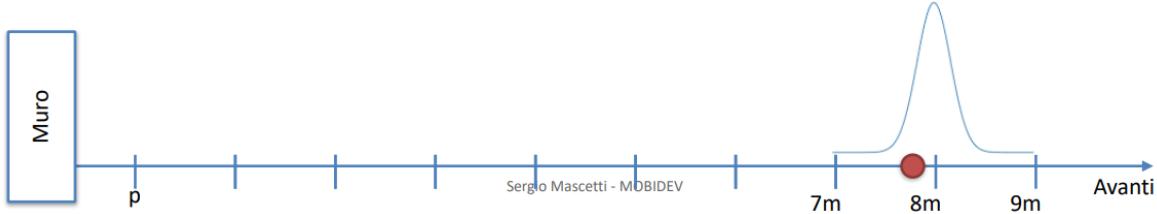
Assegno alla particelle un peso uguale a quello della gaussiana, cioè un valore molto basso, praticamente nullo alle particelle che non si trovano vicino ai 50m di distanza dall'AP.



Quando osservo uno spostamento, creo un nuovo insieme di particelle P' , ma il numero di particelle rimane uguale e tutte quante le particelle hanno lo stesso peso.

Le particelle vengono create così: parto da una particella esistente, ne scelgo una a caso pesata rispetto alla probabilità: ho maggiore probabilità di scegliere quella alta

rispetto a quella bassa. Vediamo meglio come funziona il calcolo dello spostamento.

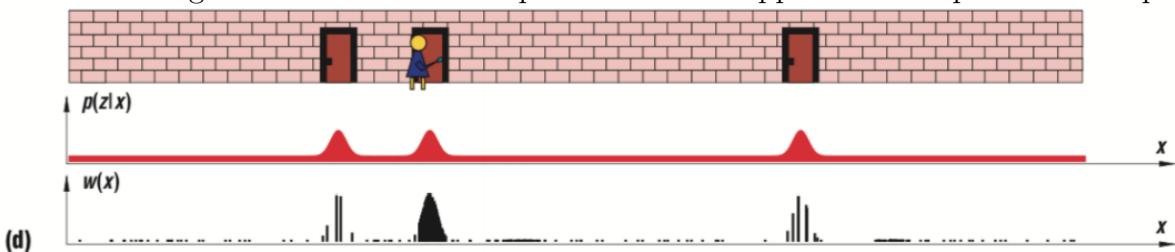


Abbiamo 1D, contiamo che l'utente fa 10 passi tutti nella stessa direzione, non sappiamo se avanti o indietro.

Partiamo da p , dalla planimetria si che l'utente non può camminare indietro perché c'è il muro, può andare solo avanti.

Sappiamo che un passo è lungo circa 80 cm (ma potremmo sbagliarci) e stimiamo dove potrebbe arrivare. In questa stima sceglieremo poi un punto. In pratica modelliamo la probabilità di arrivare in un dato punto con una gaussiana.

Scegliamo un punto di arrivo casuale all'interno della gaussiana seguendo la probabilità della gaussiana stretta. Il puntino rosso rappresenta la posizione di p' .



Quando ottengo una nuova osservazione, modifico il peso delle particelle secondo il modello percettivo. Alla prima porta e alla terza ho una bassa densità di particelle perché lo spostamento non era previsto qua. Dove invece era previsto lo spostamento, ho alta densità di particelle.

10.3.1.1 Quanta incertezza abbiamo?

Più incertezza modelliamo (cioè più è larga la nostra gaussiana), più lentamente il nostro sistema andrà a convergere, ma siamo più robusti rispetto agli errori. Se modelliamo meno incertezza (gaussiana più stretta, al limite estremo valore 1 solo per la distanza esatta di 10m), convergiamo prima, ma rischiamo di modellare casi errati, che potrebbero portarci a non convergere affatto. Con meno incertezza c'è il rischio che siccome le particelle tendono a spostarsi dove credono che sia la soluzione corretta, se in caso di errore convergono tutte verso una soluzione errata ad un certo punto tutti i pesi vanno a zero. Per prevenire questi casi, ad ogni iterazione si possono aggiungere delle particelle in posizione casuale.

10.3.1.2 Quante particelle servono?

Maggiore è il numero di particelle, meglio riesco a rappresentare il problema. Più è grande il numero di particelle più la complessità computazionale cresce. Al crescere delle dimensioni devo far crescere il numero delle particelle. Ad ogni iterazione calcolo circa 1000 particelle.

Sia l'approccio basato su griglie che il particle-filtering adottano una forma di discretizzazione dello spazio. Nell'uso delle griglie discretizzo lo spazio in modo costante e la complessità varia in base alla precisione che uso nel rappresentare lo spazio. Nel particle filter, invece la discretizzazione avviene in termini di particelle: approssimo la posizione (vera) dell'utente a quella di una particella (o di un gruppo di particelle).

La discretizzazione avviene in termini di particelle: approssimo la posizione (vera) dell'utente a quella di una particella (o di un gruppo di particelle)

Le particelle a differenza delle celle si spostano, là dove ci sono maggiori probabilità che ci sia l'utente avrò un numero più alto di particelle, dunque una discretizzazione più fine dello spazio.

10.3.1.3 Implementazione

Ci sono molte situazioni in cui devo stimare lo stato di un sistema sulla base di più dati soggetti a rumore. Il particle filter è una tecnica robusta, generalmente efficiente e abbastanza semplice da implementare. Permettono al programmatore di definire delle primitive per implementare il modello di spostamento e quello percettivo per una singola particelle. Questo è molto più semplice (dal punto di vista concettuale) che definire il modello su scala globale al sistema.

Augmented Reality

— Lezione 4 - 11 marzo 2021 —

Augmented Reality (AR) is a live, direct or indirect, view of a physical, real-world environment whose elements are augmented by computer generated sensory input such as sound, video, graphic or GPS data

AR è una vista live (in tempo reale) diretta (si può sovrapporre o sostituire al mondo reale) o indiretta (può aumentare una rappresentazione del mondo reale ad esempio una mappa) del mondo reale in cui andiamo a inserire delle informazioni, nell'ambiente, audio, video, grafica generate dal computer.

Abbiamo una sovrapposizione tra elementi del mondo reale e del mondo virtuale.

L'AR indica quindi gli ambienti in cui si mescolano elementi del mondo reale e altri del mondo virtuale. Quando si parla di sovrapporre elementi del mondo reale e del mondo virtuale, si parla di mixed reality. Si può dividere in due categorie:

- augmented reality: elementi virtuali nel mondo reale
- augmented virtuality: elementi reali nel mondo virtuale

Quando parliamo di un mondo AR dobbiamo dire cos'è del mondo reale e cosa del mondo virtuale.

Abbiamo diversi livelli di ambiente:

- reality: solo l'ambiente reale
- mixed reality
- virtual reality: solo l'ambiente virtuale

AR esiste già dagli anni '40, non era digitale ma era ottenuta con tecniche analogiche.

Un esempio di applicazione di AR è anche la pubblicità proiettata nelle partite di NBA, la pubblicità non c'è veramente, è aggiunta virtualmente.

L'AR non è solo visuale (anche se lo è per la maggior parte dei casi), ma può anche essere audio.

Abbiamo 4 principali problemi:

- Registration: ci permette di avere un modo di convertire le coordinate del mondo reale nelle coordinate dello schermo. Per fare ciò dobbiamo capire dove sta l'utente nel mondo reale
- Tracking: la registration ci permette di calcolare la conversione tra mondo reale e coordinate schermo in un dato istante, però dobbiamo tenere aggiornata questa cosa ad esempio quando muoviamo il device. L'aggiornamento si chiama tracking
- Display: gli oggetti virtuali nel mondo reale devono apparire realistici. Include sia operazioni di disegno, sia operazioni di comprensione dell'ambiente per far sì che quanto disegnato sia coerente con l'ambiente. Es: dove proiettare l'ombra dell'oggetto virtuale? Dobbiamo capire da dove arriva la luce. Abbiamo bisogno di tutta una serie di tecniche che ci permettano di capire dove devo collocare gli oggetti e dove li devo disegnare
- Interaction: Vogliamo che gli oggetti virtuali appaiano come facenti parte dell'ambiente reale, come ad esempio un oggetto virtuale che deve apparire più grande se mi avvicino e più piccolo se mi allontano. Per questo dobbiamo comprendere com'è fatto l'ambiente, dov'è l'utente, come si sta muovendo e ho bisogno di poter interagire con gli oggetti reali e virtuali. Questo problema prende il nome di interaction

11.1 Registration

L'obiettivo è quello di creare un collegamento tra il mondo reale e il sistema di output. Bisogna capire come mappare le coordinate del mondo reale in coordinate dello schermo e viceversa. La registration ha molti aspetti in comune con il calcolo della posizione perché richiede di capire dove si trova l'utente rispetto ad un sistema di riferimento globale o locale. Esistono 2 approcci alla registration:

- basato su segnale radio
- basato su visione

11.1.1 Registration con segnale radio

Uno dei problemi principali è che è difficile calcolare la direzione dell'utente. Questa tecnica si può usare sia con una visione egocentrica (con la vista dell'utente) oppure

per fare una visione allocentrica (vista dall'alto). Ad esempio un gioco in cui un utente, con una vista dall'alto, vede dei mostri virtuali muoversi su una mappa reale. L'utente stesso, perde se un mostro si avvicina a meno di 10 metri. Questo è un esempio di AR.

11.1.2 Registrazione visuale

I sistemi basati su registrazione visuale ci permettono di registrarcici rispetto ad un sistema locale. Non ci interessa la posizione rispetto al mondo ma rispetto a qualche sistema locale. Abbiamo tre soluzioni:

- marcatori esplicativi
- marcatori impliciti
- markerless

11.1.2.1 Uso dei marcatori esplicativi

I marcatori sono sempre stati utilizzati fin dai primi sistemi di AR, per definire un sistema di riferimento, e permettono di calcolare l'orientamento con 6 gradi di libertà, con buona precisione, e l'omografia: un sistema per mappare le coordinate del mondo reale nelle coordinate schermo e viceversa.

Gli oggetti visti dalla camera sono soggetti a trasformazione prospettica: le linee rette rimangono tali, ma angoli e distanze vengono alterati.

$$\begin{array}{c}
 \text{Coordinate} \\
 \text{immagine} \\
 \text{Camera}
 \end{array}
 \quad
 \begin{array}{c}
 \text{Parametri} \\
 \text{camera}
 \end{array}
 \quad
 \begin{array}{c}
 \text{Matrice omografica:} \\
 \text{traslazione e rotazione}
 \end{array}
 \quad
 \begin{array}{c}
 \text{Coordinate} \\
 \text{mondo reale}
 \end{array}
 \\
 \left[\begin{array}{c} x \\ y \\ 1 \end{array} \right] = s \left[\begin{array}{ccc} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{array} \right] \left[\begin{array}{cccc} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{array} \right] * \left[\begin{array}{c} U \\ V \\ W \\ 1 \end{array} \right]$$

Se riusciamo a calcolare questa matrice, possiamo convertire le coordinate.

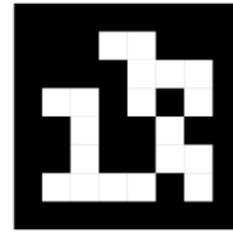
Dobbiamo creare un sistema di equazioni:

- alcuni parametri delle equazioni sono comuni (es: parametri della camera)

- altri parametri sono in relazione nota tra di loro (es: se ho un marcatore, posso sapere la distanza tra i vertici). Serve conoscere almeno 4 coppie di punti (mondo reale e camera)

Se abbiamo almeno 4 coppie di punti nel mondo reale in relazione nota tra di loro e conosciamo i 4 corrispettivi punti nelle coordinate schermo, allora riusciamo a dare uno schema di equazioni che ci dà una soluzione approssimata a questo problema.

Una tecnica per il calcolo della posizione è l'utilizzo dei marcatori ARUCO. Hanno una forma quadrata, simile ai qr-code. Includono un disegno in bianco e nero, solitamente formato da pixel abbastanza grandi. I disegni non devono essere simmetrici perché vogliamo capire anche l'orientamento della camera.



Come input abbiamo un'immagine A di un marcatore ARUCO, di cui è nota la dimensione, e un'immagine C presa dalla camera. Vogliamo calcolare se C contiene il codice ARUCO. Se non lo contiene non restituisco nulla, se lo contiene, restituisco la matrice omografica. La procedura opera in questo modo:

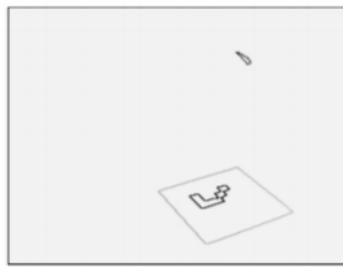
1. prende l'immagine dalla camera e la trasforma in bianco e nero. Prima l'immagine viene trasformata in scala di grigio, poi si usa un valore di soglia per trasformarla in bianco e nero



2. trova i quadrilateri. Si usano due tecniche:

- Canny edge detection: permette di trovare i bordi, ovvero dove pixel bianchi e neri sono uno di fianco all'altro
- Hough line segment detection: permette di trovare i segmenti di retta

Quando ho i segmenti di retta posso trovare i quadrilateri (4 segmenti consecutivi che si vanno a chiudere). In visione prospettica vengono alterati gli angoli, ma le linee rette rimangono rette



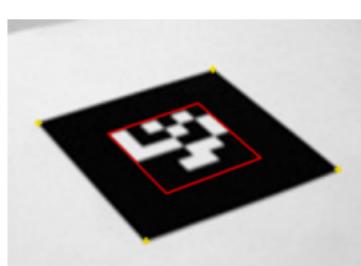
3. i quadrilateri trovati vengono processati uno ad uno e per ciascun quadrilatero Q:

- 3.1. calcola l'omografia O assumendo che Q sia A (visto in prospettiva). Sappiamo qual è la posizione degli angoli nel mondo reale? No, posizioniamo l'origine degli assi al centro dell'ARUCO.

