Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Hlastec

Razvoj mobilne aplikacije za sledenje evro bankovcev s pomočjo storitev EuroBillTracker za platformo Android

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Damjan Vavpotič

Ljubljana, 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00157 / 2013
Datum: 6.11.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ HLASTEC**

Naslov: **RAZVOJ MOBILNE APLIKACIJE ZA SLEDENJE EVRO BANKOVCEV S POMOČJO STORITEV EUROBILLTRACKER ZA PLATFORMO ANDROID**
DEVELOPMENT OF MOBILE APLICATION FOR EURO BANKNOTES TRACKING WITH USE OF EUROBILLTRACKER SERVICES FOR ANDROID

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

V okviru diplomskega dela razvijte mobilno aplikacijo, ki bo z uporabo obstoječih storitev sistema EuroBillTracker omogočala sledenje evro bankovcev. Aplikacija naj z uporabo knjižnice Tesseract ORC in s pomočjo kamere vgrajene v mobilno napravo samodejno razpoznava serijske številke bankovcev ter jih po zaključenem razpoznavanju uporabi v navezavi s sistemom EuroBillTracker. V okviru diplomskega dela najprej predstavite uporabljene tehnologije in sistem EuroBillTracker, nato pa predstavite vašo aplikacijo in njen razvoj. Predstavite tudi ključne izzive in težave s katerimi ste se srečali pri izdelavi aplikacije.

Mentor:

doc. dr. Damjan Vavpotič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matej Hlastec, z vpisno številko **24950337**, sem avtor diplomskega dela z naslovom:

Razvoj mobilne aplikacije za sledenje evro bankovcev s pomočjo storitev EuroBillTracker za platformo Android

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Damjana Vavpotiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 20. marca 2014 Podpis avtorja:

Iskrena hvala mentorju doc. dr. Damjanu Vavpotiču, staršem in moji družini, ki so me spodbujali, mi stali ob strani in verjeli vame tako v času študija kot pri pripravi diplomske naloge.

Kazalo

Povzetek

Diplomsko delo opisuje razvoj aplikacije za mobilno platformo Android mEuroBillTracker, ki omogoča vnos podatkov o bankovcu, pregled vnešenih bankovcev ter njihov prikaz na zemljevidu. Kot osnova ji služi že obstoječa spletna storitev EuroBillTracker, ki registriranim uporabnikom med drugim omogoča tudi pregled in vnos novih bankovcev. Ker pa je ročen vnos podatkov o bankovcu zamuden, je bil cilj diplomske naloge zgraditi aplikacijo, s pomočjo katere bi pohitrili celoten proces vnosa, hkrati pa bi s samodejnim vnosom določenih polj zmanjšali možnost napak. Sicer na spletu že obstaja nekaj podobnih aplikacij, ki v določeni meri pohitrijo vnos, vendar nobena ne omogoča zajema in prepoznave serijske številke. Zato je bil glavni poudarek na vnosu serijske številke, ki smo jo zajeli s pomočjo kamere, jo s specifičnimi metodami računalniškega vida ustrezno obdelali in jo kot takšno poslali v proces optične prepoznave besedila. Ker je za uspešno prepoznavo besedila pomembno, da je vhodna slika čim ostrejša, smo implementirali sistem neprekinjenega samodejnega ostrenja kamere, kjer smo si pomagali s pospeškomerom. Z njegovo pomočjo smo zaznali trenutek, ko se je naprava ustavila in šele takrat sprožili samodejno ostrenje kamere in nadalje optično prepoznavo besedila. Naslednji uporabljen senzor je bil sistem za pozicioniranje, s pomočjo katerega smo pridobili podatke o trenutni lokaciji, na osnovi katere smo s pomočjo procesa obratnega geokodiranja pridobili podatke o kraju, poštni številki in državi trenutne lokacije.

Ključne besede: Android, EuroBillTracker, evro bankovci, pospeškomer, sistem za pozicioniranje, optična prepoznava besedila, Tesseract, OpenCV

Abstract

The thesis describes the development of an application for the mobile platform Android mEuroBillTracker that enables data entry about banknotes, review of entered banknotes and their map display. The application is based on the web application EuroBillTracker that enables registered users also review and entry of new banknotes. Since the manual entry of banknote data is time comsuming the aim of the thesis was to develop an application that would accelerate the complete data entry process. At the same the possibility of mistakes would be reduced through an automatic entry of specific fields. There are some similar applications that accelerate the data entry process to some extent, yet none of them enables the capture and recognition of the serial number. Therefore the main emphasis was on the entry of the serial number that was captured with a camera, processed with computer vision methods and finally underwent the optical character recognition process. For the optical character recognition to be successful it is very important that the entry image is as sharp as possible. Therefore we implemented a system of continuous auto focusing while using the accelerometer. With its help we were able to define the moment when the device stopped and only then we triggered further optical character recognition. The next sensor used was the global positioning system which gave us the data about the current location. On its basis and with the help of the reverse geocoding system we acquired the data about the place, post office and the country of the current location.

Key words: Android, EuroBillTracker, euro banknotes, accelerometer, global positioning system, optical character recognition, Tesseract, OpenCV

Poglavje 1

Uvod

EuroBillTracker[1] je spletna storitev, ki omogoča spremljanje in vnos evro bankovcev. Vsak, ki želi vedeti, kako njegov bankovec potuje iz njegove denarnice po svetu, se mora najprej na omenjeni spletni strani registrirati, nato pa ročno vnesti osnovne podatke o bankovcu, katerega gibanje želi spremljati. Preko posebnega obrazca se najprej vnesejo podatki o serijski številki, kodi tiska in nominaciji bankovca, nato pa se še vpišejo podatki o trenutni lokaciji bankovca. Na osnovi vnešenih podatkov nam sistem vrne podatke o tem, v kateri državi je bil bankovec natisnjen in katera država ga je izdala. Če je bil bankovec predhodno že vnešen, nam sistem to javi in nam vrne tudi podatke o tem, kje in kdaj je bil vpisan v sistem. Vsem uporabnikom, ki so isti bankovec vnesli pred nami, bo sistem samodejno preko elektronske pošte javil podatke o najdbi bankovca.

Omenjena storitev EuroBillTracker se nam zdi sicer zanimiva, ker pa ni bila razvita za delovanje izključno na pametnih telefonih, je njena uporaba nerodna, vnos podatkov o bankovcih pa počasen in okoren. Zato smo v diplomski nalogi razvili mobilno aplikacijo za pametne telefone, ki so kot nekakšni "mini računalniki" zelo razširjeni v sodobni družbi in ki sodobnega človeka spremljajo takorekoč od jutra do večera, doma, v službi, na poti ter omogočajo najširšo rabo omenjene storitve.

[1] http://www.eurobilltracker.com

Razvita aplikacija omogoča vnos podatkov o bankovcu, pregled vnešenih bankovcev in prikaz potovanja bankovca na zemljevidu. Dodatno nam aplikacija izriše grafičen prikaz števila vnešenih bankovcev glede na nominacijo bankovca ter glede na državo, ki je bankovec natisnila oz. izdala. Vnos podatkov je preko aplikacije hitrejši, saj od uporabnika zahtevamo samo še vnos kode tiska ter nominacije bankovca. Serijske številke uporabniku ni potrebno ročno vnesti, ampak je dovolj, da jo zajame s kamero. Iz zajete slike sistem s pomočjo računalniškega vida in sistema za prepoznavo znakov serijsko številko samodejno prepozna. Prav tako ni potreben ročni vnos podatkov o lokaciji bankovca, saj le-to zajamemo preko sistema za pozicioniranje.

V prvem delu diplomsko delo opisuje uporabljene tehnologije in orodja. Med drugim sta opisani odprtokodni knjižnici za optično prepoznavo besedila (Tesseract) ter računalniški vid (OpenCV). Opisana je tudi uporaba senzorja za merjenje pospeška, s pomočjo katerega smo implementirali neprekinjeno samodejno ostrenje kamere.

Drugi del je namenjen predstavitvi aplikacije mEuroBillTracker, kjer opisujemo, kako smo omenjene tehnologije dejansko implementirali. Ker se je izkazalo, da so nam določene funkcionalnosti na voljo samo na novejših napravah oz. novejših verzijah operacijskega sistema Android (npr. neprekinjeno samodejno ostrenje kamere, uporaba senzorja za pospešek, ki izolira vpliv gravitacije na napravo), smo razvili alternativne rešitve in jih vgradili v aplikacijo.

Poglavje 2

Uporabljene tehnologije in orodja

2.1 Operacijski sistem Android

Android je odprtokodni operacijski sistem, ki teče na pametnih telefonih, tabličnih računalnikih in ostalih mobilnih oz. elektronskih napravah. Temelji na Linux jedru, ki je prilagojeno napravam z na dotik občutljivim zaslonom. Za razvoj Androida je najbolj zaslužen Google, ki je ravno v ta namen ustanovil poslovno združenje več podjetij, imenovano Open Handset Alliance (OHA), ter pod svoje okrilje vzel hitro rastoče podjetje Android, Inc. Prvo komercialna verzija sistema je bila izdana septembra 2008. Trenutna zadnja verzija nosi kodno ime KitKat. Tabela 2.1 prikazuje datume objav vseh trenutno izdanih verzij operacijskega sistema Android [1].

2.2 Android SDK

Razvoj programske opreme za Android operacijski sistem poteka v programskem jeziku Java, s pomočno zbirke orodij Android SDK [2, 3], ki zajema:

- razhroščevalnik,

Verzija	Kodno ime	Datum izdaje	API level
1.0	/	23.09. 2008	1
1.1	/	09.02. 2009	2
1.5	Cupcake	30.04. 2009	3
1.6	Donut	15.09. 2009	4
2.0	Eclair	26.10. 2009	5
2.0.1	Eclair	03.12. 2009	6
2.1	Eclair	12.01. 2010	7
2.2–2.2.3	Froyo	03.12. 2009	8
2.3–2.3.2	Gingerbread	06.12. 2010	9
2.3.3–2.3.7	Gingerbread	09.02. 2011	10
3.0	Honeycomb	22.02. 2011	11
3.1	Honeycomb	10.05. 2011	12
3.2	Honeycomb	15.07. 2011	13
4.0–4.0.2	Ice Cream Sandwich	19.10. 2011	14
4.0.3–4.0.4	Ice Cream Sandwich	16.12. 2011	15
4.1	Jelly Bean	09.07. 2012	16
4.2	Jelly Bean	13.11. 2012	17
4.3	Jelly Bean	24.07. 2013	18
4.4	KitKat	31.10. 2013	19

Tabela 2.1: Zgodovina verzij Android operacijskega sistema.

- knjižnice programskih vmesnikov,
- emulator za simulacijo navideznih naprav, kjer lahko testiramo obnašanje aplikacije na različnih verzijah operacijskega sistema,
- dokumentacijo,
- vzorčne aplikacije.

Uradno podprto razvojno okolje (IDE) je Eclipse[1] z dodatkom ADT vtičnika[2]. Naštejmo nekaj orodij, ki so vgrajena v ADT vtičnik [4]:

- Traceview
  Omogoča nam pregled izvajanja aplikacije.
- Android
  Med drugim nam omogoča dostop do upravitelja orodij za razvoj (Android SDK Manager), s pomočjo katerega lahko dodajamo oz. nadgrajujemo zbirke orodij, dostop do upravitelja naprav (Android AVD Manager), kjer lahko upravljamo že obstoječe oz. kreiramo nove navidezne naprave.
- Hierarchy Viewer
  Prikaže nam morebitno neoptimalno postavitev uporabniškega vmesnika.
- Pixel Perfect
  Omogača nam natančnejše postavljanje grafičnih elementov na uporabniški vmesnik.
- DDMS
  Orodje za razhroščevanje aplikacije, ki nam omogoča vpogled v vse izvajalne niti, sklad, zajem zaslonske slike, simulacijo dohodnega klica, itd.

[1] http://www.eclipse.org
[2] http://developer.android.com/tools/sdk/eclipse-adt.html

- ADB (Android Debug Bridge)
  Omogoča nam komunikacijo z napravo preko ukazne vrstice. Izvajamo lahko različne ukaze, kot so namestitev aplikacije na ciljno napravo, odpiranje ukazne lupine (shell), itd.

- ProGuard
  Orodje za optimizacijo in zakrivanje kode. Rezultat je zmanjšana .apk datoteka, nad katero je otežen povratni inženiring.

Google ponuja tudi novo razvojno okolje imanovano Android Studio[3], ki temelji na razvojnem orodju IntelliJ IDEA[4]. Omenjeno orodje je sicer še v začetni fazi razvoja, vendar smo se ga odločili uporabiti, saj je namenjen izključno razvoju aplikacij za operacijski sistem Android.

2.3 Storitve Google Play

Storitev Google Play [6] je Google-ova aplikacija, ki omogoča enostaven dostop do Google-ovih storitev in je tesno integrirana v Android operacijski sistem. Gre za nabor knjižnic, ki nam na hiter in enostaven način pomagajo pri implementaciji določenih funkcionalnosti, kot so pridobivanje lokacije, Google zemljevid, Google+, Google Drive, itd. Gre za močno orodje, ki preko Google Play trgovine prinaša nove funkcionalnosti tudi uporabnikom starejših verzij operacijskega sistema.

V diplomski nalogi smo uporabili storitev za pridobivanje lokacije (Location API) ter storitev za prikazovanje lokacije na zemljevidu (Google Maps Android API).

2.3.1 Location API

Večina naprav, ki temeljijo na operacijskem sistemu Android, nam omogoča določitev trenutne lokacije s pomočjo sistema za pozicioniranje (GPS), brezžičnega

[3]http://developer.android.com/sdk/installing/studio.html
[4]http://www.jetbrains.com/idea/index.html

(WIFI) ali mobilnega omrežja. Prednost uporabe določitve lokacije s pomočjo sistema za pozicioniranje je v natančnejši določitvi lokacije, vendar se poveča poraba električne energije.
Do trenutne lokacije lahko dostopamo preko dveh API-jev:

- Location Manager,
- Location Client.

