Univerzita Pardubice
Dopravní fakulta Jana Pernera

Spracovanie dát o polohe z GPS lokátora

Šimon Skotnický

Bakalárska práca

2019

Univerzita Pardubice
Dopravní fakulta Jana Pernera
Akademický rok: **2018/2019**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Šimon Skotnický**
Osobní číslo:	**D16206**
Studijní program:	**B3709 Dopravní technologie a spoje**
Studijní obor:	**Aplikovaná informatika v dopravě**
Název tématu:	**Zpracování dat o poloze z GPS lokátoru**
Zadávající katedra:	**Katedra informatiky v dopravě**

Zásady pro vypracování:

Cílem bakalářské práce je vývoj algoritmu zpracování, parametrizovaného uživatelem nebo automatizovaného, vyhlazování a filtrace dat o poloze z GPS lokátoru, a jeho implementace do aplikací.
V teoretické části práce budou popsány základní pojmy a principy lokace pomocí satelitního navigačního systému, a zpracování lokačních dat a jejich vizualizace na mapě.
V praktické části práce budou připraveny datové struktury, algoritmy pro filtraci naměřených bodů, a ve vhodném programovacím jazyce implementovány pro jejich grafickou prezentaci v podobě zobrazení na podkladu reálné mapy.

Rozsah grafických prací:

Rozsah pracovní zprávy: **30 normostran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **ŠEBESTA, Jiří. Globální navigační systémy. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2012. ISBN 978-80-214-4500-0.**
2. **RAPANT, Petr. Družicové navigační a polohové systémy: průvodce studiem. Ostrava: VŠB - Technická univerzita, Regionální centrum celoživotního vzdělávání, 2003. ISBN 80-248-0417-4.**
3. **CHAFFEE, James W. a Jonathan S. ABEL. The GPS filtering problem. In: EEE PLANS 92 Position Location and Navigation Symposium Record [online]. IEEE, 1992, s. 12-20 [cit. 2018-12-10]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=185813**
4. **Dokumentace API pro mapy**
5. **Dokumentace API pro GPS lokátory**
6. **RYANT, Ivan. Algoritmy a datové struktury objektově. V Praze: Ivan Ryant, [2017]. ISBN 978-80-270-1660-0.**

Vedoucí bakalářské práce: **Ing. Zdeněk Drvota**
Katedra informatiky v dopravě

Datum zadání bakalářské práce: **11. prosince 2018**

Termín odevzdání bakalářské práce: **24. května 2019**

L.S.

doc. Ing. Libor Švadlenka, Ph.D.
děkan

doc. Ing. Karel Greiner, Ph.D.
vedoucí katedry

V Pardubicích dne 11. prosince 2018

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích 24.5.2019 Šimon Skotnický

Anotácia

V tejto bakalárskej práci sa zabývame problematikou GPS lokátorov. Najskôr sme teoreticky popísali problematiku a potom vyvinuli funkcie v programovacom jazyku Javascript, ktoré dokážu vyhladzovať výsledky a z nepresných zhlukov bodov to vypočíta a nakoniec vykreslí Heat mapu. Poskytuje tak možnosť pre akýkoľvek komerčný software, aby na základe týchto funkcií zlepšili užívateľský komfort.

Kľúčové slová

GPS, mapy, Heat mapa, Javascript, stred súradníc

Title

Position data processing from GPS locator

Anotation

In this bachelor thesis we deal with the problematic of GPS locators. First, we theoretically described the problematics and then we developed functions in the Javascript programming language that can smooth out the results and calculate it from inaccurate cluster points and finally draw the heat map. It provides the opportunity for any commercial software to improve user comfort based on these features.

Keywords

GPS, Maps, Heat Map, Javascript, Coordinate Center

Obsah

Zoznam obrázkov

Úvod

Polohovacie zariadenia sa dnes používajú denne po celom svete. Využitie nájdu v prakticky všetkých častiach života. Využívame ich pri doprave, sledovaniu majetku, sledovaniu migrácie zvierat, na základe aktuálnej polohy vie software poskytnúť personalizované výsledky.

Na určenie polohy je možné použiť niekoľko systémov. Celosvetovo najpoužívanejší je americký systém GPS. Bohužiaľ žiadna polohovacia služba nie je úplne presná. Niekedy nám nevadí, že nám polohovacie zariadenie ukazuje našu polohu o pár metrov vedľa, ale niekedy by sme radi videli presnejší výsledok.

Je niekoľko možností na strane hardwaru alebo softwarovej vybavenosti polohovacích zariadení. Toto bežný užívateľ ale nedokáže ovplyvniť, takže sa pozrieme na možnosti, ktoré sa dajú užívateľom poskytnúť bez toho, aby musel meniť svoj hardware. Obzvlášť nápomocné to môže byť v prípade, že GPS prijímač stojí na mieste a stále prijíma a lokalizuje zariadenie. Okolo takého zariadenia potom vznikne neprehľadný zhluk bodov, ktorý vypadá akoby sa zariadenie pohybovalo sem a tam, pričom ale stojí na mieste. Tieto zhluky potom spojíme do jedného bodu alebo do jedného ohraničeného polygónu a dokonca zobrazíme Heat mapu, ktorá bude určovať kde najpravdepodobnejšie zariadenie je.

1 Teoretický popis GPS

1.1 Čo je GPS

GPS je známa skratka pre anglickú skratku Global Positioning System, voľne preložené Globálny Polohovací Systém. Niekedy tiež označované ako NAVSTAR.

Je to pasívny diaľkomerný systém pre stanovanie polohy a času na Zemi alebo aj v jej blízkom okolí. Nonstop poskytuje signály, ktoré prijímajú a spracovávajú GPS prijímače [1].

Obrázok 1 - GPS satelit

Zdroj https://www.lockheedmartin.com/en-us/products/gps.html

1.2 Presnosť

1.2.1 Presná Polohová Služba

GPS poskytuje signály v 2 úrovniach presnosti. Presná polohová služba poskytuje takmer dokonale presnú polohu a čas. Bohužiaľ je dostupná iba pre autorizované zariadenia, ktoré majú k dispozícii iba vojenské zložky USA, NATO a prípadne ďalších krajín.

1.2.2 Štandardná Polohová Služba

Druhou a pre bežných ľudí aj bežnejšou službou je Štandardná Polohová Služba. Tá je dostupná bez nutnosti autorizácie všetkým užívateľom a zariadeniam na celom svete.

Presnosť za dobrých podmienok dosahuje spravidla 2-3 metre.

1.2.3 Kedy sa GPS nedá použiť

Medzi nevýhody GPS patrí jej veľká nepresnosť až úplná nemožnosť použiť v podzemí, v budovách, v hustej zástavbe alebo lesoch. Dôvod je celkom prostý. Zariadenie, ktoré sa snaží prijímať GPS signál ho nemá k dispozícii, inak povedané nevidí na žiadny vysielací satelit. Druhá možnosť je, keď sa signál odráža od okolitých objektov, čo spôsobuje nepresnosti.

1.3 Družice a vysielané frekvencie

Družice vysielajú signály v niekoľkých rôznych frekvenciách. Rozlišuje sa až 5 frekvencií: L1, L2, L3, L4 a L5 [9]. Popíšeme prvé dve z nich:

- L1
    - základný signál o frekvencii 1575,42 MHz. Signál obsahuje 2 kódy
    - P(Y) kód, z anglického Precision, ktorý je pre vojenské účely navyše zašifrovaný
    - Druhý je C/A kód, podľa anglického vzoru Coarse/Acquisition. Tento kód nie je šifrovaný a môže ho použiť ktokoľvek. Aj tento signál je dosť presný a bol až do 1.5.2000 zámerne znespresňovaný, takže udával iba orientačnú polohu aj čas. GPS prijímač bol pôvodne schopný určiť svoju polohu s presnosťou na až 100m, dnes to už sú rádovo iba 3m
- L2
    - Vysiela signál o frekvencii 1227,62 MHz. Táto frekvencia je tvorená iba P(Y) kódom.

1.4 Zlepšovanie výsledku

Pokiaľ nie je zariadenie v oblasti bez signálu, vidí na hneď niekoľko satelitov a prijíma tak viac signálov naraz. Lepšie zariadenia dokážu prijať všetky dostupné signály a následne ich spriemerovaním spresniť výslednú polohu.

1.4.1 DGPS

Diferenciálne GPS, v skratke DGPS je technika používaná na ďalšie zlepšovanie presnosti. Do bodu o známych súradniciach sa umiestni referenčná stanica GPS. Tá porovnáva skutočnú a nameranú hodnotu polohy alebo času a vysiela signál s informáciou o týchto rozdieloch/diferenciách. Prijímacie zariadenie potom použije tieto diferencie na vlastné lepšie spočítanie svojej polohy a času. Výsledná presnosť s použitím DGPS sa môže zvýšiť až na jednotky centimetrov.

1.5 Vznik

Prvá experimentálna družica bola vypustená v roku 1978. Celosvetovo plne funkčný sa stal 8.12.1993, kedy bol dosiahnutý stav Initial Operational Capability (IOC). IOC znamená, že je v prevádzke 18 plne funkčných družíc.

Stav Full Operational Capability bol vyhlásený 17.7.1995. V praxi to znamená, že okolo zeme obiehalo 24 plne funkčných družíc typu Blok II a Blok IIA.

Odvtedy sa GPS sa nepostrádateľným nástrojom pre určenie polohy, času a navigáciu na celom svete. Pomáha pri tvorbe máp a v zememeračstve.

Systém vyvinulo Ministerstvo obrany USA pod oficiálnym názvom Navigation Signal Timing And Ranging GLobal Positionig System. Skrátene NAVSTAR GPS.

1.5.1 História

Prvý druživcový systém bol v USA prvý krát spustený v roku 1960. Systém obsahoval 5 družic a bol schopný určiť polohu prilbližne raz za hodinu. O 7 rokov neskôr v USA vypustli prvý krát družicu, ktorá má vo svojom vybavení presné hodiny. Nazvali ju príznačne Timation. Prvým celosvetovým rádiovým navigačným systémom bol systém Omega, ktorý pracoval na princípe porovnávania fází signálu. Do prevádzky sa dostal postupne v sedemdesiatych rokoch minulého storočia.

Na podobnom princípe, ako je dnešné GPS, fungoval už v roku 1957 Sovietsky Sputnik. Vedci už v tej dobe prišli na to, že s použitím známej obežnej dráhy družice a Dopplerovho efektu, kedy je frekvencia vysielaného signálu vyššia, keď sa približuje a naopak nižšia, keď sa oddiaľuje, môžu spoľahlivo určiť jej polohu.

Prvá experimentálna GPS družica Bloku I (BLock-I GPS) bola vypustená vo februári 1978. Prvé družice vyrobila spoločnosť Rockwell International. Neskôr výroba prešla pod spoločnosť Lockheed Martin [2].

1.5.2 Časový prehľad

Za sprístupnenie GPS aj civilnému použitiu môžeme paradoxne poďakovať nešťastnej udalosti, pri ktorej zahynulo 269 ľudí. V roku 1983 sa totiž stalo, že kórejské civilné lietadlo vletelo do Sovietského zakázaného vzdušného priestoru a bolo zostrelené. Americký prezident Ronald Reagan potom oznámil, že GPS sprístupní aj civilnému použitiu, aby sa taká udalosť už nikdy neopakovala.

V roku 1985 bolo vypustených ďalších 10 družíc Bloku I. Modernejšia družica Block-II bola vyspustená až o 4 rokov neskôr, presne 14.2.1989.

V roku 1992 sa zmenila organizačná štruktúra riadenia GPS, keby bol zrušený „2nd Space Wing“ a bol nahradený „50th Space Wing“.

