Rok akademicki 2012/2013

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Automatyki i Informatyki Stosowanej

PRACA DYPLOMOWA INŻYNIERSKA

Maciej Sikora

Mobilny system monitorowania zawodnika na torze wyścigowym

Opiekun pracy
Dr inż. Mariusz Kamola

STRESZCZENIE

Niniejsza praca ma na celu sprawdzenie i przedstawienie możliwości platformy mobilnej Android do realizacji systemu wspomagania zawodnika na torze wyścigowym.

Zasadniczą częścią pracy jest aplikacja mobilna *Lap Magic*, która w oparciu o system GPS mierzy czasy okrążeń pokonywanych przez kierowcę na określonym torze (mogącym zawierać punkty podziału na sektory – punkty kontrolne pomiaru czasu). Ponadto aplikacja rejestruje i zapisuje inne dane, jak np. dane dotyczące lokalizacji w określonym czasie czy szybkości uzyskiwane przez zawodnika.

Projekt *Lap Magic* jest projektem open source, a sama aplikacja darmowa.

Słowa kluczowe:

Android, GPS, wyścigi, tor wyścigowy, rejestrowanie danych, chronometraż

Mobile support system for racer on a racetrack

ABSTRACT

The goal of this thesis is to explore and present capabilities of Android mobile platform to create system that supports driver on a racetrack.

Fundamental part of this thesis is mobile application *Lap Magic*, which uses GPS system to measure lap times on a track (which can be divided into sectors with “check points”). Moreover, the application logs and saves other data like GPS location at specific time or speed achieved by the racer.

Lap Magic is an open source project and the application is free.

Keywords:

Android, GPS, racing, racetrack, data logging, chronometric

Spis treści

1 WSTĘP 4

1.1 WPROWADZENIE 4

1.2 CEL PRACY 5

1.3 ZAWARTOŚĆ MERYTORYCZNA 7

2 PLATFORMA ANDROID 8

2.1 PODSTAWOWE INFORMACJE 8

2.2 ISTOTNE CECHY 8

2.3 IOS I INNE PLATFORMY 10

3 WYMAGANIA PROJEKTOWE 12

3.1 OGÓLNE ZAŁOŻENIA 12

3.2 WYMAGANIA FUNKCJONALNE 13

3.3 WYMAGANIA POZAFUNKCJONALNE 14

3.4 MOŻLIWE ZASTOSOWANIA 15

4 LAP MAGIC – PROJEKT ORAZ REALIZACJA 17

4.1 WYBÓR URZĄDZENIA 17

4.2 SYSTEM GPS – CHARAKTERYSTYKA I ZNACZENIE DLA PRACY 18

4.3 MODEL DANYCH SYSTEMU 20

4.4 GŁÓWNY ALGORYTM 23

4.5 WYKORZYSTANE TECHNOLOGIE ORAZ API 25

4.5.1 Baza danych 25

4.5.2 Definiowanie torów – Google Maps API 29

4.5.3 Wydajność oraz wielowątkowość przy przetwarzaniu w czasie rzeczywistym 29

4.5.4 Wsparcie dla starszych wersji Androida 34

4.6 PRZEGLĄD INTERFEJSU UŻYTKOWNIKA 35

5 PODSUMOWANIE 41

5.1 WNIOSKI 41

5.2 ROZWÓJ SYSTEMU 42

6 BIBLIOGRAFIA 44

1 Wstęp

1.1 Wprowadzenie

W dzisiejszych czasach coraz większą popularność zyskuje sport motorowy, nie tylko zawodowy, ale przede wszystkim amatorski. Mowa tutaj przede wszystkim o sporcie motocyklowym oraz samochodowym. Kierowcy nabierają świadomości, że tylko treningi na zamkniętych obiektach, w bezpiecznych warunkach, zapewnią efektywny rozwój umiejętności, które często ratują życie ludzkie w ruchu ulicznym. Dlatego buduje się coraz więcej asfaltowych torów wyścigowych, na których amatorzy mogą podnosić swoje zdolności kierowania pojazdem, a ponadto sportowcy mogą rywalizować ze sobą w wyścigach.

Nie tylko zawodnicy, ale również amatorzy chcą kontrolować swoje postępy podczas treningu. Miarą postępów na torze są przede wszystkim czasy okrążeń. Jednak zmieniające się czasy okrążeń danego zawodnika nie wynikają wyłącznie z rozwoju jego techniki. Duży wpływ mają również zmiany w ustawieniach kierowanego pojazdu. Biorąc za przykład motocykl, podstawowe parametry, które możemy zmieniać, to:

- opony, ich model, rozmiar, ciśnienie w oponach
- ustawienia zawieszenia, jego twardość, sposób jego pracy
- przełożenie na łańcuchu napędowym[1]

Duża część kierowców chce znać nie tylko czasy całych okrążeń, lecz także pewnych sektorów danego toru, tzw. międzyczasy. Z reguły podział na sektory ma na celu zróżnicowanie toru na fragmenty bardziej kręte oraz takie, które zawierają więcej odcinków prostych. Na międzyczasy wpływ mają powyższe parametry, ale także linie przejazdu zakrętów, obrane przez zawodnika czy wybrany bieg w poszczególnych miejscach toru.

[1] Motocykle są z reguły napędzane łańcuchem, łączącym dwie zębatki – napędową, połączoną ze skrzynią biegów oraz napędzaną, połączoną z kołem. Przełożenie to stosunek ilości zębów tych zębatek.

Kolejną istotną kwestią, zwłaszcza z punktu widzenia zawodnika, jest możliwość rejestrowania danych zbieranych podczas treningu. Dane, które możemy rejestrować, to:

- bieżąca pozycja (współrzędne)
- chwila w czasie
- szybkość chwilowa

Zebrane dane mogą posłużyć do różnych analiz. Dzięki nim zawodnik może odpowiedzieć sobie na pytanie, dlaczego w danym sektorze ma słabe wyniki. Może wywnioskować, co powinien zmienić (zarówno jeśli chodzi o ustawienia pojazdu, jak również o linię swojego przejazdu) aby zyskać przewagę w poszczególnych fragmentach toru. Na podstawie zebranych danych może również odtworzyć przebieg swojego treningu – np. określić, w którym momencie, w jakim punkcie toru był i jaką miał wtedy prędkość.

1.2 Cel pracy

Z roku na rok coraz większą popularnością cieszą się smartfony. Rynek jest ogromny, podobnie jak gama modeli, z bardzo zróżnicowaną konfiguracją sprzętową. Od prostych i tanich urządzeń, z procesorami o taktowaniu rzędu 200 MHz i pamięcią RAM na poziomie 150 MB (jak np. Samsung S5230 Avila), po zaawansowane maszyny obliczeniowe, gdzie funkcja telefonu jest właściwie dodatkiem, z wielordzeniowymi procesorami o taktowaniu rzędu 1,4 GHz, z pamięcią RAM rzędu 2 GB, posiadające liczne, precyzyjne sensory, tj. odbiornik GPS, akcelerometr, żyroskop (przykładem jest Samsung Galaxy S III).

Rynek smartfonów to nie tylko sprzęt – to również systemy operacyjne działające na urządzeniach. Badania światowego rynku przyprowadzone pod koniec 3. kwartału roku 2012 przez IDC[2] (źródło: http://www.idc.com/getdoc.jsp?containerId=prUS23771812) pokazują następujący udział mobilnych systemów operacyjnych:

- Google Android – 75%

[2] International Data Corporation, czołowy światowy instytut zajmujący się badaniami rynku w zakresie technologii informacyjnych oraz telekomunikacji.

- Apple iOS – 14,9%
- BlackBerry OS – 4,3%
- Symbian – 2,3%
- Windows Phone 7 / Windows Mobile – 2%
- pozostałe platformy[3] – 1,5%

Celem niniejszej pracy jest zbadanie oraz przedstawienie możliwości platformy Android do realizacji tytułowego systemu. Przedstawione zostały powody, dla których zdecydowałem się na tą właśnie platformę, m. in. jej architektura oraz uwarunkowania biznesowe.

Istotnym elementem pracy jest aplikacja mobilna *Lap Magic*, w której zaimplementowałem pomiar czasów okrążeń na torze wyścigowym w oparciu o system GPS (system opisano szerzej w rozdziale 4.2). Aplikacja pracuje na urządzeniu, które posiada przy sobie zawodnik podczas treningu. Rejestruje ona dodatkowo wspomniane dane, tj. pozycja oraz prędkość w czasie. Analiza zebranych danych służy do wnioskowania w celu uzyskania lepszych wyników treningu po odpowiednich zmianach ustawień pojazdu i/lub techniki jazdy zawodnika (np. zmiany linii przejazdu przez wybrane zakręty).

Projekt *Lap Magic* jest projektem open source i będzie on dalej rozwijany. Więcej informacji można znaleźć pod tym adresem: http://code.google.com/p/lapmagic/. Znajduje się tam repozytorium projektu (w oparciu o system kontroli wersji GIT). Są to już działania wykraczające poza cel niniejszej pracy, ale planuję opublikować aplikację na platformie Google Play[4]. Ponadto system ma możliwość rejestrować również dane określające bieżący stan pojazdu, w szczególności stan silnika. To jednak wymaga dodatkowego sprzętu, ale w tym właśnie kierunku planuję rozwój projektu.

[3] W roku 2013 planowane jest wypuszczenie na rynek kolejnego mobilnego systemu – jest to Ubuntu Phone OS. Biorąc pod uwagę prezentacje firmy Canonical, właściciela Ubuntu, system zapowiada się bardzo interesująco i zapewne mocno wpłynie na rynek technologii mobilnych.

[4] Google Play to oficjalna platforma dystrybucji aplikacji na Androida. Więcej o niej w rozdziale nr 2.

1.3 Zawartość merytoryczna

Rozdział 2 zawiera krótki opis platformy mobilnej Android. Przedstawiono główne wady i zalety w kontekście systemu monitorowania zawodnika. Porównano platformę z konkurencyjnymi rozwiązaniami. Wyjaśniono powody wybrania systemu Android jako środowisko aplikacji *Lap Magic*.

Rozdział 3 przedstawia wymagania postawione systemowi. Opisane zostały ogólne założenia oraz wymagania z punktu widzenia inżynierii oprogramowania – funkcjonalne oraz pozafunkcjonalne. Przedstawione zostały również przykładowe zastosowania systemu spełniającego dane wymagania.

W rozdziale 4 został opisany projekt systemu oraz sposób realizacji aplikacji *Lap Magic*. Przedstawiono urządzenie, na którym przeprowadzano testy pisanego oprogramowania. Scharakteryzowano system GPS, na którym opiera się utworzone oprogramowanie. Zaprezentowano model danych systemu. Opisano algorytm działania oraz sposób realizacji poszczególnych funkcjonalności – implementację bazy danych, sposób definiowania torów, zapewnienie odpowiedniej wydajności obliczeń, wielowątkowość podczas przetwarzania w czasie rzeczywistym, wsparcie dla starszych wersji systemu Android.

Rozdział 5 zawiera podsumowanie pracy, wnioski, plany dalszych testów oraz plany rozwoju systemu.

2 Platforma Android

2.1 Podstawowe informacje

Android to system operacyjny oparty na jądrze systemu Linux. Middleware oraz biblioteki systemowe zostały napisane w głównie w językach C/C++. Został zaprojektowany dla urządzeń mobilnych, takich jak smartfony i tablety. Android jest projektem open source, kod został opublikowany na licencji Apache. Obecnie jest to najpopularniejsza i najprężniej rozwijająca się platforma mobilna.