Uporaba zgoraj omenjenih storitev zahteva, da mora biti omogočena vsaj ena izmed spodaj naštetih dovoljenj:

- android.permission.ACCESS_COARSE_LOCATION
  Za določitev lokacije uporablja WIFI/mobilno omrežje.
- android.permission.ACCESS_FINE_LOCATION
  Za določitev lokacije uporablja GPS modul.

2.3.1.1 Location Manager API

Knjižnica za uporabo Location Manager API-ja [7] (primer 2.1) je vgrajena že v sam operacijski sistem (paket android.location) in jo je možno začeti uporabljati brez dodatno nameščenih komponent. Omogoča nam dostop do sistemskih lokacijskih storitev. Uporabimo jo tako, da navedemo ponudnika storitev (GPS - LocationManager.GPS_PROVIDER oz. WIFI/mobilno omrežje - LocationManager.NETWORK_PROVIDER) in frekvenco osveževanja lokacije. Po uspešni vzpostavitvi povezave začenjamo dobivati informacijo o lokaciji.

Primer 2.1: Pridobivanje lokacije s pomočjo LocationManager API-ja.

```
LocationManager locationManager = (LocationManager) this.getSystemService(
    Context.LOCATION_SERVICE);
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
      // Klic ob spremembi lokacije.
      ...
    }
```

```
    public void onStatusChanged(String provider, int status, Bundle extras)
        {}
    public void onProviderEnabled(String provider) {}
    public void onProviderDisabled(String provider) {}
  };
// Prijava na storitev obvescanja o spremembi lokacije.
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0,
    0, locationListener);
...
// Pridobivanje lokacije
Location currentLocation = locationManager.getLastKnownLocation(
    locationProvider);
...
//Na koncu se moramo od storitve obvescanja o spremembi lokacije odjaviti.
locationManager.removeUpdates(locationListener);
```

2.3.1.2 Location Client API

Naslednja možnost je uporaba Location Client API-ja [8] (primer 2.2), ki je del Google Play SDK. API je bil predstavljen na konferenci Google IO 2013. V primerjavi s prej opisano metodo določevanja trenutne lokacije je enostavnejši za uporabo, natančnejši, učinkovitejši in za opravljeno delo porabi manj električne energije. Vsebuje tudi funkcionalnosti obveščanja uporabnika, ena izmed njih je ta, da uporabnika, kadar se je ta približal določeni lokaciji (Geofencing API), o tem tudi obvesti.

Primer 2.2: Pridobivanje lokacije s pomočjo LocationClient API-ja.

```
LocationClient locationClient = new LocationClient(this, this, this);
LocationRequest locationRequest = new LocationRequest();
LocationListener locationListener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        // Klic ob spremembi lokacije.
        ...
    }
};
locationClient.requestLocationUpdates(locationRequest, locationListener);
locationClient.connect();
...
// Pridobivanje lokacije
Location currentLocation = locationClient.getLastLocation();
```

```
...
//Na koncu se moramo od storitve obvescanja o spremembi lokacije odjaviti.
locationClient.disconnect();
```

2.3.2 Google Maps Android API v2

Google Maps Android API v2 [9] nam omogoča enostaven način dodajanja zemljevida v aplikacijo. API sam poskrbi za nalaganje novih zemljevidov in drugih podatkov. Z minimalnim naporom lahko nastavimo različne atribute zemljevida, kot so različni pogledi (običajen, satelitski, hibridni), dodajanje poligonov, oznak, animiranih prehodov med različnimi lokacijami, itd. Preden začnemo uporabljati storitev, moramo preko Google API konzole pridobiti API ključ in prijaviti aplikacijo v morebitno že prijavljene aplikacije, katerim dovolimo uporabo storitve. Prijava zahteva vnos razpoznavnega odtisa ter imena paketa aplikacije. Pridobljen ključ tako dodamo v aplikacijo.

Za delovanje storitve morajo biti omogočena naslednja dovoljenja:

- android.permission.INTERNET
  Nalaganje delčkov zemljevida iz Google Maps strežnikov.

- android.permission.ACCESS_NETWORK_STATE
  Preverjanje internetne povezave, ki je potrebna za nalaganje delčkov zemljevida iz Google Maps strežnikov.

- android.permission.WRITE_EXTERNAL_STORAGE
  Predpomnjenje podatkov že prej naloženih delčkov zemljevida.

2.4 Geokodiranje

Geokodiranje je proces, ki nam na osnovi naslova lokacije (npr. Kasaze 107b, Petrovče, Slovenija) vrne zemljepisno širino in dolžino (46.223763, 15.183557).

Obratno geokodiranje je nasprotje zgoraj opisanega procesa, pri katerem iz znane zemljepisne širine in dolžine dobimo naslov lokacije.

Ena izmed zahtev pri razvoju aplikacije je bila ravno ta, da na osnovi trenutne lokacije, ki je podana z zemljepisno širino in dolžino, pridobimo naslov (kraj, poštno številko in državo) le-te. Za to imamo na voljo dve možnosti:

- uporaba storitve Geocoder;
- uporaba storitve Google Geocoding.

2.4.1 Geocoder

Gre za storitev [10, 15], ki je podprta samo na določenih napravah. Z njeno pomočjo lahko na enostaven način pridobimo naslov določene lokacije.
Ali naprava podpira omenjeno storitev, izvemo na naslednji način:

```
if (Geocoder.isPresent()) {
    ....
}
```

Nadalje lahko na enostaven način pridobimo podatke o naslovu lokacije z določeno zemljepisno širino in dolžino.

```
Geocoder geocoder = new Geocoder(this, Locale.getDefault());
List<Address> addressList = geocoder.getFromLocation(46.22376, 15.18355, 1);
```

2.4.2 Google Geocoding API

Pri napravah, ki ne podpirajo storitve Geocoder, smo se morali znajti na drug način. Rešitev je bila uporaba Google Geocoding storitve [11]. Storitev je dostopna preko zahtevka HTTPS, v katerem navedemo iskano zemljepisno širino in dolžino (primer 3.3), kot rezultat pa dobimo človeško berljiv naslov iskane lokacije. Rezultat je lahko v obliki XML ali JSON (primer 2.4).

Primer 2.3: Zahtevek za pridobivanje lokacije s pomočjo Google Geocoding API-ja.

```
https://maps.googleapis.com/maps/api/geocode/json?latlng
    =46.223763,15.183557&sensor=true
```

Primer 2.4: Odgovor Google Geocoding API-ja.

```
{
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "107b",
               "short_name" : "107b",
               "types" : [ "street_number" ]
            },
            {
               "long_name" : "Kasaze",
               "short_name" : "Kasaze",
               "types" : [ "route" ]
            },
            {
               "long_name" : "Kasaze",
               "short_name" : "Kasaze",
               "types" : [ "locality", "political" ]
            },
            {
               "long_name" : "Slovenia",
               "short_name" : "SI",
               "types" : [ "country", "political" ]
            },
            {
               "long_name" : "3301",
               "short_name" : "3301",
               "types" : [ "postal_code" ]
            },
            {
               "long_name" : "Petrovče",
               "short_name" : "Petrovče",
               "types" : [ "postal_town" ]
            }
         ],
         "formatted_address" : "Kasaze 107b, 3301 Petrovče, Slovenia",...
```

2.5 Senzorji

Večina mobilnih naprav, ki temeljijo na Android operacijskem sistemu, ima vgrajene naslednje tipe senzorjev [12]:

- Senzorji gibanja (pospeškomer, senzor za zaznavo gravitacije, žiroskop). Merijo rotacijske sile in sile pospeška po vseh treh oseh koordinatnega sistema.
  Tabela 2.2 prikazuje pregled senzorjev gibanja, ki so na voljo na Android platformi.

- Senzorji za zaznavo okolice (barometer, fotometer, termometer).
  Merijo različne parametre okolice npr. zračni tlak, temperaturo, osvetlitev, vlažnost.
  Tabela 2.3 prikazuje pregled senzorjev za zaznavo okolice, ki so na voljo na Android platformi.

- Senzorji za določitev položaja (magnetometer, kompas, sistem za pozicioniranje).
  Merijo fizično postavitev naprave v prostoru.
  Tabela 2.4 prikazuje pregled senzorjev za določitev položaja, ki so na voljo na Android platformi.

2.5.1 Android Sensor API

Android operacijski sistem vsebuje programsko ogrodje, ki nam omogoča komunikacijo s senzorji naprave. Izvajamo lahko različne operacije, kot so:

- pridobivanje podatkov o tem, kateri senzorji so v mobilni napravi na voljo,

- pridobivanje podatkov o zmožnostih senzorja (maksimalen obseg delovanja, poraba energije, ime proizvajalca, maksimalna ločljivost, itd.),

Senzor	Kaj meri	Enota
TYPE_ACCELEROMETER	Silo pospeška	$^{m}/_{s^2}$
	(vključuje silo gravitacije)	
TYPE_GRAVITY	Silo gravitacije	$^{m}/_{s^2}$
TYPE_GYROSCOPE	Hitrost rotacije	$^{rad}/_{s}$
TYPE_LINEAR_ACCELERATION	Silo pospeška	$^{m}/_{s^2}$
	(ne vključuje sile gravitacije)	
TYPE_ROTATION_VECTOR	Rotacijo	/

Tabela 2.2: Podprti senzorji gibanja na Android platformi.

Senzor	Kaj meri	Enota
TYPE_AMBIENT_TEMPERATURE	Temperaturo okolice	°C
TYPE_LIGHT	Osvetljenost	lx
TYPE_PRESSURE	Zračni tlak	hPa ali mbar
TYPE_RELATIVE_HUMIDITY	Relativno vlažnost	%
TYPE_TEMPERATURE	Temperature naprave	°C

Tabela 2.3: Podprti senzorji za zaznavo okolice na Android platformi.

Senzor	Kaj meri	Enota
TYPE_MAGNETIC_FIELD	Moč magnetnega polja	μ
TYPE_ORIENTATION (opuščeno)	Nagib, naklon, odklon	°
TYPE_PROXIMITY	Oddaljenost od objekta	cm

Tabela 2.4: Podprti senzorji za določitev položaja na Android platformi.

Senzor	4.0	2.3	2.2	1.5
TYPE_ACCELEROMETER	✔	✔	✔	✔
TYPE_AMBIENT_TEMPERATURE	✔	✗	✗	✗
TYPE_GRAVITY	✔	✔	✗	✗
TYPE_GYROSCOPE	✔	✔	✗	✗
TYPE_LIGHT	✔	✔	✔	✔
TYPE_LINEAR_ACCELERATION	✔	✔	✗	✗
TYPE_MAGNETIC_FIELD	✔	✔	✔	✔
TYPE_ORIENTATION	✔	✔	✔	✔
TYPE_PRESSURE	✔	✔	✗	✗
TYPE_PROXIMITY	✔	✔	✔	✔
TYPE_RELATIVE_HUMIDITY	✔	✗	✗	✗
TYPE_ROTATION_VECTOR	✔	✔	✗	✗
TYPE_TEMPERATURE	✔	✔	✔	✔

Tabela 2.5: Razpoložljivost senzorjev glede na verzijo operecijskega sistema.

- pridobivanje surovih podatkov in definiranje frekvence zajema podatkov,
- prijava in odjava na obveščanje o spremembi stanja senzorja.

Dejanski dostop do senzorja pridobimo preko programskega vmesnika SensorManager, kjer navedemo tip senzorja (tabela 2.2, 2.3, 2.4), do katerega želimo dostopati, ter frekvenco zajema podatkov (primer 2.5).

Ali naprava podpira določen senzor (npr. pospeškomer), izvemo na naslednji način:

```
public boolean isAccelerometerSupported() {
    SensorManager sensorManager = (SensorManager) getSystemService(Context.
        SENSOR_SERVICE);
    List<Sensor> sensors = sensorManager.getSensorList(Sensor.
        TYPE_ACCELEROMETER);
    return sensors.size() > 0;
}
```

Primer 2.5: Odčitavanje podatkov pospeškomera s pomočjo Android Sensor API-ja.

```
SensorManager sensorManager = (SensorManager) getSystemService(Context.
    SENSOR_SERVICE);
Sensor accelerometerSensor = sensorManager.getDefaultSensor(Sensor.
    TYPE_ACCELEROMETER);
SensorEventListener sensorEventListener = new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            // Pospesek v smeri osi X.
            float accX = event.values[0];
            // Pospesek v smeri osi Y.
            float accY = event.values[1];
            // Pospesek v smeri osi Z.
            float accZ = event.values[2];
            ...
        }
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }
```

```
};
// Prijava na obvescanje o spremembi stanja senzorja.
sensorManager.registerListener(sensorEventListener, accelerometerSensor,
    SensorManager.SENSOR_DELAY_NORMAL);
...
// Odjava od obvescanja o spremembi stanja senzorja.
sensorManager.unregisterListener(sensorEventListener);
```

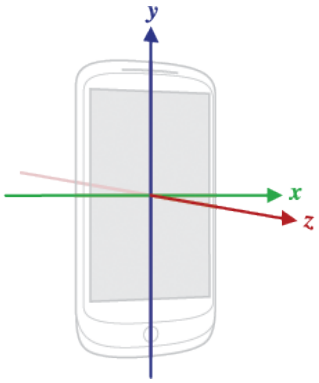
V diplomski nalogi smo izmed vseh naštetih senzorjev uporabili pospeškomer. Uporabljen je bil pri zajemu slike serijske številke bankovca. Z njegovo pomočjo smo zaznali trenutek, ko se je mobilna naprava umirila in šele potem sprožili samodejno ostrenje kamere. Tako zajeta slika je bila izostrena, kar pa je predpogoj za naslednji korak prepoznave serijske številke bankovca. S tem postopkom smo izboljšali uporabniško izkušnjo na cenejših napravah, ki ne podpirajo t. i. neprekinjenega samodejnega ostrenja.