Behom roku 1993 bol dosiahnutý minimálny počet družíc na obežnej dráhe zeme na určenie času a polohy kdekoľvek na zemi a následne 17. januára 1994 bolo na obežnej dráhe prvý krát 24 družíc.

V roku 1996 ďalší americký prezident, Bill Clinton, oficiálne uznal dôležitosť GPS pre civilný aj armádny sektor a vydal smernicu, v ktorej GPS definuje ako systém dvojakého použitia. Tiež založil Správny orgán GPS, v originálne Interagency GPS Executive Board, pre správu GPS ako národného majetku.

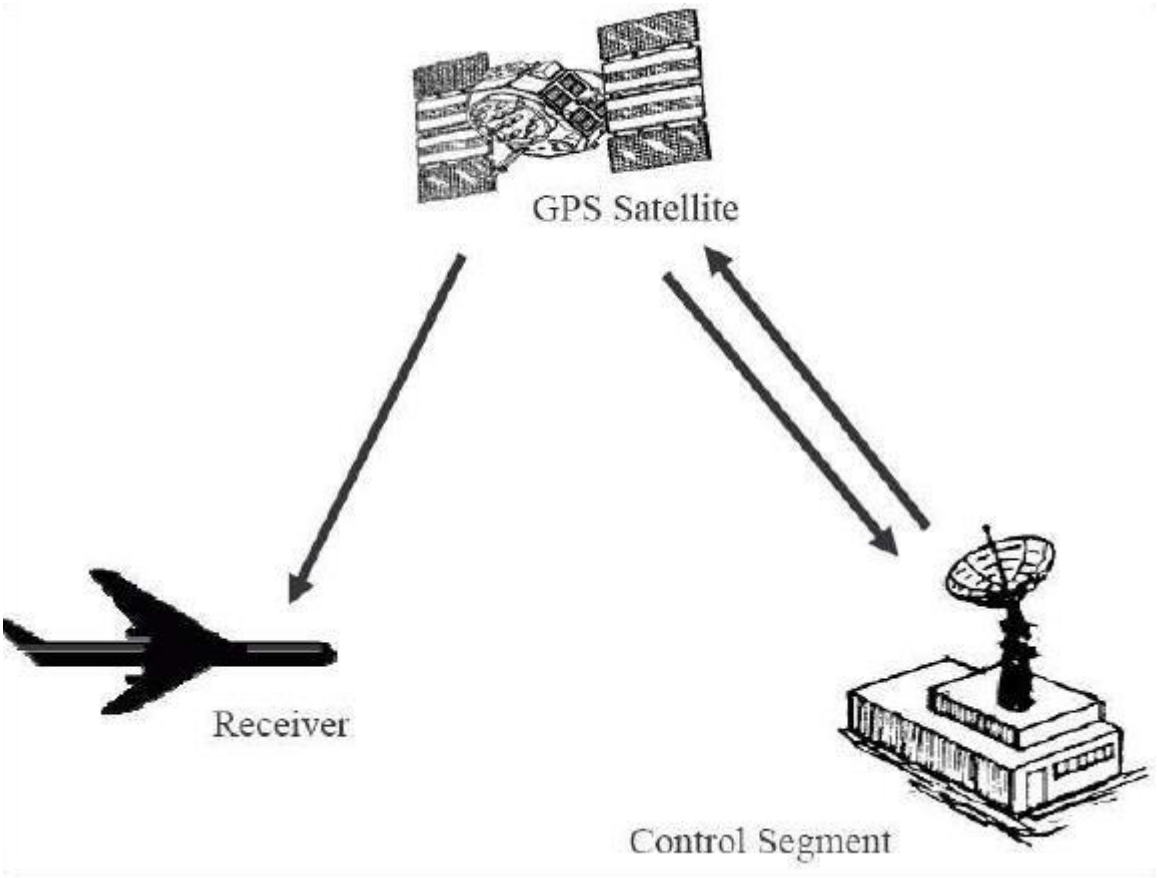
V roku 1998 viceprezident Al Gore oznámil plán na modernizáciu, a to pridaním 2 civilných signálov pre zlepšenie presnosti aj spoľahlivosti, obzvlášť kvôli leteckej bezpečnosti.

Konečne 2.5.2000 bola vypnutá tzv. „Selective Availability“, teda zámerné rušenie a znepresňovanie signálu. Dramaticky sa tak zvýšila presnosť.

Neskôr, v roku 2004 americký prezident George W. Bush nahradil Správny orgán GPS iným orgánom. Nahradil ho National Space-Based Positioning, Navigation, and Timing Executive Committee [2].

1.6 Segmenty

Celý systém GPS môžeme rozlíšiť do niekoľkých častí, takzvaných segmentov. Do jednej zahrnieme satelity, ktoré vysielajú signál. Druhá časť je riadenie a kontrolovanie satelitov a nakoniec tretia, čo bude využitie GPS zariadeniami po celom svete [2].



Obrázok 2 – Segmenty

Zdroj https://www.vectornav.com/support/library/gps

1.6.1 Kozmický segment

Projekt bol projektovaný na 24 družíc, dnes sa ich ale používa maximálny počet 32. Viac sa ich pri súčasnej softwarovej výbave nedá použiť, pretože pre číslo satelitu je určených iba 5 bitov. A pretože 2^5 je práve 32, viac ich teraz nejde použiť.

Družice obiehajú po výške 20 350 km nad zemským povrchom na šiestich kruhových dráhach so sklonom 55°. Dráhy sú vzájomne posunuté o 60° a na každej dráhe boli pôvodne rozmiestnené pravidelne 4 družice. Dnes ich tam je už 5-6 v nepravidelných intervaloch.

Družica váži 1,8t a obieha na strednej obežnej dráhe. Pohybuje sa rýchlosťou 38km/s a okolo zeme obehne za 11hod 58min, čo je presne polovica hviezdneho času.

Plánovaná životnosť jednej družice je 10 rokov. V priebehu tohto času sú niekoľko krát odstavené kvôli údržbe atómových hodín a korekcie obežnej dráhy.

Súčasti každej družice

- Atómové hodiny
- 12 antén RHCP pre vysielanie rádiových kódov v pásmu L 2000–1000 MHz
- antény pre komunikáciu s pozemnými kontrolnými stanicami v pásmu S (2204,4 MHz)
- antény pre vzájomnú komunikáciu družíc v pásmu UHF
- optické, röntgenové a pulzno-elektromagnetické detektory odhaľujúce štarty balistických rakiet a jadrové výbuchy (IONDS)
- solárne panely a batérie

Aktuálne najnovší typ GPS satelitu je GPS III. Prvý satelit tohto typu bol vynesený na obežnú dráhu 23.12.2018 raketou Falcon 9, ktorú vyvinula spoločnosť SpaceX [10].

1.6.2 Riadiaci segment

Riadiaci segment GPS systému monitoruje a koriguje činnosť družíc. Upravuje atómové hodiny a nahráva do družíc dáta o navigácii. Vydáva správy o každej družici, v akom je stave, aká je jej predpokladaná trajektória a stav atómových hodín. V prípade výpadku činnosti pozemných riadiacich staníc by GPS fungovala v nezmenenej podobe ešte niekoľko mesiacov. Potom by postupne začalo dochádzať ku znižovaniu presnosti, pretože hodiny ani trajektória by neboli kalibrované. Po ďalších mesiacoch až rokoch by začalo dochádzať ku výpadkom a väčším nepresnostiam. Táto situácia nebola a pravdepodobne ani nebude nikdy otestovaná, takže čo by sa naozaj stalo, dnes nikto s určitosťou povedať nevie.

1 Veliteľstvo

Hlavné veliteľstvo „Navstar Headquarters“ je na leteckej základni Los Angeles v Kalifornii, USA.

2 Riadiace Stredisko

Riadiace stredisko, po anglicky Master Control Station je na Schrieverovej leteckej základni USAF v Colorado Springs.

Existuje aj záložné riadiace stredisko Backup Master Control Station sa nachádza v Gaithersburgu, štát Maryland v USA. Záložné stredisko 4x ročne preberie riadenie systému a v prípade potreby je schopné byť v plnej prevádzke do 24hod.

3 Povelové stanice

Povelové stanice, alebo tiež Ground Antenna, sú umiestnené na 3 základniach USAF po svete.

4 Monitorovacie Stanice

Monitorovacích staníc je celkom 18. Nachádzajú sa na vojenských základniach USAF po celom svete.

GPS prijímače a ich užívatelia sa delia na 2 základné skupiny:

1.6.3 **Užívateľský segment**

Naivná predstava fungovania GPS napríklad v mobile je, že mobil vyšle signál k satelitu, ten mu spočíta polohu a pošle naspäť. Takto to ale nefunguje, všetky GPS zariadenia sú pasívne prijímače.

Prijímač prijíma signály signály zo všetkých družíc, ktoré sú v danú chvíľu nad obzorom. Na základe prijatých dát je zariadenie schopné spočítať presnú polohu, dátum a čas.

Určovanie polohy funguje v princípe tak, že zariadenie porovnáva čas vyslania signálu z družice s časom prijatia signálu prijímačom. Keď prijíma signál z aspoň 4 družíc, dokáže spočítať svoju polohu.

Obrázok 3 - GPS prijímač

Zdroj https://www.vectornav.com/support/library/gps

Prijímače sa delia podľa niekoľkých kritérií.

Podľa frekvenčných pásiem:

- Jednofrekvenčné
- Dvoufrekvenčné
- Viacfrekvenčné

Podľa kanálov:

- Jednokanálové
- Viackanálové

Podľa princípu výpočtov polohy a času:

- Kódové

- Fázové a kódové

Bežné civilné prijímače sú jednofrekvenčné, jednokanálové, viackanálové a kódové. Každý prijímač sa skladá minimálne z týchto súčasní:

- Anténa
- Predzosilňovač
- Procesor
- Časová základňa
- Komunikačné rozhranie

GPS prijímače a ich užívatelia sa delia na základné skupiny:

- Autorizovaní – používajú hlavne armády USA a NATO. K dispozícii majú tzv. Precise Positioning Service, teda presnú polohovaciu službu. Títo užívatelia majú zaručenú vyššiu presnosť systému
- Neautorizovaní – toto je hlavne bežný civilný komerčný sektor. Používajú Standard Positioning Service

1.7 Súradnicový systém

Základným systémom je geocentrický súradnicový systém na elipsoidu WGS84 (World Geodetic System 1984), ktorý poskytuje údaje v tvare zemepisnej dĺžky a šírky. Je to celosvetovo uznávaný štandard, ktorý vydalo americké ministerstvo obrany práve v roku 1984. Polohu určuje pomocou zemepisnej dĺžky a šírky. Naberajú rôzne hodnoty podľa toho, kde na zemi sa nachádzame [3]:

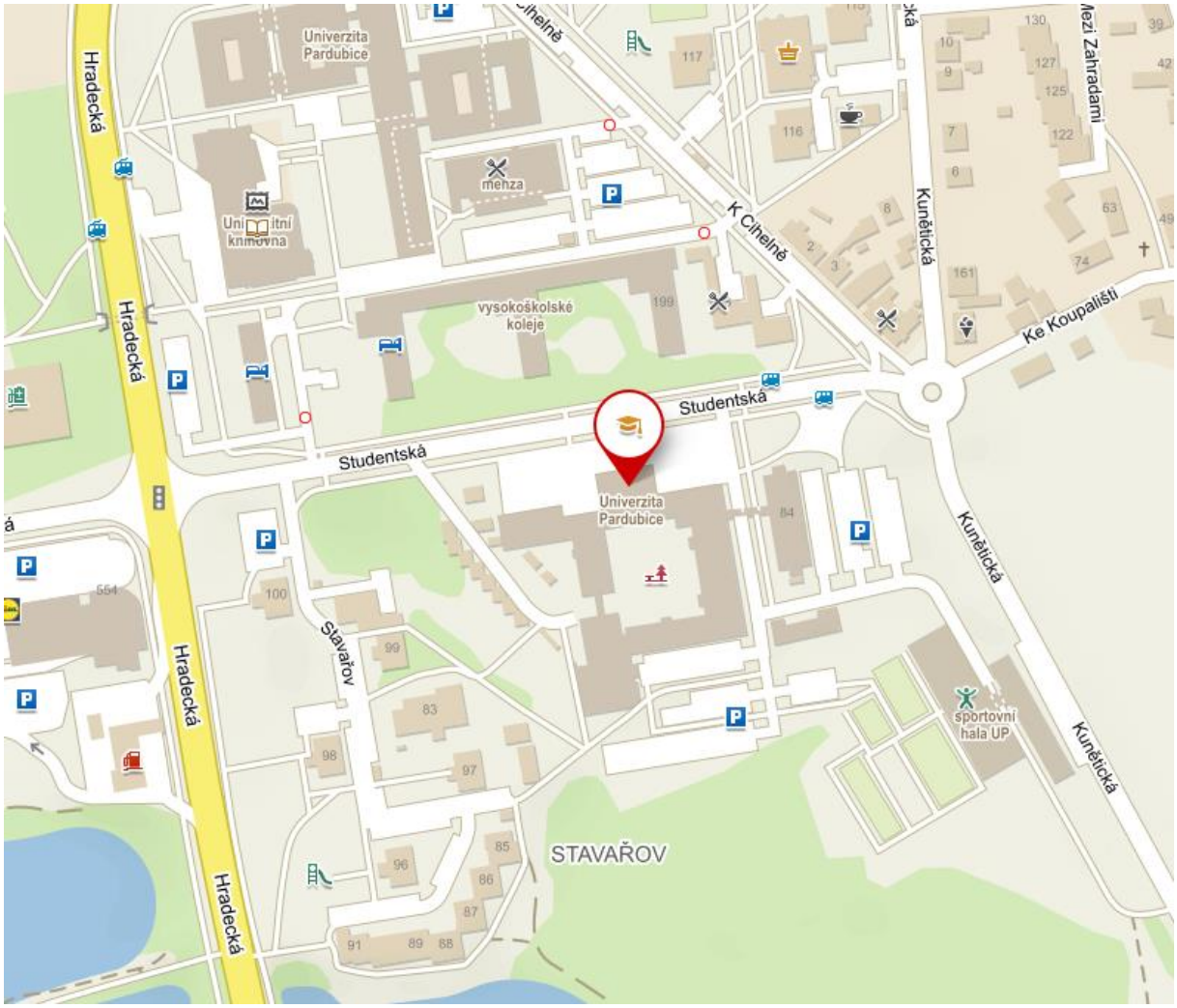
- Šírka má hodnoty
    - 0 – -90°na juh od rovníka
    - 0 – 90° na sever od rovníka
- Dĺžka má hodnoty
    - 0 – -180° na západ od nultého poludníka
    - 0 – 180° na východ od nultého poludníka

Príklady rôznych zápisov súradníc budovy rektorátu Univerzity Pardubice:

- WGS84 stupne: 50.0481606N, 15.7693950E
- WGS84 stupne a minúty: N 50°2.88963', E 15°46.16370'

- WGS84 stupne, minúty a sekundy: 50°2'53.378"N, 15°46'9.822"E

Existujú aj ďalšie systémy súradníc, napr. UTM, S-42, OLC, MGRS. Tie ale nie sú tak bežné a teda sa nimi ani nebudeme ďalej zaoberať. V Československu sa používal aj súradnicový systém S-JTSK, ktorý je postavený na tzv. Křovákovom zobrazení [15].



Obrázok 4 - UPCE na mape

Zdroj https://mapy.cz/

1.7.1 Prevod medzi súradnicami

Prevod medzi jednotlivými formátmi je pomerne jednoduchý. Zo stupňov, minút a sekúnd urobíme iba stupne týmto výpočtom:

$$stupne2 = stupne1 + \frac{minúty1}{60} + \frac{sekundy1}{3600}$$

V opačnom smere je postup trochu zložitejší. Najskôr spočítame stupne. Na to stačí zobrať jednoducho celú časť. Povedzme, že funkcia Round(x) je celá časť čísla x. Potom ak chceme spočítať stupne1°minúty1'sekundy1'', tak budeme postupovať nasledovne:

$stupne1 = Round(stupne2)$

$$min\acute{u}ty1 = Round((stupne2 - stupne1) * 60)$$
$$sekundy1 = (stupne2 - \ stupne1 - min\acute{u}ty1) * 3600$$

1.8 Praktické využitie

Pripomeňme, že pre civilné účely má GPS presnosť jednotky metrov a všetky zariadenia sú iba pasívne prijímače signálu. Možných využití je nespočetne veľa, vymenujeme si aspoň pár z nich:

1.8.1 Navigácia v automobilovej doprave

Keď som doma babke povedal, že píšem bakalárku o GPS, začala mi hovoriť historku ako kamionista zablúdil do hôr, pretože ho tam naviedla GPS. Značí to o tom, že pre navigačné systémy v automobilovej doprave zľudovateľ samotný názov GPS. Je to nepresné, ale množstvo ľudí to tak používa.

S použitím navigácie odpadá pri cestovaní nutnosť sledovať mapu, je nižšia pravdepodobnosť zablúdenia a systém ponúka najkratšiu, najrýchlejšiu alebo najlacnejšiu (z pohľadu diaľničných poplatkov) trasu.

1 Ostatné navigačné systémy

Samozrejme sa nemusí jednať iba o automobilovú dopravu. Navigovať sa môže nechať turista, keď zdoláva krásy prírody. GPS pomáha s navigáciou v leteckej aj námornej doprave. Skrátka všade.

1.8.2 Personalizácia reklám & implicitný jazyk aplikácií

Akýkoľvek používaný software, či už ide o mobilnú alebo desktopovú aplikáciu, o internetovú stránku alebo celý operačný systém, môže sledovať našu polohu. Obvykle to robí na základe

IP adresy alebo wifi sietí. Kvôli zvýšeniu presnosti ale môže použiť GPS prijímač, ak je ním zariadenie vybavené.

V tom prípade sa poloha odošle napríklad na server internetovej stránky a tá môže následne inzerovať služby a produkty, ktoré sú v bezprostrednom okolí používateľa. Ďalšie možné využitie je napríklad pre taxislužby. Ak má taxi aplikáciu, používateľ nemusí zadávať svoju polohu, pretože tá sa odošle automaticky.

Niektoré veľké celosvetovo používané programy môžu už pri inštalácii ponúknuť jazyk, ktorý bude daná aplikácia využívať práve na základe aktuálnej polohy. Určite málokto odcestuje do zahraničia na dovolenku, aby si tam inštaloval napríklad Windows, internetový prehliadač, balík Microsoft Office, emailový klient alebo čokoľvek iné.

1.8.3 Sledovanie dravcov

GPS sa využíva aj pri sledovaní dravých vtákov. Dravec sa odchytí, pripevní sa mu na nohu alebo telo GPS prijímač a pustí sa do sveta. Sledujú sa potom migračné trasy a vzorce správania.

Používa sa to zatiaľ iba na väčšie dravce, pretože zatiaľ nie je možné vyrobiť dostatočne malé a ľahké zariadenie, ktoré by uniesli aj menšie vtáky.

Do jedného sledovacieho zariadenia musí zmeniť Nano SIM karta, ktorá pravidelne odosiela svoju polohu, malá baterka, solárny panel, procesor a samozrejme plastová krabička, do ktorej sa to všetko zabalí.

Obrázok 5 - Dravec so sledovacím zariadením

Zdroj https://anitra.cz

1.9 Alternatívy

Alternatív ku GPS je celá rada. GPS je americký systém, ktorý je aktuálne v našich zemepisných šírkach najpoužívanejší. Vlastný systém Galileo vyvíja Európska Únia, svoj Glonass má Ruská Federácia a nezabudnúť nesmieme ani Čínu, ktorá má svoj systém Compass [11].

V prípade vojenského konfliktu sa na tieto systémy aj tak nedá spoľahnúť, pretože signál je možné jednoducho vyrušiť.

1.9.1 Glonass

Glonass je ruský polohovací systém, založený na podobnom princípe ako GPS. Začal vznikať ešte v roku 1976 v vtedajšom Sovietskom zväze. Počas politicky aj ekonomicky búrlivých rokoch sa na systéme nijak nepracovalo, aby na začiatku tretieho tisícročia zase vstúpil do hry.

Rozdiel oproti GPS je hlavne v kozmickom segmente. V prípade GPS sa každá družica vráti na to isté miesto každý hviezdny deň. V prípade Glonassu to je trochu inak. Tam sa na miesto

jednej družice po jednom hviezdnom dni vráti iná družica. Volá sa to neidentické opakovanie [4], [11].

1.9.2 Galileo

Galileo je polohovací systém, ktorý vyvíja a prevádzkuje Európska Únia. Oproti Glonass a GPS sú jeho satelity najvyššie, obiehajú vo výške 23 222 km. To je o 3 000 km viac ako GPS a o 4 000 km viac ako Glonass.

Galileo nie je primárne vyvíjaný ako vojenský systém, ale komerčný. Aj pre bežných užívateľov sľubuje určenie polohy s presnosťou menšou ako jeden meter a za úplatu potom ešte presnejšie.

Pôvodne plánovaný rok spustenia bol 2010. V súčasnosti Galileo stále nefunguje v plnej prevádzke, ale iba v obmedzenom režime. Na obežnej dráhe je vynesených 18 satelitov z plánovaných 32 a ešte ho nepodporujú všetky prijímače a ešte určite potrvá, kým budú [5], [11].

1.9.3 Mobilné a Wifi siete

Mobilné a Wifi v dnešnej dobe vedia poskytovať alebo spresňovať polohu. Ich problém je, že pokrytie signálom nie je 100% a v husto zastavaných oblastiach vzniká rovnaký problém, ako pri GPS. Vysielač a prijímač na seba dobre nevidia, dochádza ku skresleniu signálu a následne aj polohy.

Jednou z možných metód, ako sa toto dá riešiť, je metóda DCM. Tá predpokladá, že práve v husto zastavaných oblastiach je v každom mieste sú vlastnosti signálu jedinečné. Pomocou iných metód sa zistí aktuálna poloha a tá sa pošle operátorovi, ktorý ju uloží do databáze. Keď potom ďalší krát požiada zariadenie o svoju polohu, na základe vlastností signálu mu bude operátorom vrátená jeho poloha. Nevýhoda systému spočíva napr. v náročnej infraštruktúre na strane operátora alebo aj nepresností, ktoré vzniknú v dôsledku stavebných prác [6].

1.9.4 Radar

Určiť polohu zariadenia je možné aj pomocou rádiových vĺn, ktoré vysiela radar. Tu je ale základný problém, že je možné zistiť polohu okolitých objektov vzhľadom ku polohe radaru, ale nie polohu samého seba [7].

Na princípe radaru funguje napríklad systém RTLS, čo znamená Real Time Location System, voľne preložené lokalizačný systém v reálnom čase. Medzi jeho výhody patrí, že dokáže nonstop určiť veľmi presne polohu hľadaného zariadenia. Nevýhoda je, že hľadané objekty musia byť vybavené špeciálnym hardwarom, ktorý reaguje na rádiové vlny. Rovnako musí byť daná oblasť pokrytá signálom a mimo túto oblasť lokalizácia nefunguje. Tento systém nájde využitie napríklad na veľkých skladoch vozidiel, kde každé vozidlo dokážeme vybaviť prijímačom a zároveň dokážeme pokryť signálom celé parkovisko/sklad [8].