Platforma dostarcza rozbudowany framework do tworzenia aplikacji w jej środowisku. Jest on oparty na języku Java i używa wirtualnej maszyny Dalvik, która uruchamia pliki wykonywalne dex – w uproszczeniu, powstają one poprzez konwersję kodu bajtowego Javy. Framework zapewnia określony, pojedynczy model aplikacji, który pozwala uruchamiać jeden kod na urządzeniach szerokiej gamy, o różnej specyfikacji sprzętowej.

2.2 Istotne cechy

Głównym powodem, dla którego wybrałem system Android, to jego wielka popularność i wiodący udział w rynku. Aplikacje tworzone na Androida mają szansę dotrzeć do ogromnej liczby użytkowników. Wynika to w dużej mierze z tego, że jest to platforma otwarta. Każdy może pobrać kod Androida i uruchomić go na wybranym urządzeniu. Producenci telefonów dostarczają czasem jedynie tzw. nakładki systemowe, które dodają specyficzne aplikacje albo zmieniają interfejs graficzny, dzięki czemu np. Samsung będzie wyglądał inaczej niż HTC[1], mimo że mają ten sam system operacyjny. Dzięki temu urządzenia pracujące na Androidzie są stosunkowo tanie – w pewnym sensie odchodzi koszt zakupu oprogramowania.

[1] Samsung oraz HTC to czołowi producenci urządzeń pracujących z systemem operacyjnym Android. Ponadto, firmy te są mocno zaangażowane w rozwój tej platformy.

Kolejną kwestią jest dopracowany framework oraz SDK[2], który umożliwia budowanie, testowanie oraz debugowanie aplikacji na Androida i dostarcza emulator urządzenia, na którym możemy zainstalować różne wersje systemu operacyjnego. Wszystko jest darmowe. Dzięki temu możemy wstępnie testować kompatybilność tworzonych aplikacji z urządzeniami ze starszymi wersjami Androida. To skupia wokół Androida wielką społeczność projektantów i programistów, którzy rozszerzają możliwości platformy o nowe aplikacje i dzielą się swoją wiedzą na wielu grupach dyskusyjnych.

Platforma jest elastyczna jeśli chodzi o dystrybucję oprogramowania. Jest na to kilka sposobów, płatnych i bezpłatnych.

1. Oficjalnym kanałem dystrybucji jest platforma Google Play, która umożliwia sprzedawanie napisanego oprogramowania jak również darmową publikację. Twórca aplikacji musi się zarejestrować, wnieść jednorazową opłatę w wysokości 25 $ i wtedy może publikować swoje produkty. Każdy właściciel urządzenia może skorzystać z Google Play, aby znaleźć aplikację, której potrzebuje, pobrać ją (po wniesieniu opłaty na rzecz autora, jeśli aplikacja nie jest darmowa), w końcu wystawić w systemie recenzję i/lub ocenę. Dzięki temu oprogramowanie jest poddawane kontroli, co skutkuje wzrostem jakości dostępnych aplikacji.

2. Aplikacje można instalować na urządzeniach przy pomocy SDK. Jest to sposób dla programistów. Kod bajtowy Javy jest przekształcany w pliki dex, które (wraz z innymi zasobami aplikacji, jak np. pliki jpg) są umieszczane w pojedynczym archiwum apk – jest to ostateczny efekt budowania kodu źródłowego. Odpowiednie narzędzie z SDK wgrywa plik apk do urządzenia i instaluje aplikację. Urządzenie musi być podłączone do komputera, na którym piszemy kod, poprzez kabel USB.

3. Można wreszcie instalować aplikacje bezpośrednio z plików apk, mając je na urządzeniu. Wystarczy pobrać za pomocą urządzenia zamieszczony w internecie plik apk lub przesłać go mailem. Android potraktuje go jako instalator. Sposób

[2] Software Development Kit – zestaw narzędzi dla programistów niezbędny do tworzenia aplikacji w danym środowisku.

ten jest pozbawiony jakiejkolwiek kontroli, jednak przydaje się, zwłaszcza do testów, kiedy możemy udostępnić naszą aplikację pewnej grupie użytkowników.

Ważnym zagadnieniem, szczególnie w kontekście tytułowego systemu, jest kompatybilność platformy Android z różnorodnym sprzętem (akcesoriami) rozszerzającym standardowe możliwości danego smartfona czy tabletu. Nie jesteśmy ograniczeni poprzez zamkniętą listę wspieranych akcesoriów. Przykładowo, możemy połączyć smartfona z jakimś akcesorium przez Bluetooth i mieć gwarancję, że system operacyjny obsłuży komunikację między nimi, mimo że sprzęt został wyprodukowany przez różne firmy i nie był projektowany z myślą o współpracy ze sobą. Jest to związane z ideą Androida, która zakłada, że na każdym urządzeniu mobilnym możemy uruchomić system. W myśl tej idei „otwartości" platformy, urządzenia mobilne są kompatybilne z szeroką gamą dodatkowego sprzętu. Dzięki temu mamy bardzo duże możliwości rozszerzania lub ulepszania naszych smartfonów, co zostało również wykorzystane w tej pracy.

2.3 iOS i inne platformy

Z pozostałych systemów, właściwie rozważałem jedynie Apple iOS wraz z którymś smartfonem z rodziny iPhone. Chciałem uniknąć mało popularnych platform, stanowiących zaledwie kilka procent rynku (dla przypomnienia z rozdziału 1.2, Android posiada 75% rynku, iOS prawie 15%, co oznacza 90% rynku zajętego przez tylko dwie platformy). Ostatecznie zdecydowałem się na Androida, który miał przewagę nad iOS w kilku względach.

Platforma iOS również posiada świetny framework, oparty na języku Objective–C[3]. Jest wiele opinii wśród informatyków, że jest on lepszy niż w Androidzie. SDK również jest darmowe, jednak pracuje on tylko i wyłącznie na systemie operacyjnym OS X (wymagany jest komputer klasy Mac). Ponadto nie można darmowo instalować swoich aplikacji na iPhonie, a jedynie na emulatorze urządzenia. Aby móc uruchamiać swoje aplikacje na urządzeniu należy skorzystać z programu dla programistów (iOS Developer Program), co

[3] Objective-C został opracowany przez firmę Apple. Rozszerza język C o cechy obiektowe, systemy operacyjne OS X oraz iOS zostały napisane głównie w Objective-C.

kosztuje 99 $ rocznie. Możemy wtedy zarejestrować swoje urządzenie i wgrywać na nie tworzone aplikacje. Uczestnictwo w programie pozwala nam również na udostępnianie (darmowe lub płatne) swojego oprogramowania na platformie Apple Store – system firmy Apple, działający na podobnej zasadzie jak Google Play.

iOS jest środowiskiem „zamkniętym", zgodnie z ideą firmy Apple. Występuje tu wiele ograniczeń, które trzeba spełnić, żeby móc tworzyć oprogramowanie. Przede wszystkim wiąże się to z dużymi kosztami. Komputery klasy Mac oraz iPhone'y to sprzęt znacznie droższy od konkurencji. Do tego dochodzi kosztowny iOS Developer Program, w którym udział jest niezbędny do testowania swojego oprogramowania na fizycznym urządzeniu (warunek konieczny realizacji tytułowego systemu). Nie ma tu standardu podobnego do plików apk w Androidzie, z których możemy zainstalować aplikację na każdym urządzeniu z Androidem. Przy pomocy SDK możemy uruchamiać aplikację tylko na własnym, zarejestrowanym iPhonie, instalacja na innych urządzeniach musi odbywać się za pośrednictwem Apple Store.

W przeciwieństwie do Androida, iOS jest przeznaczony wyłącznie dla urządzeń firmy Apple. Te z kolei są kompatybilne wyłącznie z określoną ilością wspieranych akcesoriów, z reguły droższych niż odpowiednie dodatki dla telefonów z Androidem. Nie mamy tu tak dużych możliwości rozszerzania funkcjonalności standardowego iPhone'a. W tym przypadku Android ma wyraźną przewagę.

Platforma iOS została stworzona przede wszystkim z myślą o telefonach i tabletach, w ich standardowym rozumieniu. iPhone jest bardzo udanym urządzeniem, a iOS stabilnym systemem operacyjnym. Jednak platforma ma ww. ograniczenia, co ma swoje zalety – wszystko jest poddawane kontroli i dzięki temu użytkownik końcowy dostaje produkt wysokiej jakości, co tyczy się również dostępnych aplikacji. Android z kolei, dzięki mniejszym ograniczeniom, pozwala lepiej wykorzystywać go do niestandardowych celów. Pozwala nam również precyzyjnie dostosować sprzęt do naszych potrzeb.

3 Wymagania projektowe

3.1 Ogólne założenia

System monitorowania zawodnika ma być zrealizowany w postaci aplikacji mobilnej. Rejestrowanie wszystkich danych ma się odbywać przez urządzenie, opcjonalnie połączone z innym sprzętem (dodatkowymi sensorami). Dane te są rejestrowane w czasie rzeczywistym, a urządzenie poddaje je wstępnemu przetwarzaniu i prezentacji. Może również eksportować zapisane dane do oddzielnych systemów, np. do komputera PC.

Zgodnie z rozdziałem nr 2, aplikacja ma być przeznaczona na system operacyjny Android. W swojej podstawowej formie ma być aplikacją darmową, ogólnodostępną, a kod źródłowy ma zostać opublikowany (projekt open source). Dzięki temu jest większa szansa na popularność projektu i na jego rozwój.

Na rynku obecne są urządzenia z różnymi wersjami Androida. Jak pokazują aktualne badania przeprowadzone przez Google (zakończone 3 stycznia 2013 roku, źródło: http://developer.android.com/about/dashboards/index.html), najpopularniejsze są wersje 2.3.0–2.3.7, oznaczone kodem Gingerbread, wydane na przełomie 2010 i 2011 roku, z udziałem 47,6% wszystkich urządzeń. Następne w kolejności są wersje 4.0.3–4.0.4, kod Ice Cream Sandwich, wydane w grudniu 2011 roku, z udziałem 29,1%. Najbardziej aktualne wersje 4.1–4.2, Jelly Bean to udział w sumie 10,2%, podobnie jak stosunkowo stare wersje 2.0–2.2 (Eclair oraz Froyo, wydane na przełomie 2009 i 2010 roku, w sumie 12,4% udziału). W związku z tym wymagane jest, aby aplikacja działała poprawnie na wszystkich znaczących na rynku wersjach systemu Android. To oznacza najnowszą wersję 4.2 Jelly Bean i wszystkie starsze aż po 2.1 Eclair (taki zakres to ok. 97% rynku).

3.2 Wymagania funkcjonalne

System ma mierzyć czas okrążeń na torze wyścigowym z możliwie największą dokładnością w oparciu o standard GPS (dokładność zależna przede wszystkim od możliwości czujnika GPS w danym smartfonie). Ma również zapewniać pomiar międzyczasów (opisane w rozdziale 1.1), jeśli dany tor zostanie opcjonalnie podzielony na sektory.

Musi być zapewniona możliwość tworzenia oraz edycji torów i zapisywania ich w systemie. Tor jest określony nazwą i składa się z punktu startowego, będącego jednocześnie punktem końcowym (start/meta). Opcjonalnie w skład toru wchodzą inne punkty, które go dzielą. Każdy punkt jest opisany parą współrzędnych geograficznych, szerokością i długością geograficzną, nazwą oraz typem (start/meta lub punkt podziałowy).