2.5.2 Pospeškomer - TYPE_ACCELEROMETER

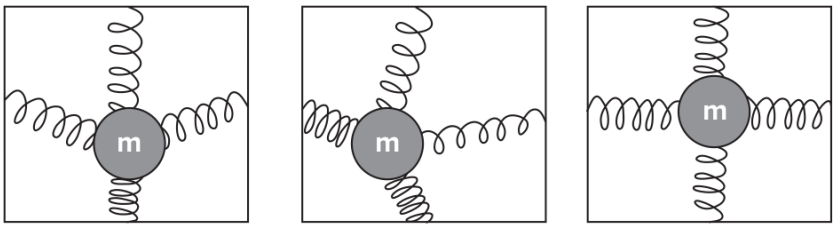
Senzor meri silo (vključno z silo gravitacije) in smer pospeška po vseh treh oseh koordinatnega sistema, ki deluje na napravo. Kadar napravo držimo v privzetem položaju (slika 2.1) in le-ta miruje, senzor zaznava gravitacijski pospešek $9.81 \mathrm{m/s^2}$ v smeri navpično od tal navzgor (koordinata Y), pospešek na preostalih dveh koordinatah pa meri $0 \mathrm{m/s^2}$. Podobno v trenutku, ko napravo izpustimo iz rok (naprava je v prostem padu), senzor po vseh treh oseh koordinatnega sistema zaznava pospešek $0 \mathrm{m/s^2}$. Opisano lahko predstavimo na primeru vzmeti, na katero je pripeta utež [15] (slika 2.2).

Ker komponenta sile gravitacije vpliva na meritve dejanskega pospeška, jo je bilo potrebno iz rezultata meritev nekako izločiti. Pri napravah, ki imajo verzijo operacijskega sistema višjo od 2.2, lahko uporabimo poseben tip pospeškomera imenovan TYPE_LINEAR_ACCELERATION, pri katerem je iz rezultata meritev že izločena komponenta sile gravitacije.

Pri nižjih verzijah operacijskega sistema smo si pomagali z metodo filtriranja, ki jo prikazuje primer 2.6 [13, 15]:



Slika 2.1: Privzeti položaj naprave in pripadajoč koordinatni sistem, ki je uporabljen pri senzorjih gibanja na Android napravah.



Slika 2.2: Sile, ki delujejo na utež, pripeto na vzmeti (od leve proti desni: naprava počiva na mizi, naprava potisnjena iz leve strani, trenutek ko napravo izpustimo iz rok).

Primer 2.6: Odstranjevanje komponente sile gravitacije iz meritev pospeškomera.

```
public void onSensorChanged(SensorEvent event){
    final float alpha = 0.8;

    gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
    gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
    gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

    linear_acceleration[0] = event.values[0] - gravity[0];
    linear_acceleration[1] = event.values[1] - gravity[1];
    linear_acceleration[2] = event.values[2] - gravity[2];
}
```

2.6 OpenCV

OpenCV [16] (Open Source Computer Vision) je odprtokodna knjižnica, ki ima implementiranih več sto algoritmov s področja računalniškega vida. Algoritmi so računsko optimizirani in prilagojeni za izvajanje v realnem času. Knjižnica je napisana v C/C++ programskem jeziku in je na voljo za različne operacijske sisteme (Windows, Linux, Mac OS, iOS, Android). Vsebuje vmesnike za programske jezike C, C++, Python in Java. Podpira tudi uporabo večjedernih procesorjev ter Intelovo knjžnico Intel IPP[5] (Integrated Performance Primitives), ki še dodatno pospeši izvajanje nekaterij funkcij.
Knjižnico sestavljajo naslednji moduli [17]:

- core (osnovne funkcije in podatkovne strukture, ki jih uporabljajo ostali moduli),
- imgproc (modul za obdelavo slike),
- video (modul za obdelavo videa),
- calib3d (umerjanje kamere in 3D rekonstrukcija),
- features2d (modul za prepoznavanje značilnih potez objektov),

[5] http://software.intel.com/en-us/intel-ipp

- objdetect (modul za zaznavanje objektov),
- highgui (vmesnik za zajem videa, gradniki za izdelavo enostavnega uporabniškega vmesnika),
- gpu (GPU pospešeni algoritmi).

V diplomski nalogi je bil glavni cilj zajem serijske številke bankovca in prepoznava le-te. Sistem za prepoznavo besedila sicer že sam poskrbi za ustrezno predprocesiranje slike, vendar se je izkazalo, da dobimo boljše rezultate, če prej vhodno sliko sami pretvorimo v binarno (črno-belo) sliko. To smo dosegli z metodo upragovanja, ki nam jo nudi knjižnica OpenCV.

2.6.1 Upragovanje

Upragovanje je proces konverzije vhodne (sivinske) slike v binarno (črno-belo) sliko, kjer vsakemu slikovnemu elementu, ki je pod določenim pragom, priredimo npr. vrednost 0 in vsakemu slikovnemu elementu, ki je nad določenim pragom vrednost 1.

Če je $g(x, y)$ pražna vrednost slikovnega elementa $f(x, y)$ pri določenem pragu T, potem velja

$$g(x,y) = \begin{cases} 1 & \text{če } f(x,y) \geq T \\ 0 & \text{sicer} \end{cases} \tag{2.1}$$

Prag T lahko definiramo kot

$$T = T[x, y, (p(x, y), f(x, y)] \tag{2.2}$$

kjer je $f(x, y)$ vrednost slikovnega elementa v točki (x, y), $p(x, y)$ predstavlja lastnost točke (npr. povprečna vrednost sosednjih slikovnih elementov s središčem v točki (x, y)).

2.6.1.1 Globalno upragovanje

Kadar je prag T (definicija 2.2) odvisen samo od $f(x, y)$, potem govorimo o globalnem upragovanju. V praksi je uporaben samo v kontroliranem okolju, kjer imamo zagotovljene pogoje enakomerne osvetlitve slike.

Metoda globalnega upragovanja, ki je implementirana v knjižnici OpenCV, se imenuje *threshold* [18, 20]. Definirana je kot:

```
Imgproc.threshold(Mat src,
                  Mat dst,
                  double thresh,
                  double maxval,
                  int type);
```

kjer je parameter *src* vhodna slika, *dst* izhodna slika, *thresh* prag, *maxval* vrednost, ki jo priredimo slikovnemu elementu, ki je nad določenim pragom, in *type* tip upragovanja.
Za parameter *type* imamo na voljo naslednje vrednosti:

- THRESH_BINARY

$$dst(x,y) = \begin{cases} maxval & \text{če } src(x,y) > thresh \\ 0 & \text{sicer} \end{cases}$$

- THRESH_BINARY_INV

$$dst(x,y) = \begin{cases} 0 & \text{če } src(x,y) > thresh \\ maxval & \text{sicer} \end{cases}$$

- THRESH_TRUNC

$$dst(x,y) = \begin{cases} thresh & \text{če } src(x,y) > thresh \\ src(x,y) & \text{sicer} \end{cases}$$

- THRESH_TOZERO

$$dst(x,y) = \begin{cases} src(x,y) & \text{če } src(x,y) > thresh \\ 0 & \text{sicer} \end{cases}$$

- THRESH_TOZERO_INV

$$dst(x,y) = \begin{cases} 0 & \text{če } src(x,y) > thresh \\ src(x,y) & \text{sicer} \end{cases}$$

2.6.1.2 Lokalno upragovanje

Kadar je prag T (definicija 2.2) poleg $f(x, y)$ odvisen tudi od $p(x, y)$, govorimo o lokalnem upragovanju. Ta tip upragovanja smo uporabili v diplomski nalogi, saj se je globalno upragovanje (poglavje 2.6.1.1) zaradi pogojev neenakomerne osvetlitve slike izkazalo kot neprimerno.

Metoda lokalnega upragovanja, ki je implementirana v knjižnici OpenCV, se imenuje *adaptiveThreshold* [19, 20]. Definirana je kot:

```
Imgproc.adaptiveThreshold(Mat src,
                          Mat dst,
                          double maxValue,
                          int adaptiveMethod,
                          int thresholdType,
                          int blockSize,
                          double C);
```

kjer je parameter *src* vhodna slika, *dst* izhodna slika, $maxValue$ vrednost, ki jo priredimo slikovnemu elementu, ki je nad določenim pragom, $adaptiveMethod$ algoritem za upragovanje, $thresholdType$ tip upragovanja, $blockSize$ velikost bloka, ki zajema slikovne elemente, s pomočjo katerih izračunamo prag za določen slikovni element in C konstantna vrednost, ki jo odštejemo od povprečja slikovnih elementov v trenutnem bloku.
Za parameter $thresholdType$ imamo na voljo naslednje vrednosti:

- THRESH_BINARY

$$dst(x, y) = \begin{cases} maxValue & \text{če } src(x, y) > T(x, y) \\ 0 & \text{sicer} \end{cases}$$

- THRESH_BINARY_INV

$$dst(x, y) = \begin{cases} 0 & \text{če } src(x, y) > T(x, y) \\ maxValue & \text{sicer} \end{cases}$$

kjer je $T(x, y)$ prag, ki je izračunan za vsak slikovni element glede na tip algoritma za upragovanje $adaptiveMethod$:

- ADAPTIVE_THRESH_MEAN_C

  Prag $T(x, y)$ je definiran kot povprečje slikovnih elementov velikosti

*blockSize*blockSize*, ki se nahajajo okoli slikovnega elementa (x, y). Od dobljenega povprečja odštejemo konstantno vrednost C.

- ADAPTIVE_THRESH_GAUSSIAN_C
  Prag $T(x, y)$ je definiran kot utežena vsota slikovnih elementov velikosti *blockSize*blockSize*, ki se nahajajo okoli slikovnega elementa (x, y). Od dobljenega povprečja odštejemo konstantno vrednost C.

2.6.2 OpenCV4Android SDK

Paket OpenCV4Android SDK [21] je zbirka orodij, ki nam omogoča uporabo OpenCV knjižnice v aplikacijah, ki temeljijo na Android operacijskem sistemu.
Knjižnico lahko v projekt vključimo na dva načina:

- vključitev celotnega paketa,
- vključitev paketa preko modula OpenCV Manager.

Uporaba prve možnosti ima mnogo slabosti, kot so: v vsako aplikacijo, ki želi uporabljati OpenCV knjižnico, moramo vključiti celotno kopijo paketa OpenCV; aplikacija mora vsebovati prevedeno kodo za vse platforme (ARM, ARMv7, x86), čeprav bo na določeni napravi koristila samo prevedeno kodo za lastno platformo; ob izidu nove verzije paketa OpenCV je potrebno posodobiti vse odjemalčeve aplikacije, ki uporabljajo paket OpenCV.
Omenjene probleme rešuje uporaba načina vključitve paketa OpenCV preko modula OpenCV Manager, ki je tudi bila uporabljena v diplomski nalogi.

2.6.2.1 OpenCV Manager

OpenCV Manager [22] je samostojna aplikacija, ki omogoča souporabo knjižnice OpenCV v več aplikacijah nameščenih na isti napravi. Uporaba omenjene storitve prinaša mnoge prednosti, kot so: zmanjšana poraba pomnilnika; vse aplikacije koristijo isto prevedeno kodo za lastno platformo; optimizacije za

specifično platformo; redno posodabljanje knjižnice preko Google Play trgovine.

Primer 2.7: Implementacija asinhrone inicializacije knjižnice OpenCV.

```
...
private BaseLoaderCallback mOpenCVCallBack = new BaseLoaderCallback(this) {
    @Override
    public void onManagerConnected(int status) {
        switch (status) {
            case LoaderCallbackInterface.SUCCESS: {
                // OpenCV knjiznica uspesno nalozena.
            } break;
            default: {
                super.onManagerConnected(status);
            } break;
        }
    }
};

@Override
protected void onResume() {
    super.onResume();
    if (!OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_4, this,
        mOpenCVCallBack)) {
        // Napaka pri inicializaciji OpenCV knjiznice.
    }
}
...
```

2.7 Tesseract OCR

Optično prepoznavanje besedila (OCR) je proces mehanske ali elektronske konverzije slik, ki vsebujejo ročno napisano ali tiskano besedilo, v strojno kodirano, računalniku berljivo besedilo. Uporabljen je na številnih področjih, kot npr. digitalizacija tiskanega besedila, samodejna prepoznava registerskih tablic, zajem besedila pri raznih obrazcih, itd. Začetne verzije so bile sprogramirane tako, da so za vsak znak vsebovale pripadajočo sliko, naenkrat je bila dovoljena uporaba samo ene pisave, medtem ko so naprednejše verzije pri kombiniranju različnih pisav in prepoznavi besedila veliko bolj uspešne. Za optično prepoznavanje besedila obstaja kar nekaj programov, kot so:

Abby, Tesseract, OCRopus, Ocrad, GOCR... V diplomski nalogi smo se odločili za uporabo programa Tesseract.

Tesseract [24] velja za enega najnatančnejših odprtokodnih sistemov za prepoznavanje besedila. Razvit je bil v Hewlett Packard labaratorijih med leti 1985 in 1994, po dveh desetletjih izdan kot odprtokodni sistem, katerega nadaljni razvoj je leta 2006 prevzel Google. Leta 1995 je bil v testu natančnosti UNLV izbran kot eden izmed treh najboljših sistemov. Podpira najrazličnejše formate slik in jih zna pretvoriti v več kot 60 jezikov. Na voljo je za različne operacijske sisteme: Linux, Mac OS, Windows, Android, iOS.

2.7.1 TESS-TWO

V diplomski nalogi smo za optično prepoznavo besedila uporabili odprtokodno knjižnico TESS-TWO[6], ki temelji na odprtokodnem projektu "Tesseract Tools for Android" z dodatkom nekaterih funkcionalnosti. Vsebuje programski vmesnik za dostop do izvornih prevedenih knjižnic v programskem jeziku C oz. C++. Knjižnico lahko uporabljamo na mobilnih napravah z operacijskim sistemom Android od verzije 2.2 naprej. Ena izmed zahtev je tudi, da so podatki o naučenih jezikih shranjeni na pomnilniški kartici mobilne naprave v direktoriju "tessdata".

