TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Karl Kuusik

Siseruumides positsioneerimise tarkvara loomine

Bakalaureusetöö (9 EAP)

Juhendaja(d):
Alo Peets

Tartu 2020

Siseruumides positsioneerimise tarkvara loomine

Lühikokkuvõte:

Ettevõtete jaoks on muutunud järjest olulisemaks *data economy.* mis viib klientide kohta kogutud inforatsiooni otse firmani. Selle heaks näiteks on toidupoodide puhul kliendi teekonna jälgimisest saadud andmed, mida on võimalik poe edendamiseks kasutada. Tänapäeval on mitmeid lahendusi siseruumides inimeste positsioneerimiseks, aga tihti on need väga kallid ja ebatäpsed. Marvelmind Robotics'i ultrahelil põhinev siseruumide positsioneerimissüsteem võimaldab täpset jälgimist võrdlemisi odava hinna eest. Probleemiks on Marvelmind Dashboard rakendus, mis on eelkõige mõeldud süsteemi ülesseadmiseks kui asukoha andmete salvestamiseks ja analüüsimiseks.

Töö eesmärgiks on luua tarkvara, mis lahendaks Marvelmind Dashboard rakenduse puudujäägid ning kasutada maksimaalselt ära Marvelmind'i positsioneerimissüsteemi võimalusi. Töö käigus valmis Python'i rakendus, mis peamiselt *tkinteri* ja *numpy* teeke kasutades laseb Marvelmind'i positsioneerimissüsteemi poolt jälgitud liikumist salvestada ja analüüsida. Lahendust testiti põhjalikult Tartu Ujula Konsumis.

Võtmesõnad:

positsioneerimissüsteemid, ultraheli

CERCS: P175 Informaatika, süsteemiteooria

Development of an indoor positioning system software

Abstract:

Businesses have started to greatly value data economy, which brings the collected customer data directly to the companies. A good example of this is the pathing of the customer inside supermarkets, which could be used to improve the store. Today we have many solutions for positioning in indoor environments, but usually they are either too expensive or not precise enough for commercial use. However, the ultrasound-based indoor positioning system made by Marvelmind Robotics promises both affordable pricing and high location accuracy. The problem with the solution is the Marvelmind Dashboard application, which is mainly meant for setting up the positioning system, rather than to save and analyse the collected data.

The purpose of this thesis is to create software, which would solve the shortcomings of the Marvelmind Dashboard application and would utilize the full potential of the Marvelmind positioning system. As a result of this thesis, using the tkinter and numpy libraries, an application for saving and analysing data captured by the Marvelmind positioning system was created. The application was thoroughly tested at Tartu Ujula Konsum.

Keywords:

positioning systems, ultrasound

CERCS: P175 Informatics, systems theory

Sisukord

1 Sissejuhatus

1.1 Teema tutvustus

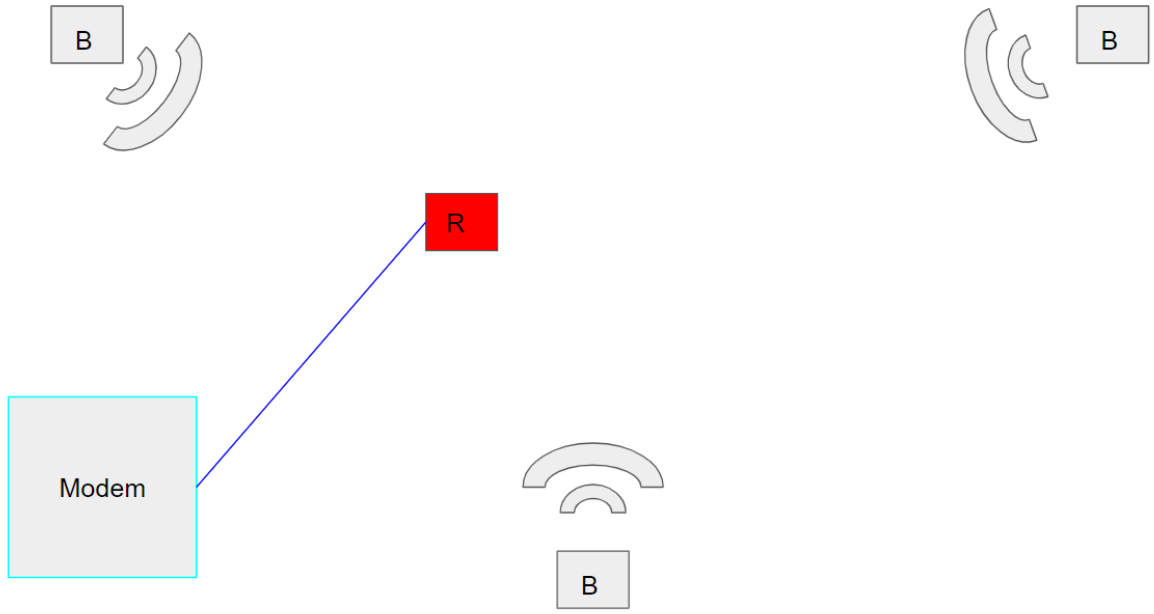
Ettevõtte jaoks on oluline teada oma kliendi eelistusi paremini kui klient ise. Toidupoodidel on vajalik teada kliendi liikumise teekonda, et suunata klienti tegema emotsioonioste. Poeketi jaoks on ostja poolt kasutamata ruutmeeter tarbetu kuluallikaks, mistõttu on oluline optimeerida müügipinna kasutamist. Selleks on oluline teada kliendi käitumist poes – kus klient käib, mis ajahetkel ja kui kaua ta seal viibib.

Tänapäeval on kulude vähendamise küsimus eriti keeruline, kuna tekkinud on poe sees kaasaskantavad triipkoodilugejad ehk ostupuldid. Nendega saab klient ise skänneerida oma tooted ja kohe enda kaasavõetud või poest ostetud kotti panna. Tavaks on panna vastupidavama pakendiga asjad (nt pudelid) enne kotti kui vähem vastupidavamad (nt munad või piimapakk). Seetõttu võib eeldada, et sellise kliendi teekond poes erineb tavalisest ostukäruga ringi kõndiva kliendi omast. Selle väite kontrollimiseks on vaja kaardistada mõlema ostleja teekond.