Assumiamo che il quadrilatero Q sia effettivamente A, cioè che Q sia la visione prospettica di A. Prendiamo i 4 angoli del quadrilatero e calcoliamo l'omografia O usando le 4 equazioni degli angoli.

Se non è l'ARUCO che mi interessa, calcolo la matrice omografica in modo sbagliato

- 3.2. non so se quel quadrilatero sia l'ARUCO. Uso O per trasformare Q in Q', che è la vista di Q senza distorsione prospettica



4. se Q' corrisponde ad A, allora ritorno O. Se non corrisponde non torno nulla

Fino ad ora non ho tenuto conto dell'orientamento del marcatore. Dovremmo calcolare ogni omografia per ciascuna possibile rotazione. Per ogni quadrilatero si possono calcolare 4 omografie (una per ogni possibile rotazione) oppure quando vado a confrontare Q' con A, provo 4 volte, ruotando Q' in tutte le possibili direzioni, poi se trovo la corrispondenza, adatto l'omografia.

Lezione 5 - 15 marzo 2021**11.1.2.2 Uso dei marcatori impliciti**

Data un'immagine, vogliamo capire dove ci troviamo rispetto ad essa. Gli ARUCO sono quadrilateri, invece un marcitore in un'immagine non è detto che sia un quadrilatero, infatti può assumere diverse forme, come ad esempio una scatola di biscotti.

Il riconoscimento di questi marker avviene con tecniche di template matching. Abbiamo due immagini:

- un template T: un'immagine di ciò che vogliamo riconoscere
- un'immagine C catturata con la camera

Vogliamo sapere se il template T compare nell'immagine C. Serve il concetto di feature points.

I feature point sono piccole parti di un'immagine, tipicamente un insieme di pixel attorno ad un altro pixel. Ogni feature point è associato ad un feature descriptor. Scegliamo dei feature points che rimangono più o meno invariati anche se viene cambiata l'angolazione da cui si guarda.

Nell'esempio in figura 11.1, il feature descriptor sono i 16 pixel attorno.

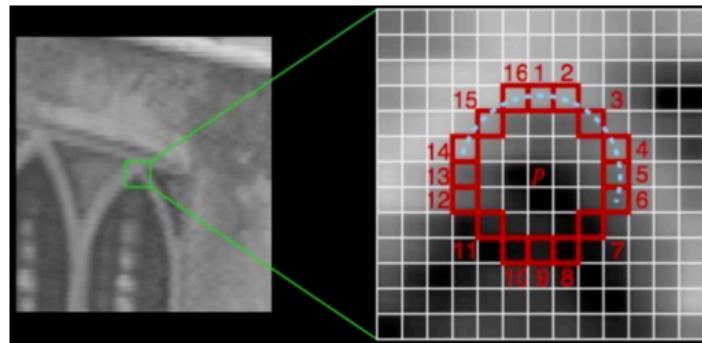


Figure 11.1

Il pattern matching opera nel seguente modo:

- identifico il feature point nel template T e nell'immagine C presa dalla fotocamera
- trovo un modo per “far combaciare” (match) i feature points di T con quelli in C
- quando trovo il match corretto, se ho almeno 4 coppie di punti (4 in T e i corrispondenti 4 in C), posso calcolare l'omografia come per i marcatori esplicativi.

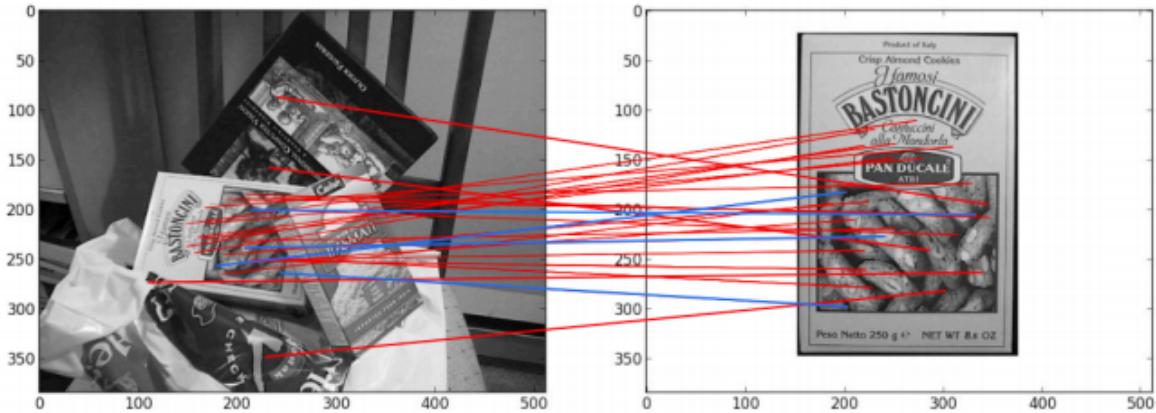


Figure 11.2

I feature points hanno un descriptor, dunque tipicamente cercherò di far combaciare due feature points (uno in T e l'altro in C) con descriptor simili, tuttavia potrebbero potrebbero esserci tanti feature points con descriptor simili, come in figura 11.2.

La prima tecnica è di provare tutte le combinazioni di punti, ma non funziona bene perché, anche se trovassi la combinazione giusta, avrei tante coppie il cui match non andrebbe bene. Quando trovo la combinazione giusta, basta qualche errore piccolo che non riesco a trovare la soluzione del sistema.

Per risolvere il problema c'è RANSAC (Random sample Consensus): l'idea è di andare in modo casuale, ma in un modo simile agli ARUCO. Sceglie un'associazione casuale di 4 punti in T e C, e assumendo che queste 4 associazioni siano corrette, calcola l'omografia O. Usa O per verificare quanti altri punti combaciano. Si ripete il procedimento fino a quando non si ottiene un certo livello minimo di punti che combaciano. L'algoritmo ha un aspetto in comune con quello usato per i marcatori esplicativi: si ipotizza che una certa soluzione sia corretta e con quella si calcola l'omografia. Poi si usa quell'omografia per verificare se la soluzione è effettivamente corretta.

11.1.2.3 Registrazione visuale senza marcatori (markerless)

In questo caso non abbiamo un'informazione predefinita da riconoscere, ma usiamo le caratteristiche dell'ambiente stesso per capire come ci stiamo spostando nell'ambiente. Il problema è identificare in un'immagine i feature point di un oggetto non noto a priori non ci dice nulla sulla nostra posizione rispetto all'oggetto stesso. Ad esempio potremmo essere molto vicini ad un oggetto piccolo oppure molto lontani da un oggetto che appare identico, ma è grande e lontano.

Per avere dei sistemi di riferimento, usiamo due immagini e confrontiamo i feature

points delle due immagini. Usiamo questi punti per capire come si rimappano tra di loro le due immagini.

Se le immagini invece sono prese da punti diversi nello spazio, prendo due immagini (o nello stesso istante se ho un dispositivo con fotocamere multiple, oppure 2 immagini a breve distanza) e uso la differenza prospettica per capire la distanza.



11.1.3 Il sistema di riferimento

Quando facciamo la registration dobbiamo preoccuparci che sistema di riferimento adottare: dove posizioniamo l'origine degli assi? Questo dipende da cosa vogliamo fare:

- alcuni sistemi adottano una registrazione su scala globale
- in molti altri casi, è sufficiente un sistema di riferimento locale che tipicamente viene scelto in corrispondenza di un punto interessante (es: un marcatore o il volto dell'utente) o in un punto qualsiasi (es: nel punto in cui si trova il dispositivo quando viene fatta la prima registration)

11.2 Il problema del tracking nei sistemi AR

Abbiamo visto che è possibile calcolare la matrice omografica, che permette di convertire un punto tra le coordinate schermo e quelle del mondo reale, ma appena ci spostiamo la matrice omografica non va più bene. Il device continua a muoversi e quindi bisogna tenere aggiornata la matrice omografica. Questo è il problema del tracking.

La registration vale quindi per un frame, ma dobbiamo fare la registration per ogni frame?

Vogliamo fare in modo di calcolare una matrice H^I che rappresenti lo spostamento rispetto a quando ho fatto la registrazione H e in modo tale che il prodotto tra le matrici sia la nuova matrice omografica.

Abbiamo 3 possibili soluzioni al problema del tracking:

1. registration ripetuta
2. tecniche visuo-inerziali
3. registration e tracking combinate

11.2.1 Registration ripetuta

È la soluzione più semplice, non si fa il tracking e si continua a ripetere la registration. È poco efficiente, la registration è complessa e non vogliamo farla ad ogni frame. Potrebbe anche non funzionare: prendiamo un marcitore, fino a quando inquadriamo il marcitore possiamo fare la registration, quando non lo inquadrano più, non la possiamo più fare. Dunque, se inquadro il marcitore una volta e poi sposto il device in modo che il marcitore non sia più inquadrato, non riesco ad aggiornare la posizione. In questo caso devo interrompere la sessione AR.

11.2.2 Tecniche visuo-inerziali

Le tecniche visuo-inerziali calcolano lo spostamento (inteso come variazione di posizione e orientamento) e combinano:

- tecniche inerziali: sono le stesse tecniche viste nell'indoor positioning. Quando supponiamo che il sistema sia in uno stato di quiete, possiamo calcolare la posizione rispetto alla gravità. Possiamo calcolare quando il device si muove grazie agli accelerometri e quando ruota grazie ai giroscopi. Durante gli spostamenti integriamo il valore dei giroscopi e doppio-integriamo quello degli accelerometri per calcolare la posizione rispetto al nostro sistema di riferimento iniziale. Queste operazioni di integrazione aumentano l'errore, che causa drift e richiede un frequente fix della posizione
- tecniche visuali: è possibile stimare lo spostamento della camera usando tecniche chiamate optical flow. L'idea è quella di cercare di stimare come è stata traslata/ruotata la camera guardando come si sono spostati alcuni punti nell'immagine

11.2.2.1 Optical flow

Dato un insieme di punti in un frame al tempo $t-1$ vogliamo calcolare come questi punti si spostano nel frame successivo al tempo t . Con punto si intende un pixel o gruppo di pixel di un oggetto del mondo reale.

Ad esempio: supponiamo di avere un'immagine perfettamente bianca, con un solo pixel verde. È molto difficile capire come si sono spostati i pixel bianchi, invece posso facilmente capire come si è spostato il pixel verde. Potrebbero comunque esserci molti casi particolari: ad esempio il pixel verde rappresenta un led che si spegne e un altro led, in posizione differente si accende.

Quali punti consideriamo?

- Approcci “dense”: tutti i punti dell’immagine
- Approcci “sparse”: solo per alcuni punti caratteristici (es: feature points)

In linea teorica, il problema non è risolvibile, perché non sappiamo nulla di come cambiano i punti tra un’immagine e l’altra, ad esempio potrebbe cambiare l’intensità per via dell’illuminazione. Ciascun punto potrebbe spostarsi in qualunque punto dell’immagine o anche sparire fuori dall’immagine al tempo t , così come al tempo t potrebbero apparire dei punti che non erano inquadrati al tempo $t-1$.

In realtà si possono fare alcune assunzioni che rendono il problema risolvibile, in modo efficace, in molti casi pratici:

- small motion: i punti non si muovono molto lontano
- brightness constancy: le forme rimangono (più o meno) simili
- spatial coherence: i punti si muovono più o meno come i loro vicini

Presi due frame a distanza molto ravvicinata, queste assunzioni sembrano ragionevoli, almeno in molti casi. Anche basandosi su queste assunzioni la soluzione del problema dell’optical flow non è banale.

La tecnica è ancora più efficiente se viene adottata su un insieme limitato di punti, per esempio si può adottare per i feature points, già identificati per la registrazione markerless. Non calcoliamo lo spostamento per tutti i punti dell’immagine ma solo per alcuni.

Le tecniche inerziali e visuali possono essere combinate, per realizzare una sistema maggiormente robusto, con tecniche come filtri di Kalman o particle filtering. Le librerie di AR esistenti adottano tecniche di calcolo dello spostamento visuo-inerziali. Queste tecniche risolvono il problema del marcatore con la registration: se non inquadro il marcitore riesco a calcolare la matrice omografica.

11.2.3 Registration e tracking combinate

Se faccio la registrazione una volta e poi continuo ad usare tecniche visuo inerziali, l'errore viene sempre più accumulato. Nei sistemi di AR questo significa che gli oggetti virtuali mostrati a schermo non sono solidali con l'ambiente.

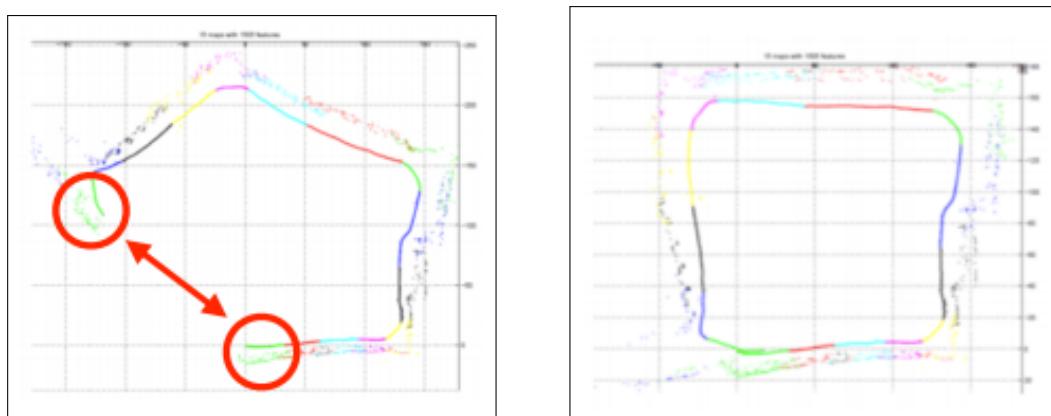
Per risolvere il problema, concettualmente vorremmo avere un modo per azzerare il drift periodicamente. Quando usiamo tecniche markerless, troviamo la posizione della camera rispetto a certi oggetti (landmark) caratteristici dell'ambiente, ma questo non ci permette di fare un fix di posizione, perché non sappiamo dove sono quegli oggetti.