2.7.2 Učenje novih jezikov

Tesseract je bil prvotno namenjen prepoznavi angleškega besedila, vendar so ga sčasoma izboljšali in ga v trenutni verziji (3.0.2) s pomočjo učenja usposobili za prepoznavo standardnih pisav za okoli 60 jezikov[7]. V diplomski nalogi smo za razpoznavo serijske številke bankovca sistem morali naučiti pisave OCR-B.
Proces učenja je bil razdeljen na več korakov [25]:

[6] https://github.com/rmtheis/tess-two
[7] https://code.google.com/p/tesseract-ocr/downloads/list

- Priprava slik z besedilom za prepoznavo.
  Na osnovi celotnega nabora dovoljenih znakov (v našem primeru velike črke abecede in cifre od 0 do 9) zgeneriramo različne primere besed, pri tem pa poskrbimo, da se pogosti znaki ponovijo vsaj 20-krat, redki vsaj 10-krat in da je tekst čimbolj realističen (če naš nabor dovoljenih znakov vsebuje tudi cifre ali posebne znake, naj bodo le-ti pomešani med črke). Paziti je potrebno, da je v omenjenih primerih uporabljena samo ena pisava, katere velikost naj bo vsaj 10 pik.

- Izdelava datoteke z okvirji.
  Datoteka vsebuje za vsak prepoznan znak na učni sliki pripadajočo koordinato ter višino in širino znaka. Obstajajo različni programi za urejanje datoteke, kjer preko grafičnega vmesnika na enostaven način popravimo morebitne napačno prepoznane znake. Eden izmed njih je jTessBoxEditor[8].

```
tesseract [lang].[fontname].exp[num].tif
          [lang].[fontname].exp[num]
          batch.nochop makebox
```

- Izdelava učne datoteke.
  V tem koraku Tesseractu pokažemo napake, ki jih delal pri prepoznavi besedila. S tem dosežemo, da v naslednjih poskusih prepoznave ne bo delal istih napak.

```
tesseract [lang].[fontname].exp[num].tif
          [lang].[fontname].exp[num]
          box.train
```

- Izdelava datoteke "unicharset".
  Proces v tem koraku poskuša iz datoteke z okvirji izvleči nabor znakov.

```
unicharset_extractor [lang].[fontname].exp0.box
                     [lang].[fontname].exp1.box ...
```

[8] http://vietocr.sourceforge.net/training.html

- Izdelava datoteke "font_properties".
  V tej datoteki opišemo lastnosti pisave, ki jo učimo. Sintaksa je naslednja:

  ```
  <fontname> <italic> <bold> <fixed> <serif> <fraktur>
  ```

  kjer je $fontname$ ime pisave, ki jo učimo, $italic$, $bold$, $fixed$, $serif$ in $fraktur$ pa so indikatorji z vrednostmi 0 oz. 1, s katerimi povemo, ali ima pisava določeno lastnost vklopljeno ali ne.

- Izdelava prototipov znakov.

  ```
  shapeclustering -F font_properties
                  -U unicharset
                  [lang].[fontname].exp0.tr
                  [lang].[fontname].exp1.tr ...

  mftraining -F font_properties
             -U unicharset
             -O [lang].unicharset
             [lang].[fontname].exp0.tr
             [lang].[fontname].exp1.tr ...

  cntraining [lang].[fontname].exp0.tr
             [lang].[fontname].exp1.tr ...
  ```

- Združitev nastalih datotek v eno datoteko.
  Nastalim datotekam *shapetable*, *normproto*, *inttemp*, $pffmtable$ dodamo pripono [lang], in jih z spodnjim ukazom združimo v [lang].traineddata.

  ```
  combine_tessdata [lang].
  ```

2.8 JSON

JSON (JavaScript Object Notation)[26] je odprt standard za izmenjavo podatkov, ki izhaja iz JavaScript skriptnega jezika. Z njegovo pomočjo lahko na

enostaven in človeku berljiv način predstavimo enostavne podatkovne strukture, ki so predstavljene kot pari atributov s pripadajočimi vrednostmi. Je alternativa jeziku XML in se največkrat uporablja za prenos podatkov med strežnikom in spletno aplikacijo. V primerjavi z jezikom XML je JSON enostavnejši, lažje ga bere tako človek kot računalnik, lažje ga preslikamo v podatkovne strukture, ki jih uporabljajo moderni programski jeziki, zaradi enostavnejših gradnikov ga je lažje procesirati, itd.
Osnovni tipi, ki jih podpira JSON:

- cela števila in števila s plavajočo vejico,
- tekstovni nizi,
- logične vrednosti (resnično, neresnično),
- seznami (urejena zaporedja vrednosti, ki so med seboj ločena z vejico in oglatimi oklepaji),
- objekti (neurejene zbirke parov ključ-vrednost, ki so med seboj ločene z vejico in zavitimi oklepaji),
- null (prazne vrednosti).

Primer iskanja uporabnikov, katerih ime se začne z besedo "mato" in pripadajoč odgovor EBT strežnika v formatu JSON:


```
{
        "users": [{
                "id": "187649",
                "name": "mato_666",
                "country": "Slovenia",
                "active": "1"
        }, {
                "id": "185941",
                "name": "mato95",
                "country": "Italy",
                "active": "0"
        }],
        "nr_users": 2,
        "nr_total": 2,
        "data": []
}
```

2.9 AChartEngine

AChartEngine [27] je odprtokodna knjižnica za operacijski sistem Android, ki nam ponuja risanje raznih grafikonov, kot so črtni grafi, grafi območij, raztreseni grafi, časovni grafi, stolpčni grafi, tortni grafi, mehurčni grafi, itd. Podpira Android operacijske sisteme 1.6 in novejše. Knjižnica je optimizirana in dobro obvladuje prikazovanje velikega števila podatkov. V našem primeru smo jo uporabili za stolpčni prikaz števila bankovcev glede na določen apoen ter za prikaz števila bankovcev glede na državo izdajatelja oz. državo tiska.

Poglavje 3

Aplikacija mEuroBillTracker

Ideja o spremljanju bankovcev, medtem ko se izmenjujejo med različnimi ljudmi, državami, kontinenti ni sicer nič novega. Eden izmed projektov, ki implementira omenjeno idejo se imenuje EuroBillTracker[1]. Ponuja nam spletno storitev, s pomočjo katere lahko spremljamo gibanje že predhodno vnešenih bankovcev in vnašamo nove bankovce. Ker je ročen vnos podatkov dokaj zamuden, smo se odločili zgraditi aplikacijo, ki pohitri celoten proces vnosa.

Arhitekturo aplikacije predstavlja diagram na sliki 3.1, kjer vidimo vse komponente, ki jih aplikacija potrebuje za svoje delovanje: Google Play Services (dostop do lokacijskih storitev, prikaz zemljevida), OpenCV Manager (lokalno upragovanje), Tesseract (prepoznavanje serijske številke bankovca - knjižnica za delovanje potrebuje pomnilniško kartico, na kateri je shranjena učna datoteka "euro.traineddata" s pisavo OCR-B), Android SDK (dostop do različnih senzorjev - sistem za pozicioniranje, pospeškomer, kamera), Jackson JSON Processor (razčlenjevanje vsebine odgovorov strežnika, ki so v formatu JSON) in AChartEngine (grafični prikaz števila vnešenih bankovcev). Vsa komunikacija s storitvami EuroBillTracker ter Google Geocoding API poteka preko povezave HTTPS.

[1] http://www.eurobilltracker.com

Izgradnjo aplikacije smo razdelili na tri funkcionalne sklope:

- vnos podatkov o bankovcu,
- komunikacija s storitvami EBT,
- pregled vnešenih bankovcev ter njihov prikaz na zemljevidu.

3.1 Vnos podatkov o bankovcu

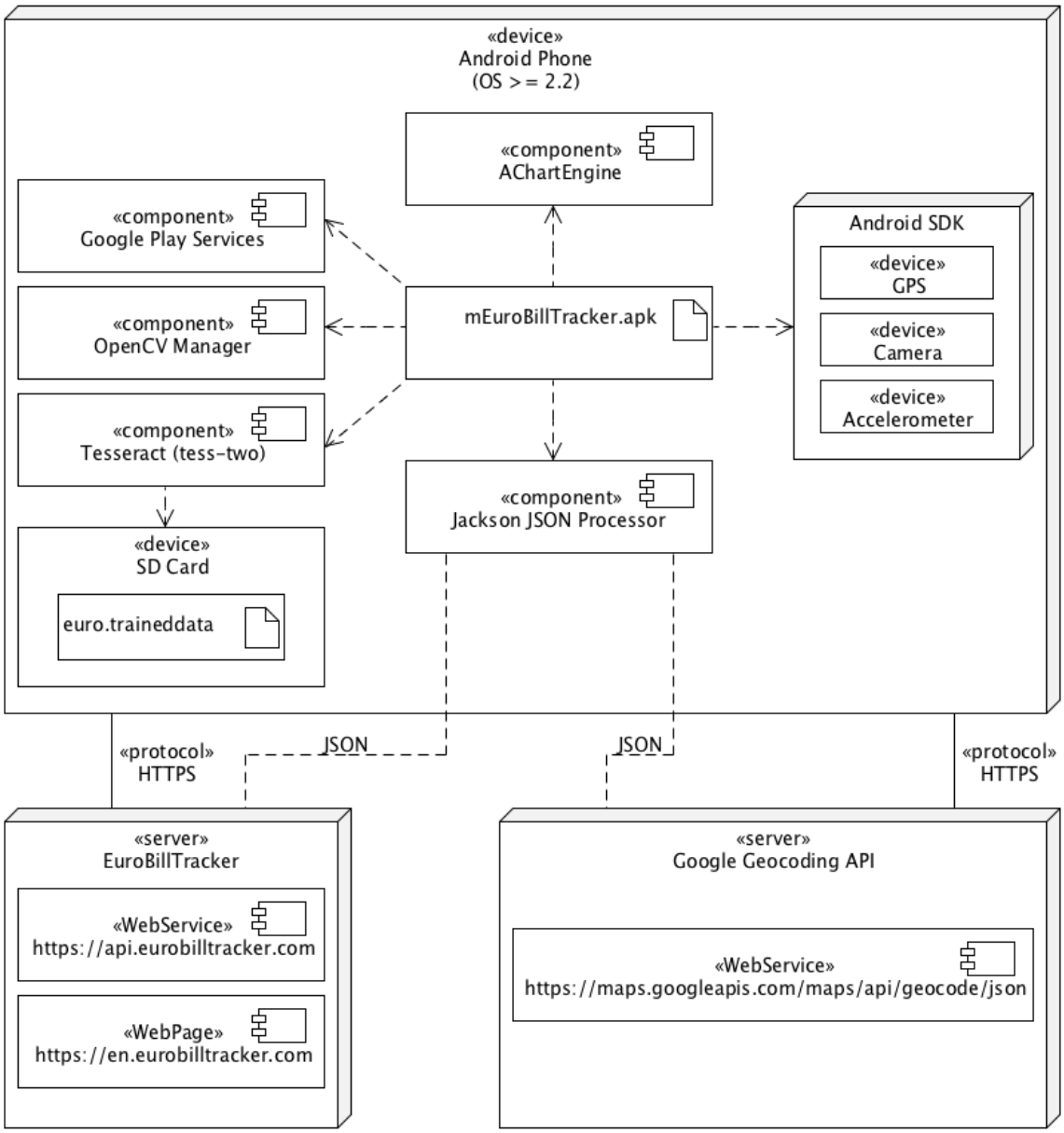
Vnesti moramo naslednje podatke (slika 3.2):

- apoen,
- serijska številka,
- koda tiska,
- kraj, poštna številka in država trenutne lokacije,
- komentar (opcijsko).

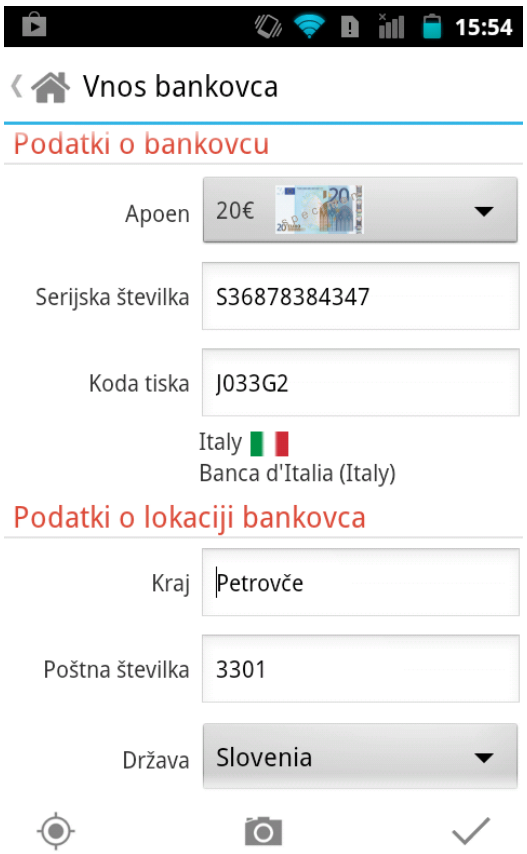
S pomočjo kamere in optične prepoznave besedila smo implementirali branje in prepoznavo serijske številke. Podatke o kraju, poštni številki in državi trenutne lokacije smo v celoti pridobili z uporabo lokacijskih storitev. Tako uporabniku ostane le še vnos apoena (izbere ga iz seznama apoenov), ter kode tiska.

3.1.1 Vnos serijske številke

Kot je že bilo omenjeno, zajamemo serijsko številko bankovca s pomočjo kamere in tako zajeto sliko z ustreznim predprocesiranjem nadalje pošljemo v proces optične prepoznave besedila.



Slika 3.1: UML postavitveni diagram, ki prikazuje arhitekturo aplikacije.



Slika 3.2: Zaslonska slika vnosa bankovca.