Asukoha andmete saamiseks on erinevaid meetodeid ning välitingimustes on neist populaarseim GPS. Poodide puhul on tegemist tavaliselt siseruumidega ja seetõttu pole GPS piisavalt täpne. Alternatiivideks on Wi-Fi ja Bluetooth põhjal triangulatsioon. Wi-Fi puhul on täpsusviga 5-15m (infsoft, 2019). Bluetooth 5.1 positsioneerimisega tuleb väiksem mõõtmisviga ning 36 Bluetoothi saatjaga on see parimal juhul kuni 5m (Cominelli, Patras, & Gringoli, 2019). Poes riiulite vahel liikumise jälgimiseks aga ei piisa 5m täpsusest, sest sellise mõõtmisveaga võib rakendus liikuva inimese riiulite vahel valesse vahesse paigutada. Suurema täpsuse saavutamiseks tuli otsida alternatiivseid lahendusi ja Marvelmind'i poolt pakutud ultrahelilahendusel oligi lubatud headel tingimustel täpsusveaks kuni 2cm (Marvelmind Robotics, 2019). Täpsuse tõttu sai otsustatud Marvelmind'i lahendust testida ning algse testimise käigus mõõtmisviga kontrollida. Kuna ebatäpsus jäigi lubatu sisse, siis mindigi edasi Marvelmind'i süsteemi kasutades.

Ultrahelil põhinevad positsioneerimissüsteemid kasutavad sarnaselt Bluetooth ja Wi-Fi süsteemidele signaali vastuvõtjaid ja saatjaid signaali tugevuse mõõtmisteks, kuid ultraheli puhul saab seda teha mitmel viisil. Esimene võimalus on paigaldada mitu erineva sagedusega ultraheli levitavat statsionaarset saatjat ning vastuvõtja mõõdab siis signaali tugevuse alusel kauguse kõigist saatjatest (vt Joonis 1). Mõõdetud kaugused saadetakse

Marvelmind'i lahenduse puhul raadiosignaali kaudu modemisse, millest seda infot lõpuks lugeda saab.



Joonis 1. Ultraheli positsioneerimissüsteem 3 saatjaga (B) ja 1 vastuvõtjaga (R)

1.2 Töö eesmärk

Töö peamine eesmärk on luua graafiline kasutajaliides, mis suhtleks Marvelmind'i modemiga ja salvestaks modemist saadud vastuvõtjate asukohad vastavatel ajahetkedel, et neid hiljem analüüsida saaks. Kliendi nõudel peab rakendus samuti suutma visualiseerida salvestatud andmeid ning seetõttu ei piisa ainult Marvelmind Dashboard rakendusest. Lahenduse realiseerimiseks on vaja kasutada Marvelmind Dashboard'i rakendust ülesseatud positsioneerimissüsteemi sätestamiseks ja erinevaid Python'i teeke salvestatud andmete lugemiseks ja visualiseerimiseks, sealhulgas ka Marvelmind'i enda loodud teeki modemiga suhtlemiseks. Süsteemi ülesseadmine ja rakenduse arendamine toimub paralleelselt, korduvalt ülesseatud süsteemi abil loodavat tarkvara katsetades ning ultraheli süsteemi täiendades.

2 Protsess enne rakenduse arendamist

Idee sellise ultraheli põhjal toimivale siseruumides positsioneerimise süsteemile sündis 2019 aasta alguses, kui Andres Kuusik ja Coop-i esindajad arutlesid omavahel siseruumides positsioneerimise süsteemide kasutuselevõtmise üle.

Andres Kuusik pöördus seejärel Tartu Ülikooli poole ning kontakteerus Alo Peetsiga, kuna vajas tehnilist abi ultraheli süsteemi ülesseadmise juures. Lisaks oli vaja nõu süsteemi abil saadud andmete salvestamiseks ja analüüsimiseks vajaliku rakendamise loomise koha pealt. Alo omakorda soovitas töö autorit ning rakenduse loomise protsessi põhjal lõputöö kirjutada.

Põhjus eraldi rakenduse loomise vajalikkuseks oli Marvelmind Dashboard rakenduse puudujäägid analüüsimise, jälgitud liikumise salvestamise ja visualiseerimise juures. Võimalik on küll pilti taustale lisada ja reaalajas saatjate asukohti jälgida, kuid protsess selle tegemiseks on aeganõudev, ebamugav ja ei paku head visuaalset tagasisidet. Lisaks ei oma Marvelmind Dashboard rakendus võimalust andmete analüüsimiseks, sealhulgas näiteks *heatmapi* genereerimine.

2.1 Ultraheli süsteemi ülesseadmine

Enne Tartu Ujula Konsumisse minemist läbisime mitu testimise faasi. Progressi jooksul õppisin palju Marvelmind'i NIA (*Non-Inverse Architecture*) ja IA (*Inverse Architecture*) positsioneerimissüsteemide kohta ja Marvelmind Dashboard rakenduse funktsionaalsuste kohta (vt Lisa 1 ja Lisa 2).