Należy grupować dane dotyczące danego treningu w postaci sesji treningowych. Sesja jest skojarzona z konkretnym torem, zawiera dane dotyczące pewnej liczby okrążeń, które nastąpiły po sobie. Ponadto sesja jest określona dokładnym czasem jej rozpoczęcia.

System, w trakcie aktywnej sesji, ma zapisywać dane dotyczące położenia (współrzędne) oraz szybkości w danej chwili. Takie dane są potrzebne, aby po skończonej sesji można było odtworzyć to co się działo w trakcie treningu, w ciągu kilku okrążeń – jaką trajektorią jechał zawodnik, jakie uzyskiwał szybkości w różnych punktach toru. Maksymalna prędkość w danym okrążeniu jest również ważna i razem z czasem przebiegu, charakteryzuje okrążenie[1].

Model danych ma uwzględniać parametry pojazdu opisane w rozdziale 1.1. Zestaw parametrów jest związany z daną sesją. Ma to umożliwić proces wnioskowania – jak zmienić poszczególne parametry, aby osiągać lepsze rezultaty w trakcie treningu.

[1] Tory wyścigowe są tak skonstruowane, że ich najdłuższym prostym (bez zakrętów) fragmentem jest tzw. prosta start/meta, posiadająca linię mety. Na tej prostej pojazdy osiągają największą prędkość w ciągu okrążenia. Wartość ta wynika ze sposobu przejechania ostatniej sekcji toru (przede wszystkim ostatniego zakrętu) oraz z dobranego przełożenia. Dlatego jest to cenny wskaźnik.

3.3 Wymagania pozafunkcjonalne

Aplikacja pełni swoją zasadniczą funkcję w czasie rzeczywistym. Bardzo ważna jest w takim razie szybkość wykonywanych obliczeń. Współpraca z systemem GPS wiąże się z pewnymi ograniczeniami. Częstotliwość odświeżania pozycji GPS (np. 2 Hz) odzwierciedla częstotliwość z jaką aplikacja odbiera z czujnika dane dotyczące położenia urządzenia oraz prędkości chwilowej. Dlatego nie możemy pominąć żadnego przychodzącego sygnału. Szybkość działania implementacji wpływa w tym przypadku na dokładność wyników.

Dokładność jest kolejną bardzo ważną kwestią. Standardowe moduły GPS w smartfonach mają z reguły niską częstotliwość odświeżania pozycji, rzędu 1–3 Hz. Do większości standardowych zastosowań taka częstotliwość jest wystarczająca, lecz gdy potrzebujemy precyzyjnie określić chwilę w czasie, gdy znajdziemy się w najbliższym otoczeniu zadanego punktu, tym bardziej jeśli przemieszczamy się z prędkościami rzędu 200 km/h, taka częstotliwość jest zbyt mała. Jadąc z prędkością 220 km/h, pokonujemy dystans 60 m w ciągu sekundy. Korzystając z wbudowanego czujnika o częstotliwości 1 Hz jesteśmy w stanie określić nasze położenie co 60 m, mając czujnik 2 Hz – co 30 m itd. Nie są to zadowalające wyniki. Dlatego w celu zapewnienia należytej dokładności, system musi wspierać obsługę zewnętrznego modułu GPS, łączonego ze smartfonem przez standard Bluetooth. Takie czujniki osiągają częstotliwość na poziomie 5–10 Hz, co jest już dobrym wynikiem. Mając czujnik 10 Hz, przy podanym przykładzie 220 km/h, rejestrujemy położenie co 6 m. Ponadto uzyskujemy hipotetyczną dokładność określenia czasu okrążenia do 0,1 s (pomiar dokonujemy 10 razy w ciągu sekundy). Są to zadowalające wyniki biorąc pod uwagę potrzeby treningowe. Należy również wspomnieć, że czujnik GPS o wysokiej częstotliwości wymaga od urządzenia odpowiednio szybkiego procesora, aby móc obsłużyć każdy przychodzący sygnał.

Ponadto aplikacja musi zapisywać obliczenia oraz rejestrowane dane. Sesja może trwać długo, więc nie możemy przechowywać danych w pamięci operacyjnej i dopiero po zakończeniu sesji dokonywać trwałego zapisu. Jest to zresztą obarczone ryzykiem utraty wszystkich danych z sesji, np. gdy proces z jakichś powodów zostanie zatrzymany lub urządzenie zgaśnie z powodu wyładowanej baterii. Z kolei nie możemy pozwolić na regularne przydzielanie procesora do operacji trwałego zapisu danych – wtedy wiele sygnałów

przepadnie i aplikacja będzie zachowywać się błędnie. Dlatego należy zapewnić cykliczny zapis danych w oddzielnym wątku. W ten sposób aplikacja ciągle obsługuje przychodzące z GPS sygnały, jednocześnie regularnie zapisuje trwale nowe dane „w tle".

Ostatnim wymaganiem jest zapewnienie prostego, wygodnego i intuicyjnego interfejsu użytkownika. Aplikacja ma być zgodna ze standardami tworzenia oprogramowania na Androida – to zapewnia jakość oraz dobre oceny w oczach wielu użytkowników platformy. Używanie aplikacji ma być łatwe i wygodne, a prezentacja danych przejrzysta. Należy również dostosować oprogramowanie do poprawnej pracy w trakcie sesji zarówno z wygaszonym ekranem, ze schowanym urządzeniem np. w kieszeni jak i zapewniając podgląd podstawowych danych, tj. czas bieżącego okrążenia, czas najlepszego okrążenia, numer okrążenia oraz aktualna prędkość.

3.4 Możliwe zastosowania

Aplikacja jest skierowana przede wszystkim dla motocyklistów[2] oraz kierowców samochodów trenujących na torach wyścigowych. Wymagania funkcjonalne zostały tak określone, aby „ułatwić życie" tej właśnie grupie użytkowników. Jednak z powodzeniem mogą jej używać inni sportowcy, gdzie dyscyplina wymaga wykonania pewnej czynności w czasie po zadanym, zamkniętym torze, na otwartej przestrzeni (trzeba koniecznie zapewnić łączność urządzenia z satelitami GPS). Dobrym przykładem są biegacze oraz rowerzyści trenujący na obranej trasie.

Głównym zastosowaniem systemu jest monitorowanie treningu zawodnika na torze wyścigowym. Zbierane dane służą zawodnikowi do kontroli swoich osiągnięć oraz śledzenia postępów. Startujący w amatorskich wyścigach, dzięki analizie danych, mogą lepiej przygotować się do zawodów na danym obiekcie. Znając ustawienia swojego pojazdu (opisane w rozdziale 1.1) i osiągane wyniki przy takiej konfiguracji, mogą wprowadzać zmiany mające na celu poprawę osiągnięć i stale to weryfikować.

[2] Wynika to z zainteresowań oraz hobby autora pracy.

System można również zastosować do odtworzenia pełnego toru przejazdu zawodnika w trakcie sesji treningowej. Posiadając dane dotyczące położenia oraz prędkości w każdej zarejestrowanej chwili, można graficznie wyznaczyć (wspomagając się interpolacją) linie przejazdu w poszczególnych okrążeniach zaznaczając również prędkość w każdym punkcie toru, a przede wszystkim w charakterystycznych miejscach, jak obszar przed wybranym zakrętem, środek tego zakrętu, fragment toru następujący po zakręcie, linia mety itp.

Wspomniani wcześniej biegacze, rowerzyści oraz inni sportowcy również mogą stosować system, w uproszczony sposób. W tym przypadku będzie to pewnego rodzaju automatyczny stoper. Jednak może on również służyć do kontroli osiąganych prędkości – szczególnie dla rowerzystów oraz biegaczy ważne są ich maksymalne prędkości, jakie są w stanie osiągać.

Możliwe zastosowania mogą być znacznie szersze. System w takiej formie jest podatny na rozwój, przede wszystkim dzięki możliwościom Androida oraz dostępnego sprzętu. System może się rozwijać w kierunku rejestrowania i przetwarzania jeszcze większej ilości danych, co zależy od dodatkowego sprzętu i czujników. Może on również w dalszej perspektywie wprowadzać funkcje społecznościowe i np. umożliwiać dzielenie się wynikami na torze ze znajomymi lub ich porównywanie, tworzenie list rankingowych związanych z danym torem itd.

4 Lap Magic – projekt oraz realizacja

4.1 Wybór urządzenia

Smartfon, który wybrałem jako urządzenie testowe to model Samsung GT–I9000 Galaxy S (Rysunek 4.1.1). Telefon wyposażony jest w system Android w wersji 2.1, Eclair (stosunkowo stara wersja). Jest to przedstawiciel segmentu wydajnych smartfonów, jego najważniejsze cechy to[1]:

- procesor o taktowaniu 1 GHz
- pamięć RAM 512 MB
- pamięć wbudowana 8 GB
- wbudowany czujnik GPS o częstotliwości odświeżania pozycji do 2 Hz

Rysunek 4.1.1 – Samsung GT–I9000 Galaxy S

[1] Na podstawie danych z oficjalnej strony – http://www.samsung.com/uk/galaxys/specs.html.

Galaxy S bardzo dobrze nadaje się do tego projektu. Posiada wbudowany moduł GPS, ponadto jest to dość wydajne obliczeniowo urządzenie na przewidywane potrzeby. Fakt, że ma przestarzałą wersję Androida pomaga przy pisaniu aplikacji wspierającej poprzednie wersje systemu. Moduł Bluetooth to już standard w tego typu urządzeniach i Galaxy również go posiada.

4.2 System GPS – charakterystyka i znaczenie dla pracy

GPS (Gloal Positioning System)[2] – system nawigacji satelitarnej stworzony przez Departament Obrony Stanów Zjednoczonych, zapewniający informacje dotyczące lokalizacji oraz czasu w każdym punkcie na Ziemi (pod warunkiem istnienia niezakłóconych linii widoczności z czterema lub więcej satelitami GPS). Ma ogromne znaczenie dla celów militarnych, cywilnych oraz komercyjnych. Jest utrzymywany przez Stany Zjednoczone Ameryki i jest ogólnodostępny (darmowy) dla każdego, kto posiada odbiornik GPS.

System składa się z trzech segmentów: kosmicznego – zbiór satelitów okrążających kulę ziemską, naziemnego – stacji kontrolnych i monitorujących na Ziemi, użytkownika – odbiorniki GPS i osprzęt służący do obliczania, m. in. dokładnej pozycji użytkownika systemu w trzech wymiarach – długość i szerokość geograficzna oraz wysokość. Pierwsze dwa segmenty są zarządzane i utrzymywane przez Siły Powietrzne Stanów Zjednoczonych.

1. Segment kosmiczny. Składa się z satelitów krążących po orbitach na wysokości 20200 km nad Ziemią, transmitujących sygnały drogą radiową. Utrzymanie segmentu polega na zapewnieniu przynajmniej 24 aktywnych satelitów przez 95% dostępnego czasu. Każda satelita okrąża Ziemię dwa razy na dobę. Orbity są tak ułożone, że z każdego punktu na Ziemi, w każdej chwili jest widok na przynajmniej 4 satelity.

2. Segment naziemny. Składa się z globalnej infrastruktury (stacji kontrolnych), która śledzi satelity GPS, nadzoruje ich poprawne transmisje i aktualizuje ich

[2] Głównym źródłem informacji zawartych w tym podrozdziale jest oficjalna strona internetowa http://www.gps.gov/systems/gps/.

dane nawigacyjne. Występuje 1 centralna stacja kontrolna, na terenie Stanów Zjednoczonych, 1 alternatywna centrala, 12 stacji dowodzących oraz 16 stacji monitorujących.