3.1.1.1 Neprekinjeno samodejno ostrenje kamere

Ker je za proces optične prepoznave zajete slike med drugim pomembno, da je le-ta čim ostrejša, je potrebno poskrbeti za to, da uporabnik pred zajemom slike sproži samodejno ostrenje kamere. Nekatere novejše naprave sicer že podpirajo neprekinjeno samodejno ostrenje. To pomeni, da medtem ko uporabnik premika napravo, se le-ta odzove na premik in v trenutku, ko se naprava umiri, samodejno sproži ostrenje. Tako uporabniku ni potrebno ročno prožiti akcije ostrenja. Podobno funkcionalnost smo želeli podpreti tudi na preostalih napravah, ki neprekinjenega samodejnega ostrenja kamere ne podpirajo. Prišli smo na idejo uporabe senzorja za pospešek, ki ga podpira skoraj vsaka mobilna naprava. Ker ima ostrenje slike smisel samo v trenutku, ko se gibanje naprave ustavi, smo s pomočjo senzorja za pospešek ta trenutek ujeli in šele takrat sprožili ostrenje kamere.
Najenostavnejši način uporabe senzorja za pospešek (poglavje 2.5.2) poteka po naslednjih korakih:

- Implementiramo vmesnik SensorEventListener.

```
public class MotionDetector implements SensorEventListener {

    SensorManager sensorManager;
    private Sensor accelerometerSensor;

    public MotionDetector(Context context, OnMotionListener
        motionListener, LowPassFilter lowPassFilter) {
        sensorManager = (SensorManager) context.getSystemService(
            Context.SENSOR_SERVICE);
        accelerometerSensor = sensorManager.getDefaultSensor(Sensor.
            TYPE_ACCELEROMETER);
    }
...
```

- Na začetku uporabe se z metodo registerSensorListeners prijavimo na storitev uporabe pospeškomera, na koncu se od storitve odjavimo z metodo unregisterSensorListeners.

```
public void registerSensorListeners() {
    if (accelerometerSensor != null) {
```

```
            sensorManager.registerListener(this, accelerometerSensor,
                SensorManager.SENSOR_DELAY_UI);
        }
    }

    public void unregisterSensorListeners() {
        sensorManager.unregisterListener(this);
    }
    ...
```

Pri prijavi navedemo s kakšnim zamikom želimo prejemati podatke senzorja.
SensorManager.SENSOR_DELAY_NORMAL: zamik 200000μs
SensorManager.SENSOR_DELAY_UI: zamik 60000μs
SensorManager.SENSOR_DELAY_GAME: zamik 20000μs
SensorManager.SENSOR_DELAY_FASTEST: zamik 0μs

- Zajem podatkov senzorja za pospešek.

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        //podatek o pospesku hranimo v tabelo acceleration
        System.arraycopy(event.values, 0, acceleration, 0, event.
            values.length);
    }
}
...
```

Zajeti podatki vsebujejo informacijo o pospešku po vseh treh oseh koordinatnega sistema (acceleration[0]: pospešek po osi X, acceleration[1]: pospešek po osi Y, acceleration[2]: pospešek po osi Z).
Poleg pospeška je v rezultat vpletena tudi komponenta gravitacije.

Takoj smo spoznali, da so podatki, ki jih pridobimo po opisani metodi, zaradi vpletene komponente gravitacije in zaznave manjših tresljajev dokaj neuporabni.
Rešitev problema je bila uporaba nizkopasovnega filtra [13], ki pridobljene vrednosti zgladi in izolira silo gravitacije, naknadno pa z visokopasovnim filtrom izločimo silo gravitacije.

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        System.arraycopy(event.values, 0, acceleration, 0, event.values.
            length);
        final float alpha = 0.6;

        // Z nizkopasovnim filtrom izoliramo silo gravitacije.
        gravity[0] = alpha * gravity[0] + (1 - alpha) * acceleration[0];
        gravity[1] = alpha * gravity[1] + (1 - alpha) * acceleration[1];
        gravity[2] = alpha * gravity[2] + (1 - alpha) * acceleration[2];

        // Z visokopasovnim filtrom odstranimo prispevek gravitacije.
        linearAcceleration[0] = acceleration[0] - gravity[0];
        linearAcceleration[1] = acceleration[1] - gravity[1];
        linearAcceleration[2] = acceleration[2] - gravity[2];
    }
}
...
```

Z omenjeno metodo smo odstranili vpliv nenadnih tresljajev pri premikanju mobilne naprave. Pri napravah, ki imajo verzijo operacijskega sistema višjo od 2.2, nam podobno filtriranje opravi uporaba tipa pospeškomera TYPE_LINEAR_ACCELERATION.

Da bi zaznali, kdaj se je naprava dejansko ustavila, smo morali postaviti minimalen prag pospeška, pod katerim smatramo, da naprava miruje, in preveriti, ali trenutni maksimalen pospešek v kateri koli smeri ne presega minimalnega praga. Ko pademo pod minimalen prag, je to znak, da se je naprava umirila in da moramo sprožiti samodejno ostrenje kamere. To storimo preko vmesnika *OnMotionListener*, kjer pošljemo informacijo o prenehanju gibanja naprave.

```
private static final float MIN_MOTION_ACCELERATION = 0.2f;
private boolean isMoving = false;

public interface OnMotionListener {
    public void onStartMoving();
    public void onStopMoving();
}

@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
```

```
        ...
        // Poiscemo maksimalen pospesek v katerikoli smeri.
        float maxLinearAcceleration = linearAcceleration[0];
        if (linearAcceleration[1] > maxLinearAcceleration) {
            maxLinearAcceleration = linearAcceleration[1];
        }
        if (linearAcceleration[2] > maxLinearAcceleration) {
            maxLinearAcceleration = linearAcceleration[2];
        }

        if (maxLinearAcceleration > MIN_MOTION_ACCELERATION) {
            motionListener.onStartMoving();
            isMoving = true;
        } else {
            if (isMoving) {
                motionListener.onStopMoving();
                isMoving = false;
            }
        }
    }
}
...
```

Zadnji korak je implementacija vmesnika *OnMotionListener*, kjer sprožimo samodejno ostrenje kamere. Ko se ostrenje kamere konča, si to informacijo zapomnimo, saj jo potrebujemo v procesu optične prepoznave. Proces sprožimo šele takrat, ko je ostrenje kamere končano.

```
public class NoteScanActivity extends Activity implements MotionDetector.
    OnMotionListener, Camera.AutoFocusCallback {
    private boolean cameraSteady = true;
    private boolean focused;
    private boolean autoFocusCompleted = true;

    @Override
    public void onStartMoving() {
        cameraSteady = false;
    }

    @Override
    public void onStopMoving() {
        cameraSteady = true;
        if (autoFocusCompleted) {
            autoFocusCompleted = false;
            getCamera().autoFocus(NoteScanActivity.this);
        }
    }
```

```
    @Override
    public void onAutoFocus(boolean success, Camera camera) {
        focused = success;
        autoFocusCompleted = true;
    }

    ...
}
```

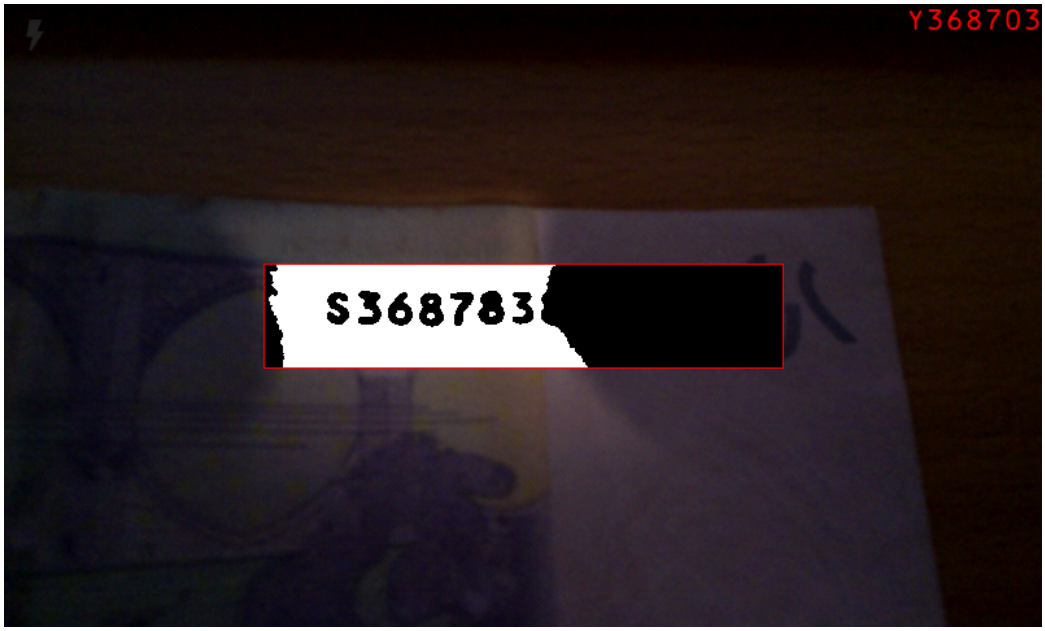
3.1.1.2 Zajem in predprocesiranje zajete slike

Ko je mobilna naprava umirjena in slika na kameri izostrena, za kar smo poskrbeli v prejšnjem koraku (poglavje 3.1.1.1), moramo preko kamere sliko zajeti in jo kot takšno poslati naslednjemu procesu optične prepoznave serijske številke. Izkazalo se je, da brez predhodnje obdelave zajete slike rezutati optične prepoznave niso najboljši, sploh pri slabših, neenakomernih pogojih osvetlitve. Sicer modul za optično prepoznavo besedila Tesseract sam opravi predprocesiranje slike (pretvorba v črno-belo sliko), vendar je le-to prilagojeno pogojem, kjer so pogoji osvetlitve idealni (npr. namenska naprava za skeniranje, ki zagotavlja enakomerno osvetlitev). Za pretvorbo slike v črno-belo uporablja metodo globalnega upragovanja (algoritem Otzu), ki v slabših pogojih osvetlitve ne deluje najbolje. Zato smo s pomočjo knjižnice OpenCV implementirali lokalno upragovanje, ki nima problemov z neenakomerno osvetlitvijo slike. Razliko med obema metodama upragovanja lahko vidimo na slikah 3.3 in 3.4, kjer se lepo vidi vpliv neenakomerne osvetlitve na proces pretvorbe slike v črno-belo sliko.
Metoda za lokalno upragovanje je opisana v poglavju 2.6.1.2. Poigrati smo se morali z dvema parametroma omenjene metode. Prvi parameter je $blockSize$, ki smo ga določili s poskušanjem. Izbrali smo fiksno vrednost 47. Tudi drugi parameter C smo najprej postavili na fiksno vrednost, vendar se je izkazalo, da je izbira vrednosti odvisna od trenutnih pogojev osvetlitve. Tako smo za vrednost izbrali povprečno vrednost slikovnih elementov sivinske slike in jo delili s številom 3. Ker na hitrost optične prepoznave besedila vpliva tudi velikost slike, smo iz celotne slike izrezali samo delček (pravokotnik), nad

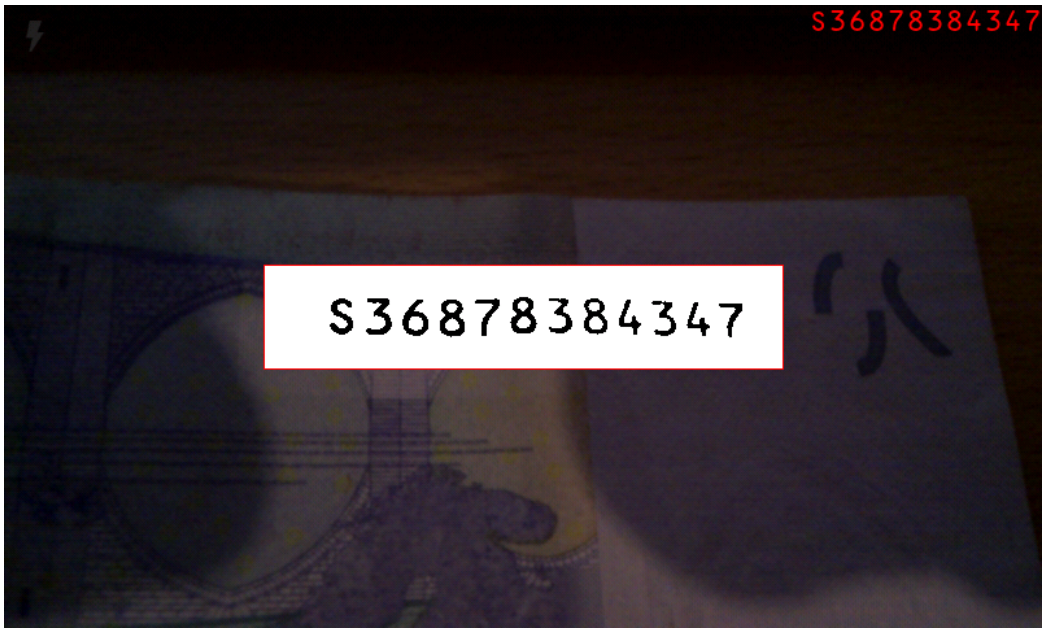
katerim smo izvajali predprocesiranje slike in kasneje optično prepoznavanje besedila. Implementacija opisanega je sledeča:

```
public class NoteScanActivity extends Activity implements
    CameraBridgeViewBase.CvCameraViewListener2 {
    private Mat srcRgba;
    private Mat srcGray;
    private Rect roiRect;

    @Override
    public void onCameraViewStarted(int width, int height) {
        srcRgba = new Mat();
        srcGray = new Mat();
        int rectWidth = width / 2;
        int rectHeight = height / 6;
        roiRect = new Rect((width - rectWidth)/2, (height - rectHeight)/2,
            rectWidth, rectHeight);
    }

    @Override
    public void onCameraViewStopped() {
        srcRgba.release();
        srcGray.release();
    }

    @Override
    public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame
        inputFrame) {
        srcRgba = inputFrame.rgba();
        srcGray = inputFrame.gray();
        Mat roiRgba = srcRgba.submat(roiRect);
        Mat roiGray = srcGray.submat(roiRect);

        double mean = Core.mean(roiGray).val[0];
        Imgproc.adaptiveThreshold(roiGray, roiGray, 255, Imgproc.
            ADAPTIVE_THRESH_MEAN_C, Imgproc.THRESH_BINARY, 47, mean/3);
        Imgproc.cvtColor(roiGray, roiRgba, Imgproc.COLOR_GRAY2RGBA);

        return srcRgba;
    }

    ...
}
```

Izkazalo se je, da je za predprocesiranje slike dovolj samo omenjena metoda upragovanja, ki odlično loči serijsko številko od samega ozadja bankovca.



Slika 3.3: Globalno upragovanje (metoda Otzu).



Slika 3.4: Lokalno upragovanje.