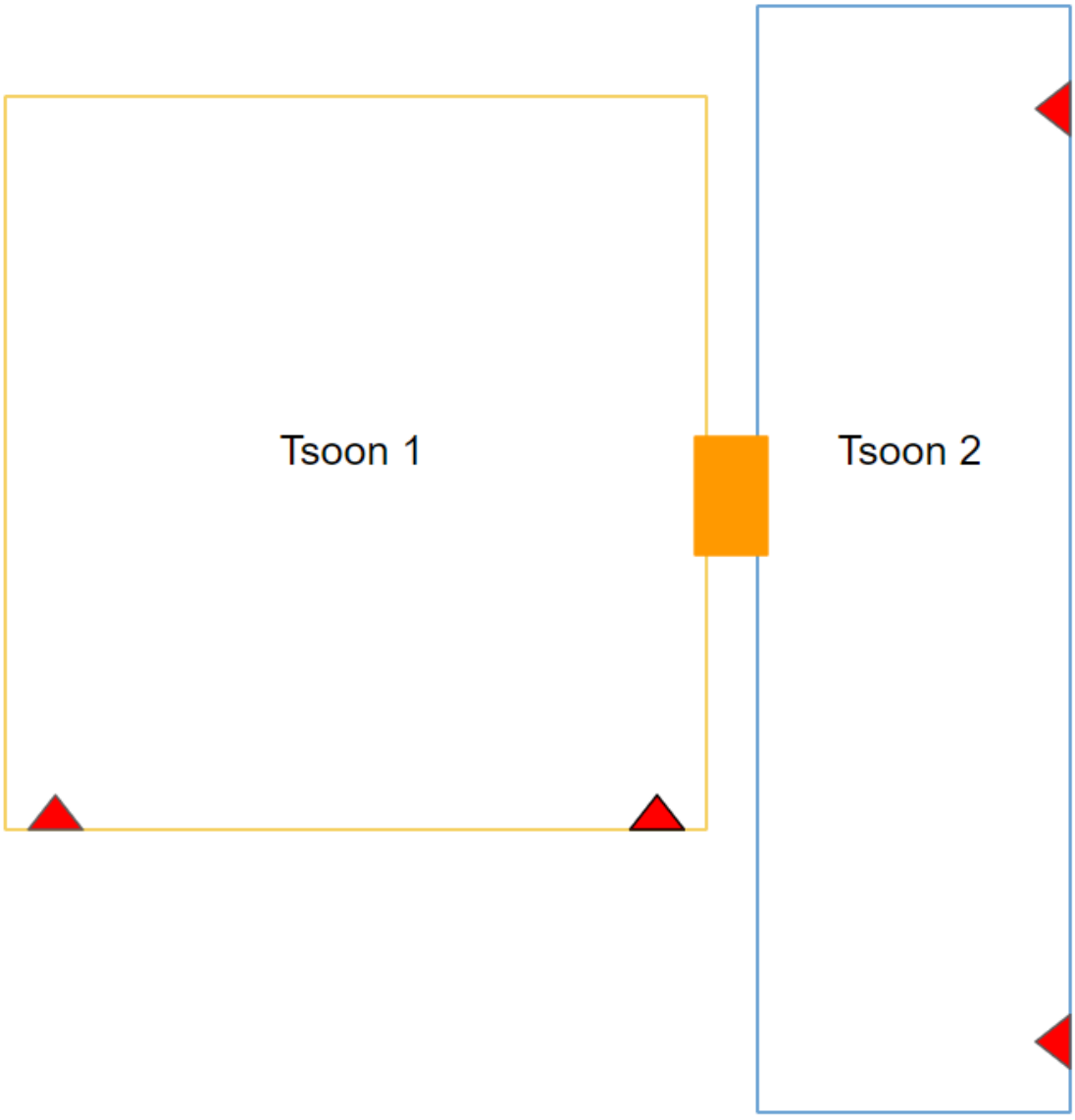
Esimene faas oli osta NIA süsteemi saatjad, mille abil esimest korda Marvelmind Dashboard rakendusega tutvuda (vt Lisa 1). Saatjad sättisime üles Andres Kuusiku kabinetis majandusteaduskonnas Liivi tn. 4 majas Marvelmind'i teegi testimiseks.

Pärast esimese faasi testimist tellisime IA süsteemi saatjad ja vastuvõtjad teise faasi jaoks (vt Lisa 2). Need seadsime samuti algseks testimiseks üles Liivi tn 4 majas koridoris ja klassiruumis, et katsetada erinevate jälgimisstoonide ülesseadmist ning vaadata, kuidas mõjutab tsoonide vahel liikumine vastuvõtja trajektoori (vt Joonis 2).

Nende esimeste IA süsteemi katsetuste põhjal tekkis arusaam, et mida vähem tsoone teha, seda funktsionaalsem on kogu lahendus, kuna 4 saatjaga lahendus (vt Joonis 2) töötas palju täpsemalt ja töökindlamalt kui 8 saatjaga lahendus. Töökindluse paranemine tulenes sellest, et IA süsteemi saatjad on omavahel paaris ning üks paar saab jälgida Marvelmind Dashboard rakenduses defineeritud tsooni või kogu võimalikku ala. Väikse ruumi peale ei ole aga

tingimata vaja teha mitu jälgimise tsooni ja kuna tsoonide vahel defineeritud ülekandetsoonides liikumine ei ole sama sujuv kui ühes suures tavalises tsoonis, siis on kogu liikumine ebatäpsem ja edasi-tagasi hüplev. Üks saatja suudab kuni 30 meetri kaugusel olevat vastuvõtjat jälgida otsese nähtavuse olemasolul ning meie kasutatud alad ei olnud saatjate paigutuse suhtes ei pikemad ega laiemad kui 30m. Otsene nähtavus saatja ja vastuvõtja vahel oli kasutatud ruumides garanteeritud.

Testimise käigus tuli ilmsiks, et kui otsene nähtavus saatja ja vastuvõtja vahel puudub, siis rakenduses näidatud saatja asukohta enam ei liigutata või saatja hüppab ringi vigaste mõõtmiste tõttu. Selle saime teada minnes läbi Tsoon 1 ja Tsoon 2 vahel olevast uksest (vt Joonis 2).



Joonis 2. Esimene katsetus mitme tsooniga IA süsteemiga Liivi tn 4 majas.