Se noi inquadrriamo un landmark, poi ci spostiamo e poi ritorniamo ad inquadrarlo, possiamo usare quello stesso landmark come fix di posizione. Queste operazioni di registration e tracking diventano parte intergrante di un'unica tecnica che si chiama SLAM (simultaneous localization and mapping).

11.2.3.1 Esempio SLAM

Supponiamo di girare attorno ad un edificio e mentre lo facciamo, identifichiamo i feature points. L'edificio è rettangolare, ma il sistema visuo-inerziale sta sbagliando: ha calcolato gli angoli in modo errato e dunque il rettangolo non si chiude.

Se però riusciamo ad accorgerci che siamo tornati nello stesso punto della partenza (perché riconosciamo gli stessi feature points) possiamo correggere l'errore, non solo nella posizione corrente, ma anche in tutte quelle precedenti.



Augmented Reality - display e interaction

———— Lezione 6 - 19 marzo 2021 ——

12.1 Display visuale

Il problema dell'output verso l'utente può essere affrontato in due modi:

- visuale
- sonoro

Per poter posizionare gli oggetti virtuali, serve un sistema di riferimento, chiamato sessione, che definisce dove sono gli assi e come sono orientati. Il sistema di riferimento ci serve perché quando posizioniamo gli oggetti, dobbiamo capire in base a cosa li posizioniamo.

Il programmatore può posizionare un oggetto virtuale indicando le coordinate 3D di dove esso deve essere posizionato, quindi viene indicata la posizione dell'oggetto virtuale nel mondo reale. Il programmatore posiziona l'oggetto nel mondo e poi la libreria di AR usa l'omografia per convertire le coordinate dell'oggetto virtuale da quelle 3D del mondo a quelle 2D dello schermo.

Ci sono un po' di problemi pratici da risolvere, ad esempio più punti 3D corrispondono ad un singolo punto 2D. Il punto che scelgo da visualizzare è quello più vicino all'utente.

Quando vogliamo mostrare un oggetto 3D abbiamo due strade:

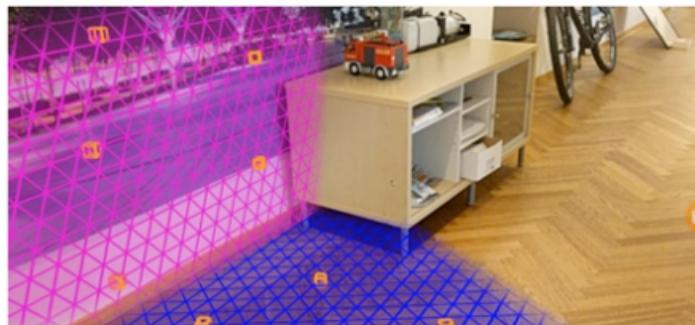
- definire un oggetto 3D statico (a tempo di programmazione) e lo inseriamo nella sessione, ad esempio creo un modello di un aereo con un programma di moderazione 3D e da codice lo inserisco nella scena quando necessario
- inserire dinamicamente degli oggetti 3D all'interno della scena e modificarne le

proprietà, ad esempio viene inserito un cubo di una certa dimensione in una certa posizione, una determinata texture, ecc.

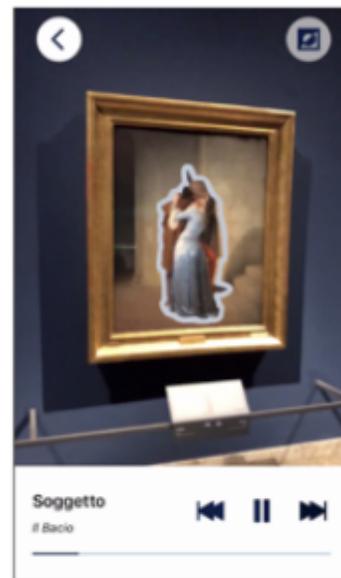
Il problema della visualizzazione è quello di mostrare oggetti in modo realistico e questo richiede di comprendere l'ambiente. Per questo motivo le librerie di AR includono una serie di funzionalità.

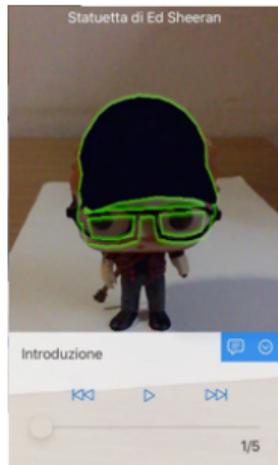
12.1.1 Visualizzazione realistica

Quando posizioniamo un oggetto vogliamo che rispetti i principi della fisica, ad esempio ci aspettiamo che la sedia sia appoggiata a terra. Una delle caratteristiche principali è quella di riconoscere i piani verticali e orizzontali. Quando gli oggetti vengono riconosciuti, la libreria di AR ne calcola la posizione, sempre nel sistema di riferimento della scena, dunque è semplice posizionare un oggetto “sopra”.



In alcuni casi gli oggetti virtuali hanno significato solo in presenza di altri oggetti del mondo reale che possono essere sia 2D che 3D. Molte librerie di AR rendono disponibili delle funzioni per riconoscere dei template 2D ed anche in questo caso, la libreria di AR fornisce la posizione nello spazio di queste immagini, così da rendere possibile il disegno di altri oggetti virtuali. Un esempio ne è un sistema che riconosce un quadro e disegna una linea che evidenzia una parte di esso.





Alcune librerie di AR (tra cui ARKit) permettono anche di riconoscere un template 3D. L'app riconosce il template (la statuetta) fisico e disegna oggetti virtuali sopra di esso.

Un altro problema è relativo alla sorgente luminosa. Dobbiamo identificare la sorgente luminosa per disegnare l'oggetto e per disegnare le ombre.

12.1.2 Problema dell'occlusione

Il problema nasce quando dobbiamo far interagire il sistema reale con quello virtuale. Si ha quindi il problema quando un oggetto fisico si frappone tra la camera e un oggetto virtuale, ma il sistema di AR non lo sa e disegna l'oggetto virtuale come se apparisse di fronte a quello reale.

Per risolvere il problema dell'occlusione è necessario comprendere se c'è un oggetto reale che si interpone tra la camera e un oggetto virtuale e capire esattamente dove si trova. In questo modo è possibile identificare quali parti dello schermo non debbano essere disegnate con gli oggetti virtuali. Questo richiede delle capacità di riconoscere la distanza fisica degli oggetti dalla camera.

12.1.3 Riconoscimento della scena

Le librerie di AR rilasciano nuove funzionalità per riconoscere la scena. Questo sviluppo delle funzionalità procede di pari passo con la potenza computazionale dei device e con la disponibilità di nuovi sensori. Per esempio, grazie al lidar, ARKit riesce a riconoscere un insieme di oggetti.

Ci sono casi in cui vorremmo togliere oggetti reali aggiungendo oggetti virtuali, ad esempio tolgo il mobile reale, per mettere quello virtuale dell'ikea. Il problema è che devo stimare cosa c'è dietro l'oggetto. Attraverso l'uso di tecniche di machine learning si riesce a stimare l'immagine da mostrare sulla base di quella nota, ad esempio ripetendo un pattern noto sullo sfondo.

12.2 Display audio

L'audio può essere usato come componente di output di un sistema di AR:

- in assenza di informazioni visive: es, in un sistema di supporto alle persone non vedenti
- in combinazione ad informazioni visive, per dare maggiore realismo agli elementi virtuali

L'audio si ottiene sfruttando un principio percettivo del suono. Percepiamo il suono in un modo diverso da un orecchio all'altro. Le differenze tra i due canali possono essere:

- interaural level difference (ILD): simulo una differenza di volume nell'orecchio destro e sinistro. Se c'è un suono a sinistra di una persona, la persona sentirà il volume del suono maggiore nell'orecchio sinistro
- interaural time difference (ITD): impostare una differenza di tempo nel canale. Se c'è un suono a sinistra di una persona, l'orecchio sinistro di quella persona riceverà il suono prima dell'orecchio destro

Conosciamo il suono nell'ambiente simulato, ma non conosciamo la propagazione del suono nell'ambiente reale e per questo ci sono diversi problemi:

- della posizione della testa: sappiamo com'è orientato il dispositivo ma non sappiamo com'è orientata la testa dell'utente. È rischioso assumere che l'utente sia orientato come il device (es: se gira la testa?). Si possono pensare varie soluzioni come l'uso di hardware posizionato sulla testa dell'utente, come ad esempio cuffie con sensori inerziali o il riconoscimento della posizione del volto rispetto al device (al momento non è possibile effettuare una sessione AR sia con la camera frontale che con quella posteriore)
- dell'ambiente: com'è l'ambiente virtuale per cercare di capire come si propaga il suono nel mondo reale. Lo stesso suono sarà percepito in modo differente cambiando l'ambiente (es indoor o outdoor). La comprensione dell'ambiente può aiutare a generare un suono più realistico
- anatomia dell'ascoltatore: la forma della testa e del torso impattano su come una persona percepisce il suono. Esistono dei modelli chiamati HRTF (Head Related Transfer Functions) che permettono di modellare come un utente percepisce il suono in base a molti fattori

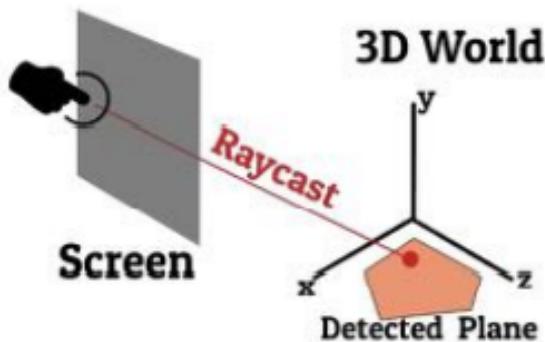
12.3 Interazione

I sistemi di AR possono richiedere all'utente di interagire con oggetti reali e virtuali. Abbiamo due problemi:

- come selezionare un elemento del mondo 3D partendo dalle coordinate 2D dello schermo
- come permettere all'utente di manipolare oggetti nel mondo 3D

In entrambi i casi abbiamo coordinate schermo 2D e del mondo reale 3D. Quando abbiamo un oggetto virtuale 3D, ogni punto si rimappa in al più una coordinata dello spazio bidimensionale. Quando vogliamo la cosa inversa, abbiamo che in un punto 2D corrispondono infiniti punti nel mondo 3D che corrispondono a quel punto.

Le librerie di AR rendono disponibile Hit test, una funzione che preso in input un punto delle coordinate schermo, ritorna l'elenco di tutti gli oggetti riconosciuti che sono toccati da questo raggio. Si proietta una linea (chiamata raycast) che parte dal punto 2D indicato e si estende seguendo l'apertura focale della camera. L'hit test ritorna l'elenco di oggetti intersecati.



Quando vogliamo manipolazione gli oggetti virtuali, abbiamo il problema della dimensionalità. Gli oggetti virtuali hanno 9DOF (posizione, rotazione e dimensione sui 3 assi) e devo capire come gestire questi 9 gradi. Ci sono due approcci principali:

- integrality: l'utente può interagire con tutte le dimensioni in contemporanea
- separability: l'utente può manipolare una dimensione per volta

L'interazione con gli oggetti può avere 3 soluzioni:

- tramite schermo:

- tramite proxy fisici
- movimenti delle mani nello spazio

12.3.1 Interazione tramite schermo

Si può usare un approccio basato su integrality, perché possiamo usare gesti diversi per dimensioni diverse, ad esempio spostare un oggetto con tocco con due dita, cambiare la dimensione con il pinch ecc. Tuttavia questi gesti non sono standard e l'utente deve essere istruito su come fare.

Il problema è che con il tocco sullo schermo posso spostare a destra/sinistra, e in alto/basso, ma non posso avvicinare/allontanare l'oggetto. Le possibili soluzioni:

- aggiunta di un controllo per la distanza: stiamo applicando il principio della separability
- faccio in modo che almeno una dimensione dell'oggetto virtuale sia vincolata al mondo fisico (es: l'oggetto non vola, ma è appoggiato per terra). In questo caso l'oggetto ha due gradi di libertà sulla posizione e lo posso riappare più facilmente sulle schermi

12.3.2 Interagire tramite proxy fisici

Possiamo avere oggetti virtuali controllati tramite oggetti fisici, ad esempio spostiamo un oggetto virtuale spostando un marcatore esplicito. Un esempio potrebbe essere la bacchetta nell'immagine, che ha diversi marcatori aruco, grazie ai quali è possibile calcolare non solo la posizione ma anche la rotazione. L'immagine è tratta da un sistema che permette di toccare un oggetto e capire quale oggetto è stato toccato.



Le soluzioni basate su gesti sullo schermo sono limitate dal fatto che lo schermo è 2D, mentre lo spazio fisico è 3D. Un'idea potrebbe essere che l'utente possa fare gesti direttamente nello spazio 3D in modo tale da rendere l'interazione più naturale, ad esempio "prendo" un oggetto virtuale con due dita e lo sposto/ruoto dove voglio.