3.1.1.3 Optična prepoznava serijske številke

V prejšnjem poglavju smo opisali, kako pripraviti sliko za korak optične prepoznave serijske številke bankovca. Na sliki 3.4 vidimo, da nam proces lokalnega upragovanja sliko spremeni v črno belo in tudi odstrani morebiten šum v sliki. Takšen format slike namreč orodju, ki smo ga uporabili za optično prepoznavo (Tesseract) najbolj ustreza. Skeniranje številke bankovca smo si zamislili tako, da bi prepoznava potekala v realnem času. To pomeni, da uporabniku ni potrebno slikati celotnega bankovca in čakati na odgovor, ali je prepoznava uspela ali ne, ampak sredinski pravokotnik samo pridrži nad serijsko številko (slika 3.4), medtem ko se v ozadju neprekinjeno izvaja optična prepoznava serijske številke. Ko je serijska številka prepoznana, kamero samodejno zapremo in prepoznano številko prenesemo na ekran za vnos bankovca (slika 3.2). Seveda prepoznava ni vedno uspešna, saj smo lahko npr. med zajemom slike kamero preveč zatresli in zaradi tega dobili motno sliko, zato proces prepoznave ponavljamo tako dolgo, dokler prepoznava ni uspešna. Ali je številka pravilno prepoznana stestiramo tako, da skupaj seštejemo vsa števila, to ponavljamo toliko časa, dokler ne pridemo do enomestnega števila, ki ni nič drugega kot kontrolna vsota [28], ki jo določa prva črka v serijski številki (tabela 3.1).
Primer kontrole serijske številke:

$$S36878384347$$

$$3 + 6 + 8 + 7 + 8 + 3 + 8 + 4 + 3 + 4 + 7 = 61$$

$$6 + 1 = 7$$

Dobili smo rezultat 7, ki ga na osnovi prve črke v serijski številki (črka S) preverimo v tabeli 3.1.
Serijsko številko lahko stestiramo tudi tako, da kot v prejšnjem primeru seštejemo skupaj vsa števila, kjer črko nadomestimo z njeno ASCII vrednostjo. Seštevanje ponavljamo toliko časa, dokler ne pridemo do enomestnega števila. Rezultat mora biti deljiv s številom 9 [29].

Koda	Država	Kontrolna vsota
Z	Belgija	9
Y	Grčija	1
X	Nemčija	2
V	Španija	4
U	Francija	5
T	Irska	6
S	Italija	7
P	Nizozemska	1
N	Avstrija	3
M	Portugalska	4
L	Finska	5
H	Slovenija	9
G	Ciper	1
F	Malta	2
E	Slovaška	3
D	Estonija	4

Tabela 3.1: Identifikacijska kode.

$$S36878384347$$

$$83 + 3 + 6 + 8 + 7 + 8 + 3 + 8 + 4 + 3 + 4 + 7 = 144$$

$$1 + 4 + 4 = 9$$

Da smo dosegli prepoznavanje v realnem času, je bilo med drugim potrebno zagotoviti, da slika, ki jo pošljemo v proces optične prepoznave ni prevelika. Zato smo sliko prej obrezali na velikost pravokotnika in slika, ki jo vidi uporabnik skozi sredinski pravokotnik, je slika, iz katere Tesseract poskuša prepoznati besedilo. Kot smo opisali v poglavju 2.7.2, je bilo potrebno za uspešno prepoznavo serijske številke Tesseract naučiti nove pisave, konkretno gre za pisavo OCR-B. Pomagali smo si s programom jTessBoxEditor, kjer smo najprej zgenerali sliko z besedilom (slika 3.5) s pisavo

OCR-B, ki je vsebovalo nekaj sto naključno generiranih besed dolžine 12 znakov. Besede so vsebovale velike črke abecede in števila od 0 do 9. Kot rezultat smo dobili sliko "euro.ocrb.exp0.tif" s pripadajočo datoteko okvirjev "euro.ocrb.exp0.box", ki jo lahko nadalje urejamo in popravimo morebitne napačno prepoznane znake (slika 3.6). Potem ko smo v datoteki okvirjev odpravili morebitne napake, lahko začnemo s postopkom izdelave učne datoteke.

```
tesseract euro.ocrb.exp0.tif euro.ocrb.exp0 box.train
```

Naslednji korak je izdelava datoteke "unicharset",

```
unicharset_extractor euro.ocrb.exp0.box
```

datoteke "font_properties" z vsebino

```
ocrb 0 0 0 0 0
```

ter izdelava prototipov znakov.

```
shapeclustering -F font_properties -U unicharset euro.ocrb.exp0.tr
mftraining -F font_properties -U unicharset -O euro.unicharset euro.ocrb.exp0.tr
cntraining euro.ocrb.exp0.tr
```

Na koncu nastalim datotekam "shapetable", "normproto", "inttemp", "pffmtable" dodamo pripono "euro"

```
mv shapetable euro.shapetable
mv normproto euro.normproto
mv inttemp euro.inttemp
mv pffmtable euro.pffmtable
```

in jih združimo v končno datoteko "euro.traineddata".

```
combine_tessdata euro.
```

Dobljeno datoteko "euro.traineddata" moramo nekako prenesti v aplikacijo. Pomagamo si lahko z mapo "assets" (slika 3.7). Ker uporabljena knjižnica Tesseract zahteva, da se omenjena datoteka nahaja na pomnilniški kartici mobilne naprave, jo ob zagonu aplikacije prekopiramo iz mape "assets" na ustrezno mesto na pomnilniški kartici. Preden začnemo s kopiranjem, moramo dovoliti branje oz. pisanje na pomnilniško kartico. V AndroidManifest.xml moramo vpisati:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Nadalje sprožimo kopiranje naučenega jezika "euro" na pomnilniško kartico mobilne naprave.

```
String path = Environment.getExternalStorageDirectory().toString();
AssetManager as = getAssets();
InputStream is = as.open("tessdata/euro.traineddata");
OutputStream os = new FileOutputStream(path + "/tessdata/euro.traineddata");
byte[] buffer = new byte[1024];
int read;
while ((read = is.read(buffer)) != -1) {
    os.write(buffer, 0, read);
}
os.close();
```

Tako nam preostane še samo, da Tesseract knjižnici povemo, da naj za prepoznavo besedila uporabi omenjeno datoteko.

```
String path = Environment.getExternalStorageDirectory().toString();
TessBaseAPI tessBaseAPI = new TessBaseAPI();
tessBaseAPI.init(path, "euro");
```

Dodatno lahko Tesseract knjižnici pomagamo s tem, da ji naštejemo vse dovoljene znake, ki lahko nastopajo v besedilu.

```
tessBaseAPI.setVariable(TessBaseAPI.VAR_CHAR_WHITELIST,
                        "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789");
```

Prepoznavo serijske številke sprožimo s klicem metode:

```
Bitmap bitmap = ...
tessBaseAPI.setImage(bitmap);
String recognizedText = tessBaseAPI.getUTF8Text();
```

Seveda je priporočljivo, da se omenjena metoda kliče v svoji niti, saj lahko samo tako zagotovimo, da ostane uporabniški vmesnik odziven [5]. Na koncu preverimo, ali je prepoznana serijska številka bankovca pravilna. Če se izkaže, da ni pravilna, celoten postopek ponovimo in ga ponavljamo dokler serijska številka ni pravilno prepoznana. Potem kamero zapremo in serijsko številko prenesemo na ekran za vnos bankovca (slika 3.2).
Preden začnemo s prepoznavo serijske številke bankovca, moramo zadostiti naslednjim pogojem:

- kamera mora mirovati ($cameraSteady == true$),

```
public class NoteScanActivity extends Activity implements
    MotionDetector.OnMotionListener {
    private boolean cameraSteady = true;
    ...
    @Override
    public void onStartMoving() {
        cameraSteady = false;
    }

    @Override
    public void onStopMoving() {
        cameraSteady = true;
    }
    ...
}
```

- slika na kameri mora biti izostrena ($focused == true$),

```
public class NoteScanActivity extends Activity implements Camera.
    AutoFocusCallback {
    private boolean focused;
    ...
    @Override
    public void onAutoFocus(boolean success, Camera camera) {
        focused = success;
    }
    ...
}
```

- predhodnji proces prepoznave serijske številke se je končal ($ocrRunning == false$),

```
private class OcrTask extends AsyncTask<Bitmap, Void, String> {

    @Override
    protected void onPreExecute() {
        ocrRunning = true;
    }

    @Override
    protected String doInBackground(Bitmap... params) {
        tessBaseAPI.setImage(params[0]);
        String recognizedText = tessBaseAPI.getUTF8Text();
        return recognizedText;
    }

    @Override
```

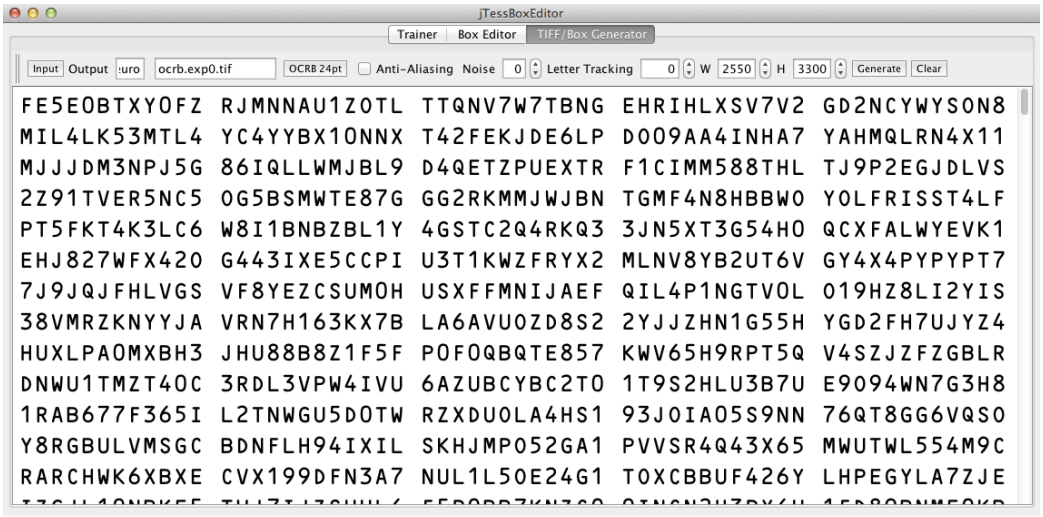
```
        protected void onPostExecute(String recognizedText) {
            ocrRunning = false;
            //preverjanje serijske stevilke bankovca (recognizedText)
            ...
        }
    }
```

Ko je zadoščeno vsem pogojem, sprožimo postopek prepoznavanja besedila vhodne slike.

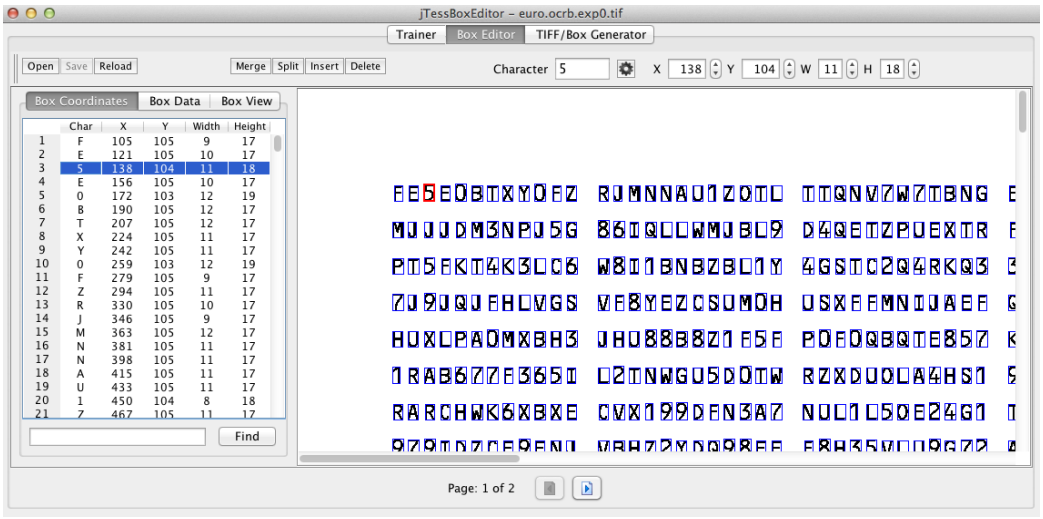
```
...
@Override
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame)
    {
    srcRgba = inputFrame.rgba();
    srcGray = inputFrame.gray();
    Mat roiRgba = srcRgba.submat(roiRect);
    Mat roiGray = srcGray.submat(roiRect);

    double mean = Core.mean(roiGray).val[0];
    Imgproc.adaptiveThreshold(roiGray, roiGray, 255, Imgproc.
        ADAPTIVE_THRESH_MEAN_C, Imgproc.THRESH_BINARY, 47, mean/3);
    Imgproc.cvtColor(roiGray, roiRgba, Imgproc.COLOR_GRAY2RGBA);

    if (cameraSteady && focused && !ocrRunning) {
        OcrTask ocrTask = new OcrTask();
        Bitmap bitmap = Bitmap.createBitmap(roiRgba.cols(), roiRgba.rows(),
            Bitmap.Config.ARGB_8888);
        Utils.matToBitmap(roiRgba, bitmap);
        ocrTask.execute(bitmap);
    }
    return srcRgba;
}
```

3.1.2 Vnos trenutne lokacije

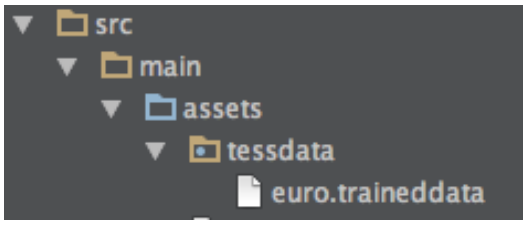
Vnesti moramo kraj, poštno številko in državo trenutne lokacije. Ker gre za zamudno opravilo, smo si pomagali z geolokacijskimi storitvami (poglavje 2.4), ki nam na osnovi določene zemljepisne širine in dolžine med drugimi vrne ravno zahtevane podatke. Ker nas zanima trenutna lokacija, le-to najlažje pridobimo z uporabo sistema za pozicioniranje (poglavje 2.3.1).
Zajem podatkov o trenutni lokaciji lahko tako razdelimo na dva dela:



Slika 3.5: Generiranje testnih vzorcev in izdelava okvirjev (jTessBoxEditor).