2.1.1 Süsteemi ülesseadmise kulu

Kuna alguses oli eesmärk tutvuda süsteemi kasutusvõimalustega ja piirangutega, siis osteti nii NIA kui IA süsteemi saatjad ja vastuvõtjad. Kui süsteemiga tuttav inimene pakuks lahendust mõnele ettevõttele sellise positsioneerimissüsteemi ülesseadmiseks, siis sõltub hind suuresti jälgitava ala iseärasustest. Mida avaram ala on, seda töökindlam ja odavam lahendus on, kuid suurte alade puhul tuleb arvestada sellega, et üks statsionaarne saatja katab kuni 30m ala. Samuti sõltub lahenduse hind ka jälgitavate vastuvõtjate kogusest, kuna iga vastuvõtja maksab. Kõige odavam IA süsteem 2 saatja ja 1 vastuvõtjaga maksab 299$ ning selle abil saaks katta ühe kuni 30m laiuse ja pikkusega ruumi (Marvelmind Robotics, 2019). IA süsteem oleks kergesti laiendatav ostes rohkem Mini-RX vastuvõtjaid, millest igaüks maksab 99$ (Marvelmind Robotics, 2019). Ala suurendamiseks peaks ostma paarikaupa HW v4.9 saatjaid, millest igaüks maksab 89$ (Marvelmind Robotics, 2019). Hinnad on võetud kevad 2020 seisuga.

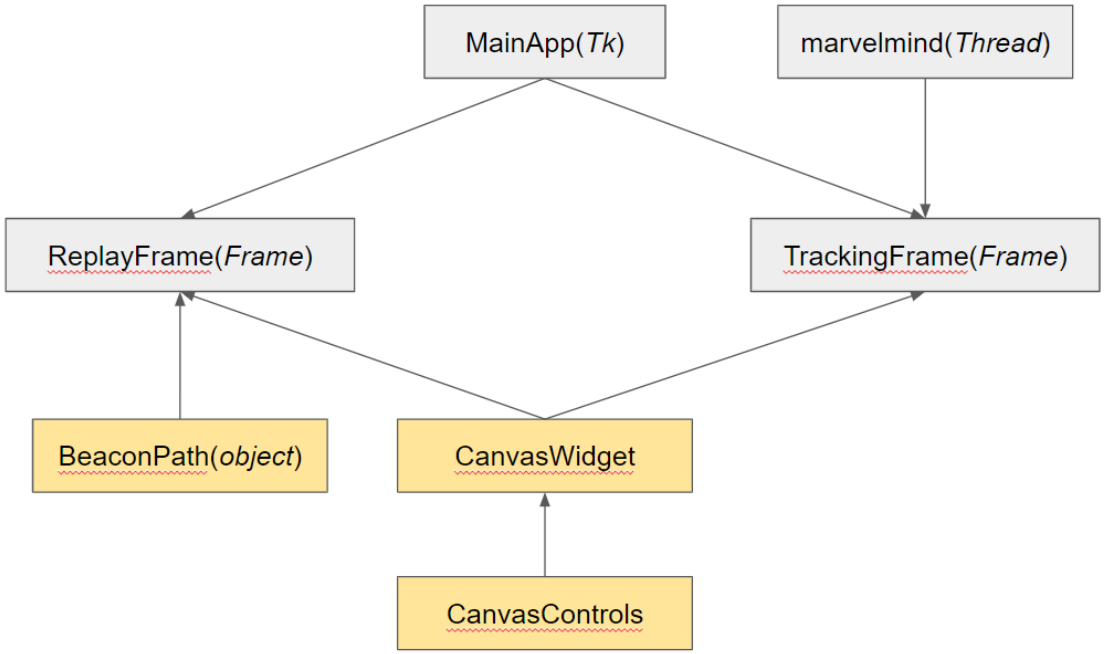
2.2 Nõuded rakendusele

Arendustöö suunamiseks ja vajalike nõuete väljaselgitamiseks, tegin mitu intervjuud Andres Kuusikuga, kes rakendust peamiselt analüüsimiseks kasutama hakkab. Intervjuude põhjal tekitasin nõuete nimekirja, mida rakendus täitma peab. Saadud nõuded on peamiselt funktsionaalsed, sest programmi praeguses faasis on kasutajate hulk maksimaalselt 5 inimest. Andres Kuusiku intervjuudest sain järgnevad nõuded:

- Esimene vaade rakenduse avamisel peab sisaldama poe plaani ja selle sisestamise võimalust
- Plaanil peab saama navigeerida hiirega lohistades ja *zoomida* sisse-välja punkti, kus hetkel kursor asub.
- Plaani peal peavad olema inimeste liikumispunktid (teisisõnu trajektoor)
- Liikuvate vastuvõtjate asukohti peab salvestama reaalajas iga sekundi tagant
- Plaanil vaadeldavat ajaperioodi peab saama kergelt muuta.
- Plaanil peab olema võimalik valida, milliste saatjate trajektoore näidatakse.
- Plaanile peab saama ka genereerida *heatmapi* vastavalt kõigile andmetele.
- Plaani peab saama kalibreerida vastavalt poe mõõtmetele.
- Saadud andmeid peab olema võimalik arvutite vahel eksportida ja importida.

2.3 Nõuete analüüs

Intervjuust ja nõuetest sai järeldada, et rakendus peab koosnema peamiselt kahest vaatest: reaalajas liikumise vaade (*TrackingFrame*) ja salvestatud andmete analüüsimise vaade (*ReplayFrame*). Kuna mõlemad vaated joonistavad *Canvas* elemendi peale kas trajektoori või vastuvõtja asukoha näitamiseks, siis vajavad mõlemad klassi *Canvas* elemendiga suhtlemiseks (vt Joonis 3).



Joonis 3. Rakenduse klassid ja nende kasutus.

Joonis 3 kujutatud *BeaconPath* hoiab informatsiooni vastuvõtjate trajektooride kohta, täpsemalt trajektoori alguspunkti, läbitud punkte, vektoreid nende punktide vahel ja vastuvõtjat identifitseerivat täisarvu. Nende abil saab trajektoorid *Canvas* elemendile joonistada vastavalt vaadeldavale ajaperioodile.