3. Segment użytkownika. W jego skład wchodzą setki tysięcy amerykańskich i sprzymierzonych wojskowych użytkowników GPS PPS[3] oraz dziesiątki milionów cywilnych, komercyjnych i naukowych użytkowników GPS SPS[4] (usługi działają na innych częstotliwościach transmisji). Typowy odbiornik GPS (w przypadku niniejszej pracy jest to wbudowany moduł smartfona Samsung Galaxy S) składa się z anteny dostrojonej do odpowiedniej częstotliwości, procesora obliczeniowego oraz zegara.

W uproszczeniu system działa następująco. Odbiornik odbiera sygnały z przynajmniej 4 satelitów. Każda satelita przesyła swoją pozycję oraz dokładny czas nadania sygnału t_1 ze swojego zegara atomowego[5]. Sygnał przemieszcza się z prędkością światła (300 km/s), odbiornik rejestruje nadejście sygnału w chwili t_2. Mając dane czas przebiegu sygnału (różnica $t_2 - t_1$) oraz jego prędkość, odbiornik wyznacza swoją odległość od satelity. Gdy odbiornik zna swoją odległość od przynajmniej 4 satelitów (oraz ich położenia), może określić swoje dokładne położenie na Ziemi w oparciu o zależności geometryczne. Wiele odbiorników dostarcza użytkownikowi również innych obliczeń, np. szybkości przemieszczania się liczonej na podstawie zmian ostatnich położeń odbiornika – tak wyliczona szybkość charakteryzuje się dużą dokładnością.

Rząd Stanów Zjednoczonych zobowiązuje się do zapewnienia cywilnym użytkownikom dokładności systemu takiej jak opisano w standardzie SPS – jest to w najgorszym przypadku blisko 8 m na poziomie wiarygodności 95%. Z reguły dokładność jest więc bardzo dobra, jest to zakres kilku (3–5) metrów.

[3] PPS – Precise Positioning Service, usługa GPS przeznaczona dla wojska, umożliwiająca zwiększenie dokładności systemu.

[4] SPS – Standard Positioning Service, standardowa usługa GPS, dostępna dla wszystkich.

[5] Szczególny rodzaj zegara, którego działanie opiera się na zliczaniu okresów atomowego wzorca częstotliwości.

Dokładność cywilnego systemu SPS jest właściwie taka sama jak wojskowego systemu PPS. Jednakże SPS przeprowadza transmisję na jednej częstotliwości, podczas gdy PPS na dwóch. Dzięki temu PPS może skorzystać z pewnych technik redukujących zakłócenia radiowe powodowane przez ziemską atmosferę. Brak zakłóceń zapewnia nieznacznie lepszą dokładność dla użytkowników wojskowych.

Niniejszy projekt opiera się na systemie GPS. Jego dokładność pozwala jedynie z dużym przybliżeniem określić moment kiedy znaleźliśmy się w konkretnym punkcie (aby pobrać czas z licznika). Biorąc pod uwagę zastosowania treningowe oraz wyznaczanie trendów wynikających z zestawień wyników, taka dokładność jest wystarczająca i akceptowalna. Implementacja musi jedynie uwzględniać zmieniające się na bieżąco odległości od zadanych punktów na torze i odpowiednio reagować, gdy urządzenie znajdzie się w dostatecznej bliskości punktu. Z kolei szybkość przemieszczania się wyliczana przez moduł GPS jest na tyle dokładna, że możemy przyjmować ją bezpośrednio, bez żadnego uśredniania wartości.

4.3 Model danych systemu

Przyjęto relacyjny model danych, w oparciu o który działa system (Rysunek 4.3.1). Model składa się z następujących tabel z danymi:

1. `Tracks`

   Reprezentuje tory wyścigowe. Posiada identyfikator oraz nazwę.

2. `Points`

   Przedstawia punkty wchodzące w skład toru. Punkt posiada nazwę, typ (start/koniec lub punkt podziału toru), współrzędne geograficzne (szerokość oraz długość geograficzną) oraz promień wyznaczający najbliższe otoczenie danego punktu. Posiada klucz obcy w postaci identyfikatora toru, do którego dany punkt należy.

3. `Sessions`

   Tabela reprezentuje przebyte sesje treningowe. Posiada identyfikator, nazwę oraz datę rozpoczęcia sesji. Posiada klucz obcy – identyfikator toru, na którym odbywa się sesja treningowa.

4. `Setting_Types`

   Tabela będąca słownikiem możliwych ustawień pojazdu, zawiera identyfikator typu oraz jego nazwę.

5. `Settings`

   Zawiera wpisy ustawień pojazdu dla danej sesji treningowej. Reprezentowane są przez wartość oraz opcjonalny komentarz. Tabela posiada klucze obce w postaci identyfikatora typu danego ustawienia oraz identyfikatora związanej z nim sesji. Para tych kluczy stanowi klucz główny tabeli.

6. `Laps`

   Przedstawia przejechane okrążenia. Każde okrążenie jest określone jego numerem porządkowym, czasem okrążenia oraz maksymalną zanotowaną prędkością w trakcie jego przebiegu. Tabela zawiera klucz obcy w postaci identyfikatora sesji. Identyfikator ten wraz z numerem porządkowym okrążenia stanowią klucz główny tabeli.

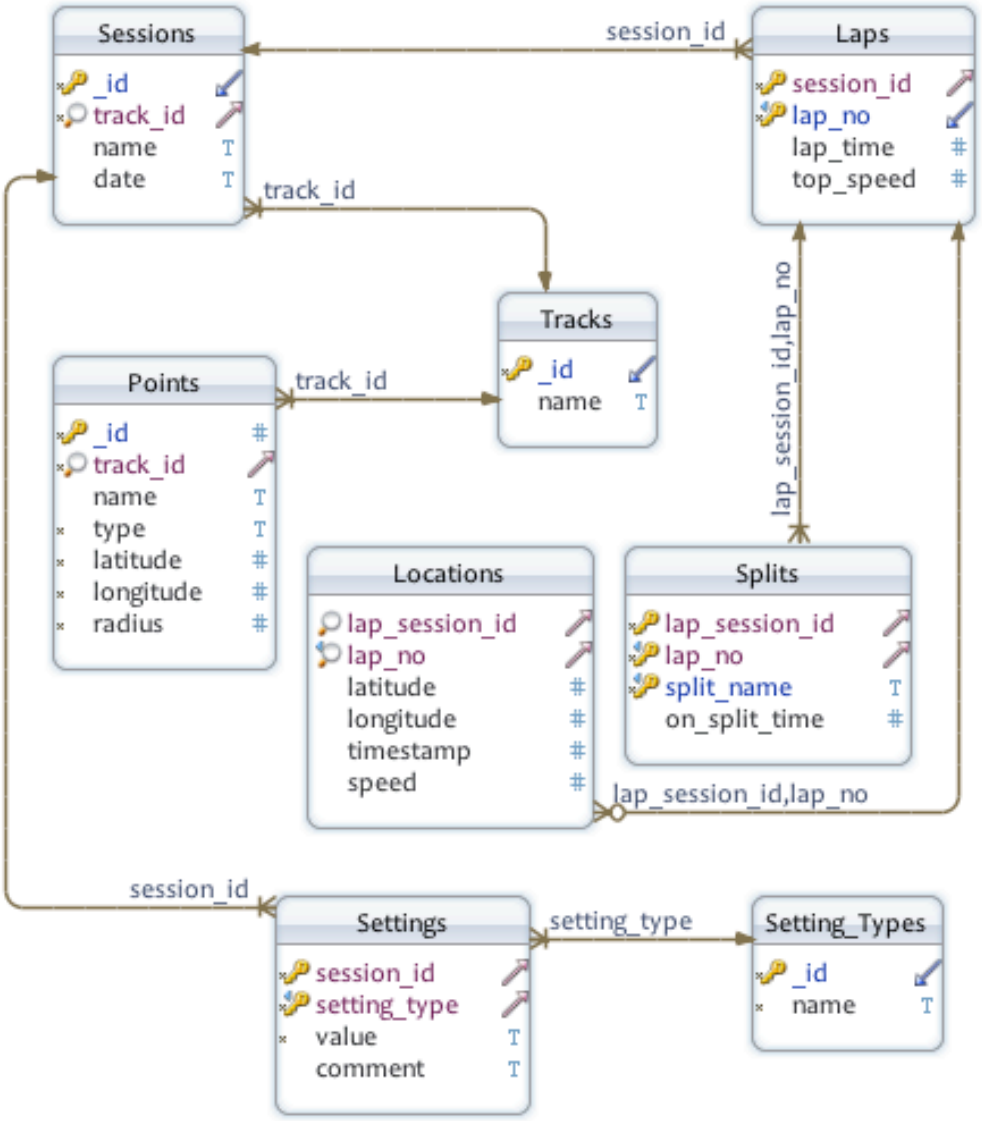
7. `Splits`

   Reprezentuje opcjonalne międzyczasy. Składają się z nazwy, związanej z nazwą odpowiedniego punktu podziału toru oraz czasu, po którego upływie dotarliśmy do tego punktu (licząc od początku okrążenia). Zawiera złożony klucz obcy w postaci identyfikatora okrążenia (identyfikator sesji wraz z numerem okrążenia). Klucz ten wraz z nazwą stanowi klucz główny tabeli.

8. `Locations`

   Tabela zawiera opcjonalne, zarejestrowane położenia zawodnika w trakcie trwania sesji. Są one reprezentowane przez szerokość oraz długość geograficzną, prędkość chwilową oraz czas, który upłynął od początku okrążenia do momentu

zarejestrowania położenia. Tabela zawiera klucz obcy taki jak w tabeli `Splits` – identyfikator sesji wraz z numerem okrążenia.