Slika 3.6: Urejanje okvirjev (jTessBoxEditor).



Slika 3.7: Assets mapa, kamor odložimo datoteko z naučenim jezikom "euro".

- pridobivanje zemljepisne širine in dolžine trenutne lokacije,
- pretvorba zemljepisne širine in dolžine v naslov lokacije.

3.1.2.1 Pridobivanje zemljepisne širine/dolžine trenutne lokacije

Trenutno lokacijo pridobimo z uporabo sistema za pozicioniranje. Najprej moramo aplikaciji dovoliti dostop do lokacijskih storitev, kar storimo tako, da v AndroidManifest.xml vpišemo:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Trenutno lokacijo pridobimo preko Location Client API-ja (poglavje 2.3.1.1).

```
public class InsertNoteActivity extends Activity implements
    GooglePlayServicesClient.ConnectionCallbacks, GooglePlayServicesClient.
    OnConnectionFailedListener {
    private LocationClient lc = new LocationClient(this, this, this);
    Location lastLocation;
    ...
    @Override
    public void onConnected(Bundle bundle) {
        lastLocation = lc.getLastLocation();
    }

    @Override
    public void onDisconnected() {

    }

    @Override
    public void onConnectionFailed(ConnectionResult connectionResult) {
        //Napaka pri povezavi do storitve
    }
    ...
```

Vidimo, da lokacije ne pridobimo v trenutku inicializacije LocationClient-a, ampak šele pri povratnem klicu metode *onConnected*. V primeru, da je prišlo do napake pri povezavi do storitve, lahko ta trenutek ujamemo pri povratnem klicu metode *onConnectionFailed*.
Ko storitve pridobivanja trenutne lokacije ne potrebujemo več, se od nje odjavimo. Ponavadi to storimo ob zaključku aktivnosti.

```
...
@Override
protected void onStop() {
    lc.disconnect();
    super.onStop();
}
...
```

3.1.2.2 Pretvorba zemljepisne širine in dolžine v naslov lokacije

Ko smo pridobili koordinate trenutne lokacije, lahko nadaljujemo z iskanjem naslova lokacije preko procesa obratnega geokodiranja (poglavje 2.4). Pri napravah, ki podpirajo storitev Geocoder, je stvar preprosta.

```
...
List<Address> addressList = null;
if (Geocoder.isPresent()) {
    Geocoder geocoder = new Geocoder(this, Locale.getDefault());
    addressList = geocoder.getFromLocation(lastLocation.getLatitude(),
                                           lastLocation.getLongitude(),
                                           1);
    Address address = addressList.get(0);
    String postnaStevilka = address.getPostalCode();
    String kraj = address.getLocality();
    String drzava = address.getCountryCode();
}
...
```

Pri napravah, ki omenjene storitve ne podpirajo, lahko uporabimo storitev Google Geocoding, kjer preko zahtevka HTTPS za zahtevane koordinate pridobimo naslov lokacije, ki je lahko v formatu XML ali JSON. Storitev je dosegljiva na naslovu

```
https://maps.googleapis.com/maps/api/geocode/json
```

ki nam rezultat vrača v formatu JSON, oz. na

```
https://maps.googleapis.com/maps/api/geocode/xml
```

če želimo rezultat v formatu XML. Iskane koordinate podamo s parametrom *latlng*.

```
https://maps.googleapis.com/maps/api/geocode/json?latlng=46.223763,15.183557&sensor=true
```

Z navedbo obveznega parametra *sensor* storitvi povemo, ali zahtevek prihaja iz naprave, ki podpira sistem za pozicioniranje. Odgovor klica storitve prikazuje primer 2.4.

3.2 Komunikacija s storitvami EBT

3.2.1 Opis storitev EBT

Storitev EuroBillTracker nam ponuja programski vmesnik za dostop do osnovnih funkcionalnosti pregledovanja in vnosa novih bankovcev. Storitve, ki nam jih ponuja programski vmesnik, lahko razdelimo na dve skupini [30]:

- Storitve, ki tečejo v uporabniškem kontekstu.
  Uporaba storitev je dovoljena samo registriranim uporabnikom. Če želimo uporabljati metode programskega vmesnika, moramo biti prijavljeni v sistem. Ob prijavi v sistem nam EBT strežnik vrne identifikator seje, s katerim se strežniku predstavimo pri nadaljnih klicih storitev.

- Storitve, ki tečejo v globalnem kontekstu.
  Uporaba storitev je dovoljena vsakomur. V primeru, da smo prijavljeni v sistem, nam strežnik pri nekaterih storitvah vrača bogatejši nabor podatkov.

Do programskega vmesnika dostopamo preko zahtevka HTTPS

```
https://api.eurobilltracker.com/?m=<method name>&
                                 v=<version>&
                                 phpsessid=<session id>
```

kjer je m ime metode, v njena verzija in *phpsessid* identifikator seje, ki ga dobimo ob prijavi v sistem. Vsi rezultati klicev metod so podani v formatu JSON in kodni strani UTF-8.
Ker namen diplomske naloge ni podrobno opisovanje ponujenih metod, njenih

vhodnih parametrov in razčlenjevanju rezultatov, jih naštejmo le nakratko[2].

Uporabniški kontekst vsebuje naslednje metode: *login* (prijava v sistem EBT), *logout* (odjava iz sistema EBT), *sessioncheck* (preverjanje veljavnosti uporabniške seje), *insertbills* (vnos bankovcev), *search* (iskanje uporabnikov, krajev in držav), *mycities* (seznam vseh mest, kjer smo vnesli bankovec), *myzipcodes* (seznam vseh poštnih številk, kjer smo vnesli bankovec), *mycomments* (seznam vseh komentarjev, ki so bili vpisani na določeni lokaciji).

V globalnem kontekstu so nam na voljo metode: $globalstats\_profile\_note$ (informacije o določenem bankovcu), $globalstats\_profile\_user$ (informacije o določenem uporabniku), $globalstats\_profile\_city$, $globalstats\_profile\_region$, $globalstats\_profile\_country$ (informacije o določenem mestu, regiji in državi).

Pri izgradnji aplikacije smo uporabili le nekaj ponujenih metod (prijava, odjava, preverjanje veljavnosti seje, vnos bankovca, podrobnejše informacije o bankovcu, iskanje uporabnika na osnovi uporabniškega imena). Pogrešali smo metodo za iskanje vseh lastno vnešenih bankovcev. Podatke nam je nazadnje uspelo pridobiti preko domače spletne strani, kjer obstaja neposredna povezava na datoteko, ki vsebuje seznam bankovcev, ki smo jih vnesli.

3.2.2 Pretvorba JSON struktur v javanske razrede

Vsebina vseh odgovorov strežnika, ki jih dobimo ob klicu posameznih metod, opisanih v prejšnjem poglavju, je v formatu JSON (poglavje 2.8). Ročno razčlenjevanje posameznih odgovorov in pretvorba le-teh v javanske strukture je zamuden proces, kjer lahko zaradi površnosti kaj hitro pride do napak. Da bi omenjen proces čimbolj avtomatizirali, smo uporabili princip, kjer za vsak strežnikov odgovor pripravimo ustrezen javanski razred, katerega vsak

[2]Podrobnejši opis programskega vmesnika se nahaja na strani http://api.eurobilltracker.com/index.html

atribut povežemo s pripadajočim atributom v odgovoru strežnika. Povezavo definiramo tako, da nad vsakim atributom javanskega razreda naredimo zaznamek, v katerega vpišemo ime atributa iz strežnikovega odgovora [31]. Opisan proces ponazarja spodnji primer prijave na strežnik EBT:

Primer 3.1: Prijava v storitev EBT - zahteva.

```
https://api.eurobilltracker.com/?my_password=xxx&v=2&my_email=matej.hlastec
    %40gmail.com&m=login
```

Primer 3.2: Prijava v storitev EBT - odgovor.

```
{
        "sessionid": "ngp8gf2ddldq8o1pa8rl648v21",
        "username": "mato_666",
        "my_city": "Petrov\u010de",
        "my_country": "Slovenia",
        "my_zip": "3301",
        "totalbills": "222",
        "totalhits": "2",
        "data": []
}
```

Najprej zgradimo ustrezen javanski razred, kjer z zaznamki "@JsonProperty" opremimo posamezne atribute razreda ter v oklepaju navedemo, na katero polje v strukturi JSON se atribut navezuje.

Primer 3.3: Opremljanje javanskega razreda "Session" z zaznamki "@JsonProperty".

```
@JsonIgnoreProperties(ignoreUnknown = true)
public class Session implements Serializable {

    private static final long serialVersionUID = 1L;

    @JsonProperty("sessionid")
    private String sessionId;

    @JsonProperty("username")
    private String username;

    private String email;

    @JsonProperty("my_city")
    private String myCity;
```

```
    @JsonProperty("my_country")
    private String myCountry;

    @JsonProperty("my_zip")
    private String myZip;

    @JsonProperty("totalbills")
    private Integer totalBills;

    @JsonProperty("totalhits")
    private Integer totalHits;

    public Session() {
    }
    // getters, setters
    ...
}
```

Sledi korak preslikave vhodnega besedila v JSON formatu v javanski razred, ki smo ga ustrezno opremili z zaznamki.

Primer 3.4: Preslikava odgovora strežnika v javanski razred "Session".

```
String loginResponse = ...;
ObjectMapper objectMapper = new ObjectMapper();
ObjectReader objectReader = objectMapper.reader(Session.class);
Session session = objectReader.readValue(loginResponse);
...
```

3.3 Pregled vnešenih bankovcev ter njihov prikaz na zemljevidu

Programski vmesnik EBT nam ne ponuja metode za pridobivanje seznama lastno vnešenih bankovcev, zato smo morali do podatkov priti na drug način. Pomagali smo si z domačo spletno stranjo, na kateri lahko kot registriran uporabnik preko neposredne povezave dostopamo do datoteke v CSV formatu, ki vsebuje seznam bankovcev. Omenjeno datoteko pridobimo preko zahtevka HTTPS.

```
https://en.eurobilltracker.com/my_notes/?tab=5
```

Rezultat je datoteka, v kateri se vsak zapis nahaja v svoji vrstici, posamezna polja pa so med sabo ločena z ločitvenim znakom ";".

```
5;2013;NA5016546055;;2014-02-11 11:57:29;Zalec;Slovenia;3310;N014J5;
135695470;2;0;46.252;15.1637
...
```

Pridobljeni podatki so nam dovolj, da lahko prikažemo enostaven seznam bankovcev, ki ga lahko vidimo na sliki 3.8. Če želimo podrobnejši vpogled v vnešen bankovec in si na zemljevidu ogledati njegovo potovanje (v primeru, da je bankovec v sistem vnesel že nekdo pred nami), nam pridobljen seznam ne ponuja dovolj podatkov. Lahko pa iz seznama pridobimo identifikacijsko številko, pod katero je v sistem vnešen bankovec. Na osnovi identifikacijske številke preko zahtevka HTTPS pridemo do podrobnejših informacij o bankovcu.

```
https://api.eurobilltracker.com/?note_id=135695470&v=1&m=globalstats_profile_note
```

V odgovoru vidimo, da obstaja poleg osnovnih podatkov o bankovcu tudi opis celotne poti s pripadajočimi koordinatami zemljepisne širine in dolžine.


```
{
        "data": {
                "note_id": "135695470",
                "serial_number": "NA5016546055",
                "anonymized_printer_code": "N",
                "printer_code": "N014J5",
                "is_moderated": "0",
                "is_edited": "0",
                "denomination": "5",
                "year": "2013",
                "cardinality": "2",
                "step": [{
                        "step_id": 0,
                        "timestamp": "2013-12-18␣03:47:00",
                        "timestamp_adjusted": "2013-12-18␣01:47:00",
                        "user_id": "174014",
                        "user_name": "mizar",
                        "city_id": "55755",
                        "city_name": "Ljubljana",
                        "city_name_translated": "Ljubljana",
                        "location_name": "Ljubljana",
                        "postal_code": "1000",
                        "country_name": "Slovenia",
                        "country_name_translated": "Slovenia",
                        "latitude": "46.0579",
                        "longitude": "14.5048",
                        "latlong_quality": 0,
```

```
                        "comment": "restaurant",
                        "hit_km": 0,
                        "hit_days": 0
                }, {
                        "step_id": 1,
                        "timestamp": "2014-02-11␣12:57:29",
                        "timestamp_adjusted": "2014-02-11␣10:57:29",
                        "user_id": "187649",
                        "user_name": "mato_666",
                        "city_id": "123321",
                        "city_name": "\u017dalec",
                        "city_name_translated": "\u017dalec",
                        "location_name": "\u017dalec",
                        "postal_code": "3310",
                        "country_name": "Slovenia",
                        "country_name_translated": "Slovenia",
                        "latitude": "46.252",
                        "longitude": "15.1637",
                        "latlong_quality": 0,
                        "comment": "",
                        "hit_km": 0,
                        "hit_days": 55
                }],
                "anonymized_serial_number": "Nxxxx6546xxx",
                "total_km": 55,
                "total_days": 55,
                "img_url_europe": "http:\/\/www.eurobilltracker.com\/map\/
                    notereport.php?area=europe;lat=:46.0579:46.252;long
                    =:14.5048:15.1637",
                "img_url_world": null
        }
}
```


Sedaj imamo na voljo dovolj podatkov, da lahko na zemljevidu narišemo celotno pot potovanja bankovca (slika 3.9).