Augmented Reality in pratica

———— Lezione 7 - 25 marzo 2021 ———

13.1 Panoramica

Le due librerie principali per lo sviluppo di applicazioni AR sono:

- ARKit di Apple: è utilizzabile sui device mobili Apple recenti e non utilizzabile su simulatore
- ARCore di Google: può essere utilizzata in nativo su Android ed è utilizzabile, con alcune limitazioni, su simulatore

13.1.1 Macro funzionalità

Per realizzare applicazioni in realtà aumentata, ARKit e ARCore forniscono funzionalità per semplificare le seguenti operazioni:

- Registration e Tracking
- Scene
- Display
- Interaction

ARKit e ARCore condividono alcuni concetti base:

- scene (ARKit) / session (ARCore): due nomi diversi per lo stesso concetto, che rappresenta l'ambiente rispetto al quale effettuiamo la registration. Gli oggetti virtuali e reali sono posizionati nel sistema di riferimento della scena/sessione. Fornisce le chiamate per accedere agli altri elementi di AR
- configuration: definisce le caratteristiche della scena / sessione da creare:

- rispetto a cosa facciamo la registration, ad esempio rispetto all’ambiente 3D circostante, viene creato un sistema di riferimento in una posizione iniziale e si tiene traccia dello spostamento rispetto a questo sistema
- rispetto a cosa siamo interessati a riconoscere
- ancora: rappresentano dei punti di riferimento agli oggetti virtuali o reali identificati. Es: quando un oggetto viene riconosciuto, viene associato ad un’ancora. Attraverso l’ancora possiamo risalire alla posizione e rotazione (6DOF) dell’oggetto rispetto alla scena. Le ancore si usano ad esempio quando riconosciamo i piani ed ogni piano è rappresentato dalla propria ancora
- hitTest/rayCasting: si tratta di un’operazione per convertire le coordinate schermo (2D) in un punto 3D. Concettualmente devo considerare il raggio, che “esce” dal device verso lo spazio, che interseca zero o più oggetti (reali o virtuali) ed hitTest ritorna l’insieme di questi oggetti

13.1.2 Comprendere la scena

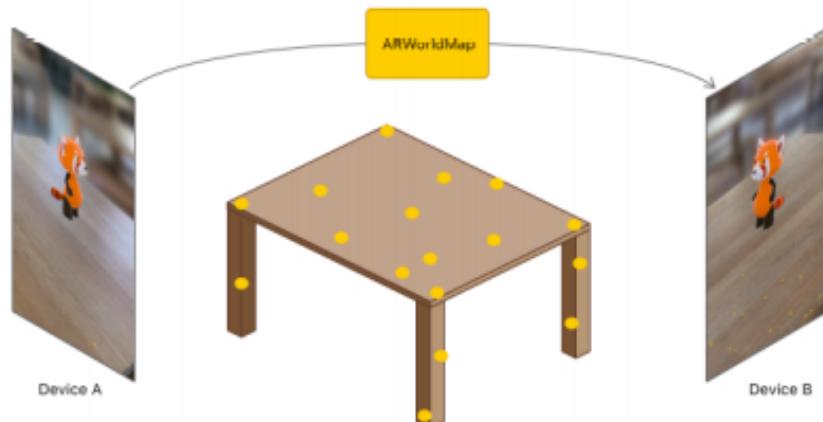
ARKit e ARCore forniscono alcuni strumenti che semplificano il processo di comprensione della scena:

- riconoscimento piani
- riconoscimento immagini (2D): la procedura avviene in 3 passi:
 - aggiunta dell’immagine alle risorse del progetto: si fornisce ad ARKit/ARCore l’immagine da riconoscere e la dimensione attesa nel mondo reale
 - modifica della configurazione della scena, indicando, da codice, che si vuole riconoscere quell’immagine
 - specificare cosa fare quando l’immagine viene riconosciuta: si gestisce l’evento di quando l’ancora relativa a quell’immagine viene aggiunta
- riconoscimento dei volti
- riconoscimento oggetti (3D): dobbiamo prima ottenere un modello 3D dell’oggetto e poi il procedimento di riconoscimento è analogo a quanto avviene per le immagini 2D

13.2 Alcune funzionalità di ARKit 4

13.2.1 AR multiuser

Sia in ARKit che in ARCore si possono avere più scene/sessioni condivise tra più utenti. Prendiamo ad esempio un utente identifica dei feature point e poi si identifica tramite l'ambiente. Se quelle stesse informazioni vengono condivise con un altro utente e il device di questo, riesce a capire dove si trova rispetto ai fp del primo utente, possiamo fare la registrazione rispetto ad un sistema di riferimento comune. Questo permette di mostrare un oggetto virtuale nello stesso punto a diversi utenti con diverse angolazioni.



13.2.2 Scene geometry

I device che hanno un sensore di profondità (Lidar) possono creare una ricostruzione topologica dell'ambiente. La libreria AR per i dispositivi con Lidar mette a disposizione funzionalità per:

- riconoscere gli oggetti
- calcolare quando un oggetto si interpone tra la camera e un oggetto virtuale, così da risolvere il problema dell'occlusione
- permettere agli oggetti virtuali di rispettare alcune regole della fisica nell'interazione con oggetti del mondo reale (ad esempio due oggetti che non si possono compenetrare)

Attraverso il lidar è anche possibile andare a creare una configurazione chiamata body tracking dove il sistema riconosce le principali articolazioni del corpo e ce le restituisce come nodi, ovvero punti che possiamo tracciare, cosicché possiamo calcolare la posizione del corpo e come si muove l'utente. Questa procedura si chiama **Motion capture**.

Activity recognition - Gabriele Civitarese

———— Lezione 8 - 8 aprile 2021 ——

14.1 Introduzione al riconoscimento di attività

L'activity recognition si colloca nel context-awareness, dove le applicazioni sono consce del contesto.

Ci sono diversi esempi di uso di contesto nelle applicazioni mobili:

- app che ricorda di scendere alla fermata del tram giusta
- app che suggerisce la strada in base al traffico attuale (tipo waze)
- app che ricorda di prendere l'ombrellino prima di uscire di casa quando piove (capisce quando l'utente è a casa e che c'è un tempo sfavorevole)

Uno dei più importanti aspetti dell'analisi del contesto è capire cosa fa l'utente, attraverso le tecnologie sensoristiche, e questo prende il nome di activity recognition.

Le attività svolte dall'utente possono essere:

- fisiche: attività di basso livello, relative a specifici movimenti, che vengono monitorate con dispositivi mobili o dispositivi wearable. Sono attività come correre, camminare, pedalare, ecc.
- quotidiane: attività di alto livello che possono essere monitorate tramite sensori ambientali (es: smart-home). Sono attività come mangiare, cucinare, dormire, ecc.

Ci sono diverse soluzioni commerciali che monitorano le nostre attività e ci sono diversi servizi con le relative app che le mostrano, ad esempio il monitoraggio dello stile di vita, la qualità del sonno, il riconoscimento di attività in base ai luoghi visitati, ecc..

Sia Android che iOS mettono a disposizione delle API per riconoscere l'attività svolta dall'utente e possono essere usate per creare app che si adattano all'attività svolta dall'utente.

Il riconoscimento di questi sistemi non sempre è accurato, infatti una stessa attività può essere fatta da diversi utenti in modi differenti. C'è anche il problema che due attività diverse ma simili, vengono riconosciute come se fosse la stessa attività.

14.1.1 Riconoscimento

I dispositivi mobili dispongono di diversi tipi di sensori che permettono di tracciare in modo continuo i movimenti fisici delle persone e che producono dati grezzi, cioè non ancora processati. I dati vengono pre-processati e viene fatta pulizia. Vengono passati ad un modello di riconoscimento, che può essere un sistema di AI, che capisce come mappare i dati pre-processati in attività e fornisce una predizione, cioè una distribuzione di probabilità (stai correndo al 80%, stai camminando al 10%, stai salendo le scale al 10%). Viene poi fornito l'output del sistema, secondo l'attività più probabile.

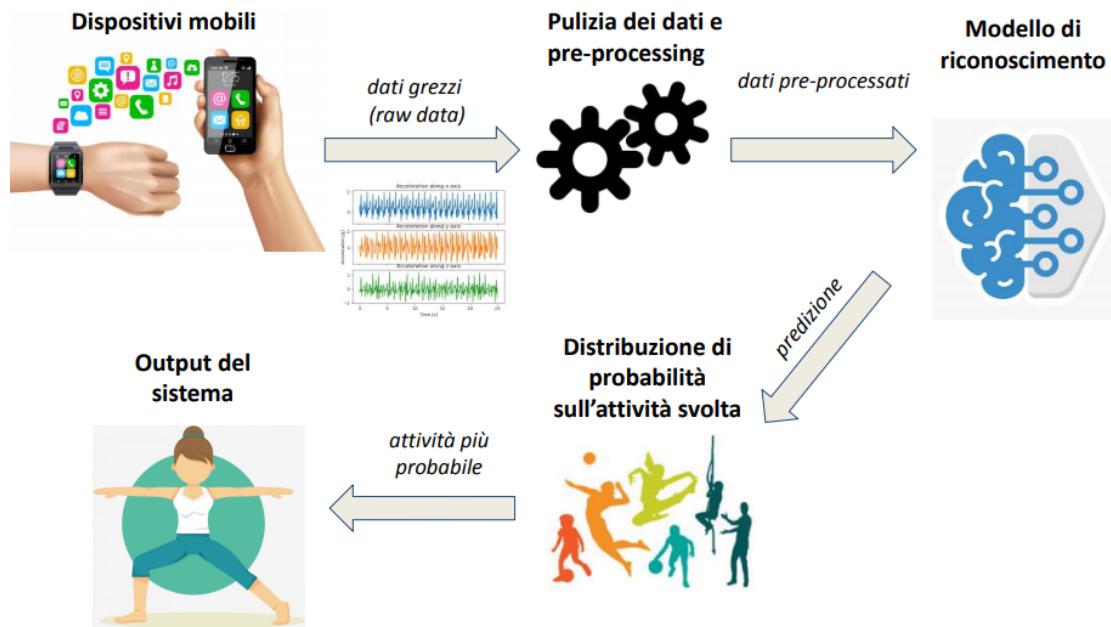


Figure 14.1: Riconoscimento

14.2 Acquisizione dei dati grezzi di movimento dai dispositivi mobili

I sensori inerziali, visti nella sezione 1.1.5.1, sono sensori che ci dicono qualcosa sui movimenti delle persone:

- accelerometro: sui 3 assi ci dice la variazione di velocità che lo smartphone sta subendo. Si misura in m/s^2
- giroscopio: sui 3 assi indica la velocità angolare. Si misura in $grad/s$
- magnetometro: rileva la variazione nel campo magnetico. Si misura in $micro - Tesla$

Ognuno di questi sensori produce dati su tutti e tre gli assi.

Oltre ai sensori fisici, i sistemi operativi mettono a disposizione dei sensori virtuali, come ad esempio il contapassi.

I sensori devono campionare una misurazione in intervalli e più la frequenza di campionamento è alta, più l'informazione è accurata, ma la computazione è più pesante e ci sono più dati da analizzare in fase di analisi.

14.3 Tecniche di machine learning per il riconoscimento di attività

Ci sono tecniche analitiche che permettono di analizzare il segnale e da cui possiamo capire cosa sta facendo una persona, ad esempio "se il valore dell'accelerometro sull'asse x è maggiore di 0.4 e il valore del giroscopio... allora l'utente sta correndo".

Le soluzioni analitiche hanno diversi limiti:

- ogni attività può essere svolta in modi molto diversi dalla stessa persona
- diverse persone possono svolgere la stessa attività in modo molto diverso
- attività diverse hanno dei pattern di movimento molto simili
- i dati dei sensori sono soggetti a rumore

Il riconoscimento di attività fisiche si basa principalmente su tecniche di machine learning (ML), ovvero metodi statistici il cui scopo è trovare in modo automatico "pattern nascosti" nei dati. Queste tecniche permettono di costruire un modello di attività che viene utilizzato per classificare i dati dei sensori.

Un problema su cui si basa activity recognition nel machine learning è la classificazione: dato un input X , vogliamo sapere a quale delle classi dell'insieme Y appartiene. Nel ML, $X (\in \mathbb{R}^n)$ è un vettore n-dimensionale che rappresenta il dato che vogliamo classificare. Y è l'insieme delle attività, l'insieme delle possibili classi $\{y_1, y_2, \dots, y_k\}$. Ad ogni X corrisponde una classe y_i , ma non è nota. Vogliamo trovare una funzione $h(X)$ che approssimi al meglio il mapping tra elementi di \mathbb{R}^n ed elementi di Y . Un sistema che implementa h è detto **classificatore**.

$$h : \mathbb{R}^n \rightarrow Y$$

Il ML non è una tecnica perfetta, ma un'approssimazione e si cerca che sia il più precisa possibile.

Ci sono diversi algoritmi di classificazione e sono divisi in categorie:

- apprendimento supervisionato: si insegna con dei dati, in una fase precedente all'algoritmo, a riconoscere il mapping tra dati di sensori e attività. Una prima fase iniziale in cui vengono forniti (da parte del programmatore) degli esempi, e il sistema di ML sulla base di questi, impara ad associare i dati alle attività. Si ha un'etichetta associata all'attività dei dati
- apprendimento non supervisionato: abbiamo i dati iniziali, ma non si sa quale dato corrisponde a quella attività. Si usano tecniche di clustering per raggruppare, con delle metriche di similarità, i dati dei sensori più simili. Si ottengono così dei cluster dove per ognuno si cerca di capire quale attività corrisponde. Non si ha un'etichetta associata all'attività
- apprendimento semi-supervisionato: cerca di mischiare apprendimento supervisionato e non supervisionato

14.3.1 Apprendimento supervisionato

Sono algoritmi che vanno “allenati” su un insieme di dati di esempio, chiamati **training set**. Ogni esempio è un vettore X n-dimensionale associato con l'etichetta Y corrispondente. Il training set viene utilizzato dall'algoritmo di classificazione per creare un modello, chiamato **classificatore**. Il modello risultante viene usato per associare un'etichetta a vettori n-dimensionalni di cui non si conosce l'etichetta, ovvero la **classificazione**.