Rysunek 4.3.1 – model bazy danych systemu monitorowania zawodnika

4.4 Główny algorytm

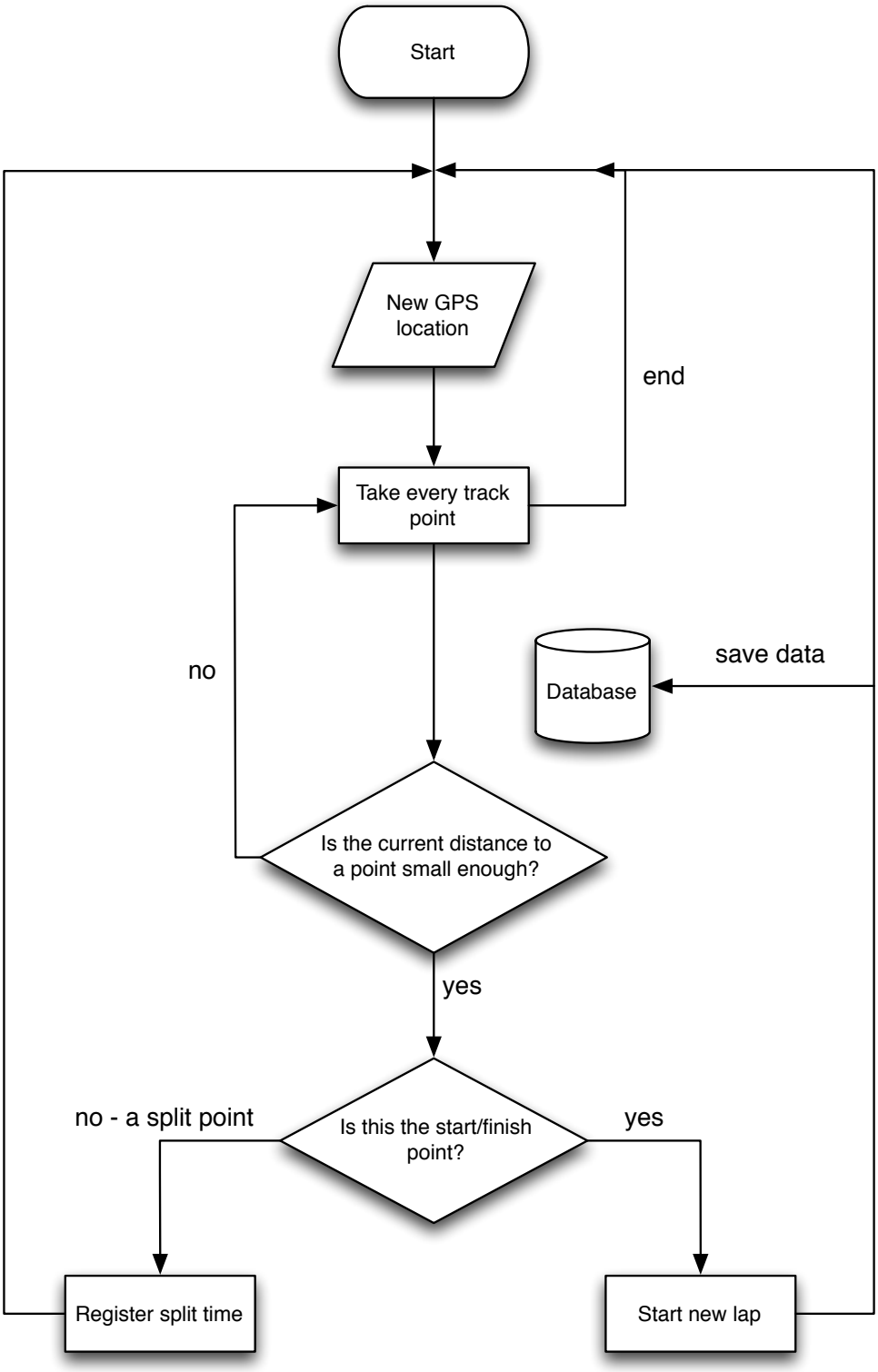
Najważniejszą funkcjonalnością systemu jest rejestrowanie danych w trakcie wykonywania okrążeń oraz wyznaczanie ich czasów. Ogólny algorytm przedstawia Rysunek 4.4.1. Algorytm zakłada, że jest dostępna lista punktów wchodzących w skład konkretnego toru.

Z każdym nadejściem sygnału GPS pobierane jest aktualne położenie oraz prędkość. Następnie przeglądany jest każdy punkt z listy. Jeśli urządzenie jest dostatecznie blisko danego punktu (odległość ta jest właściwością punktu, o czym zostało napisane w rozdziale 4.3) to algorytm stwierdza wejście w jego obszar. Następnie stara się uśrednić kilka kolejnych pomiarów położenia, aby wyeliminować możliwy błąd GPS (przypadkowe wejście w obszar punktu podczas gdy urządzenie nie jest w jego dostatecznie bliskim otoczeniu). Jeśli błąd nie zostaje stwierdzony, to w zależności od typu punktu:

- Aktualizowany jest międzyczas dla tego punktu, jeśli jest to punkt podziału toru.

- Rozpoczynane jest kolejne okrążenie, gdy natrafiono na punkt start/meta. W oddzielnym wątku zlecony zostaje zapis do bazy danych wszystkich informacji dotyczących okrążenia – numer kolejny, czas przebiegu, maksymalna zanotowana prędkość, międzyczasy tego okrążenia oraz zachowane wszystkie położenia oraz prędkości w poszczególnych chwilach czasu.

Po którejś z powyższych czynności algorytm dalej rejestruje sygnały z satelitów GPS. Zatrzymać go może jedynie użytkownik. Po ukończeniu sesji wybiera polecenie „stop” i system przestaje rejestrować dane. Mamy wtedy dostęp do wszystkich danych zapisanych w trakcie sesji.



Rysunek 4.4.1 – uproszczony schemat algorytmu przedstawiającego przebieg sesji

4.5 Wykorzystane technologie oraz API

W trakcie pisania aplikacji *Lap Magic* należało wykorzystać pewne konkretne technologie do realizacji poszczególnych funkcjonalności. Czasami nie było możliwości wyboru, który był narzucony przez framework Androida. W innych przypadkach trzeba było najlepiej dopasować do aktualnych potrzeb to, co udostępnia nam Android.

4.5.1 Baza danych

Android (podobnie jak inne platformy mobilne) wspiera system relacyjnych baz danych SQLite[6]. Jest to transakcyjna baza danych SQL przeznaczona do systemów wbudowanych (jakimi są urządzenia mobilne). Na kompletną bazę, z wieloma tabelami, indeksami, triggerami itd. składa się pojedynczy plik na dysku.

Framework Androida dostarcza kompletne API do bazy SQLite. Z poziomu kodu Javy można wygodnie tworzyć bazy danych, tworzyć i edytować obiekty bazodanowe i wreszcie zasilać tabele danymi oraz pobierać je za pomocą zapytań. *Lap Magic* wykorzystuje SQLite do zarządzania własnymi danymi.

Poniżej zamieszczono fragmenty kodu prezentujące implementację poszczególnych struktur. Fragment kodu 4.5.1 przedstawia klasę do zarządzania bazą danych, Fragment kodu 4.5.2 prezentuje implementację przykładowej tabeli `Laps`, Fragment kodu 4.5.3 pokazuje przykład operacji `insert`.

[6] http://www.sqlite.org/about.html

```
public class LapMagicDatabaseHelper extends SQLiteOpenHelper {

	private static LapMagicDatabaseHelper helperInstance = null;

	private static final String TAG = "LapMagicDatabaseHelper";

	private static final String DB_NAME = "lapmagic.db";
	private static int DB_VERSION = 7;

	private Context appContext;

	public static LapMagicDatabaseHelper getInstance(Context context) {
		if (helperInstance == null) {
			Log.d(TAG, "Creating the Singleton Database Helper instance");
			helperInstance = new LapMagicDatabaseHelper(
					context.getApplicationContext());
		}
		Log.d(TAG, "Returnign the Database Helper instance");
		return helperInstance;
	}

	private LapMagicDatabaseHelper(Context context) {
		super(context, DB_NAME, null, DB_VERSION);
		Log.d(TAG, "Setting the Singleton Database Helper application context = "
			+ context);
		appContext = context;
	}

	@Override
	public void onCreate(SQLiteDatabase db) {
		TracksTable.onCreate(db);
		SessionsTable.onCreate(db);
		...
	}

	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
		TracksTable.onUpgrade(db, oldVersion, newVersion);
		SessionsTable.onUpgrade(db, oldVersion, newVersion);
		...
	}

}
```

Fragment kodu 4.5.1 – baza danych aplikacji w postaci Singleton.

```
public class LapsTable {

	private static final String TAG = "LapsTable";

	public static final String TABLE_LAPS = "laps";
	public static final String COLUMN_SESSION_ID = "session_id";
	public static final String COLUMN_LAP_NO = "lap_no";
	public static final String COLUMN_LAP_TIME = "lap_time";
	public static final String COLUMN_TOP_SPEED = "top_speed";

	private static final String TABLE_CREATE = "create table "
			+ TABLE_LAPS
			+ "("
			+ COLUMN_SESSION_ID + " integer not null, " // references sessions
			+ COLUMN_LAP_NO + " integer not null, "
			+ COLUMN_LAP_TIME + " integer not null, " // in milliseconds
			+ COLUMN_TOP_SPEED + " integer not null, "
			+ "primary key(" + COLUMN_SESSION_ID + ", " + COLUMN_LAP_NO + ")"
			+ ");";

	public static void onCreate(SQLiteDatabase database) {
		Log.d(TAG, "creating table laps with statement '" + TABLE_CREATE
				+ "'");
		database.execSQL(TABLE_CREATE);
	}

	public static void onUpgrade(SQLiteDatabase database, int oldVersion,
			int newVersion) {
		Log.w(TAG, "Upgrading database from version " + oldVersion
				+ " to " + newVersion + ", which will destroy new all old data");
		database.execSQL("drop table if exists " + TABLE_LAPS);
		onCreate(database);
	}

}
```

Fragment kodu 4.5.2 – implementacja przykładowej tabeli.

```
SQLiteDatabase db = LapMagicDatabaseHelper.getInstance(this)
				.getWritableDatabase();

ContentValues trackValues = new ContentValues();
trackValues.put(TracksTable.COLUMN_NAME, track_name);

		if (!deleteMode) {

			// a new track - insert everything
			if (track_id == -1) {
				Log.d(TAG, "inserting new track " + track_name);
				track_id = db.insert(TracksTable.TABLE_TRACKS, null,
						trackValues);
	...
```

Fragment kodu 4.5.3 – przykład operacji insert.

Wystąpił jednak problem z kluczami obcymi. Otóż wsparcie dla kluczy obcych (oraz ograniczeń typu `on delete cascade`) jest obecne od wersji SQLite 3.6.19. Testowy

Samsung Galaxy S jest wyposażony w SQLite w wersji 3.5.9, dlatego sam zaimplementowałem triggery odzwierciedlające model danych aplikacji. Triggery działają również poprawnie na nowszych wersjach Androida, z nowszym SQLite, w wersjach 3.6.22, 3.7.x (testowane na symulatorze).

Triggery, które powstały (Fragment kodu 4.5.4 – część implementacji):

- na usuwanie toru – usuwane są również powiązane punkty
- na edycję id toru – edytowane są odpowiednie punkty
- na usuwanie sesji – usuwane są dane okrążeń związanych z sesją
- na usuwanie okrążeń – usuwane są zapisane międzyczasy związane z okrążeniem oraz wszystkie zarejestrowane położenia w tym okrążeniu

```
public class Triggers {

	private static final String TAG = "Triggers";

	public static final String DELETE_TRACK = "delete_track";
	public static final String UPDATE_TRACK = "update_track";
	public static final String DELETE_SESSION = "delete_session";
	public static final String DELETE_LAP = "delete_lap";

	private static final String DELETE_TRACK_CREATE = "create trigger "
			+ "if not exists " + DELETE_TRACK + " before delete on "
			+ TracksTable.TABLE_TRACKS + " begin delete from "
			+ PointsTable.TABLE_POINTS + " where "
			+ PointsTable.COLUMN_TRACK_ID + " = old."
			+ TracksTable.COLUMN_ID + "; end;";

	private static final String UPDATE_TRACK_CREATE = "create trigger "...
	...

	public static void onCreate(SQLiteDatabase database) {
		Log.d(TAG, "creating trigger " + DELETE_TRACK + " with statement '"
				+ DELETE_TRACK_CREATE + "'");
		database.execSQL(DELETE_TRACK_CREATE);

		Log.d(TAG, "creating trigger " + UPDATE_TRACK + " with statement '"
				+ UPDATE_TRACK_CREATE + "'");
		database.execSQL(UPDATE_TRACK_CREATE);
	...
	}
}
```

Fragment kodu 4.5.4 – implementacja triggerów.

4.5.2 Definiowanie torów – Google Maps API

Moim celem było zrealizowanie definiowania torów w prosty i wygodny sposób. Do tej funkcjonalności skorzystałem z API Google Maps, dostępnego jako standardowa część frameworku. Charakterystyczną cechą tej technologii jest to, że smartfon pobiera obraz map z serwera, przez internet. Także bez połączenia z internetem nie zobaczymy poprawnie map (chyba, że pewna część map została uprzednio pobrana i zapisana w cache'u).