Iz pridobljenih podatkov smo na koncu naredili še grafičen prikaz analize vseh vnešenih bankovcev glede na državo izvora oz. državo tiska bankovca. Za prikaz grafov smo uporabili odprtokodno knjižnico AChartEngine[3], v kateri najprej definiramo osnovne lastnosti grafa (naslov grafa, robove, barve osi, ozadja, label, itd.),

[3]http://www.achartengine.org

```
XYMultipleSeriesRenderer renderer = new XYMultipleSeriesRenderer();
renderer.setAxesColor(Color.BLACK);
renderer.setXLabelsColor(Color.BLACK);
renderer.setYLabelsColor(0, Color.BLACK);
renderer.setBackgroundColor(Color.TRANSPARENT);
...
```

nastavimo lastnosti posameznih stolpcev (barva, ali naj se nad stolpcem prikazujeje tudi vrednost, itd.),

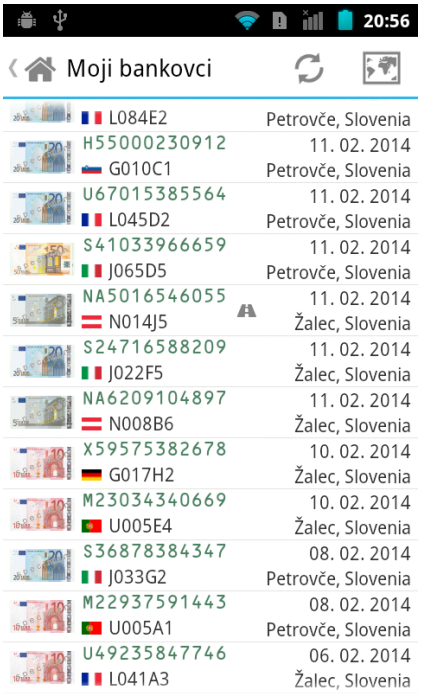
```
XYSeriesRenderer seriesRenderer = new XYSeriesRenderer();
seriesRenderer.setColor(Color.RED);
seriesRenderer.setDisplayChartValues(true);
...
```

ter dodamo posamezne stolpce s pripadajočimi vrednostmi.

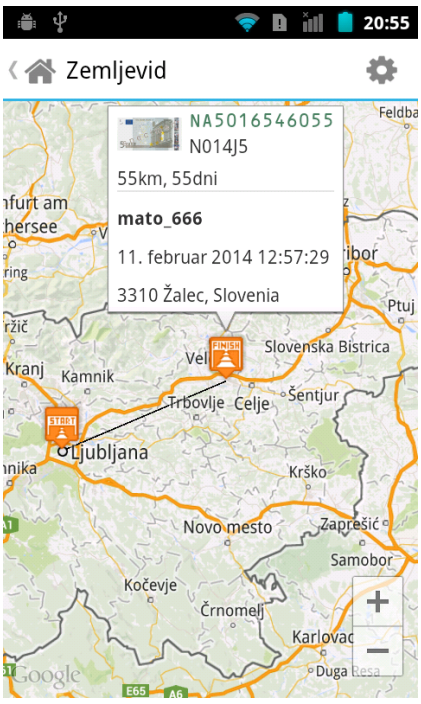
```
XYSeriesRenderer seriesRenderer = new XYSeriesRenderer();
seriesRenderer.setColor(Color.RED);
seriesRenderer.setDisplayChartValues(true);
XYSeries series = new XYSeries("");
renderer.addXTextLabel(1, "5EUR");
series.add(1, 27);
renderer.addXTextLabel(2, "10EUR");
series.add(2, 40);
renderer.addXTextLabel(3, "20EUR");
series.add(3, 60);
...
renderer.addSeriesRenderer(seriesRenderer);
XYMultipleSeriesDataset dataset = new XYMultipleSeriesDataset();
dataset.addSeries(series);
```

Na koncu samo še prikažemo celoten graf, ki ga lahko vidimo na sliki 3.10 in 3.11 iz katerega je razvidno, da večina bankovcev izhaja iz sosednjih držav (Italija, Nemčija, Avstrija, Francija).

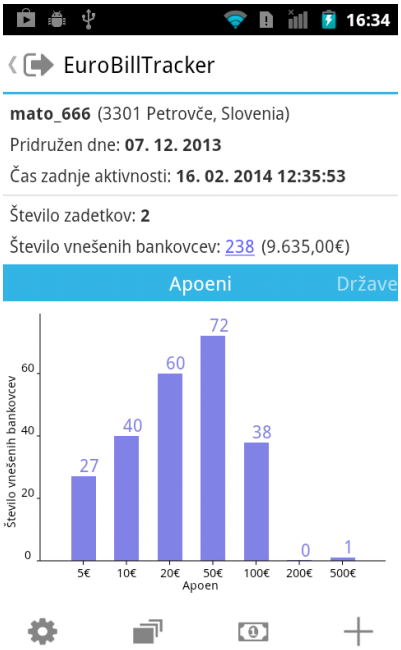
```
GraphicalView graphicalView = ChartFactory.getBarChartView(getActivity(),
    dataset, renderer, BarChart.Type.DEFAULT);
relativeLayout.addView(graphicalView);
```



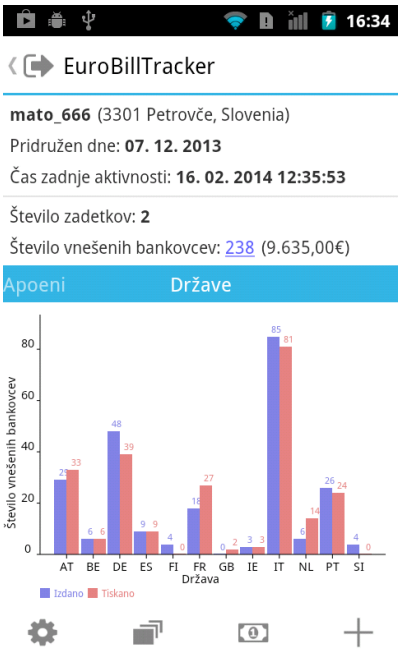
Slika 3.8: Prikaz seznama lastno vnešenih bankovcev.



Slika 3.9: Prikaz potovanja bankovca.



Slika 3.10: Prikaz števila vnešenih bankovcev glede na apoen.



Slika 3.11: Prikaz števila vnešenih bankovcev glede na državo izdajatelja oz. državo tiska.

Poglavje 4

Sklepne ugotovitve

V diplomski nalogi smo razvili aplikacijo mEuroBillTracker za mobilno platformo Android, ki se povezuje z že obstoječo spletno storitvijo EuroBillTracker. Cilj naloge, tj. poenostaviti in pohitriti celoten proces vnosa podatkov o bankovcu, je bil dosežen, saj aplikacija zahteva le ročen vnos apoena in kode tiska, vnos serijske številke ter podatkov o kraju, poštni številki in državi pa je v celoti avtomatiziran.

Najbolj zanimiv del naloge je bil zajem serijske številke bankovca, kjer smo se spoznali z uporabo kamere, zajemom in obdelavo zajete slike, pospeškomerom ter procesom optične prepoznave besedila. Ker sistem za prepoznavo besedila zahteva, da je slika čim ostrejša, smo na začetku uporabljali neprekinjeno samodejno ostrenje kamere. Ko pa smo ugotovili, da tega podpirajo le določene mobilne naprave, smo morali poiskati nadomestno rešitev. Prišli smo do ideje, da bi lahko samodejno ostrenje kamere programsko sprožili v trenutku, ko se mobilna naprava neha premikati. Na pomoč nam je priskočil pospeškomer, s pomočjo katerega smo zaznali gibanje naprave. Tudi pri uporabi pospeškomera se je izkazalo, da lahko iskani linearni pospešek na napravah z verzijo operacijskega sistema 2.3 naprej odčitamo brez večjih problemov, pri starejših verzijah operacijskega sistema pa smo morali motečo kompontento gravitacije odstraniti na drugačen način. Naslednji korak je bila obdelava zajete slike, kjer se je najbolje izkazala metoda

lokalnega upragovanja, ki je odporna na slabše pogoje osvetlitve slike. Pri procesu optične prepoznave besedila, smo ugotovili, da uporabljena knjižnica že vsebuje precej naučenih pisav, vendar so rezultati optične prepoznave pri pisavi OCR-B, ki se uporablja za zapis serijske številke, slabši. Zato smo si morali ustvariti množico učnih slik s celotnim naborom znakov, ki lahko nastopajo v serijski številki. Nad množico smo izvedli učenje in tako naučeno pisavo vključili v sistem prepoznave, ki je posledično potekala brez problemov.

Pri zajemu podatkov o trenutni lokaciji smo si pomagali z uporabo sistema za pozicioniranje. Podatke o zemljepisni širini in dolžini smo z metodo obratnega geokodiranja pretvorili v opisne podatke. Ker pa je ta storitev podprta samo na določenih mobilnih napravah, smo na preostalih napravah uporabili storitev Google Geocoding. Na splošno se je izkazalo, da imamo pri določevanju trenutne lokacije na voljo dva principa. Prvi (starejši) je uporaba Location Manager API-ja, ki je vgrajen že v sam operacijski sistem. Druga možnost pa je uporaba Location Client API-ja, ki je del Google Play SDK in si ga je potrebno na mobilno napravo naložiti naknadno. Uporabili smo slednjega, saj je v primerjavi s starejšim API-jem varčnejši, učinkovitejši in enostavnejši za uporabo.

Kar nekaj težav smo imeli pri uporabi že obstoječih storitev, ki jih ponuja spletna stran EuroBillTracker. API, ki nam je na voljo, je namreč dokaj osiromašen, prav tako določene metode API-ja za vhodne parametre zahtevajo identifikatorje, do katerih imamo posreden dostop samo preko prijave na domačo spletno stran.

Mobilno aplikacijo bi lahko izboljšali v smislu iskanja rešitve zajema in prepoznave kode tiska in nominacije bankovca, v okviru katere bi se spoznali z naprednejšimi metodami računalniškega vida. Zaradi omenjenih težav pri uporabi EuroBillTracker API-ja bi lahko med napravo in strežnik postavili vmesni strežnik, na katerem bi implementirali modernejši vmesnik do storitev EuroBillTracker, verjetno po principu REST.

Sistem za zajem in prepoznavo serijske številke bankovca bi lahko z minimalnim posegom razširili tudi na druga področja uporabe, na primer na področje mobilnega bančništva, kjer bi brez večjih težav lahko izvedli optično branje univerzalnega plačilnega naloga.

Literatura

[1] Android version history. Dostopno na:
http://en.wikipedia.org/wiki/Android_version_history
(Povzeto 26.01.2014).

[2] Android SDK. Dostopno na:
http://developer.android.com/sdk/index.html (Povzeto 27.01.2014).

[3] Android software development. Dostopno na:
http://en.wikipedia.org/wiki/Android_software_development
(Povzeto 27.01.2014).

[4] Android Developer Tools. Dostopno na:
http://developer.android.com/tools/help/adt.html
(Povzeto 27.01.2014).

[5] Marko Gargenta. Learning Android. O'Reilly Media, Inc., str. 65-69, 2011.

[6] Google Play Services. Dostopno na:
http://developer.android.com/google/play-services/index.html
(Povzeto 27.01.2014).

[7] Location Strategies. Dostopno na:
http://developer.android.com/guide/topics/location/strategies.html
(Povzeto 27.01.2014).

[8] Location API. Dostopno na:
http://developer.android.com/google/play-services/location.html
(Povzeto 27.01.2014).

[9] Google Maps Android API v2. Dostopno na:
http://developers.google.com/maps/documentation/android/intro
(Povzeto 27.01.2014).

[10] Geocoder. Dostopno na:
http://developer.android.com/reference/android/location/Geocoder.html
(Povzeto 27.01.2014).

[11] The Google Geocoding API. Dostopno na:
http://developers.google.com/maps/documentation/geocoding
(Povzeto 27.01.2014).

[12] Sensors Overview. Dostopno na:
http://developer.android.com/guide/topics/sensors/sensors_overview.html
(Povzeto 27.01.2014).

[13] Motion Sensors - Accelerometer. Dostopno na:
http://developer.android.com/guide/topics/sensors/sensors_motion.html#sensors-motion-accel (Povzeto 30.01.2014).

[14] Motion Sensors - Linear Accelerometer. Dostopno na:
http://developer.android.com/guide/topics/sensors/sensors_motion.html#sensors-motion-linear (Povzeto 30.01.2014).

[15] Greg Milette, Adam Stroud. Professional Android Sensor Programming.
John Wiley & Sons, Inc., str. 46-48, 92-93, 111-112, 2012.

[16] OpenCV. Dostopno na:
http://opencv.org (Povzeto 02.02.2014).

[17] OpenCV - Introduction. Dostopno na:
http://docs.opencv.org/modules/core/doc/intro.html
(Povzeto 02.02.2014).

[18] OpenCV - Thresholding. Dostopno na:
http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#threshold (Povzeto 02.02.2014).

[19] OpenCV - Adaptive Thresholding. Dostopno na:
http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#adaptivethreshold (Povzeto 02.02.2014).

[20] Gary Bradski, Adrian Kaehler. Learning OpenCV. O'Reilly Media, Inc., str. 135-140, 2008.

[21] OpenCV4Android SDK. Dostopno na:
http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/O4A_SDK.html#opencv4android-sdk (Povzeto 02.02.2014).

[22] OpenCV Manager. Dostopno na:
http://docs.opencv.org/android/service/doc/Intro.html
(Povzeto 02.02.2014).

[23] Optical character recognition. Dostopno na:
http://en.wikipedia.org/wiki/Optical_character_recognition
(Povzeto 04.02.2014).

[24] Tesseract OCR. Dostopno na:
http://code.google.com/p/tesseract-ocr (Povzeto 04.02.2014).

[25] TrainingTesseract3. Dostopno na:
http://code.google.com/p/tesseract-ocr/wiki/TrainingTesseract3
(Povzeto 07.02.2014).

[26] JSON. Dostopno na:
http://en.wikipedia.org/wiki/JSON (Povzeto 14.02.2014).

[27] AChartEngine. Dostopno na:
http://code.google.com/p/achartengine (Povzeto 16.02.2014).

[28] The Euro Information Website - Banknote Serial Numbers. Dostopno na:
http://www.ibiblio.org/theeuro/bnk.serialnumbers.htm
(Povzeto 09.02.2014).

[29] Euro banknotes - Security features. Dostopno na:
http://en.wikipedia.org/wiki/Euro_banknotes#Security_features
(Povzeto 10.02.2014).

[30] EuroBillTracker - API. Dostopno na:
http://api.eurobilltracker.com/index.html (Povzeto 11.02.2014).

[31] Jackson Annotations. Dostopno na:
http://wiki.fasterxml.com/JacksonAnnotations (Povzeto 14.02.2014).