Ci sono due fasi:

- addestramento:

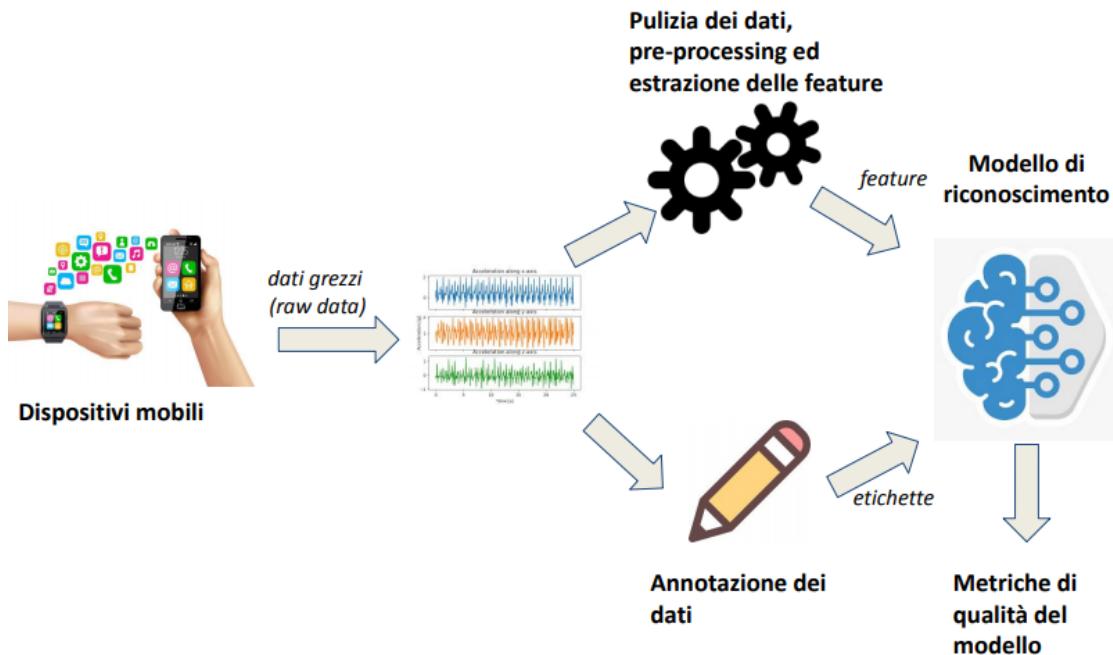


Figure 14.2: fase di addestramento

- riconoscimento: come da figura 14.1.

Nelle due fasi ci sono passaggi che sono comuni.

14.3.1.1 Training set

Per allenare un algoritmo di riconoscimento è necessario collezionare un dataset etichettato di dati di sensori. È quindi nota la corrispondenza tra dati di sensori e attività. È un'operazione molto costosa, e c'è:

- la necessità di volontari che svolgono le attività
- la necessità di un'infrastruttura complessa per la raccolta di dati etichettati

Durante la pianificazione del training set è necessario pianificare:

- quali attività considerare
- quali dispositivi mobili
- quante persone coinvolgere

- dove svolgere le acquisizioni
- decidere se il setting è realistico o guidato
- come annotare i dati
- quanti minuti di attività acquisire per ogni utente

Le due modalità principali di annotazione dei dati sono:

- self annotation: l'attività viene annotata dalla persona stessa, segnando l'inizio e la fine dell'attività. È una modalità intrusiva perché ci possono essere la possibilità di fornire annotazioni sbagliate e svogliatezza nell'annotare
- tramite video: si registra ciò che fa una persona tramite dei video e in una seconda fase vengono guardati e i dati vengono annotati. Ci sono problemi di privacy, è soggetto ad errori ed è oneroso in termini di tempo

Una volta effettuate le fasi di acquisizione e annotazione, si hanno a disposizione:

- flussi di dati diversi per ogni sensore, per ogni utente e per i relativi timestamp
- informazioni temporali relative all'inizio e alla fine delle attività svolte dagli utenti (etichette)

———— Lezione 9 - 12 aprile 2021 ———

14.4 Pre-processing

I flussi di dati devono essere trasformati in vettori n-dimensionali in modo tale da poter usare tecniche di apprendimento supervisionato per riconoscere attività.

Il pre-processing è una fase che precede la classificazione e necessita di tuning. Dopo la misurazione dei dati, abbiamo le seguenti fasi, come in figura 14.3:

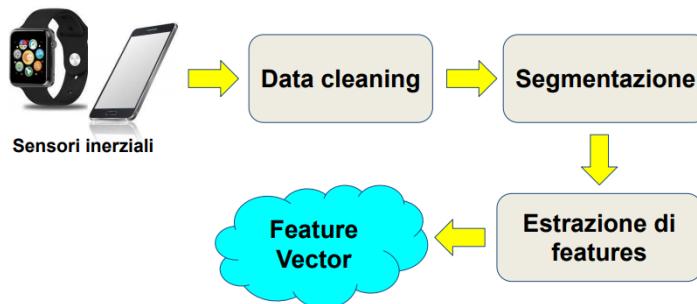
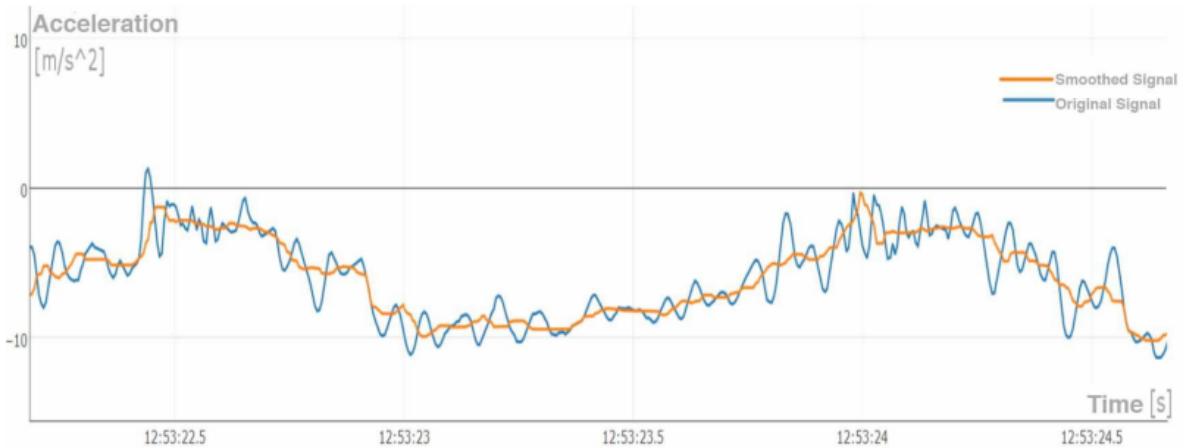


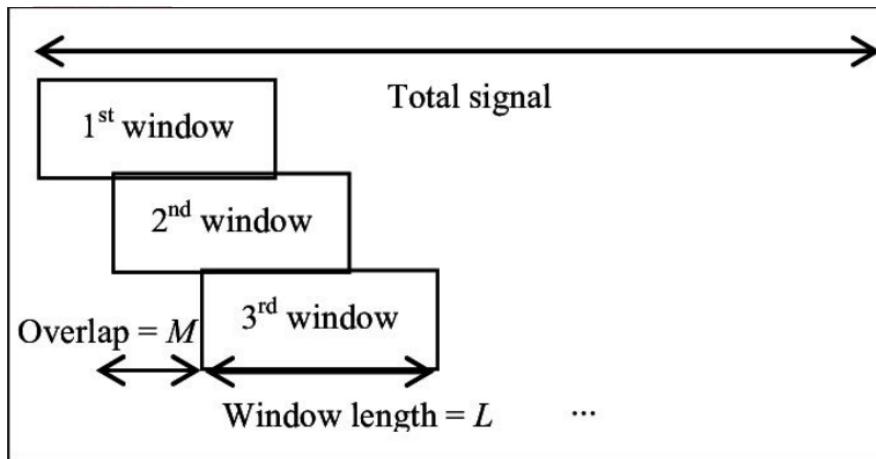
Figure 14.3: Pipeline pre-processing

- data cleaning: i dati dei sensori sono affetti da rumore e per poter rimuoverlo si utilizzano i filtri, come il filtro mediano:



La linea arancione rappresenta il valore del filtro, quella blu rappresenta i dati grezzi. Ogni valore del segnale viene sostituito con la mediana considerando i valori nell'intorno

- segmentazione: il segnale viene diviso in segmenti temporali dove ogni segmento contiene i dati di tutti i sensori presi nello spazio temporale $t_1 t_2$. La tecnica più nota è lo sliding window. Abbiamo un insieme di finestre, dove una finestra può partire a metà della finestra precedente facendo un overlap (la seconda finestra può partire a metà della prima).



Questo avviene perché andremo a classificare ognuna di queste finestre. Se le finestre sono affiancate senza overlap rischiamo che ci sia un'azione caratteristica che finisce a cavallo tra due e non riusciamo a riconoscerla.

Se consideriamo attività semplici (camminare, stare seduti), di solito la finestra è di 3 o 4 secondi con un overlap del 50%. Attività più complesse come lavare i piatti, richiedono una finestra più lunga. La dimensione della finestra può essere scelta anche in base a quanto "real-time" vuole essere il sistema

- feature extraction: dai segmenti estraiamo informazioni per creare i vettori, i **feature vector**.

14.4.1 Features

Ci sono due tipi di feature, basate:

- sul tempo: caratteristiche del segnale che riflettono il suo andamento nel tempo (es: media, varianza, ecc.)
- sulla frequenza: caratteristiche del segnale che riflettono la sua periodicità

14.4.1.1 Generazione del feature vector

Per ogni dispositivo mobile dell'utente si calcolano le features per ogni sensore inerziale al suo interno. Le features vengono concatenate per generare un vettore n-dimensionale, dove n è il numero di features calcolate. Le features sono solitamente grandi un centinaio. I range di valori delle diverse features possono variare significativamente.

14.4.1.2 Feature scaling

Per costruire un modello di riconoscimento robusto, abbiamo due tecniche:

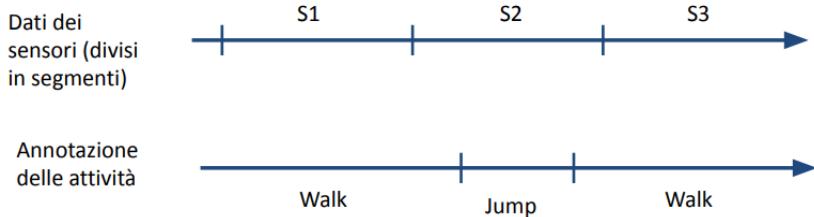
- normalizzazione: vengono normalizzati i range delle features e i valori sono scalati nel range [0,1]. È utile per rappresentare i dati usando una scala comune
- standardizzazione: cerca di riportare feature con range diversi, su una distribuzione simile. I valori sono centrati in 0. È particolarmente adatta se i dati hanno una distribuzione gaussiana

Nell'ambito dell'activity recognition, non sempre normalizzare le features migliora il riconoscimento. Va valutato empiricamente se normalizzare e come.

14.5 Riconoscimento di attività: classificazione

Abbiamo ottenuto i dati dei sensori divisi in segmenti e la notazione delle attività (etichette). La lunghezza dei segmenti dei dati non combacia con la lunghezza dei

segmenti delle attività svolte. Serve un modo per associare un'etichetta ad ogni segmento.

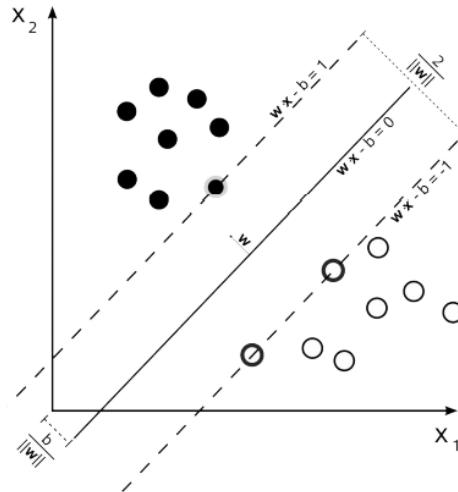


La soluzione più comune è associare l'attività prevalente del segmento. Ad esempio in S2, l'attività prevalente è Jump. In alcune situazioni, questa tecnica non funziona.