Obrázok 6 - Radar

Zdroj https://medium.com/virtuslab/i-read-last-thoughtworks-technology-radar-vol-18-so-you-dont-need-to-b08db0a79997

2 Praktická časť

2.1 Vyhladzovanie odchyliek a nepresností

Ako už bolo popísané, GPS nie je úplne presné. Najviac to je vidieť, ak GPS prijímač stojí na jednom mieste. Pri zobrazení bodov do mapy je potom vidieť, že okolo zariadenia sa vytvorí akýsi zhluk bodov.

Užívateľ, ktorý také body vidí, je potom zmätený kde vlastne je a môže vyvodiť nesprávne dôsledky. Napríklad pri turistike môže zvoliť nesprávnu cestu. Nevýhoda je aj pri spätnej kontrole cesty. Modelová situácia môže byť bežec, ktorý chce po tréningu zdieľať svoju trasu na sociálne siete. Obzvlášť v mestskom prostredí sa môže stať, že na chvíľu musí počas behu zastaviť, napríklad na semafore. V trase potom vznikajú zhluky bodov a to nevyzerá dobre.

Pre takéto prípady by sa hodilo, keby sa tieto zhluky bodov označili a vypočítal sa ich stred. Minimálne by mohol mať užívateľ možnosť označiť body, ktoré sú takpovediac duplicitné. Ešte lepšie by bolo, keby sa to všetko dokázalo urobiť automaticky. Ďalej v tejto práci sa pokúsime pripraviť a ukázať presne túto funkcii.

2.2 Druhy stredov

V prvom rade si musíme povedať, čo je vlastne stred geografických súradníc. Máme niekoľko možností:

- Prvá a najjednoduchšia cesta, ako počítať stred je spočítať jednoducho aritmetický priemer Zemepisnej šírky a dĺžky. Toto je rýchla cesta, ktorá dá základnú predstavu o strede, ale jej zásadná nevýhoda je, že neberie do úvahy zakrivenie zemského povrchu. Pre malé vzdialenosti by tu bol rozdiel naproste minimálny, pravdepodobne ešte menší než je chyba zaokrúhľovania súčasnej výpočetnej techniky, ale pri veľkých vzdialenostiach, rádovo stovky kilometrov, by tu bol už rozdiel vidieť.
- Keby sa počítal stred 3 bodov, mohlo by sa použiť myšlienku, že 3 body predsa určujú kružnicu a keď stred jednotlivých bodov by potom bol stred kružnice. Táto úvaha bohužiaľ padá už na tom, že kružnicu je síce možné definovať 3 bodmi, ale opačná implikácia neplatí. Napr. body [1,1], [1,2], [1,4] rozhodne žiadnu kružnicu nedefinujú. A aj keby to tak bolo, máme ďalší problém, ako sa vysporiadať so situáciou, keď máme bodov ďaleko vyššie množstvo, ako iba 3.

- Najprechodnejším variantom sa tak ukazuje byť modifikovaná prvá možnosť. Budeme musieť iba urobiť úpravu, že zo štandardné WGS64 súradnice prevedieme do súradníc v Euklidovskom priestore. Súradnice tam tvoria karteziánsku sústavu troch súradníc navzájom kolmých na osy. Súradnice jedného bodu sú jeho vzdialenosť od troch rovín, ktoré sa pretínajú v osách x, y, z.

2.3 Počítanie stredu

2.3.1 Prevod na stupne

Prvý krok k vypočítaniu stredu je, aby sme mali súradnice vo formáte desatinného čísla, teda aby sme mali iba stupne. Ak je náš zdrojový formát stupne a minúty, alebo dokonca stupne, minúty a sekundy, je to treba previesť na stupne.

Tento postup už bol popísaný, pre prehľadnosť uvediem iba znovu iba vzorec, kde „stupne2“ je náš očakávaný výsledok.

$$stupne2 = stupne1 + \frac{minúty1}{60} + \frac{sekundy1}{3600}$$

2.3.2 Prevod na radiány

Prevod na radiány je druhý nutný krok. Tento prevod je ešte jednoduchší, ako ten predchádzajúci. Výsledok dostaneme, keď použijeme tento vzorec:

$$radiány = stupne * \pi / 180$$

2.3.3 Prevod do karteziánskych súradníc

Karteziánska sústava je v našom prípade trojrozmerná sústava so stredom v strede zeme. Predstavuje zjednodušený model, ktorý predpokladá, že zem je guľatá. To síce nie je dokonale presné, ale dosiahneme lepšie výsledky, akoby sme považovali zem za rovinu.

Karteziánske súradnice je výsledný formát, ktorý potrebujeme, aby sme mohli vypočítať stred. Potrebujeme trojrozmernú sústavu súradníc, ku ktorej sa opäť dopracujeme matematickým vzorcom:

$$x = Cos(latitude) * Cos(longitude)$$
$$y = Cos(latitude) * Sin(longitude)$$

$$z = Sin(latitude)$$

Použili sme premenné latitude a longitude, kde latitude je zemepisná šírka a longitude je zemepisná dĺžka.

Rozdiel oproti predchádzajúcim dvom krokom je, že kým tam sme počítali a prevádzali zemepisnú dĺžku a šírku oddelene, tu sme ich spojili dokopy a máme jeden spoločný výsledný formát.

2.3.4 **Priemer**

Už máme body prevedené na karteziánske súradnice. Nemáme body rozdielnych váh. Keby boli, tak sa počítanie priemeru musí ešte trochu upraviť.

Pre výpočet musíme sčítať jednotlivé zložky súradníc všetkých bodov a nakoniec vydeliť ich počtom. Kód, ktorý urobí túto akciu v jazyku Javascript vyzerá takto:

```
var centerX = 0;
var centerY = 0;
var centerZ = 0;

cartesians.forEach(function(cart) {

    centerX += Number(cart[0]);
    centerY += Number(cart[1]);
    centerZ += Number(cart[2]);
});

centerX = centerX / cartesians.length;
centerY = centerY / cartesians.length;
centerZ = centerZ / cartesians.length;
```

Kde v poli *cartesians* sú uložené karteziánske súradnice všetkých bodov.

Týmto postupom vypočítame stred niekoľkých bodov. Výsledok je ale vo formáte karteziánskych súradníc, takže ho potrebujeme previesť na stupne. Na to sa pozrieme v ďalšej podkapitole.

2.3.5 **Prevod stredu na stupne**

Pri prevode naspäť budeme postupovať po rovnakých krokoch a budeme používať inverzné funkcie. Prejdeme teda z karteziánskych súradníc na radiány, odtiaľ na stupne a prípadne na formát stupne a minúty, alebo stupne, minúty a sekundy.

Najnáročnejší krok je z karteziánskych súradníc vypočítať radiány, ale aj to sa dá zvládnuť so správnym postupom [13]:

$$r = \sqrt{centerX^2 + centerY^2}$$

$$d = \frac{1}{\tan\left(\frac{centerY}{centerX}\right)}$$

$$lontitude = r * \cos(d)$$

$$latitude = r * \sin(d)$$

V jazyku Javascript ale máme funkcie, ktoré tento postup trochu zjednodušujú. Konkrétne nám pomôže funkcia atan2(y, x) [12], ktorá počíta radiány súradníc x y. To je presne to, čo potrebujeme

```
var Lon = Math.atan2(centerY, centerX);
var Hyp = Math.sqrt(centerX * centerX + centerY * centerY);
var Lat = Math.atan2(centerZ, Hyp);
```

Nasledovať bude prevod z radiánov na stupne. Na to použijeme podobný vzorec ako pri prevode na radiány, ale opačný:

$$stupne = radians * (\frac{180}{\pi})$$

Konečne máme výsledok, ktorý môžeme zobraziť na mape.

2.4 mapy.cz API

API máp je dobre popísané v dokumentácii [14]. Z množstva poskytovaných funkcií potrebujeme iba pár. Postupne prejdeme všetky, ktoré sa využívajú v tejto práci.

2.4.1 **Zobrazenie mapy na stránke**

Úplne najzákladnejšia požiadavka, ktorú od mapy máme, je aby sa vôbec zobrazila. Na to musíme najskôr zavolať Javacsriptovú knižnicu mapy.cz:

```
<script type="text/javascript" src="https://api.mapy.cz/loader.js"></script>
<script type="text/javascript">Loader.load();</script>
```

Po odkázaní sa na samotné mapy sme ešte spustili príkaz Load, čím sa mapa inicializuje. Ten sa musí spustiť ešte pred začatím vykresľovania samotnej stránky, inak by mapa nefungovala. Po tomto nutnom základe už môžeme zvoliť miesto na našej stránke, kde sa mapa zobrazí užívateľom:

```
<div id="m" style="height:250px"></div>
```

Ako vidíme, stačí jeden div s id=“m“ a mapa sa do tohto zobrazí. V ukážke html kódu je ešte pridaný jeden style atribút, ktoré nastavuje výšku mapy na striktných 250px.

Toto nám ale stále nestačí, mapa sa bez dodatočného nastavenia nezobrazí. Ešte musíme aktivovať základnú vrstvu, až potom sa zobrazí. My si ku tomu pridáme ešte vycentrovanie mapy na Univerzitu Pardubice a zapnutie štandardného ovládania myšou:

```
// vycentrovanie na súradnice Univerzity Pardubice
var center = SMap.Coords.fromWGS84(15.7691231, 50.0481623);
var m = new SMap(JAK.gel("m"), center, 18);
// regovanie na veľkosť pohľadu
m.addControl(new SMap.Control.Sync());
// zapnutie turistického podkladu
m.addDefaultLayer(SMap.DEF_BASE).enable();
// ovladanie myšou
var mouse = new SMap.Control.Mouse(SMap.MOUSE_PAN | SMap.MOUSE_WHEEL |
SMap.MOUSE_ZOOM);
m.addControl(mouse);
```

2.4.2 **Odchytávanie udalosti Click**

Keď už máme mapu zobrazenú, môžeme zaregistrovať Click Event Handler. Bude nám slúžiť ku tomu, aby sme vedeli kam užívateľ kliká. My budeme tieto body nie len ukladať, ale ich aj funkciou addPointToMap() zobrazíme na mape. Na funkciu sa bližšie pozrieme v ďalšej podkapitole, teraz si ukážeme spracovanie udalosti:

```
// funkcia spracovávajúca klik
```

```
    function click(e, elm) {

        // súradnice bodu, kam sa kliklo
        var coords = SMap.Coords.fromEvent(e.data.event, m);
        // pridanie súradnic do našeho globálneho poľa
        clickedCoordinates.push(coords.toWGS84());
        // zobrazenie bodu na mape
        var cc = clickedCoordinates[clickedCoordinates.length - 1];
        addPointToMap(cc);
    }

    // registrácia udalosti
    m.getSignals().addListener(window, "map-click", click);
```

2.4.3 **Pridanie bodu do mapy**

Funkciu pridania bodu do mapy využijeme hneď v niekoľkých prípadoch. V prvom rade treba zobrazovať body, na ktoré užívateľ klikol. Potom zobrazujeme stred aktuálne naklikaných bodov v reálnom čase a nakoniec ju použijeme, keď automaticky detekujeme a vyhladzujeme clustery.

```
function addPointToMap(coords) {

    // konverzia bodu do pozadovaneho formatu
    var c = SMap.Coords.fromWGS84(coords[0], coords[1]);

    // nastavia sa vlastnosti pridavaneho bodu
    var options = {
        // nastavi sa znacka, ktora bude graficky interpretovat bod
        url: obrazek,
        // bodu by slo nastavit nadpis
        // title: marker.name,
        // miesto ukotvenia na kliknutom mieste
        anchor: {left:10, bottom: 1}
    }

    // clickedCoordinates.length je zaroven unikatne ID bodu
    var znacka = new SMap.Marker(c, clickedCoordinates.length, options);
    vrstva.addMarker(znacka);
}
```

2.4.4 **Vykreslenie polygónu na mape**

Zobrazovať a vykresľovať polygón je funkcia, ktorú využijeme po jeho detekcii na zobrazenie užívateľovi. Technicky to funguje tak, že sa vytvorí nová vrstva na mape. Potom sa do definovanej funkcie pošle parametrom zoznam bodov, ktoré sú vrcholy polygónu, ktorý sa následne zobrazí:

```
var gVrstva = new SMap.Layer.Geometry();
m.addLayer(gVrstva);
gVrstva.enable();

var options1 = {
    color: "#f00"
};

var polyline = new SMap.Geometry(SMap.GEOMETRY_POLYGON, null,
    clearedPoints, options1);
gVrstva.addGeometry(polyline);
```

2.5 Príklad vypočítania a zobrazenia stredu

Ukážeme sin a príklade, ako funguje vypočítavanie stredu súradníc. Budeme mať 5 bodov, u ktorých prejdeme postupne všetkými krokmi a vypočítame stred.