Samuti peab reaalajas jälgimisel vastuvõtjate asukohad *Canvas* elemendile joonistama. Marvelmindi teegi abil tekitatud lõim suhtleb *TrackingFrameiga*, et salvestada ja joonistada asukohainformatsiooni. Seda peab tegema mitte rohkem kui 1 sekundi tagant kõikide vastuvõtjate puhul, mistõttu *hedge.position()* meetodist ei piisa ja tuleb kasutada Marvelmindi lõimes hoitavat asukohtade puhvrit *hedge.valuesUltrasoundPosition*. Puhver sisaldab kõiki lähiajal mõõdetud asukohti ja selle suurus on muudetav lõime loomise hetkel, mistõttu saab puhvrit teha alati suuremaks kui liikuvate vastuvõtjate arv.

3 Rakenduse komponendid

Rakenduse loomiseks oli vaja kõigepealt luua funktsionaalne süsteem Marvelmind'i saatjate ja vastuvõtjate vahel ning see üles seada Tartu Ujula Konsumis. Sellele järgnes Marvelmind'i teegi katsetamine ülesseatud süsteemiga.

3.1 Marvelmind HW 4.9 saatjad ja Mini-RX vastuvõtjad

Positsioneerimissüsteemid koosnevad tavaliselt 2 osast: statsionaarsest signaali saatjast (*beacon*) ja liikuvast vastuvõtjast (*receiver*). Vastuvõtja omakorda saadab oma infot tsentraalsesse kolmandasse kohta ja selle põhjal saab näha reaalajas vastuvõtja asukohta saatjate suhtes. Samal põhimõttel töötab ka Marvelmind'i süsteem. Vastavalt ala suurusele, nähtavusele ja kasutatavale taustsüsteemile tuleb asetada vähemalt 2 saatjat, mis teatud sagedusega saadavad välja helisignaali. Saatjate poolt saadetud helisignaal põrkub vastu liikuvat vastuvõtjat ja selle põrkumise jaoks läinud aeg ja signaali tugevus laseb välja arvutada kauguse saatjast. Saatjad on suunatud ning sellest tingituna peab olema 2 saatjat ühel seinal. Saatjad lasevad välja signaali erineva helisagedusega, et saaks neid omavahel eristada ja signaali tugevuse põhjal selgitada kaugus esimesest saatjast ja kaugus teisest saatjast. Nende kahe kaugusega saab juba asetada vastuvõtja teatud punkti kindlas taustsüsteemis.

Taustsüsteeme on Marvelmind'il aga kahte tüüpi: IA (*Inverse Architecture*) ja NIA *(Non-Inverse Architecture*). Peamiseks erinevuseks nende süsteemide vahel on see, et NIA süsteem oskab automaatselt leida saatjate omavahelised kaugused, samas kui IA süsteemil peab saatjate vahelisi kauguseid ise mõõtma, seejärel need Marvelmind Dashboard'i sisestama ja alamjaotusteks jagama. Sellest tingituna on NIA süsteemi ülesseadmiseks vaja minimaalselt 3 saatjat, samas kui IA süsteem vajab minimaalselt kahte saatjat. NIA süsteemi probleemiks on aga vähene vastuvõtjate arv, mida süsteem toetada suudab ühes alamjaotuses (kuni 5). IA süsteem seevastu suudab toetada mitusadat vastuvõtjat ning sellest ka valik kasutada seda, kuna süsteem peab olema laiendatav ka suurematele poodidele, mis on tihedama klientuuriga.

Joonis 4. HW 4.9 saatja (paremal), Mini-TX saatja (keskel) ja ainult IA arhitektuuri toetav Mini-RX vastuvõtja (vasakul).

Pärast sobiva taustsüsteemi leidmist tuleb leida saatjate optimaalsed asukohad ruumis. Tuleb katta maksimaalne pind, samal ajal garanteerides, et igas punktis on vähemalt 2 saatjat vastuvõtjale otseselt nähtavad. See vähendab vigaseid andmeid ja vastuvõtja ebaregulaarset liikumist. Vastuvõtja teekonda saab hiljem töödelda maksimaalse kiiruse põhjal, välistades võimatult kaugel asuvad punktid, et elimineerida veel rohkem mõõtmisvigu ja luua usutavad kliendi teekonnad.

3.2 Marvelmind'i teek

Marvelmind'i poolt arendatud teek Python'i jaoks laseb luua ühenduse modemiga ja jookseb eraldi lõimel, valmis tagastama infot vastuvõtja asukoha kohta. Teegi kasutamiseks on vaja isendi loomisel ette anda ühenduse asukoht, mis Windowsi arvutitel on "\\.\COMX", kus X on pordinumber, mida saab leida kas Marvelmind Dashboard'ist või Windowsi Seadmehaldurist (Rudykh, 2019). Pärast isendi loomist ja isendi *start()* meetodi kutsumist saab pärida vastuvõtjate kohta informatsiooni (Rudykh, 2019).

Asukoha informatsiooni pärimiseks on olemas meetod on hedge.*position()*, mis tagastab viimase puhvris oleva vastuvõtja kohta nullpunkti suhtes relatiivsed X, Y ja Z-koordinaadid, identifitseerimisnumbri ja millisekudites modemi käivitamisest mööda läinud aja (Rudykh,

2019). Isendi loomise hetkel saab lisada ka filtri, et jälgida ainult kindlaid vastuvõtjaid, kuid see pole rakenduse puhul vajalik, sest on vajalik teada kõikide seadmete asukohta igal ajahetkel.

Teegil on ka muid meetodeid telemeetria, signaali tugevuse ja saatjate kauguste näitamiseks (Rudykh, 2019). Kuna rakendus on veel algfaasis, siis need meetodid ei osutunud vajalikeks kliendi nõuete täitmiseks. Tulevikus võib neid sellele vaatamata veel vaja minna, kui tahta juurde arendada süsteemi, mis lahendaks vastuvõtja ja saatja vahelise signaali puudumise või tõkestamise korral tekkiva hüppe.

4 Arendusprotsess

Rakenduse arendamine toimus mitmes faasis ning iga faasi käigus sai rakenduse tegelik eesmärk progressiivselt selgemaks. Põhjus selleks oli teadmiste puudumine Marvelmind Dashboard rakenduse võimaluste ja piirangute kohta. Samuti kulus oluline osa tööajast positsioneerimissüsteemi ülesseadmise peale, sest iga testimise jaoks kasutatud asukohal olid oma iseärasused.