14.5.1 Algoritmi di machine learning "noti" in AR

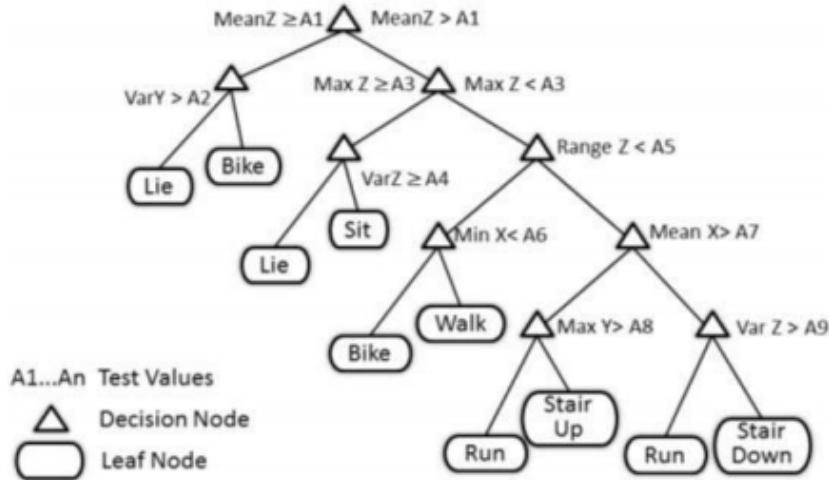
Una volta etichettati i segmenti possiamo addestrare il classificatore. Tra gli algoritmi di classificazione, i più usati sono:

- Support Vector Machine (SVM): supponiamo di avere 2 attività. Pallini neri attività 2, pallini bianchi attività 1. Lo scopo è di trovare un iperpiano che cerca di dividere al meglio gli esempi dell'attività 1 da quelli dell'attività 2. Bisogna quindi trovare la retta che distingue meglio le attività

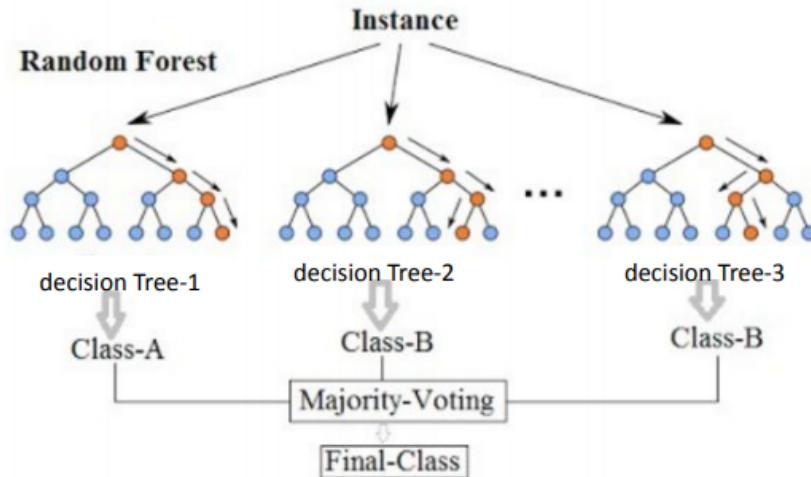


- Random Forest (basati su Decision Tree): il decision tree è un albero costruito in fase di training. Ogni nodo di questo albero è una condizione su una specifica feature (ad esempio valore dell'accelerometro < 34). Sono condizioni calcolate automaticamente in base ai dati di training. Le foglie di questo albero sono le attività. Si parte dalla radice e si passa attraverso l'albero in base alle condizioni

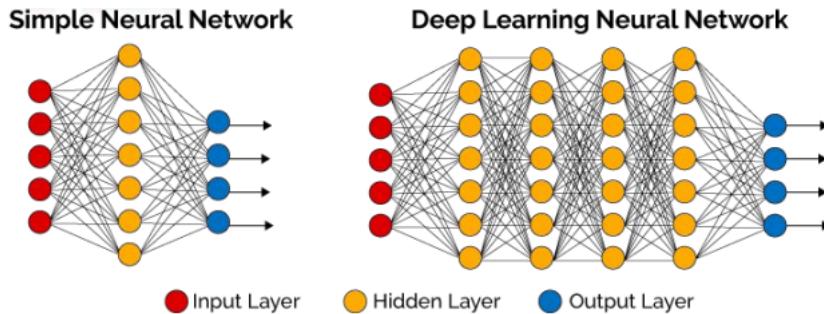
che si trovano in ogni nodo.



Il random forest è una foresta di alberi di decisione. Il feature vector attraversa tutti i decision tree. Ogni albero di decisione è formato in modo randomico ed ognuno arriva alla propria predizione di attività. Attraverso un majority-voting viene presa l'attività prevalente tra tutti i decision tree.



- Reti neurali: ci sono tecniche di deep learning che permettono di calcolare le feature in modo indipendente. Per estrarre le feature in modo affidabile, sono necessari molti dati



14.5.2 Modello di riconoscimento

Il modello di riconoscimento può essere:

- personalizzato: allenato solo con dati dell'utente che utilizza il sistema. È molto accurato, ma non è realizzabile
- generalizzato: allenato usando dati di utenti diversi da colui che usa il sistema. Il training viene fatto una volta sola, ma il riconoscimento spesso è meno accurato. Le API di Google e Apple adottano questo modello

L'ideale sarebbe avere un modello che combini i vantaggi dei due modelli: il training set viene creato inizialmente con un set di utenti diversi da colui che utilizza il sistema e poi viene chiesto all'utente di annotare una piccola porzione di dati per personalizzare il riconoscimento.

Dobbiamo capire se fare il riconoscimento:

- offline: raccoglie i dati per un certo periodo e successivamente fa il riconoscimento
- online: processa i dati per riconoscere le attività in real-time. Lato online abbiamo pro e contro:

Lato server	
pro	contro
capacità computazionali elevate	grosso overhead in termini di comunicazione di rete

Lato mobile	
pro	contro
miglior supporto al riconoscimento real-time	bisogna creare modelli di riconoscimento che possano girare efficientemente su dispositivi mobili

Nei dispositivi mobili, il modello deve essere creato offline usando un training set (esistono svariati tool di machine learning che permettono di farlo) e ci sono apposite librerie che caricano il modello e lo usano per riconoscere in tempo reale le attività. Per poter creare e deployare il modello è necessario usare la stessa SDK.

Le SDK più famose sono:

- Weka per Android
- CoreML per iOS: si può processare un modello offline ad esempio con librerie di Python, viene validato e una volta soddisfatti lo si può convertire in un formato apposito per CoreML e poi può essere deployato sull'applicazione ed utilizzato per il riconoscimento
- TensorFlow Lite per deep learning che può essere integrato sia in Android che in iOS

Un tool di Apple che fa parte di XCode e che permette di addestrare un classificatore senza dover scrivere codice è CreateML.

14.5.3 Uso del modello

Una volta che il modello è stato addestrato viene usato nel seguente modo:

- il dispositivo acquisisce lo stream di dati dai sensori
- segmenta lo stream
- per ogni segmento estrae il feature vector
- il feature vector viene dato al classificatore che restituisce l'etichetta

14.5.4 Il risultato del classificatore

Il classificatore non restituisce solo l'attività ma una distribuzione di attività. Ad esempio abbiamo due distribuzioni di probabilità:

- Caso 1: $< A, 0.5 > < B, 0.4 > < C, 0.1 >$
- Caso 2: $< A, 0.5 > < B, 0.25 > < C, 0.25 >$

In entrambi i casi l'etichetta più probabile è A, anche se nel primo caso il classificatore è più incerto (A con confidenza 0.5 e B di 0.4).

In base alla situazione si può decidere cosa fare del risultato:

- usare sempre l'etichetta con confidenza più alta

- definire qualche metrica per decidere quando prendere per buono il risultato del classificatore

———— Lezione 10 - 15 aprile 2021 ——

14.6 Validazione

Il ML è una tecnica statistica, si può sbagliare (hanno una % di errore nel riconoscimento), quindi si vuole minimizzare l'errore rendendo il classificatore il più accurato possibile.

È necessario avere un dataset di attività etichettate e poi dobbiamo dividerlo in due parti:

- training set: una parte per allenare il classificatore
- test set: una parte per testare il classificatore

Le etichette del test set servono per vedere se il modello allenato produce le stesse etichette di quelle del dataset, cioè per capire quanto il classificatore "ci ha azzeccato".

Non dobbiamo mai fare il test su dati usati per il training perché bisogna capire come funziona il modello su dati mai visti.

14.6.1 Validazione "naive"

Una delle opzioni più semplici è quella di fare uno split del dataset prendendo 80% di training e 20% di test. Si crea il modello usando i dati di training e si valuta la qualità di predizioni sui dati di test.

Il problema di questa validazione è che il risultato non può essere robusto. Se dividiamo in 80 e 20, possiamo avere uno split fortunato che ci dà il risultato migliore, oppure possiamo avere uno split sfortunato che ci dà risultati pessimi.

Ci sono diversi passaggi per irrobustire il risultato:

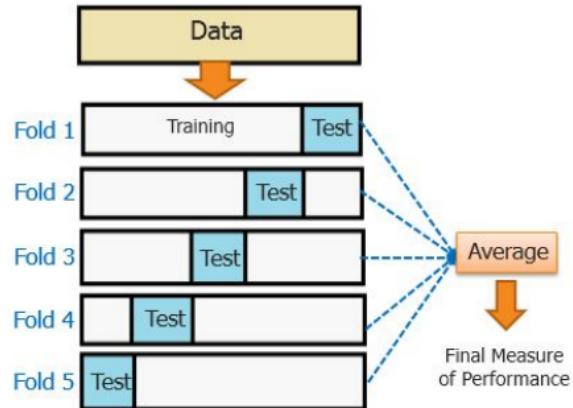
- cross-validation
- leave one subject out cross validation
- true/false positives/negatives
- matrice di confusione

14.6.2 Cross-validation

Dividiamo il dataset esattamente in k -partizioni di uguale dimensione.

Abbiamo k -iterazioni dove in ognuna iterazione (fold) prendiamo una certa parte del dataset da usare come test e una certa parte da usare come training.

La valutazione finale è la media della valutazione fatta ad ogni fold.



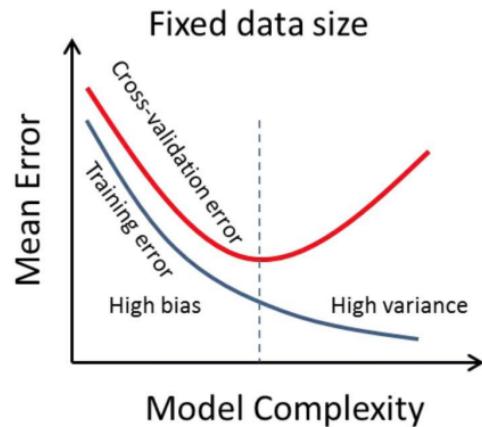
14.6.3 Leave one subject out cross validation

Simile alla cross-validation dove ad ogni fold, i dati di un utente vengono usati come test, i restanti utenti come training. Ad esempio se abbiamo 5 utenti, nel test abbiamo i dati di un utente, nel training i restanti 4. Ogni fold usa come test un utente diverso infatti, abbiamo tante iterazioni quanti gli utenti.

Il grafico ci fa vedere sull'asse x la complessità del modello, sull'asse y l'errore medio che il modello fa nella classificazione. La linea blu è il training error, un errore che fa il classificatore nel predire l'attività sugli stessi dati del training. Più il modello è semplice e più l'errore è alto (high bias). Più andiamo a rendere il modello sofisticato più l'errore scende (high variance). Cross-validation error rappresenta l'errore di predizione usando la cross-validation. Lo scopo è di arrivare in un punto in cui minimizziamo l'errore sulla cross-validation error, cioè dove abbiamo dati che non abbiamo mai visto.

Possiamo avere due fenomeni:

- **underfitting** che avviene con high bias: il modello è troppo semplice che non descrive sufficientemente i dati
- **overfitting** che avviene con high variance: il modello calza a pennello con il training set, ma che fatica a generalizzare su dati mai visti



Entrambi i problemi vanno evitati.

14.6.4 True/false positives/negatives

Per ogni singola attività abbiamo i due set, training e test, e per valutare l'accuratezza di un classificatore è necessario calcolare (per ogni attività):

- quante volte il classificatore ha calcolato l'attività A ed effettivamente l'utente ha fatto l'attività A (true positive)
- quante volte è stata predetta un'attività errata (false positive)
- quante volte un'attività non è stata predetta (false negative)

Ad esempio prendiamo un sistema binario, che vuole capire quando sto saltando, dove accende la lampadina verde se salto.

- se salto e sto saltando è true positive
- se salto e non me lo dice è un falso negativo
- se sto fermo e dice che salto è un falso positivo

Partendo da TP, FP, FN, è possibile ricavare delle metriche per valutare la qualità di predizione del classificatore. Le metriche più usate sono (hanno tutte un range tra 0 e 1, più la metrica va verso 1 più è buona, più tende allo 0 più sbaglia):

- precision: più è bassa e più sono presenti FP $Precision = TP/(TP + FP)$ Se il classificatore dice che corro sempre, la precision è bassa
- recall: più è bassa e sono presenti FN $Recall = TP/(TP + FN)$ Se il classificatore dice che corro sempre, la precision è alta
- F1: media armonica di precision e recall $F1 = 2 * (precision * recall) / (precision + recall)$

14.6.5 Matrice di confusione

È una matrice che ci fa capire quali attività possono essere confuse tra di loro. Le colonne rappresentano le attività predette dal sistema, le righe invece le volte in cui l'utente stava facendo effettivamente quell'attività. In ogni cella abbiamo le volte in cui il sistema ha predetto l'attività ed effettivamente l'utente stava facendo quell'attività. Nella diagonale principale abbiamo i TP e tutto ciò che sta fuori dalla diagonale, equivale ad un errore.

	laying	0	0	0	0	0
laying	423	0	0	0	0	0
sitting	0	365	17	0	0	0
standing	0	21	387	0	0	0
walk	0	0	0	351	17	11
walkdown	0	0	0	14	246	18
walkup	0	0	0	14	12	310
Random Forest						
	laying	sitting	standing	walk	walkdown	walkup

14.6.6 Tecniche di tuning

Un modello di riconoscimento di attività ha tantissimi iper-parametri ed il tuning di questi valori non è banale.