Predstavme si teda, že sedíme pred rektorátom Univerzity Pardubice na lavičke a máme pustený GPS prijímač. Počas sedenia nám v dôsledku nepresností vráti prijímač tieto body:

- [15.768723466097043, 50.04810372983722]
- [15.76856789797418, 50.04819673746039]
- [15.768728830515073, 50.04824840828425]
- [15.768980958162473, 50.04824840828425]
- [15.76903460234277, 50.048076171988406]

Môžeme si všimnúť, že máme WGS84 formát, kde sú použité iba stupne. Keby sme mali aj minúty alebo sekundy, prepočítali by sme ich jednoduchým postupom na stupne a mali by sme rovnaký výsledok.

Každý bod sa musí najskôr prepočítať na radiány. Pripomeňme vzorec a postupne dosaďme všetky body:

$$radiány = stupne * \pi / 180$$

- [15.768723466097043, 50.04810372983722]

    - po prevode: [0.27521614331988586, 0.8735041944653141]
- [15.76856789797418, 50.04819673746039]
    - po prevode: [0.27521342814393074, 0.8735058177545678]
- [15.768728830515073, 50.04824840828425]
    - po prevode: [0.27521623694664293, 0.8735067195805715]
- [15.768980958162473, 50.04824840828425]
    - po prevode: [0.27522063740422537, 0.8735067195805715]
- [15.76903460234277, 50.048076171988406]
    - po prevode: [0.27522157367179606, 0.8735037134901175]

Keď máme radiány, môžeme vypočítať karteziánske súradnice. Opäť pripomeňme postup a vypočítajme výsledok:

$$x = Cos(latitude) * Cos(longitude)$$
$$y = Cos(latitude) * Sin(longitude)$$
$$z = Sin(latitude)$$

- [0.27521614331988586, 0.8735041944653141]
    - prevod na [x, y, z]
    - [0.6179780898369482, 0.737734590712433, 0.27175495238879427]
- [0.27521342814393074, 0.8735058177545678]
    - prevod na [x, y, z]
    - [0.6179773660904644, 0.7377361595001791, 0.2717523393934604]
- [0.27521623694664293, 0.8735067195805715]
    - prevod na [x, y, z]
    - [0.6179762106317344, 0.7377361316713372, 0.2717550424920459]
- [0.27522063740422537, 0.8735067195805715]
    - prevod na [x, y, z]
    - [0.6179754427221044, 0.7377352149456281, 0.271759277342186]
- [0.27522157367179606, 0.8735037134901175]
    - prevod na [x, y, z]
    - [0.6179774970309415, 0.7377331622028405, 0.27176017837345173]

Toto je výsledný formát jednotlivých bodov, pre ktoré môžeme počítať stred aritmetickým priemerom. Sčítame teda jednotlivé súradnice všetkým bodov:

- ∑x = 3.089884606312193

- ∑y = 3.688675259032418
- ∑z = 1.3587817899899384

A vydelíme počtom bodov, v našom prípade 5:

- $\frac{3.089884606312193}{5} = 0.6179769212624386$
- $\frac{3.688675259032418}{5} = 0.7377350518064836$
- $\frac{1.3587817899899384}{5} = 0.2717563579979877$

Už máme vypočítaný stred, ale máme ho v karteziánskych súradniciach. Potrebujeme ho prerobiť cez radiány na stupne a zobraziť na mape. Keďže je demo naprogramované v jazyku Javascript, použijeme jeho funkciu atan2(y, x) a spočítame radiány.

```
var Lon = Math.atan2(centerY, centerX);
var Hyp = Math.sqrt(centerX * centerX + centerY * centerY);
var Lat = Math.atan2(centerZ, Hyp);
```

Bod $[0.6179769212624386, 0.7377350518064836, 0.2717563579979877]$ teda bude po prevode vyzerať takto:

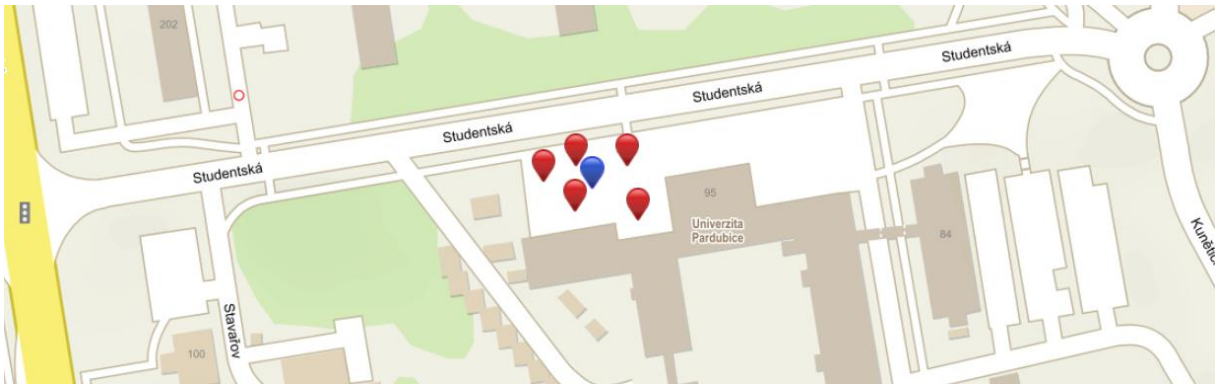
[0.2752176038975041, 0.8735054329744814]

Prevod z radiánov na stupne je posledný nutný krok:

$$stupne = radians * (\frac{180}{\pi})$$

- [0.2752176038975041, 0.8735054329744814]
    - po prevode: [15.768807151030222, 50.048174691185395]

A to už je výsledný bod, ktorý zobrazíme na mape. Graficky to vyzerá nasledovne. Červené značky sú jednotlivé namerané body a modrá značka je vypočítaný stred:



Obrázok 7 - Stred súradníc

Zdroj https://mapy.cz/

2.6 Automatická detekcia clusteru

Pozrime sa konečne na najdôležitejšiu a nosnú časť práce, ku ktorej sme sa postupnými malými krokmi dopracovali. Chceme, aby program dokázal automaticky detekovať ľubovoľné množstvo ľubovoľne veľkých zhlukov bodov. Nazývať ich budeme anglicizmom „cluster“.

Hlavná idea algoritmu je, aby dokázal rozpoznať jednotlivé clustery z priestorového aj časové hľadiska. Časový pohľad je veľmi dôležitý a nesmieme ho opomínať, pretože GPS prijímač môže stáť na jednom mieste a vytvoriť okolo seba cluster, potom akýkoľvek dlhý čas odísť inam a zbierať polohovacie dáta na iných miestach. Po čase sa vráti na pôvodné miesto a opäť bude si vytvorí okolo seba cluster. Tieto 2 clustery spolu nesmieme pomiešať, sú to 2 rôzne prípady a tak ku tomu musíme aj pristupovať.

Táto požiadavka je možné splniť v prípade, že máme body zoradené v takom poradí, v akom boli zachytené prijímačom. Toto je v podstate banálny požiadavka, pretože väčšina softwaru, ak nie úplne všetky, ktoré pracujú s GPS súradnicami, ukladajú body sekvenčne tak, ako prišli.

2.6.1 Výber bodov do clusteru

Prvý krok, ktorý musíme urobiť, je vybrať jednotlivé clustery a body v nich. Je to základná funkcia, ktorá oddelí jednotlivé clustery od bežných bodov.

Aby sme mohli rozpoznať, kde začína a kde končí cluster, potrebujeme jednu základnú vec. Potrebujeme vedieť, ako ďaleko očakávame, že budú body za sebou. Toto je konštanta, ktorú si môže editovať užívateľ sám podľa toho, čo sleduje GPS prijímač. Bude záležať od rýchlosti pohybu sledovaného objektu a od frekvencie zberu dát. Pri určovaní vzdialenosti musíme myslieť aj na to, že GPS má síce za dobrých podmienok chybovosť v jednotkách metrov, ale pri zhoršenej viditeľnosti na satelity to môžu byť aj desiatky až stovky metrov.

Povedzme, že náš cluster bude cluster vtedy, keď všetky body okolo neho budú vzdialené maximálne 100m. To zároveň znamená, že 2 body môžu byť navzájom vzdialené 200m.

Algoritmus výberu bude potom fungovať takto:

- Príde prvý bod
- Príde druhý bod. Pokiaľ je viac ako našich 100m od prvého bodu, ďalej nepokračujeme. Pokiaľ je vzdialený menej ako 100m, vypočítame ich stred
- Príde tretí, štvrtý až n-tý bod. Pozrieme sa, či je vzdialený najviac 100m od aktuálneho stredu. Pokiaľ áno, pridáme ho do kolekcie clustera a prepočítame stred. Pokiaľ je

vzdialený viac ako 100m, pôvodný cluster uzavrieme a považujeme ho za krok 1 ďalšej iterácie algoritmu

Týmto postupom vypočítame postupne všetky clustery na mape. Môžeme zobraziť ich stredy, alebo vyobraziť pekný polygon. To bude témou ďalšej kapitoly.

2.6.2 Výber bodov, ktoré definujú polygón

Body rozdelené do clusterov už máme, teraz ich potrebujeme zobraziť užívateľovi. Zvolil som jemne podfarbený polygón, ktorý má výhodu v ďalšej kapitole o heat mapách, ale to až v ďalšej kapitole.

Na vykreslenie polygónu potrebujeme vybrať body v správnom poradí tak, aby sa z nich stali vrcholy polygónu a po jeho vyfarbení boli všetky body vo vnútri. Síce by sme mohli zobrať všetky body, ale výsledok by mal príliš veľa vrcholov. Okrem toho by sme mali vrcholy aj v bodoch uprostred polygónu, čomu sa chceme vyhnúť. Budeme teda vyberať body tak, aby sme mali čo najmenší počet vrcholov a zároveň tak, aby boli všetky body obsiahnuté v polygóne alebo v jeho niektorom vrchole.

Náš algoritmus je rozdelený na 4 časti, resp. výsledný polygón rozdelíme na 4 kvadranty. Najskôr budeme hľadať časť od najjužnejšieho bodu po najvýchodnejší. Potom od najvýchodnejší po najsevernejší, odtiaľ po najzápadnejší a nakoniec sa vrátime naspäť na najjužnejšieho.