4.1 Esimene prototüüp

Arendustöö programmi kallal algas 2019 aasta septembrikuus, kui esimesed saatjad testimiseks saadi. Esimesed saatjad töötasid küll NIA põhimõttel, mis reaalselt kasutusele võtmiseks ei sobinud, aga algseks katsetamiseks sobisid väga hästi. Peamine eelis NIA saatjate abil arendamiseks tulenes sellest, et NIA arhitektuuriga saatjad ei vaja füüsilisi mõõtmisi saatjate vahel süsteemi ülesseadmiseks, vaid kasutavad oma ultraheli süsteemi signaali omavahel põrgatades, automatiseerides seadmete omavahelise kauguse leidmise. Tänu automatiseeritud kauguste leidmisele oli süsteemiga tutvumine oluliselt lihtsustatud, mis oli ka peamine idee esimeste saatjate ostu taga.

Nende esimeste saatjate abil sai välja selgitada, kuidas Marvelmind'i teek ühildub modemiga ning kontrollida, et teegi poolt väljastatav info kattub dokumentatsioonis kirjeldatuga. Info salvestamiseks tegin lihtsa *for*-tsükli, mis iga 1 millisekundi möödudes kirjutas faili *hedge.position()* meetodi väljundi (vt Joonis 5). Hiljem rakendust edasi arendades ning nõuete täpsustamisel suurendasin asukoha salvestamise ajavahemiku 1 sekundini, kuna lõimes oleva info uuenemiskiiruse tõttu tekkis palju korduvaid andmeid ja väiksema ajavahemikuga salvestamist ei pidanud klient oluliseks. Selle *for*-tsükli abil kogusin Tartu Ujula Konsumis paar tekstifaili andmeid, et nende abil graafilist liidest lihtsam ehitada oleks (vt Joonis 5).

[id, x, y, z, angle, timestamp]

Joonis 5. *hedge.position()* meetodi väljund.

Graafilise liidese jaoks kasutasin *tkinteri* teeki, mis võimaldab Python'is luua graafilisi liideseid. Tegin teegi abil akna, mis koosnes *Canvas* elemendist, mille peale joonistasin ruudu vastavalt *hedge.position()* funktsiooni väljundile (vt Joonis 5). Nii sai testimiseks

realiseeritud algne reaalajas jälgimise funktsionaalsus, mis toetas ainult ühte liikuvat vastuvõtjat.

Kuna *hedge.position()* funktsiooni väljund sisaldab ka liikuva vastuvõtja ID-d, siis tegin *Canvase* elemendi kõrvale ka koha, kus saab erinevate ID-dega vastuvõtjate trajektoore sisse-välja lülitada. Selle realiseerimiseks tuli uurida kuidas *tkinteri Canvas* element täpsemalt töötab. Nimelt iga joonistusfunktsioon paneb joonistatava elemendi magasini, mille sisu tervenisti uuesti *Canvas* elemendile joonistatakse. Sellest tulenevalt on vaja iga kord erinevaid trajektoore sisse-välja lülitades *Canvasel* kutsuda välja *clear()* funktsioon, mis kõik varem tehtud jooned ära kustutaks ja seejärel joonistada kõik nähtavad elemendid uuesti. Vastasel juhul jääksid vanad jooned magasini ja joonistataks koos uute joontega. Lisaks *clear()* funktsioonile on ka funktsioon *remove()*, millele saab argumendiks anda kustutatavat kujundit märgistava täisarvu.

4.2 Töö peale intervjuusid

Peale nõuete täpsustamist ilmnes, et on vaja ümber teha senine loogika *Canvas* elemendile teekonna joonistamiseks, kuna nõutud funktsionaalsus vajas paindlikumat lahendust. Sellest tulenevalt otsustasin hakata kasutama *numpy* teegi maatrikseid, sest teegis olemasolevad matemaatika funktsioonid lihtsustaks nõuetest tulenevate funktsionaalsuste implementeerimist, näiteks muutujaga vektorite korrutamine, mis on *zoom* funktsiooni juures vajalik.

4.2.1 Eksportimine ja importimine

Kuna andmete analüüsimise vaade kasutab mitut erinevat tüüpi faile, siis otsustasin kasutada *zip* formaati andmete eksportimise võimaldamiseks. See *zip*-fail sisaldab 3 faili:

- Tekstifail, mis sisaldab *TrackingFramei* poolt modemiga suhtleva lõime seest reaalajas loetud informatsiooni, mis salvestatakse reaalajas vastuvõtjaid jälgiva vaate poolt (vt Joonis 3).
- Teine tekstifail sisaldab informatsiooni algpunkti asukoha kohta *Canvase* peal, selle orientatsiooni ning kalibreerides leitud pikslite ja reaalelu meetrite vahelist suhet. Lisaks sisaldab see fail ka põrandaplaani faili nime, et meetod seda samast *zip*-failist lugeda oskaks.
- Kolmas fail on lihtsalt asukoha põrdandaplaani pildifail, mis on PNG-formaadis.