Ci sono tecniche di tuning per cercare di trovare i valori che massimizzano il testing:

- grid search: si provano delle combinazioni di iper-parametri, presi da elenchi di valori da testare, definiti dall'utente. È la tecnica utilizzata più spesso, ma computazionalmente pesante
- random search: sceglie in modo casuale delle combinazioni di iper-parametri, ma non dà garanzie sul miglior risultato
- bayesian optimization: tecnica di ottimizzazione sequenziale. Cerca di predire i valori di iper-parametri che massimizzi il miglioramento atteso. Costruisce un modello probabilistico per stimare i risultati attesi nello spazio degli iper-parametri. Date le combinazioni di ip e dati i risultati sulle combinazioni di ip, vado a scegliere al prossimo giro una combinazione di ip che cerca di migliorare i risultati. Si continua iterativamente a selezionare iper-parametri fino alla convergenza

14.6.7 Recap. activity recognition con supervised learning

1. Raccolta dei dati per l'addestramento del sistema
2. Pre-processing: definisco la tecnica e gli iper-parametri
3. Etichettatura dei feature vector estratti dal pre-processing con le etichette raccolte
4. Scelta dell'algoritmo di classificazione e dei relativi iper-parametri
5. Validazione: calcolo la qualità del riconoscimento del classificatore
6. Tuning: itero dal passo 2 (se necessario anche dal passo 1) per cercare di migliorare la qualità del riconoscimento
7. Deploy: addestro il classificatore su tutto il dataset e lo sposto sul dispositivo mobile

14.7 Approcci semi-supervisionati per il riconoscimento di attività

È poco pratico acquisire una grossa quantità di dati necessaria per ottenere un classificatore accurato. Nei sistemi supervisionati il modello di attività viene creato una volta sola ed abbiamo due problemi:

- ogni modello è fortemente dipendente da:
 - persone, che hanno fornito i dati e come hanno svolto le attività
 - attività considerate
- è necessario considerare un set limitato e pre-determinato di attività
- un soggetto può potenzialmente cambiare nel tempo il modo in cui svolge una specifica attività ed il modello iniziale può risultare non più accurato

14.7.1 Tecniche di apprendimento semi-supervisionato

Sono tecniche di ML che permettono di inizializzare il classificatore con un dataset più ristretto, dopodiché si cerca di etichettare in maniera automatica i dati che non sono stati etichettati.

Ci sono 3 tecniche principali:

- self-training: abbiamo dataset ridotto x creare modello iniziale. Vogliamo usare il modello iniziale per propagare le etichette verso i dati non etichettati. Abbiamo

il dato non etichettato e il classificatore che con il modello iniziale applica il riconoscimento. L'output del classificatore viene associato come etichetta al dato non etichettato solo quando è sicuro che l'attività sia quella. I dati etichettati automaticamente vengono aggiunti al modello

- co-training: più classificatori vengono allenati su un piccolo dataset su diverse viste dei dati, ad esempio diversi feature vector, diverse porzioni del dataset. Il majority voting, tra i classificatori, determina la predizione su dati non etichettati. I dati etichettati dalla maggioranza vengono usati per aggiornare il modello
- active learning: uno dei metodi più efficaci ma non è una tecnica automatica. Richiede l'interazione con l'utente. Guarda quando il classificatore è poco sicuro su un'attività. In questo caso viene chiesta l'etichetta all'utente. L'etichetta viene poi usata per aggiornare il modello

Gli approcci hanno diversi limiti:

- self-learning: gli errori nell'etichettatura automatica possono rinforzarsi col tempo
- co-training: più robusto del self-learning, ma può essere complicato creare più viste sensate dei dati
- active learning: sebbene sia una delle tecniche più potenti richiede un'interazione con l'utente per migliorare il modello

Le tecniche semi-supervised possono essere usate anche per creare un modello di attività che si personalizza sull'utente:

- il modello iniziale viene creato da un insieme di utenti diverso da quello finale
- l'utente finale scarica il modello e inizia ad usare il sistema
- con le tecniche semi-supervised, il modello viene aggiornato con dati etichettati dell'utente finale

Il training viene fatto una volta sola, perché una volta finito il modello non si aggiorna. Ogni volta che viene effettuato il training da capo, con i dati vecchi più le etichette nuove che abbiamo trovato, è time consuming e spesso impraticabile. L'ideale è usare algoritmi di apprendimento incrementale, algoritmi il cui modello di attività viene aggiornato continuamente con nuovi esempi etichettati.

Esistono diversi tipi di algoritmi incrementali:

- algoritmi nascono incrementali come k-NN, Linear Regression, Deep Learning
- ci sono estensioni di noti algoritmi in versione incrementale come SVM, Random Forest

Lezione 11 - 19 aprile 2021

14.8 Use Case: SmartWheels

Quello che si vuole creare è un sistema per facilitare la navigazione outdoor per persone con disabilità motorie. Una persona a sedie a rotelle non può sapere in anticipo gli ostacoli che troverà durante il percorso.

Le app per navigazione outdoor (come Maps o Waze) dovrebbero personalizzare il percorso per queste persone in base alle barriere architettoniche che può trovare. Attraverso queste app l'utente può settare ciò che preferisce evitare durante il percorso, tipo:

- small-step-up
- small-step-down
- medium-step-up
- medium-step-down

Il sistema calcola il percorso secondo le preferenze scelte. Ci sono sistemi simili, ma la raccolta di dati annotati (urban features UF come barriere architettoniche, rampe, ecc.) è molto complicata. Le informazioni vengono raccolte tramite:

- crowdsourcing: è difficile che gli utenti in sedia a rotelle vogliano fornire dati mentre circolano. Viene collezionata solo una frazione di dati utili al sistema
- computer vision: efficiente per UF come strisce pedonali, ma non sempre efficienti per altre UF

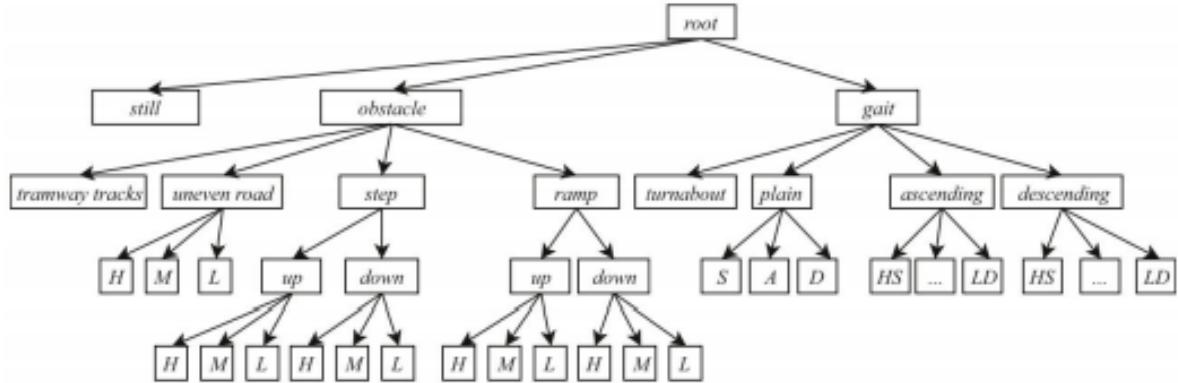
Un metodo complementare è SmartWheels, un sistema in grado di rilevare, in modo automatico, gli ostacoli (come le barriere architettoniche) utilizzando sensori inerziali installati sulla sedia a rotelle e sfruttando metodi supervisionati di activity recognition.

14.8.1 Riconoscere attività vs UF

L'idea è riconoscere le attività che rivelano la presenza di barriere architettoniche. Serve anche sapere dove si trova l'utente per sapere la posizione della barriera architettonica.

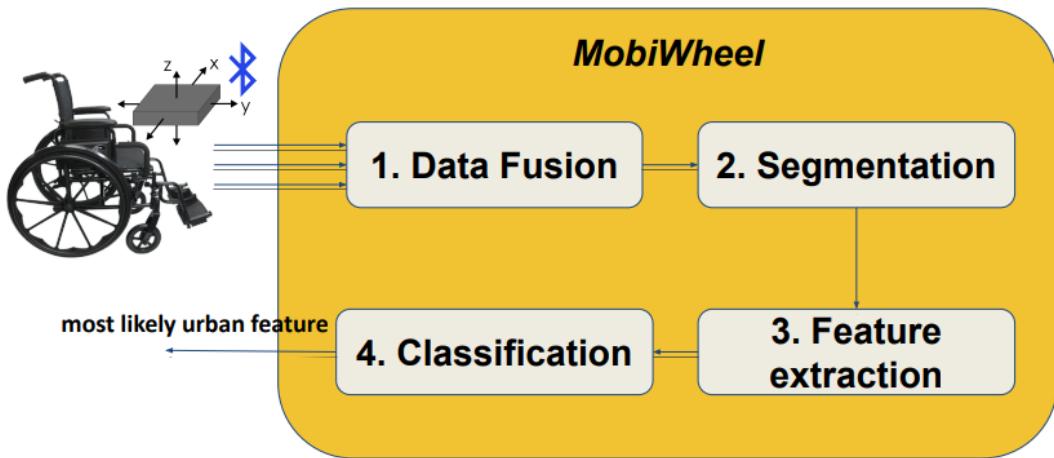
Ci sono diversi tipi di sedie a rotelle che possono essere spinte dagli utenti stessi, da un'altra persona o da propulsione elettrica, ma SmartWheel è focalizzato su utenti in grado di spingere autonomamente la propria sedia a rotelle.

In una fase iniziale si è definito il target di attività, quello che poi il classificatore dovrà riconoscere. Si è definita una gerarchia di UF dopo aver effettuato delle interviste.



La radice indica qualsiasi UF, da cui vengono identificati gli altri sotto UF.

14.8.2 Architettura del sistema



I sensori inerziali tramite bluetooth comunicano al sistema di riconoscimento. Nel data fusion vengono allineati i dati che arrivano dai diversi dispositivi in modo tale da effettuare la segmentazione (fatta tramite la tecnica di sliding window). Con le finestre temporali, estraiamo le features ovvero la rappresentazione numeri dei segmenti temporali. Forniamo al classificatore il vettore in modo tale che ci dia la UF più probabile.

14.8.2.1 Acquisizione del dataset

I dati sono stati acquisiti in un'organizzazione no-profit che ha messo a disposizione la sua area di allenamento con ostacoli urbani, come i gradini. 17 utenti in sedia a rotelle hanno seguito un percorso predefinito mentre venivano registrati, alcuni in ambiente controllato (una specie di palestra), altri in strada.

Sono stati usati 3 piccole board che comunicano ad uno smartphone tramite BLE (bluetooth low energy). Gli utenti dispongono di uno smartwatch sul polso e uno smartphone, in una fascia da braccio, sulla gamba. Poi un'app mobile colleziona e salva internamente questi dati.

La prima fase è allineare i dati, presi da ciascun dispositivo sui tre assi, temporalmente. Successivamente viene effettuata la segmentazione a sliding window ad un certo overlap.

Un problema è che alcune attività richiedono poco tempo (come scendere uno scalino), altre invece sono più lunghe (come un'andatura su piano). C'è il problema che nel segmento viene riconosciuta l'attività prevalente, in questo caso non viene riconosciuta l'attività del gradino. Una soluzione è dare priorità ad UF attraversabili in breve tempo, oppure ridurre la sliding window, ma questo avrebbe aumentato la complessità computazionale.

I classificatori principalmente investigati sono:

- Random Forest
- Hierarchical Random Forest: una versione gerarchica del Random Forest

Tramite la leave-one-subject-out cross validation sono stati determinati i migliori parametri.

Introduzione a Swift

———— Lezione 12 - 22 aprile 2021 ———

15.1 Swift

Swift è un linguaggio di programmazione realizzato da Apple. È un linguaggio moderno, recente anche rispetto ad linguaggi di programmazione come C, Java C#, Dart.

Gli obiettivi di chi crea un nuovo linguaggio di programmazione sono:

- evitare/ridurre gli errori più comuni del programmatore
- aumentare la compattezza del codice
- migliorare la leggibilità del codice
- migliorare le prestazioni

Le caratteristiche principali di Swift sono:

- multi piattaforma (iOS, watchOS, macOS, tvOS, ecc.)
- compilato
- supporta due paradigmi di programmazione:
 - Object Oriented
 - Funzionale
- si possono creare applicazioni utilizzando combinazioni di Swift, Objective-C, C e C++

15.2 Programmazione procedurale in Swift

Parole riservate:

- **var** definisce una variabile
- **let** definisce una costante

L'assegnamento viene fatto con "**=**", mentre il punto e virgola al termine di ogni istruzione è opzionale ed è preferibile ometterlo, per questioni di stile.

Swift è un linguaggio di programmazione fortemente tipizzato. Si può evitare di specificare il tipo di una variabile nel momento in cui essa viene specificata ed il tipo viene inferito automatico, ma non può essere cambiato a runtime (come invece succede in JavaScript).

```

1  var a = 5 //this is Int
2  var b:Int = 3 //this is explicitly an Int
3  var c = a+b //this is Int
4  var d = "hello!" //this is a String
5  c = d; //this gives a compile-time error

```

Come avviene in altri linguaggi si può effettuare la conversione di tipi attraverso determinate funzioni. Ad esempio per convertire un intero in stringa ci sono due modi:

```

1  var studentNumber = 50
2  let label = "Ci sono " + String(studentNumber) + " studenti"
3  //prima conversione
4  let label2 = "Ci sono \(studentNumber) studenti"
5  //seconda conversione

```

15.2.1 Tuple

Una variabile può essere associata ad una tupla di valori, es:

```

1  let httpError = (404, "Not Found") //tupla di valori
2  //in questo caso error e' di tipo (Int, String)

```

Per poter leggere i valori di una tupla si può operare nel seguente modo:

```

1  print(httpError) //prints "(404, "Not Found")"
2  //modo 1.
3  let (statusCode, statusMessage) = httpError
4  print(statusCode) //prints 404
5  print(statusMessage) //prints "Not Found"
6  //modo 2.
7  let (statusCodeAgain, _) = httpError; /*just read the
8  error number, not the type

```

```

9     note: in the line above I have used the "_" symbol
10    meaning "I don't care about the second value"*/
11    print(statusCodeAgain) //prints 404
12    //modo 3.
13    let statusCodeAgainAgain = httpError.0
14    print(statusCodeAgainAgain) //prints 404
15    let statusMessageAgain = httpError.1
16    print(statusMessageAgain) //prints "Not Found"

```

La differenza tra tupla e classe è il fatto che una tupla ha solo lo stato, mentre una classe ha sia lo stato che il comportamento. Le tuple si usano per insieme di dati da usare localmente nel codice, ad esempio valori di ritorno delle funzioni. In altri casi si usano le classi o le strutture.