- Začiatok výberu je jednoduchý, jednoducho vyberieme najjužnejší bod. To je ten, ktorého súradnica y má najnižšiu hodnotu.
- Potom nájdeme najjužnejší bod zo všetkých ostatných, ktorý je navyše na východ od predchádzajúceho bodu
- Predchádzajúci krok opakujeme tak dlho, dokým sa nedostaneme do najvýchodnejšieho bodu. To spoznáme tak, že už žiadny ďalší bod nenájdeme
- Ďalej budeme postupovať analogicky, ale ideme od najvýchodnejšieho po najsevernejší. Hľadáme teda body, ktoré sú vždy severnejšie od predchádzajúceho a zároveň čo najvýchodnejšie
- Nakoniec postup zopakujeme pre posledné dva kvadranty

Ukážeme implementáciu tohto postupu v jazyku Javascript. Na začiatku algoritmu si pomôžeme tým, že si vstupné pole bodov skopírujeme do ďalších 2 pomocných polí. Jedno

zoradíme podľa osy y a druhé zoradíme podľa osy x. To síce zvýši pamäťovú náročnosť algoritmu, ale zase zníži výpočetnú náročnosť.

```
function drawHeatMap(heatMapPoints) {

    // tu budu vrcholy, ktore budu urcovat polygón
    clearedPoints = Array();

    // skopirovanie vstupneho pola do 2 pomocnych
    var southFirst = [...heatMapPoints];
    var eastFirst = [...heatMapPoints];

    // zoradenie prveho pola
    // najjuznejsi bod je prvy, najsevernejsi posledny
    southFirst.sort(function(a, b) {

        if (a[1] < b[1]) {
            return -1;
        }

        if (a[1] > b[1]) {
            return 1;
        }

        return 0;
    });

    // zoradenie druheho pola
    // najvychodnejsi bod je prvy
    eastFirst.sort(function(a, b) {

        if (a[0] < b[0]) {
            return 1;
        }

        if (a[0] > b[0]) {
            return -1;
        }

        return 0;
    });

    // najjuznejsi bod ide prvy
    clearedPoints[0] = SMap.Coords.fromWGS84(southFirst[0][0],
southFirst[0][1]);

    // prva cast algoritmu
    // hladame body, ktore su vzdy severnejsie a zaroven vychodnejsie
```

```
    var nicePolygon = true;
    while(nicePolygon) {

        // ak sa foreach breakne, prenastavi sa na false a skusa sa to zas
        nicePolygon = false;

        // najdem najjuznejsi bod z bodov, ktore su na vychod
        for (let point of southFirst) {

            var lastPoint = clearedPoints[clearedPoints.length - 1].toWGS84();

            if (!(point[1] == lastPoint[1] && lastPoint[0] == point[0])) {

                // hladam bod, ktory je severnejsie
                // ale zaroven je vychodnejsie
                if(point[1] >= lastPoint[1] && point[0] >= lastPoint[0]) {

                    // nasiel som taky bod, vlozim ho a som spokojny
                    clearedPoints[clearedPoints.length] =
SMap.Coords.fromWGS84(point[0], point[1]);
                    nicePolygon = true;
                    break;
                }
            }
        };
    }

    // druha cast algoritmu
    // hladame body medzi najvychodnejsim a najsevernejsim
    nicePolygon = true;
    while(nicePolygon) {

        // aktualne posledny bod je ten najvychodnejsi

        // ak sa foreach breakne, prenastavi sa na false a skusa sa to zas
        nicePolygon = false;

        // od najvychodnejsieho po najsevernejsi
        for (let point of eastFirst) {

            var lastPoint = clearedPoints[clearedPoints.length - 1].toWGS84();

            if (!(point[1] == lastPoint[1] && lastPoint[0] == point[0])) {

                // hladam bod, ktory je severnejsie
                // ale zaroven je vychodnejsie
                if(point[1] >= lastPoint[1] && point[0] <= lastPoint[0]) {

                    // nasiel som taky bod, vlozim ho a som spokojny
```

```
                    clearedPoints[clearedPoints.length] =
SMap.Coords.fromWGS84(point[0], point[1]);
                    nicePolygon = true;
                    break;
                }
            }
        };
    }

    // otocime zoradene polia
    // v dalsich 2 iteraciach budeme chciet mat polia, ktore zacinaju
najsevernejsim, resp. nazapadnejsim
    southFirst.reverse();
    eastFirst.reverse();

    // tretia cast algoritmu
    // hladame body medzi najseverenejsim a najzapadnejsim
    nicePolygon = true;
    while(nicePolygon) {

        // ak sa foreach breakne, prenastavi sa na false a skusa sa to zas
        nicePolygon = false;

        // od najsevernejsieho po najzapadnejsi
        for (let point of southFirst) {

            var lastPoint = clearedPoints[clearedPoints.length - 1].toWGS84();

            if (!(point[1] == lastPoint[1] && lastPoint[0] == point[0])) {

                // hladam bod, ktory je severnejsie
                // ale zaroven je vychodnejsie
                if(point[1] <= lastPoint[1] && point[0] <= lastPoint[0]) {

                    // nasiel som taky bod, vlozim ho a som spokojny
                    clearedPoints[clearedPoints.length] =
SMap.Coords.fromWGS84(point[0], point[1]);
                    nicePolygon = true;
                    break;
                }
            }
        };
    }

    // stvrta a posledna cast algoritmu
    // hladame body medzi najzapadnejsim a najjuznejsim
    nicePolygon = true;
    while(nicePolygon) {
```

```
        // ak sa foreach breakne, prenastavi sa na false a skusa sa to zas
        nicePolygon = false;

        // od najzapadnejsieho po najjuznejsi
        for (let point of eastFirst) {

            var lastPoint = clearedPoints[clearedPoints.length - 1].toWGS84();

            if (!(point[1] == lastPoint[1] && lastPoint[0] == point[0])) {

                // hladam bod, ktory je severnejsie
                // ale zaroven je vychodnejsie
                if(point[1] <= lastPoint[1] && point[0] >= lastPoint[0]) {

                    // nasiel som taky bod, vlozim ho a som spokojny
                    clearedPoints[clearedPoints.length] =
SMap.Coords.fromWGS84(point[0], point[1]);
                    nicePolygon = true;
                    break;
                }
            }
        };
    }

    // nastavime farbu polygonu a zobrazime ho na mape
    var options1 = {
        color: "#f00"
    };

    var polyline = new SMap.Geometry(SMap.GEOMETRY_POLYGON, null,
clearedPoints, options1);
    gVrstva.addGeometry(polyline);

    // mozeme vypnut zobrazenie bodov, aby bolo polygon lepsie vidiet
    vrstva.disable();
}
```

2.6.3 Výber bodov do Kernel distribúcie bodov

Posledná funkcia, ktorú pridáme, bude Heat mapa pomocou Kernelovej distribúcie bodov. Budeme používať 25%, 50%, 75% a 100% Kernely.

- 100% Kernel nie je nič iné, ako celý polygón so všetkými bodmi, ktoré obsahuje
- 75% Kernel berie do úvahy 75% všetkých bodov, ktoré sú najbližšie ku stredu
- 50% a 25% je analogicky to isté, ako 75%

Kernelova Heat mapa je významným nástrojom, ako vyhladiť mapu, odfiltrovať najvychýlenejšie body a zvýrazniť oblasť, kde je zachytených najviac bodov. V praxi môžeme postupovať nasledovne.

Najskôr potrebujeme spočítať, ako ďaleko sú jednotlivé body od stredu. To vypočítame vcelku jednoducho. Výsledne pole distancesFromCenter bude obsahovať všetky body aj s ich vzdialenosťou od stredu.

```
var distancesFromCenter = Array();
heatMapPoints.forEach(function(point) {

    // console.log(center[0], center[0]);
    var dist = distance(center[0], center[0], point[0], point[0], "K");
    distancesFromCenter.push([dist, point]);
});
```

V druhom kroku si toto pole zoradíme od najbližšieho po najvzdialenejší.

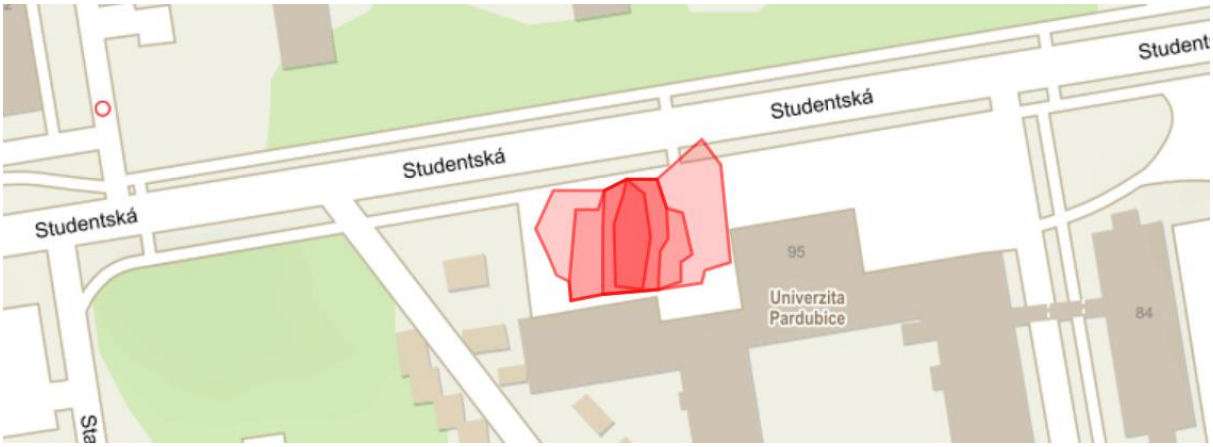
```
// zoradim od najblizsieho po najdalejsi
distancesFromCenter.sort(function(a,b) {
    if (a[0] < b[0]) {
        return -1;
    }

    if (a[0] > b[0]) {
        return 1;
    }

    return 0;
});
```

V treťom kroku prejdeme pole cyklom a budeme si ukladať jednotlivé Kernely. Vzhľadom ku tomu, že vieme zistiť, koľko bodov máme, jednoduchou podmienkou zhodnotíme, či daný bod patrí do niektorého Kernelu.

```
// deklarujem polia jednotlivych kernelov
var smallKernel = Array();
var mediumKernel = Array();
var bigKernel = Array();
var count = 1;
var totalCount = distancesFromCenter.length;
distancesFromCenter.forEach(function(point) {

    // 25% Kernel
    if (count / totalCount <= 0.25) {
```

```
            smallKernel.push(point[1]);
        }

        // 50% Kernel
        if (count / totalCount <= 0.5) {
            mediumKernel.push(point[1]);
        }

        // 75% Kernel
        if (count / totalCount <= 0.75) {
            bigKernel.push(point[1]);
        }
        count++;
    });
```

Po spočítaní Kernelov a zobrazení Heat mapy môžu výsledky vyzerať takto:

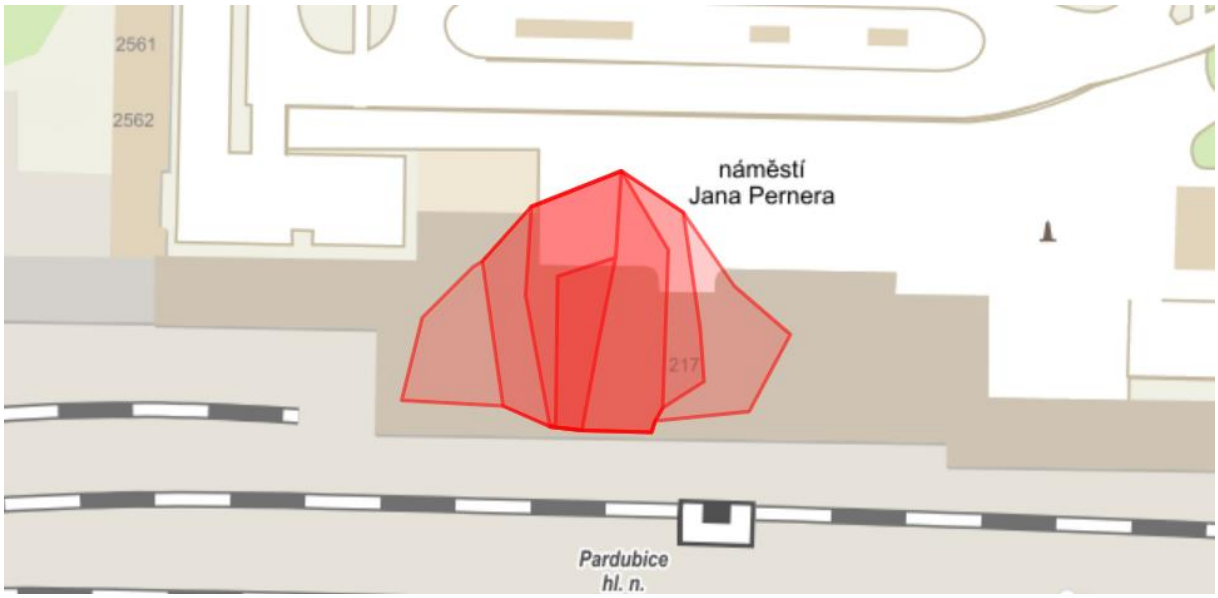


Obrázok 8- Heat mapa

Zdroj https://mapy.cz/

Obrázok 9 - Heat mapa

Zdroj https://mapy.cz/



Obrázok 10 - Heat mapa

Zdroj https://mapy.cz/

Heat mapy tu využili vlastnosť API od mapy.cz, že ak sa zobrazí polygónov cez seba, tak v miestach prekrytia je farba sýtejšia. Vo výsledku nám to urobí takéto Heat mapy.

Záver

V práci bola okrem teoretického úvodu popísaná praktická časť. V tej sa vyvinula funkcia v Javascriptu, ktorá dokáže vyhladiť mapu od množstva bodov, vypočítať ich stred, automaticky detekovať cluster, vykresliť Heat mapu a nakoniec zobraziť užívateľovi lepší výsledok. Praktické výsledky práce budú navyše implementované do komerčného softwaru sledovania dravcov. Cieľ práce tak bol splnený.

Použitá literatúra

[1] Navigační systémy GPS. *Co je GPS* [online]. [cit. 2019-04-12]. Dostupné z: http://www.gpsnavigace.cz/prispevky/co_je_gps.htm

[2] Americký družicový navigační systém NAVSTAR GPS. *Americký družicový navigační systém NAVSTAR GPS - Český Kosmický Portál - Odbor ITS, kosmických aktivit a VaVaI. Český Kosmický Portál - Odbor ITS* [online]. 2017 [cit. 2019-04-24]. Dostupné z: http://www.czechspaceportal.cz/3-sekce/gnss-systemy/gnss-mimo-evropu/americky-navstar-gps/

[3] Formát souřadnice. *Formát souřadnice - GeoWiki* [online]. 2018 [cit. 2019-04-12]. Dostupné z: http://wiki.geocaching.cz/wiki/Formát_souřadnice

[4] Ruský globální družicový navigační systém GLONASS. *Ruský globální družicový navigační systém GLONASS - Český Kosmický Portál - Odbor ITS, kosmických aktivit a VaVaI*[online]. 2017 [cit. 2019-04-18]. Dostupné z: http://www.czechspaceportal.cz/3-sekce/gnss-systemy/gnss-mimo-evropu/rusky-glonass/

[5] GALILEO - Evropský globální navigační družicový systém. *Ruský globální družicový navigační systém GLONASS - Český Kosmický Portál - Odbor ITS, kosmických aktivit a VaVaI*[online]. 2017 [cit. 2019-04-18]. Dostupné z: http://www.czechspaceportal.cz/3-sekce/gnss-systemy/galileo/

[6] LOKALIZÁCIA V MOBILNÝCH SIEŤACH POMOCOU METÓDY DATABÁZOVEJ KORELÁCIE (DCM). *LOKALIZÁCIA V MOBILNÝCH SIEŤACH POMOCOU METÓDY DATABÁZOVEJ KORELÁCIE (DCM)* [online]. 2017 [cit. 2019-05-11]. Dostupné z: https://dspace.vsb.cz/bitstream/handle/10084/83889/AEEE-2008-7-3-427-mada.pdf?sequence=1

[7] How Radar Works. *How Radar Works | HowStuffWorks* [online]. [cit. 2019-05-11]. Dostupné z: https://science.howstuffworks.com/radar.htm

[8] RTLS - Real Time Location System. *RTLS - Real Time Location System | Kodys Slovensko* [online]. [cit. 2019-05-11]. Dostupné z: https://www.kodys.sk/rtls-real-time-location-system

[9] GPS III explained: Everything you need to know about the next generation of GPS. *GPS III explained: Everything you need to know about the next generation of GPS | Digital*

Trends [online]. 2011 [cit. 2019-05-01]. Dostupné z: https://www.digitaltrends.com/cool-tech/gps-iii-explained-everything-you-need-to-know-about-the-next-generation-of-gps/

[10] SpaceX launches first GPS 3 satellite. *SpaceX launches first GPS 3 satellite - SpaceNews.com* [online]. 2018 [cit. 2019-05-01]. Dostupné z: https://www.digitaltrends.com/cool-tech/gps-iii-explained-everything-you-need-to-know-about-the-next-generation-of-gps/

[11] Srovnání navigačních systémů GPS, GLONASS, GALILEO, BEIDOU. *Srovnání navigačních systémů GPS, GLONASS, GALILEO, BEIDOU* [online]. [cit. 2019-05-08]. Dostupné z: http://wiki.cs.vsb.cz/images/8/88/Gis_lap028.pdf

[12] JavaScript atan2() Method. *JavaScript atan2() Method* [online]. [cit. 2019-05-16]. Dostupné z: https://www.w3schools.com/jsref/jsref_atan2.asp

[13] Polar and Cartesian Coordinates. *Polar and Cartesian Coordinates* [online]. 2017 [cit. 2019-05-16]. Dostupné z: https://www.mathsisfun.com/polar-cartesian-coordinates.html

[14] Mapy API verze 4.13 – Neil Armstrong. *API Mapy.cz* [online]. 2019 [cit. 2019-05-18]. Dostupné z: https://api.mapy.cz/

[15] Systém jednotné trigonometrické sítě katastrální. *Systém jednotné trigonometrické sítě katastrální - Wikipedie* [online]. 2018 [cit. 2019-05-23]. Dostupné z: https://cs.wikipedia.org/wiki/Systém_jednotné_trigonometrické_sítě_katastrální

Príloha A – Zdrojové kódy súboru index.html

```
<!DOCTYPE html>
<html>
  <head>
      <meta charset="utf-8" />
      <script type="text/javascript"
src="https://api.mapy.cz/loader.js"></script>
      <script type="text/javascript">Loader.load();</script>
      <script type="text/javascript" src="custom.js"></script>
      <script type="text/javascript" src="conversions.js"></script>

  </head>

  <body style="text-align:center;">
     <div id="m" style="height:350px; width: 650px; display: inline-
block"></div>

     <div>
        <button type="button" onclick="calculateClusters(clickedCoordinates)"
style="color: #fff; background-color: blue; height: 3em; width: 6em; border:
none; margin-top: 25px;">Clustery</button>
    </div>

  </body>
</html>


<script type="text/javascript">

    var obrazek = "https://api.mapy.cz/img/api/marker/drop-red.png";
    var obrazekStredu = "https://api.mapy.cz/img/api/marker/drop-blue.png";

    // vycentrovanie na súradnice Univerzity Pardubice
    var center = SMap.Coords.fromWGS84(15.7691231, 50.0481623);
    var m = new SMap(JAK.gel("m"), center, 18);
    // regovanie na veľkosť pohľadu
    m.addControl(new SMap.Control.Sync());
    // zapnutie turistického podkladu
    m.addDefaultLayer(SMap.DEF_BASE).enable();
    // ovladanie myšou
    var mouse = new SMap.Control.Mouse(SMap.MOUSE_PAN | SMap.MOUSE_WHEEL |
    SMap.MOUSE_ZOOM);
    m.addControl(mouse);

    var vrstva = new SMap.Layer.Marker();

    m.addLayer(vrstva);
```

```
    vrstva.enable();

    var gVrstva = new SMap.Layer.Geometry();
    m.addLayer(gVrstva);
    gVrstva.enable();

    // funkcia spracovávajúca klik
    function click(e, elm) {

        // súradnice bodu, kam sa kliklo
        var coords = SMap.Coords.fromEvent(e.data.event, m);
        // pridanie súradnic do našeho globálneho poľa
        clickedCoordinates.push(coords.toWGS84());
        // zobrazenie bodu na mape
        var cc = clickedCoordinates[clickedCoordinates.length - 1];
        addPointToMap(cc);

        // po každom pridanom bode sa okamžite prepočíta stred
        var centerCoords = calculateCenterPoint(clickedCoordinates);
        addCenterToMap(centerCoords);
    }

    // registrácia udalosti
    m.getSignals().addListener(window, "map-click", click);

</script>
```