Eksportimise ja importimise meetodite hoidmise jaoks tegin eraldi faili *exporter.py*, mis rakenduse peamisesse faili teegina kaasasin. Nii importimise meetod kui eksportimise

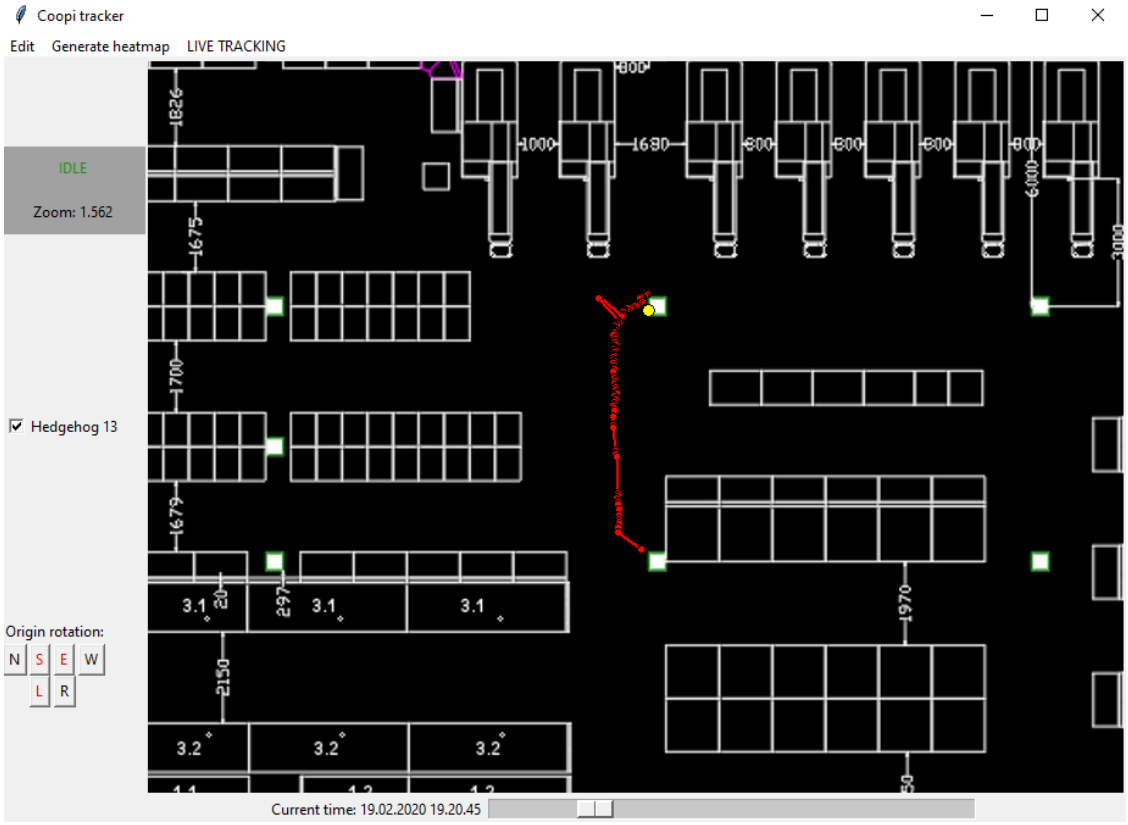
meetod kasutavad Python'is olemasoleva teegi *ZipFile()* funktsiooni *zip*-failide loomiseks, nendesse kirjutamiseks ja nendest lugemiseks (Kumar, 2020).

4.2.2 Teekonna salvestamine ja näitamine (*BeaconPath* klass)

BeaconPath klass sisaldab kõike vastuvõtjaga seotud infot, sealhulgas liikumise algpunkti, vektoreid sellest punktist *numpy* maatriksi kujul, failist sisse loetud punktide listi ja vastuvõtjat identifitseerivat täisarvu. Vektorid maatriksis sisaldavad X- ja Y-telje suhtes liikumise informatsiooni, aga punktide massiiv sisaldab ka ajahetki, millal ja mis punktis vastuvõtja mingil ajahetkel oli.

Järgnevalt muutsin olemasolevat loogikat loodud klassi abil saatja liikumise trajektoori *Canvas* elemendile joonistamiseks. Selle jaoks tegin funktsiooni *draw_lines()*, mis võttis argumentideks alguspunkti ja vektorite listi, mida alguspunktist alates joonistada. Iga vektor korrutatakse läbi pikslite ja reaalelu meetrite vahelise suhtega, et trajektoor läheks kokku taustal oleva põrandaplaaniga.

Hiljem lisasin ka võimaluse näidatavat trajektoori keerata ja peegeldada, kuna taustal olev pilt ei pruugi alati olla sama rotatsiooniga, mis salvestatud teekond. Selle jaoks tegin nupud, mis mõjutavad *draw_lines()* meetodi poolt tehtavaid operatsioone vektorite ja punktide vahel (vt Joonis 6).



Joonis 6. Trajektoor põrandaplaani peal *ReplayFrame* vaates.

Teekonna salvestamist oli samuti vaja ümber teha, sest *hedge.position()* meetod väljastas ainult viimase loetud vastuvõtja positsiooni. Antud meetod ei sobi, kuna on vaja vaadata kõikide vastuvõtjate liikumist paralleelselt iga sekund. Ümbertegemine ei olnud aga suur probleem, kuna *hedge.position()* meetod tagastas lõimes oleva puhvri viimase elemendi ning puhvris hoitavaid positsioone saab ka ilma *hedge.position()* meetodita lugeda. Lõimes oleva puhvri suurust saab lõime loomise hetkel muuta ning rohkemate vastuvõtjate puhul tuleb seda ka teha. Iga sekundi tagant puhvri lugemise jaoks kasutasin teeki *schedule* (Bader, 2020).

4.2.3 Plaanil navigeerimine

Edasi oli tarvis tegeleda põrandaplaanil navigeerimise implementeerimisega. *Tkinteri* teeki on sisse kirjutatud juba vajalikud funktsioonid, et terve *Canvas* elemendi sisu ringi liigutada. Nendeks funktsioonideks on *scan_mark()* ja *scan_dragto()*, millest esimene märgib x- ja y-koordinaatidega punkti, kust tirimist alustatakse, ja teine tirib kogu *Canvast* mingi punkti suunas, mis on samuti x- ja y-koordinaatidega tähistatud.

Lisaks nendele funktsioonidele oli vaja *Canvas* panna kuulama kursori tegevusi, et nende põhjal liigutamist teha. *Tkinteri* teeki on juba valmis kirjutatud vajalikud *event listenerid*, mis lasevad defineerida funktsioonide välja kutsumisi hiirenupu alla vajutamisel (Button-1), hiire liikumisel nuppu all hoides (B1-Motion) ja hiirenupu lahti laskmisel (ButtonRelease-1) (effbot.org, 2020).