Dzięki temu użytkownik może otworzyć mapę (w trybie satelitarnym[7]) i korzystając z interfejsu dotykowego smartfona nanieść punkty na wizerunek toru. Z tego miejsca może te punkty przesuwać po mapie oraz edytować ich właściwości. Google Maps precyzyjnie odzwierciedla punkt na mapie na współrzędne geograficzne (które zapisujemy razem z punktem), co wykazały testy. Taki sposób tworzenia oraz edytowania torów jest intuicyjny oraz praktyczny.

4.5.3 Wydajność oraz wielowątkowość przy przetwarzaniu w czasie rzeczywistym

Od aplikacji wymagane jest szybkie przetwarzanie danych przychodzących z GPS. Jednocześnie cykliczny, trwały zapis zarejestrowanych danych, tak jak zostało opisane w rozdziale 3.3. Implementacja korzysta z prostych typów danych, nie wymagających tyle pamięci oraz czasu przetwarzania jak dla złożonych obiektów. Algorytm preferuje punkt start/koniec w stosunku do pozostałych punktów – to właśnie na dokładnym określeniu czasu okrążenia zależy nam najbardziej i dlatego zawsze po nadejściu danych z GPS ten punkt jest sprawdzany w pierwszej kolejności.

Wymagane jest również, aby aplikacja realizowała zapis danych w trakcie trwania sesji w oddzielnym wątku. Każda aplikacja w systemie Android posiada swój własny proces. Android obsługuje interfejs użytkownika (UI) aplikacji w pojedynczym wątku jej procesu (w wątku interfejsu, inaczej – głównym). Jeśli nie jest to jawnie zaimplementowane, to cały kod aplikacji uruchamiany jest w jej wątku głównym – w takim przypadku wszystkie długotrwałe

[7] Widok mapy powstały ze zdjęć satelitarnych.

operacje blokowałyby całą aplikację. Nie można blokować głównego wątku, który rejestruje w czasie rzeczywistym dane GPS, więc należy w tym przypadku skorzystać z oddzielnego wątku.

Android wspiera wszystkie konstrukcje wielowątkowe dostarczane przez standard Javy, jak np. klasa `Thread` czy pakiet `java.util.concurrent`. Jednak aby bezpiecznie aktualizować w oddzielnym wątku interfejs użytkownika oraz dane zdefiniowane w wątku głównym, należy zsynchronizować oba wątki ze sobą. Android udostępnia w tym celu odpowiednie klasy i zastosowanie którejś z nich jest w tym przypadku najlepszym wyborem. Najważniejszymi przedstawicielami są klasy `Handler` oraz `AsyncTask`.

1. `Handler`.

   Obiekt klasy `Handler` utworzony w zasięgu głównego wątku może aktualizować UI. Obsługuje asynchroniczne komunikaty. Korzysta z kolejki komunikatów (`MessageQueue`) wątku, w którym został utworzony. Klasa jest głównie wykorzystywana do kolejkowania zadań z określeniem najwcześniejszego dopuszczalnego momentu rozpoczęcia.

2. `AsyncTask`.

   Klasa opakowuje obiekty typu `Thread` oraz `Handler` i zarządza ich tworzeniem. Umożliwia przetwarzanie w oddzielnym wątku, a następnie bezpieczną aktualizację UI wraz ze zdefiniowanymi w nim danymi. Aby jej użyć należy zaimplementować nową klasę, dziedziczącą po `AsyncTask`, następnie zaimplementować metodę `doInBackground`, która jest uruchamiana w oddzielnym wątku („w tle” tworzony jest obiekt `Thread`). Można również zaimplementować metody `onPreExecute` oraz `onPostExecute` – są to metody wywoływane w wątku głównym (są z nim zsynchronizowane), odpowiednio przed oraz po wykonaniu kodu metody `doInBackground`. W nich można bezpiecznie aktualizować UI oraz czytać dane (z blokadą do czytania).

Zapis danych został zrealizowany przy pomocy klasy AsyncTask. *Lap Magic*, w trakcie trwającej sesji, przechowuje zestaw zmiennych określających stan aplikacji – są to numer okrążenia, największa uzyskana prędkość w trakcie okrążenia oraz międzyczasy. W trakcie przechodzenia do następnego okrążenia dane te muszą zostać zapisane a zmienne zresetowane (aplikacja ma być od razu gotowa na kolejne sygnały GPS). W metodzie onPreExecute, która jest zsynchronizowana z wątkiem głównym, kopiowane są wartości wspomnianych zmiennych do pól obiektu AsyncTask. Następnie, w nowym wątku uruchamiana jest metoda doInBackground, w której następuje zapis skopiowanych danych do bazy SQLite – w tym momencie zmienne stanu aplikacji już są zresetowane przez główny wątek, w którym odbywa się normalna praca systemu.

Poniżej przedstawiono implementację oraz sposób wykorzystania dwóch obiektów AsyncTask.

Fragment kodu 4.5.5 przedstawia funkcję inicjującą nowe okrążenie. W przypadku rozpoczęcia pierwszego okrążenia, zlecany jest zapis nowej sesji. W przypadku kolejnych okrążeń zlecane są zapisy danych związanych z utworzoną wcześniej sesją.

Fragment kodu 4.5.6 przedstawia implementację klasy, która służy do stworzenia nowej sesji. W metodzie doInBackground wykonywane jest polecenie insert, a w metodzie onPostExecute aktualizowany jest obiekt zewnętrzny w oparciu o wynik doInBackground.

Fragment kodu 4.5.7 pokazuje implementację klasy do zapisywania danych z poszczególnych okrążeń. Pobranie potrzebnych danych z obiektu zewnętrznego odbywa się w metodzie onPreExecute, zapis do bazy danych w doInBackground.

```
private void newLap() {
        SQLiteDatabase db = LapMagicDatabaseHelper.getInstance(this)
                        .getWritableDatabase();

        if (lapNo > 0) {
                currentLap = getChronometerTime();
                restartChronometer();
                Toast.makeText(this, getString(R.string.run_session_lap_message)
                        + " " + lapNo + " - " + getFormattedTime(currentLap),
                        Toast.LENGTH_SHORT).show();

                if (currentLap < bestLap)
                        newBestLapTime(currentLap);

                (new SaveLapTask()).execute(db);
        }
        else {
                // start 1st lap and save the session
                restartChronometer();
                (new SaveSessionTask()).execute(db);
        }

        ++lapNo;
        clearSplitTimes();
        topSpeed = 0;
        lapNoView.setText(String.valueOf(lapNo));
        Toast.makeText(this, getString(R.string.run_session_lap_message) + " "
                + lapNo, Toast.LENGTH_LONG).show();
}
```

Fragment kodu 4.5.5 – funkcja inicjująca nowe okrążenie, zleca również zapis danych.

```
private class SaveSessionTask extends AsyncTask<SQLiteDatabase, Void,
    Long> {

        @Override
        protected Long doInBackground(SQLiteDatabase... db) {

                // get track name - a default for session name
                Cursor trackCursor = db[0].query(TracksTable.TABLE_TRACKS,
                        new String[] { TracksTable.COLUMN_NAME }, TracksTable.COLUMN_ID
                        + " = " + trackId, null, null, null, null);
                trackCursor.moveToFirst();
                String trackName = trackCursor.getString(trackCursor
                                .getColumnIndexOrThrow(TracksTable.COLUMN_NAME));
                trackCursor.close();

                ContentValues sessionValues = new ContentValues();
                sessionValues.put(SessionsTable.COLUMN_TRACK_ID, trackId);
                sessionValues.put(SessionsTable.COLUMN_NAME, trackName);

                Log.d(TAG, "inserting new session " + trackName + " in async
                    task");
                Long newSessionId = db[0].insert(SessionsTable.TABLE_SESSIONS,
                    null, sessionValues);
                return newSessionId;
        }

        @Override
        protected void onPostExecute(Long newSessionId) {
                // update member session id in the UI thread
                Log.d(TAG, "setting session id = " + newSessionId
                        + " after async task execution");
                sessionId = newSessionId;
        }

}
```

Fragment kodu 4.5.6 – `AsyncTask` odpowiadający za zapis w tabeli `Sessions`.

```
private class SaveLapTask extends AsyncTask<SQLiteDatabase, Void,
    Void> {

        private int taskLapNo;
        private long taskLapTime;
        private float taskTopSpeed;
        private long[] taskSplitTimes;

        @Override
        protected void onPreExecute() {
                // set up lap data in the UI thread
                taskLapNo = lapNo;
                taskLapTime = currentLap;
                taskTopSpeed = topSpeed;
                taskSplitTimes = splitTimes.clone();
        }

        @Override
        protected Void doInBackground(SQLiteDatabase... db) {
                Log.d(TAG, "Saving lap data: Lap No. " + taskLapNo
                   + ", lap time " + getFormattedTime(taskLapTime)
                   + ", top speed " + taskTopSpeed + " in async task");

                ContentValues values = new ContentValues();
                values.put(LapsTable.COLUMN_SESSION_ID, sessionId);
                ...
                db[0].insert(LapsTable.TABLE_LAPS, null, values);
                values.clear();

                values.put(SplitsTable.COLUMN_LAP_SESSION_ID, sessionId);
                values.put(SplitsTable.COLUMN_LAP_NO, taskLapNo);

                // skip the start point
                for (int i = startIndex + 1; i < points.size(); ++i) {
                        String splitName = points.get(i).getName();
                        ...
                        values.put(SplitsTable.COLUMN_SPLIT_NAME, splitName);
                        ...
                        db[0].insert(SplitsTable.TABLE_SPLITS, null, values);
                }
                return null;
        }

}
```

Fragment kodu 4.5.7 – `AsyncTask` odpowiadający za zapis w tabelach `Laps` oraz `Splits`.

4.5.4 Wsparcie dla starszych wersji Androida

System ma być kompatybilny ze starszymi, wciąż popularnymi (o których napisano w rozdziale 3) wersjami Androida. Nowe wersje zostały pozbawione niektórych elementów obecnych w tych starszych, natomiast starszym wersjom brakuje dopracowanych i bardzo

popularnych narzędzi młodszego Androida. O pewnych problemach z tym związanych wspomniano już w rozdziale 4.5.1.

Istotnym elementem, z którego aktualnie korzysta praktyczna każda aplikacja w systemie Android, jest pasek zadań – Action Bar, wprowadzony w wersji 3.0 systemu operacyjnego (Honeycomb). Korzystanie z paska zadań zamiast fizycznego przycisku „menu" na urządzeniu (którego wiele nowych modeli smartfonów już nie ma) jest jednym z najważniejszych zaleceń projektowych dla aplikacji na tą platformę.

W celu zapewnienia paska zadań skorzystałem z biblioteki ActionBarSherlock[8]. Jest to dodatek do oficjalnej biblioteki wsparcia, zawierającej pakiety systemowe, które są albo niedostępne dla starszych platform albo oferują nowe wersje API dla poszczególnych klas. ActionBarSherlock działa dynamicznie. Tworzy kompletny, w pełni funkcjonalny pasek zadań, jeśli aplikacja korzystająca z biblioteki jest uruchamiana na platformach starszych niż 3.0. Dla pozostałych wersji, biblioteka po prostu udostępnia programiście wbudowany pasek zadań. Testowy Samsung Galaxy S posiada Androida w wersji 2.1. W tym przypadku ActionBarSherlock spisał się poprawnie i zgodnie z oczekiwaniami.