Príloha B – Zdrojové kódy súboru custom.js

```
// jedina globalna premenna (okrem mapy)
// ma v sebe vsetky naklikane suradnice
var clickedCoordinates = Array();

function addPointToMap(coords) {

    // konverzia bodu do pozadovaneho formatu
    var c = SMap.Coords.fromWGS84(coords[0], coords[1]);

    var options = {
        url: obrazek,
        // bodu by slo nastavit nadpis
        // title: marker.name,
        // miesto ukotvenia na kliknutom mieste
        anchor: {left:10, bottom: 1}
    }

    // clickedCoordinates.length je zaroven unikatne ID bodu
    var znacka = new SMap.Marker(c, clickedCoordinates.length, options);
    vrstva.addMarker(znacka);
}

function removeCenterFromMap(centerCoords) {

    var c = SMap.Coords.fromWGS84(centerCoords[0], centerCoords[1]);

    var options = {
        url: obrazekStredu,
        // title: marker.name,
        anchor: {left:10, bottom: 1}
    }

    vrstva.removeMarker(new SMap.Marker(c, 99999, options) );
}

function addCenterToMap(centerCoords) {

    removeCenterFromMap(centerCoords);

    var c = SMap.Coords.fromWGS84(centerCoords[0], centerCoords[1]);

    var options = {
        url: obrazekStredu,
        // title: marker.name,
        anchor: {left:10, bottom: 1}
    }
```

```
    var znacka = new SMap.Marker(c, 99999, options);
    vrstva.addMarker(znacka);
}

function calculateCenterPoint(points) {

    var cartesians = Array();
    var nDecimal;
    var eDecimal;

    points.forEach(function(coordinate) {

        nDecimal = coordinate[0];
        eDecimal = coordinate[1];

        var nRadians = convertDegreeDecimalToRadians(nDecimal);
        var eRadians = convertDegreeDecimalToRadians(eDecimal);

        cartesians.push(convertRadiansToCartesian(nRadians, eRadians));
    });

    var centerX = 0;
    var centerY = 0;
    var centerZ = 0;

    cartesians.forEach(function(cart) {

        centerX += Number(cart[0]);
        centerY += Number(cart[1]);
        centerZ += Number(cart[2]);
    });

    centerX = centerX / cartesians.length;
    centerY = centerY / cartesians.length;
    centerZ = centerZ / cartesians.length;

    var Lon = Math.atan2(centerY, centerX);
    var Hyp = Math.sqrt(centerX * centerX + centerY * centerY);
    var Lat = Math.atan2(centerZ, Hyp);

    var centerDecimalLon = convertRadiansToDegreeDecimal(Lon);
    var centerDecimalLat = convertRadiansToDegreeDecimal(Lat);

    var centerCoords = Array();
    centerCoords[0] = centerDecimalLat;
    centerCoords[1] = centerDecimalLon;

    return centerCoords;
```

```
}

function calculateClusters(points) {

    var centerCoords = Array();
    var cartesians = Array();
    var heatMapPoints = Array();
    var clusterSize = 0;


    var previousNDecimal = -1;
    var previoudERadians = -1;

    points.forEach(function(coordinate) {

        var nDecimal = coordinate[0];
        var eDecimal = coordinate[1];

        if(previousNDecimal == -1){
            previousNDecimal = nDecimal;
            previoudERadians = eDecimal;
        }

        var nRadians = convertDegreeDecimalToRadians(nDecimal);
        var eRadians = convertDegreeDecimalToRadians(eDecimal);

        cartesians.push(convertRadiansToCartesian(nRadians, eRadians));

        var dist = distance(nDecimal, eDecimal, previousNDecimal,
previoudERadians);
        if(dist < 0.1) {

            heatMapPoints.push(coordinate);

            // vezmem body z clustera a urobim z nich stred

            var centerX = 0;
            var centerY = 0;
            var centerZ = 0;

            cartesians.forEach(function(cart) {

                centerX += Number(cart[0]);
                centerY += Number(cart[1]);
                centerZ += Number(cart[2]);
            });

            centerX = centerX / cartesians.length;
            centerY = centerY / cartesians.length;
```

```
            centerZ = centerZ / cartesians.length;

            var Lon = Math.atan2(centerY, centerX);
            var Hyp = Math.sqrt(centerX * centerX + centerY * centerY);
            var Lat = Math.atan2(centerZ, Hyp);

            var centerDecimalLon = convertRadiansToDegreeDecimal(Lon);
            var centerDecimalLat = convertRadiansToDegreeDecimal(Lat);

            clusterSize += 1;

            centerCoords[0] = centerDecimalLat;
            centerCoords[1] = centerDecimalLon;
            previousNDecimal = centerDecimalLat;
            previoudERadians = centerDecimalLon;
        }
        else {

            if(clusterSize > 1) {
                addCenterToMap(centerCoords);
                countKernelsAndDraw(heatMapPoints);
            }

            //novy cluster - vynulujem body pre centrum
            cartesians = Array();
            heatMapPoints = Array();
            previousNDecimal = -1;
            previoudERadians = -1;
            centerCoords = Array();
            clusterSize = 0;

            heatMapPoints.push(coordinate);
        }
    });

    if(clusterSize > 1) {

        addCenterToMap(centerCoords);
        countKernelsAndDraw(heatMapPoints);
    }
}

function countKernelsAndDraw(heatMapPoints) {

    var center = calculateCenterPoint(heatMapPoints);
    var distancesFromCenter = Array();
    heatMapPoints.forEach(function(point) {

        var dist = distance(center[0], center[0], point[0], point[0]);
```

```
        distancesFromCenter.push([dist, point]);
    });

    // zoradim od najblizsieho po najdalejsi
    distancesFromCenter.sort(function(a,b) {

        if (a[0] < b[0]) {
            return -1;
        }

        if (a[0] > b[0]) {
            return 1;
        }

        return 0;
    });

    // deklarujem polia jednotlivych kernelov
    var smallKernel = Array();
    var mediumKernel = Array();
    var bigKernel = Array();

    var count = 1;
    var totalCount = distancesFromCenter.length;
    distancesFromCenter.forEach(function(point) {

        // 25% Kernel
        if (count / totalCount <= 0.25) {
            smallKernel.push(point[1]);
        }

        // 50% Kernel
        if (count / totalCount <= 0.5) {
            mediumKernel.push(point[1]);
        }

        // 75% Kernel
        if (count / totalCount <= 0.75) {
            bigKernel.push(point[1]);
        }

        count++;
    });

    drawHeatMap(smallKernel);
    drawHeatMap(mediumKernel);
    drawHeatMap(bigKernel);
    drawHeatMap(heatMapPoints);
}
```

```
function drawHeatMap(heatMapPoints) {

    // tu budu vrcholy, ktore budu urcovat polygón
    clearedPoints = Array();

    // skopirovanie vstupneho pola do 2 pomocnych
    var southFirst = [...heatMapPoints];
    var eastFirst = [...heatMapPoints];

    // zoradenie prveho pola
    // najjuznejsi bod je prvy, najsevernejsi posledny
    southFirst.sort(function(a, b) {

        if (a[1] < b[1]) {
            return -1;
        }

        if (a[1] > b[1]) {
            return 1;
        }

        return 0;
    });

    // zoradenie druheho pola
    // najvychodnejsi bod je prvy
    eastFirst.sort(function(a, b) {

        if (a[0] < b[0]) {
            return 1;
        }

        if (a[0] > b[0]) {
            return -1;
        }

        return 0;
    });

    // najjuznejsi bod ide prvy
    clearedPoints[0] = SMap.Coords.fromWGS84(southFirst[0][0],
southFirst[0][1]);

    // prva cast algoritmu
    // hladame body, ktore su vzdy severnejsie a zaroven vychodnejsie
    var nicePolygon = true;
    while(nicePolygon) {
```

```
        // ak sa foreach breakne, prenastavi sa na false a skusa sa to zas
        nicePolygon = false;

        // najdem najjuznejsi bod z bodov, ktore su na vychod
        for (let point of southFirst) {

            var lastPoint = clearedPoints[clearedPoints.length - 1].toWGS84();

            if (!(point[1] == lastPoint[1] && lastPoint[0] == point[0])) {

                // hladam bod, ktory je severnejsie
                // ale zaroven je vychodnejsie
                if(point[1] >= lastPoint[1] && point[0] >= lastPoint[0]) {

                    // nasiel som taky bod, vlozim ho a som spokojny
                    clearedPoints[clearedPoints.length] =
SMap.Coords.fromWGS84(point[0], point[1]);
                    nicePolygon = true;
                    break;
                }
            }
        };
    }

    // druha cast algoritmu
    // hladame body medzi najvychodnejsim a najsevernejsim
    nicePolygon = true;
    while(nicePolygon) {

        // aktualne posledny bod je ten najvychodnejsi

        // ak sa foreach breakne, prenastavi sa na false a skusa sa to zas
        nicePolygon = false;

        // od najvychodnejsieho po najsevernejsi
        for (let point of eastFirst) {

            var lastPoint = clearedPoints[clearedPoints.length - 1].toWGS84();

            if (!(point[1] == lastPoint[1] && lastPoint[0] == point[0])) {

                // hladam bod, ktory je severnejsie
                // ale zaroven je vychodnejsie
                if(point[1] >= lastPoint[1] && point[0] <= lastPoint[0]) {

                    // nasiel som taky bod, vlozim ho a som spokojny
                    clearedPoints[clearedPoints.length] =
SMap.Coords.fromWGS84(point[0], point[1]);
                    nicePolygon = true;
```

```
                    break;
                }
            }
        };
    }

    // otocime zoradene polia
    // v dalsich 2 iteraciach budeme chciet mat polia, ktore zacinaju
najsevernejsim, resp. nazapadnejsim
    southFirst.reverse();
    eastFirst.reverse();

    // tretia cast algoritmu
    // hladame body medzi najseverenejsim a najzapadnejsim
    nicePolygon = true;
    while(nicePolygon) {

        // ak sa foreach breakne, prenastavi sa na false a skusa sa to zas
        nicePolygon = false;

        // od najsevernejsieho po najzapadnejsi
        for (let point of southFirst) {

            var lastPoint = clearedPoints[clearedPoints.length - 1].toWGS84();

            if (!(point[1] == lastPoint[1] && lastPoint[0] == point[0])) {

                // hladam bod, ktory je severnejsie
                // ale zaroven je vychodnejsie
                if(point[1] <= lastPoint[1] && point[0] <= lastPoint[0]) {

                    // nasiel som taky bod, vlozim ho a som spokojny
                    clearedPoints[clearedPoints.length] =
SMap.Coords.fromWGS84(point[0], point[1]);
                    nicePolygon = true;
                    break;
                }
            }
        };
    }

    // stvrta a posledna cast algoritmu
    // hladame body medzi najzapadnejsim a najjuznejsim
    nicePolygon = true;
    while(nicePolygon) {

        // ak sa foreach breakne, prenastavi sa na false a skusa sa to zas
        nicePolygon = false;
```

```
        // od najzapadnejsieho po najjuznejsi
        for (let point of eastFirst) {

            var lastPoint = clearedPoints[clearedPoints.length - 1].toWGS84();

            if (!(point[1] == lastPoint[1] && lastPoint[0] == point[0])) {

                // hladam bod, ktory je severnejsie
                // ale zaroven je vychodnejsie
                if(point[1] <= lastPoint[1] && point[0] >= lastPoint[0]) {

                    // nasiel som taky bod, vlozim ho a som spokojny
                    clearedPoints[clearedPoints.length] =
SMap.Coords.fromWGS84(point[0], point[1]);
                    nicePolygon = true;
                    break;
                }
            }
        };
    }

    // nastavime farbu polygonu a zobrazime ho na mape
    var options1 = {
        color: "#f00"
    };

    var polyline = new SMap.Geometry(SMap.GEOMETRY_POLYGON, null,
clearedPoints, options1);
    gVrstva.addGeometry(polyline);

    // mozeme vypnut zobrazenie bodov, aby bolo polygon lepsie vidiet
    vrstva.disable();
}
```

Príloha C – Zdrojové kódy súboru conversions.js

```
function toDegreesMinutesAndSeconds(coordinate) {

    var absolute = Math.abs(coordinate);
    var degrees = Math.floor(absolute);
    var minutesNotTruncated = (absolute - degrees) * 60;
    var minutes = Math.floor(minutesNotTruncated);
    var seconds = (minutesNotTruncated - minutes) * 60;

    return degrees + "°" + minutes + "'" + seconds;
}

// convert stupnov na radiany
function convertDegreeDecimalToRadians(degree) {

    return degree * Math.PI / 180;
}

// convert stupnov na radiany
function convertRadiansToDegreeDecimal(degree) {

    return degree * (180 / Math.PI);
}

function convertRadiansToCartesian(lat, lon) {

    var x = Math.cos(lat) * Math.cos(lon);
    var y = Math.cos(lat) * Math.sin(lon);
    var z = Math.sin(lat);

    return [x, y, z];
}

function distance(lat1, lon1, lat2, lon2) {

    if ((lat1 == lat2) && (lon1 == lon2)) {

        return 0;
    }
    else {

        var radlat1 = Math.PI * lat1/180;
        var radlat2 = Math.PI * lat2/180;
        var theta = lon1-lon2;
        var radtheta = Math.PI * theta/180;
        var dist = Math.sin(radlat1) * Math.sin(radlat2) + Math.cos(radlat1) *
Math.cos(radlat2) * Math.cos(radtheta);
```

```
        if (dist > 1) {

            dist = 1;
        }

        dist = Math.acos(dist);
        dist = dist * 180/Math.PI;
        dist = dist * 60 * 1.1515;
        // prevod na kilometre
        dist = dist * 1.609344;
        return dist;
    }
}
```