15.2.2 Valori opzionali

Di default una variabile non può assumere un valore nullo. Possiamo specificare una variabile come **optional** per indicare che può assumere il valore nullo:

```
1 var i:Int? = nil
```

Le variabili opzionali sono di un tipo diverso rispetto alle controparti non-opzionali, ad esempio Int? è diverso da Int. Se una funzione prende in input Int, non posso passarle Int?, però vale il contrario.

Ho una serie di restrizioni quando ho una variabile opzionale:

- non posso usarla nelle espressioni (non posso fare a+b se una delle due variabili è opzionale)
- non posso accedere ai metodi o accedere alle proprietà

Esistono vari metodi per estrarre ("unwrap") il valore non-opzionale da una variabile opzionale o comunque per gestire i tipi opzionali. In Java se abbiamo un attributo a di una classe che è null e proviamo a leggerlo, viene dato un NullPointerException.

Ci sono diversi metodi per fare unwrapping:

- forced unwrapping: mettendo un ! dopo un tipo opzionale, si va a leggere quel tipo come se non fosse opzionale. Se si usa questa sintassi su una variabile che è nil, viene generata un'eccezione a runtime, quindi bisogna sempre controllare prima che non sia nil.

```

1     var a = 1 //non-optional
2     var b = Int("123") //optional

```

```

3      var c:Int
4      c = a+b /*this is a is a compile-time error b cannot
5      be used in an expression because it could be nil*/
6      //solution:
7      if b != nil {
8          //we use "!" to unwrap the value
9          c = a+b!
10     }
11     /*force unwrapping of a nil variable generates
12     a runtime exception*/

```

- optional binding: si usa un costrutto sintattico comunemente detto "if let".

```

1  var possibleNumber = "123"
2  var numberOptional = Int(possibleNumber)
3  /*next line: check if numberOptional is nil,
4  if not, assign it to numberNonOptional and run
5  the if condition. Otherwise run the else
6  condition.*/
7  if let numberNonOptional = numberOptional {
8      print(numberNonOptional) //prints 123
9  } else {
10     print("cannot convert")
11 }

```

La condizione dell'if è verificata se la variabile che vogliamo controllare non è nil ed in quel caso all'interno del costrutto if abbiamo un riferimento ad un'altra variabile, dello stesso tipo ma non opzionale

- implicitly unwrapped optionals:

```

1      var a: String!
2      var b: String = ""
3      b = a /*generates a runtime error: I am assigning a
4      nil value to a non-optional variable*/
5      a = "ciao"
6      /*next line: I can use b as if it is not
7      optional (note, I am not using the !)*/
8      b=a;

```

A volte è possibile avere una variabile optional che ha sempre un valore. Se definisco una variabile con "!" questa è optional, ma la posso usare come se non lo fosse

- optional chaining: con una variabile opzionale non è possibile accedere né alle proprietà né richiamare i metodi, ma è possibile richiamare metodi e proprietà con una sintassi particolare, attraverso ?, in modo tale che, se la variabile è nil, viene restituito nil, senza generare eccezioni. I valori ritornati sono sempre opzionali

```

1      var user: Person?
2      var n = "" //an empty non-optional string
3      n=user.name /*compile time error: cannot use a
4      property for optional variable user*/
5      n=user!.name //run-time exception if user is nil
6      n=user?.name /*compile time error: name is a
7      non-optional property, but when used with
8      optional chaining can return nil, so it cannot
9      be assigned to non-optional variable n*/
10     var m: String?
11     m = user?.name

```

- nil-coalescing operator: l'operatore ?? viene utilizzato per rendere più breve un'operazione comune: data una variabile opzionale a, restituire il valore di a se a non nil, altrimenti un valore b

```

1      let defaultColorName = "red"
2      var userDefinedColorName: String? // defaults to nil
3      var colorNameToUse = userDefinedColorName ?? 
4                      defaultColorName
5      /*meaning: if userDefinedColorName is not nil (which
6      is not the case in this example), then colorNameToUse
7      is userDefinedColorName. Otherwise colorNameToUse is
8      defaultColorName*/

```

La gestione delle variabili opzionali comporta una sintassi più complicata e la necessità di passare da variabili opzionali a non opzionali.

La sintassi, però, è utile per spostare un'eccezione, a livello di errore sintattico identificato, a tempo di compilazione, a differenza di Java che con un NullPointerException ci dà errore a run-time.

Alla fine il risultato è una riduzione degli errori di gestione delle variabili con valore nullo.

15.2.3 Operatori

Sono molto simili agli operatori C:

- unari (-a, !b)
- binari (a+b)
- ternari (a?b:c che significa "if a then b, else c")

In Swift il confronto tra due oggetti funziona con `==`. L'operatore `==` confronta se due variabili puntano allo stesso indirizzo di memoria. Non esistono gli operatori `++` e `--`.

15.2.4 Gestione dell'overflow

In Java l'overflow non causa l'eccezione ma un bug.

```

1 int a = Integer.MAX_VALUE - 1;
2 int b = Integer.MAX_VALUE - 1;
3 int c = a + b;
4 System.out.println(c); // -4

```

In Swift gli overflow generano un'eccezione a run-time.

```

1 var a = Int.max
2 var b = Int.max
3 let c = a + b
4 print(c) //error: Execution was interrupted

```

La gestione dell'overflow in Swift aiuta a risolvere il problema ed evita che l'errore si propaghi. Non previene un errore comune, ma ne limita gli effetti e lo rende più semplice da risolvere.

15.2.5 Operatori di range

Gli operatori permettono di creare rapidamente dei gruppi di interi o elementi contigui di un array.

```

1 for i in 1...5 {
2     print(i)
3 } //1, 2, 3, 4, 5
4 for i in 1..<5 {
5     print(i)
6 } //1, 2, 3, 4
7 let names = ["Anna", "Alex", "Brian",
8 "Jack"]
9 for i in names[1...2] {print(i)} //Alex, Brian

```

```
10    for i in names[...1] {print(i)} //Anna, Alex
11    for i in names[2...] {print(i)} //Brian, Jack
```

15.2.6 Collezioni

Sono definite tre strutture dati principali:

- array: possono essere definiti con *Array <Element>* o *[Element]*
- set: sono definiti *Set <Element>*, non possono contenere duplicati. Abbiamo a disposizione gli operatori tipici insiemistici (unione, intersezione, ecc.)
- dictionary: sono analoghi alle mappe in Java, un insieme di coppie di elementi, in cui il tipo del primo e del secondo elemento sono definiti

15.2.7 Selezione con costrutto guard

Guida il programmatore a scrivere il codice in modo più leggibile. Il costrutto guard verifica che valga una determinata condizione ed esegue del codice nel caso in cui non sia valida. Nel caso in cui non sia valida, si esce dalla funzione, si segnala un errore o si genera un'eccezione. È un costrutto if in cui si considera solamente il caso in cui la condizione non sia verificata.

```
1 func divide(numerator: Int, denominator: Int) {
2     guard denominator != 0 else {
3         print("non si puo'")
4         return
5     }
6     print(numerator/denominator)
7 }
8
9 divide(numerator:10, denominator: 2)
```

———— Lezione 13 - 26 aprile 2021 ———

15.3 Funzioni e closures

Caratteristiche delle funzioni:

- possono avere più di un valore di ritorno (come le tuple)
- i parametri possono avere dei valori di default

- le funzioni sono un tipo di dato (un parametro di una funzione ad esempio può essere una funzione)

```

1 func greet(person:String) -> String{ //definizione
2     let greeting = "Hello, " + person + "!"
3     return greeting
4 }
5
6 print(greet(person: "Sergio")) //richiamo la funzione

```

15.3.1 I parametri

In Java un parametro è caratterizzato dal nome e dal tipo.

In Swift, invece, ogni parametro ha 3 caratteristiche:

- un argument label: specifica l'etichetta da usare quando si richiama la funzione
- il nome del parametro: nome della variabile da usare nell'implementazione della funzione
- il tipo del parametro

Se non indichiamo la label abbiamo il nome della variabile (nella funzione) preceduto da `_`.

Le label hanno un valore sintattico: in Java non possiamo avere due funzioni che hanno la stessa signature (nome del metodo, l'elenco di tipi del metodo ed il valore di ritorno), in Swift invece la label fa parte della signature, quindi possiamo avere due metodi uguali con una label diversa.

I parametri possono avere un valore di default (opzionali) e per convenzione vengono messi dopo quelli necessari.

Quando all'interno di una funzione voglio cambiare il valore di un parametro, devo andare a specificare la parola riservata **inout** e bisogna passare i parametri (quando si richiama la funzione) per riferimento.

Le funzioni sono un tipo di dato, definito dal tipo dei parametri e del valore di ritorno.
La funzione sum ad esempio associa un interno ad una coppia di interi.

```

1 func sum(_ a: Int, _ b: Int) -> Int{
2     return a + b;
3 }
4
5 func dif(_ a: Int, _ b: Int) -> Int {

```

```

6         return a-b;
7     }
8
9     func f(theFunction: (Int, Int) -> Int, _ a:
10        Int, _ b: Int) -> Int {
11         return theFunction(a, b);
12     }
13
14    let a=5, b=3
15    var operation = sum
16    print(f(theFunction: operation, a, b)); //8
17    operation = dif
18    print(f(theFunction: operation, a, b)); //2

```

È possibile assegnare un nome nuovo ad un tipo come:

```
1 typealias Pippo = Int //posso usare Pippo al posto di Int
```

È molto utile quando si usano le funzioni come tipi.

15.3.2 Ordinamento di un Array

Per ordinare un Array, è necessario passare come parametro il criterio di ordinamento:

```

1 func forward(first:String, second:String) -> Bool {
2     return first < second
3 }
4 func backward(first:String, second:String) -> Bool {
5     return first > second
6 }
7
8 let names = ["Carlo", "Daniele", "Ugo", "Bruno", "Sergio",
9             "Mattia"]
10 print(names.sorted(by: forward))
11 // ["Bruno", "Carlo", "Daniele", "Mattia", "Sergio", "Ugo"]
12 print(names.sorted(by: backward))
13 // ["Ugo", "Sergio", "Mattia", "Daniele", "Carlo", "Bruno"]

```

È anche possibile definire una funzione dentro ad altre funzioni. La visibilità delle funzioni definite cambia, infatti non è possibile accedere ad una funzione definita all'interno di un'altra.

15.3.3 Closures

Sono simili ai blocchi in C e Objective-C e alle lambda expression di Java. Servono per risolvere in modo più compatto alcuni problemi tipici di questi linguaggi come la gestione degli eventi. Sono alla base della sintassi di SwiftUI. Una closure concettualmente corrisponde a definire una funzione nel momento stesso in cui mi serve.

Indichiamo una closure tra parentesi graffe, indichiamo i parametri ed il valore di ritorno, poi usiamo la parola riservata `in` e indichiamo il procedimento

```

1  let names = ["Carlo", "Daniele", "Ugo", "Bruno", "Sergio",
2    "Mattia"]
3  print(names.sorted(by:{first: String, second: String) ->
4            Bool in
5            return first < second}))
6  // ["Bruno", "Carlo", "Daniele", "Mattia", "Sergio", "Ugo"]

```

Ci sono diverse tecniche per ridurre il codice attraverso le closures:

- type inference: quando definiamo una funzione da sola, dobbiamo definire tipi e parametri, se invece una closure è passata come argomento ad una funzione, il compilatore sa già quali tipi aspettarsi, dunque possiamo evitare di indicarli.

```

1  print(names.sorted(by:{first, second in
2    return first < second
3  }))

```

- implicit return: se una closure ha una sola istruzione, si può omettere il `return`

```

1  print(names.sorted(by:{first, second in
2    first < second}))

```

- shorthand argument names: possiamo omettere di indicare i nomi dei parametri, usiamo la notazione `$0` per il primo, `$1` per il secondo parametro, ecc. In questo caso possiamo anche evitare la parola riservata "in"

```

1  print(names.sorted(by:{$0 < $1}))

```

- operation methods: se l'unica operazione da compiere è applicare un operatore ai due parametri, possiamo omettere i parametri stessi e indicare solo l'operatore. In questo caso non indichiamo neanche le parentesi graffe

```

1  print(names.sorted(by:<))

```

- trailing closure: se la closure è l'ultimo parametro di una funzione, possiamo "spostare" la closure al di fuori dalla funzione.

```
1     print(names.sorted{$0 < $1})
```

Le closure in Swift:

- permettono di essere più compatti
- semplificano la scrittura per i programmatori esperti
- complicano la lettura del codice per i programmatore alle prime armi
- introducono tante eccezioni sintattiche

Le closures (anche le funzioni innestate) possono usare riferimenti a variabili e costanti che sono definite nel contesto dove la closure è definita. Quello che non è scontato è che queste variabili e costanti rimangono valide anche quando il contesto esterno smette di esistere