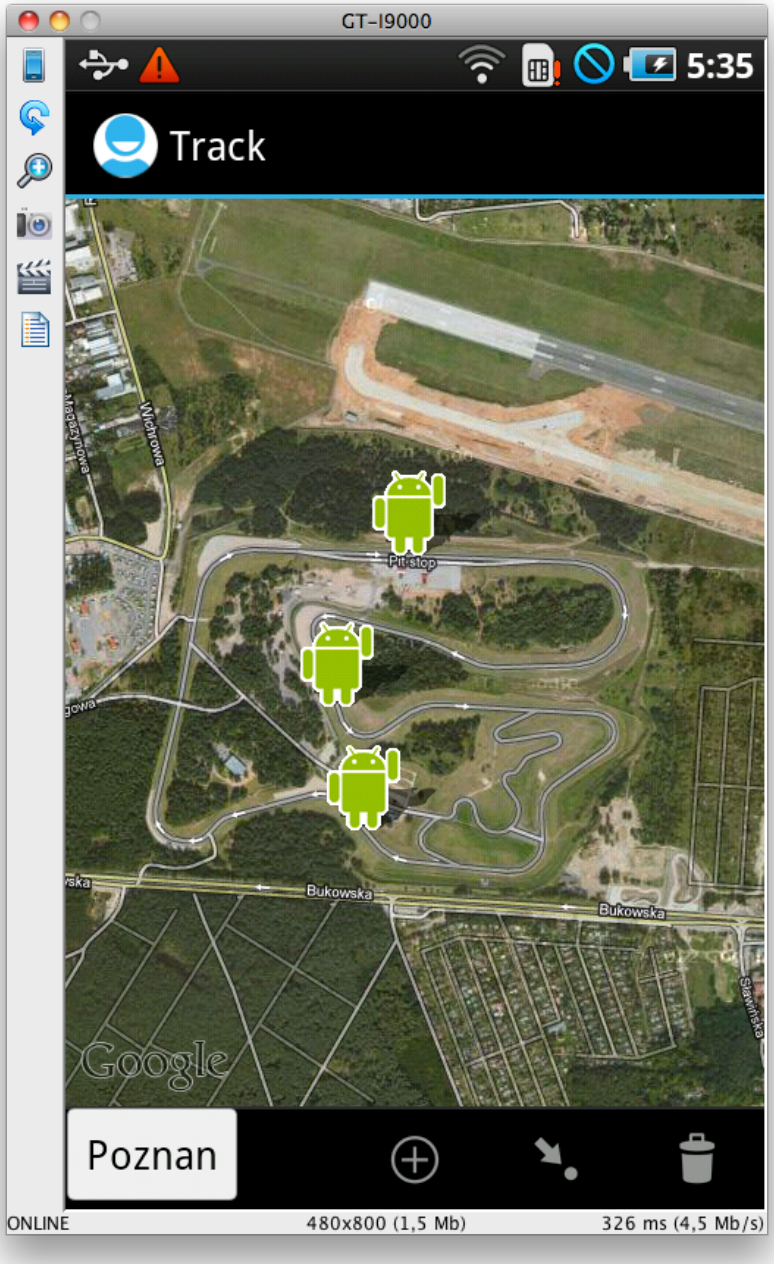
4.6 Przegląd interfejsu użytkownika

Poniżej przedstawiono kilka głównych widoków, z których składa się aplikacja. Zrzuty ekranu telefonu zostały zrobione przy pomocy programu Droid@Screen[9].

Rysunek 4.6.1 przedstawia widok na Tor Poznań wraz z punktami, które go definiują.

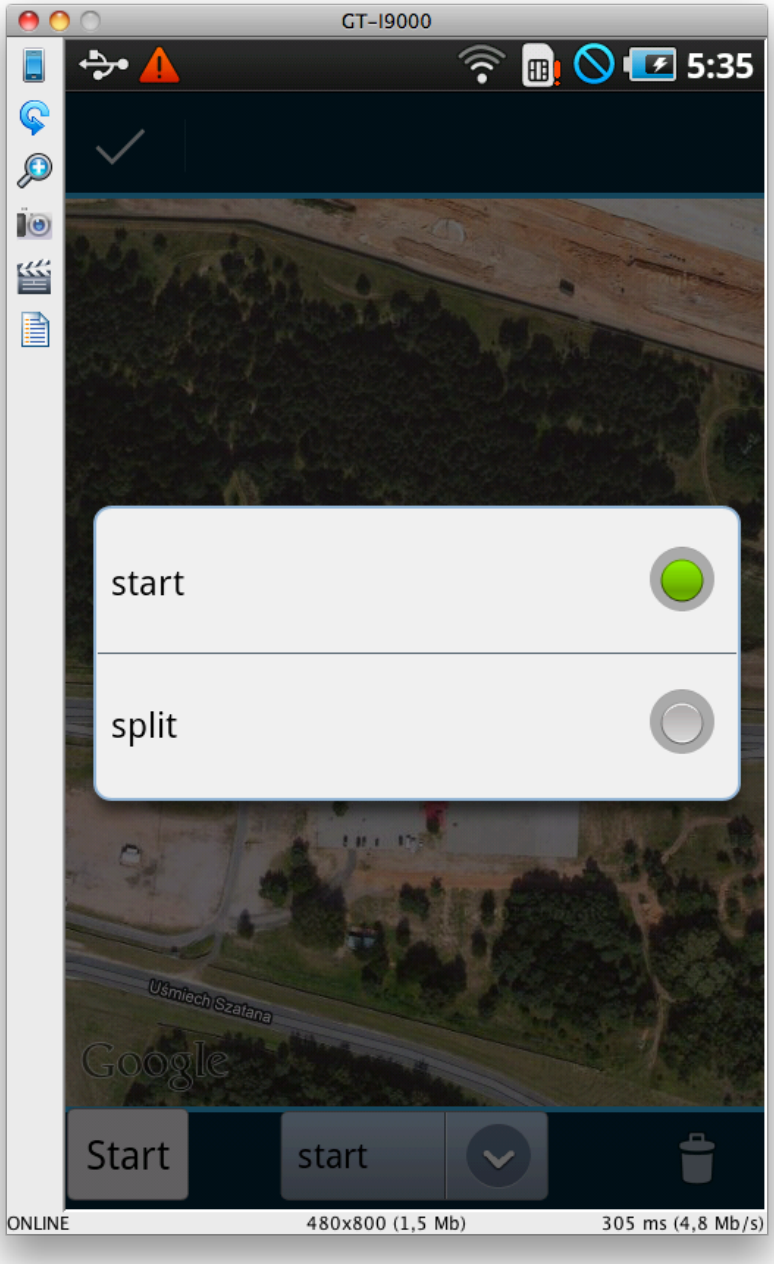
[8] Strona internetowa biblioteki – http://actionbarsherlock.com/.

[9] Droid at Screen – aplikacja, które przechwytuje obraz z telefonu i przekazuje go na ekran komputera, więcej informacji pod adresem http://droid-at-screen.ribomation.com/.



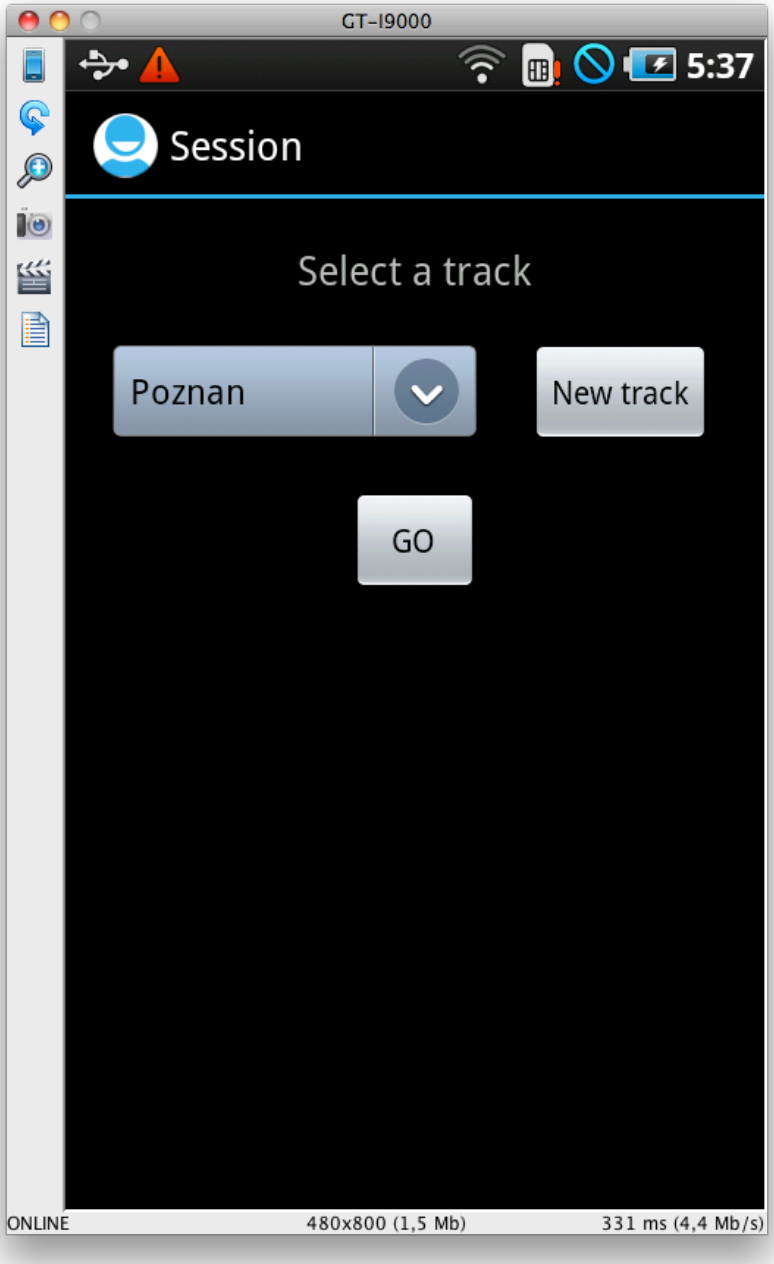
Rysunek 4.6.1 – „Tor Poznań" w ujęciu aplikacji *Lap Magic*.

Rysunek 4.6.2 pokazuje menu kontekstowe, aktywne po wybraniu zdefiniowanego punktu na mapie. Umożliwia ono edycję właściwości punktu.



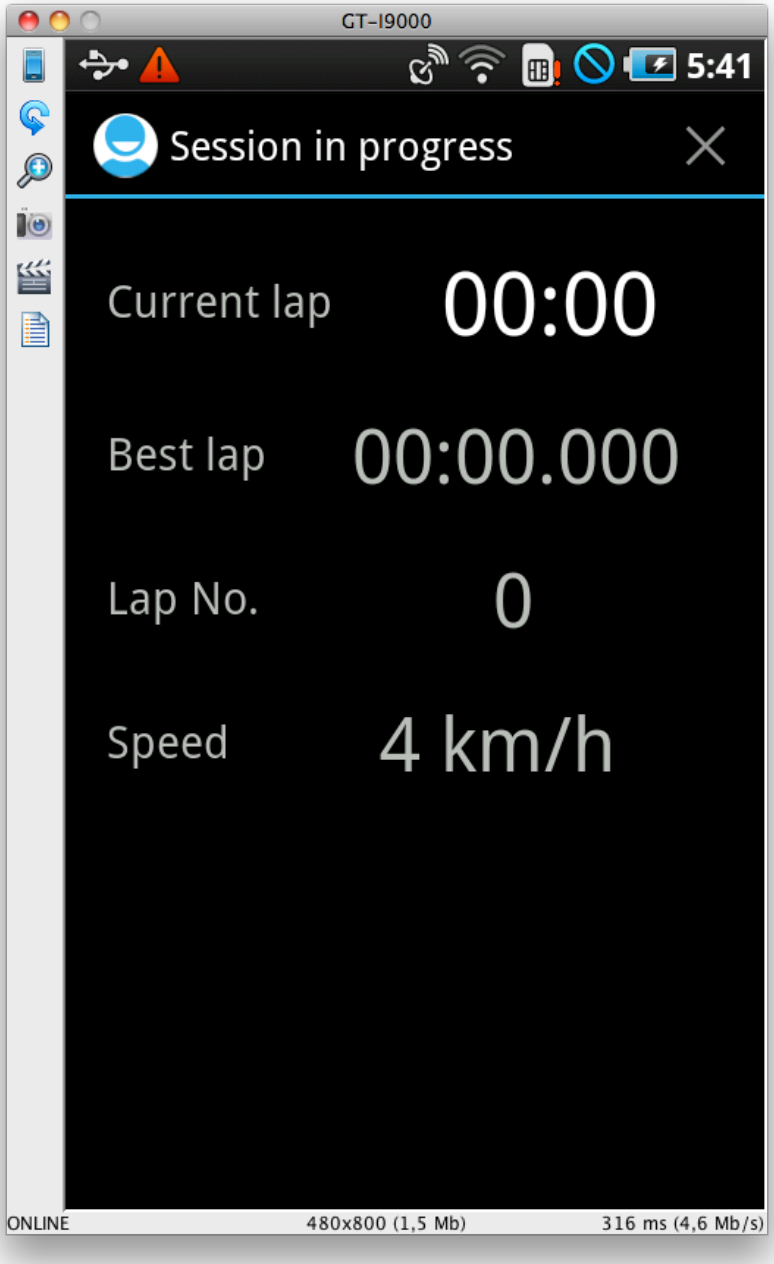
Rysunek 4.6.2 – menu umożliwiające edycje punktu.

Rysunek 4.6.3 przedstawia widok do przygotowywania sesji. Użytkownik ma możliwość wybrać tor i rozpocząć trening.



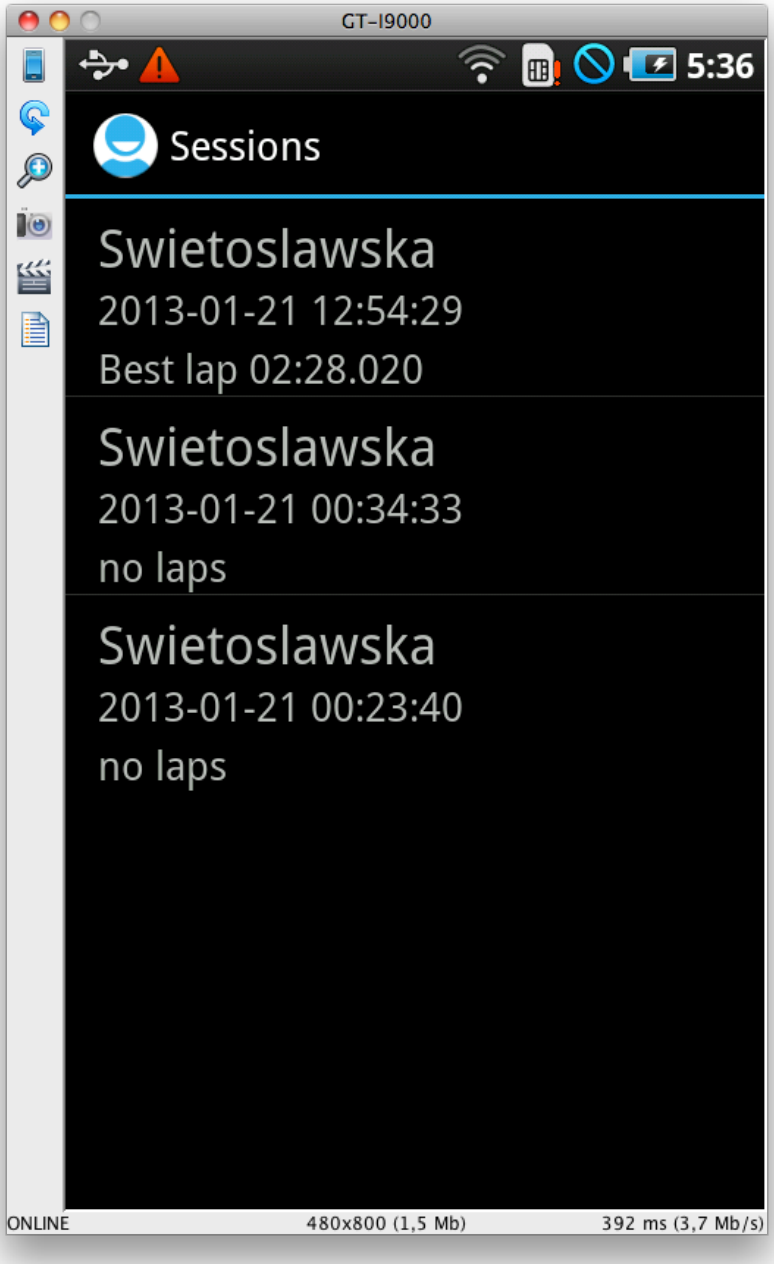
Rysunek 4.6.3 – widok umożliwiający wybór toru przed rozpoczęciem sesji.

Rysunek 4.6.4 pokazuje główny ekran w trakcie przebiegu sesji oraz ważne dane rejestrowane w jej trakcie.



Rysunek 4.6.4 – główny widok w trakcie trwania sesji.

Ten Rysunek 4.6.5 pokazuje przykładową listę zapisanych sesji, a dokładnie 3 sesje testowe, zebrane przeze mnie w trakcie testów.



Rysunek 4.6.5 – zapisane sesje.

5 Podsumowanie

5.1 Wnioski

Wynikiem niniejszej pracy jest darmowa aplikacja mobilna, która wykorzystuje system Android do realizacji systemu monitorowania zawodnika. Projekt jest open source, kod znajduje się do pobrania pod adresem http://code.google.com/p/lapmagic/. Udało się spełnić wymagania postawione w rozdziale 3. Android okazał się platformą dostarczającą to czego potrzeba w systemie o takim charakterze:

- rozbudowany framework, umożliwiający korzystanie z map, systemu GPS, wbudowanej bazy danych, przetwarzania współbieżnego

- możliwość rozszerzenia urządzenia mobilnego o dodatkowy sprzęt

Wykonane zostały podstawowe testy aplikacji, korzystając z samochodu, przy niezbyt dużych prędkościach (rzędu 70 km/h), na „torach" będących trasami biegnącymi przez ulice Warszawy. Efekty są zadowalające – aplikacja poprawnie reaguje na przejazdy przez zdefiniowane punkty. Ponadto wskazania prędkości z systemu GPS (szczególnie podczas przyspieszeń i hamowań) były zgodne ze wskazaniami prędkościomierza samochodu.

W ciągu kilku miesięcy planowane są testy na prawdziwych torach wyścigowych przy znacznie większych prędkościach – przekraczanie punktu pomiaru czasu może się odbywać przy 220 km/h lub nawet więcej. Przewiduję, że wbudowane czujniki GPS w smartfonach w takich warunkach nie spiszą się dobrze. Pomiary czasu będą niedokładne, a wręcz pojawią się błędy (np. możliwe „przegapienie" punktu pomiaru czasu). Zgodnie z symulacją opisaną w rozdziale 3.3, przy prędkościach przekraczających 200 km/h potrzebne jest odświeżanie pozycji na poziomie 10 Hz. W ten sposób jest możliwe rejestrowanie nowej pozycji co 6–8 m (najrzadziej co 10 m). To dobry wynik, podobnie jak hipotetyczna dokładność pomiaru czasu okrążenia wynosząca 0,1 s (dla czujnika 10 Hz).

Dlatego planuję wyposażyć smartfona w tego typu czujnik. Wymaganiem wobec takiego akcesorium jest łączność przez Bluetooth oraz wsparcie dla protokołu NMEA[1] (w chwili obecnej jest to standard i praktycznie nie ma czujników innego typu).
Znalazłem odpowiednie programowanie, które poddałem już wstępnym testom. Jest to program *BlueGps4Droid*[2]. Aplikacja podmienia standardowy systemowy serwis danych GPS na swój własny, który najpierw odkodowuje dane z zewnętrznego czujnika a potem w normalny sposób przesyła do innej aplikacji (w tym przypadku do *Lap Magic*). Gdy zdobędę czujnik, użyję tego oprogramowania. Jeśli stwierdzę, że nie działa zgodnie z oczekiwaniami, to mogę samemu zaimplementować serwis czytający dane GPS z wiadomości NMEA.

Testy na torze wyścigowym będą o tyle ciekawe, że będzie można odnieść wyniki aplikacji do oficjalnego systemu chronometrażu na torze, opartego o fotokomórki i odpowiednie kostki pomiarowe (tzw. transpondery). System taki wykazuje bardzo dużą dokładność (błąd jest niemal pomijalny) – takie systemy są wykorzystywane w profesjonalnych zawodach sportowych.

5.2 Rozwój systemu

W chwili obecnej rozważam dwie drogi rozwoju systemu.

Pierwsza droga zakłada wprowadzenie funkcji społecznościowych do aplikacji. Wynikami z danego toru można by się dzielić ze znajomymi. Można by nawet wchodzić z nimi w rywalizację on–line i tworzyć listy rankingowe dla poszczególnych torów. Rejestrowanie danych w każdej chwili od rozpoczęcia sesji daje też możliwość wykorzystania tej wiedzy podczas tworzenia interesujących filmów. Zawodnik posiadający kamerę na pokładzie pojazdu mógłby zsynchronizować obraz z kamery z zapisanymi lokacjami w czasie (wraz z prędkością) i w ten sposób wyświetlać na filmie przebieg swojego treningu wzbogacony o pozycję na torze w danej chwili.

[1] Protokół komunikacji między urządzeniami elektronicznymi służący do przesyłania danych GPS.

[2] http://sourceforge.net/projects/bluegps4droid/

Kolejna droga wiąże się z rozszerzeniem platformy sprzętowej. Android udostępnia możliwości przetwarzania danych z różnych sensorów. Gdyby posiadać czujniki, które logowałyby dane charakteryzujące bieżącą pracę silnika (np. obroty silnika, stopień otwarcia przepustnic, pewne stosunki gazów znajdujących się w układzie wydechowym), a następnie przesyłały je do urządzenia (np. przez Bluetooth), to w oparciu o Androida można by zrealizować swoisty komputer pokładowy z dodatkową funkcją mierzenia czasów okrążeń.

Jako autor pracy jestem zainteresowany jej rozwojem przede wszystkim wg drugiej drogi. Mam nadzieję, że znajdę odpowiednie źródła oraz uda mi się skontaktować ze specjalistami w dziedzinie elektroniki w pojazdach, dzięki czemu poszerzę swoją wiedzę na ten temat.

6 Bibliografia

[1] Dokumentacja systemu Android. http://developer.android.com/develop/index.html.

[2] Dokumentacja systemu iOS. https://developer.apple.com/devcenter/ios/index.action.

[3] The Global Positioning System. http://www.gps.gov/systems/gps/.

[4] Dokumentacja bazy danych SQLite. http://www.sqlite.org/.

[5] Dokumentacja Google Maps. https://developers.google.com/maps/.

[6] Biblioteka ActionBarSherlock. http://actionbarsherlock.com/index.html.