Kursori liigutamise kuulamisele ja *Canvase* sisseehitatud funktsioonide abil liigub terve *Canvase* elemendi sisu vastavalt kursori tegevustele. Hiire vasakut nuppu alla vajutades hakatakse jälgima hiire liikumist ja vastavalt hiire liikumisele liigub ka terve *Canvase* elemendi sisu. Pärast hiire nupu lahtilaskmist lõppeb jälgimine ja *Canvase* sisu jääb paika sinna, kus ta enne nupu lahti laskmist oli.

Zoom funktsiooni implementeerimine oli aga kordades keerulisem, kuna piltide skaleerimine ei ole toetatud meetodi *Canvas.scale()* poolt. Kõik *Canvasele* joonistatud jooned ja ringid küll skaleeruvad *Canvas.scale()* meetodi abil, aga nende algseid asukohti ja suurust selle juures muudetakse, mitte ei skaleerita tervet taustsüsteemi. Skaleerimisel kasutatakse muutujat *scale* korrutamistehtes, millest tulenevalt langeb *Canvasel* olevate elementide asukohtade täpsus (vt Joonis 7).

$$((coord - offset) * scale + offset)$$

Joonis 7. Valem, mida *Canvas.scale()* meetod kasutab koordinaatide muutmiseks.

Täpsuse kadumise probleemi lahendamiseks defineerisin *Canvase* peal 0-punkti kasutades *Canvas.create_text()* meetodit ning paigutasin kõik elemendid vastavalt selle asukohale (FooBar167, 2018). Põhjus teksti kasutamiseks oli see, et *Canvas.coords()* funktsioon tagastab teksti puhul ainult alguspunkti koordinaadid, samas kui *Canvas.create_rectangle()* või mõne muu kujundi loova funktsiooni puhul tagastatakse nii algus- kui lõpp-punkt, ning elemendi keskpunkt asub nende kahe punkti vahel (Tutorialspoint, 2020). Selle nullpunkti asukoha saamise jaoks defineerisin meetodi *get_zero_reference()* ning paigutasin taustal oleva põrandaplaani vastavalt selle meetodi väljundile.

Lisaks põrandaplaani positsioneerimisele vajas praegune vektorite joonistamise süsteem ümbertegemist, sest see jooksutatakse iga *zoomimise* korral ja ei võtnud arvesse seda, kui palju taustal olev põrandaplaan suurendatud on. Seetõttu ei muutunud joonistatud vektorite pikkus vastavalt *zoomimisele* ning vektorite lõpp-punktide asukohad olid valed. Selle kooskõla puudumise parandamiseks oli vaja joonistatava vektori pikkus korrutada taustal oleva põrandaplaani hetkese *scale* muutujaga.

4.2.4 Ajaperioodi määramine

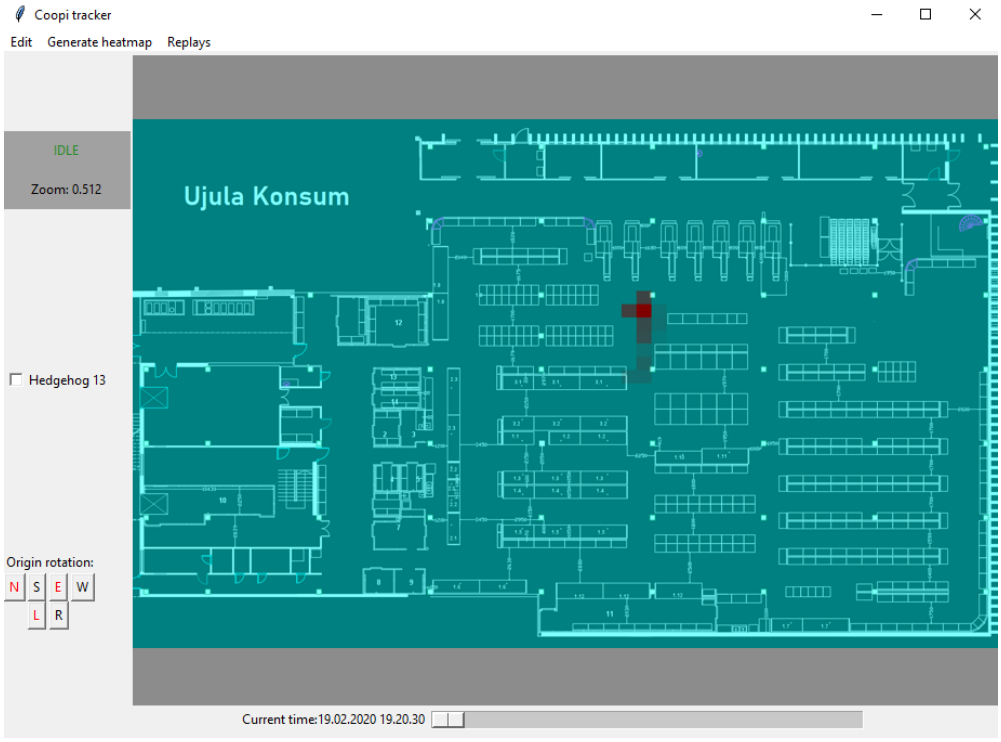
Marvelmind'i teek annab kahjuks ainult informatsiooni, et mitu millisekundit pärast modemi käivitamist kuskil mingi vastuvõtja asus. Sellest tingituna pidin võtma arvuti süsteemiaja lindistamise alustamise hetkel. Salvestatud kellaaja, kuupäeva ning punktide millisekundite abil sai liitmistehte abil täpse ajavahemiku, mis punktis mingi vastuvõtja kindlal kellaajal ja kuupäeval oli. Vaadeldavat kellaaega ja kuupäeva näidatakse tõmmatava slaideri kõrval (vt Joonis 8).

Joonis 8. Ajaperioodi muutmise slaider.

4.2.5 *Heatmapi* genereerimine

Heatmapi genereerimiseks oli mitu erinevat võimalust ning palju erinevaid variante sai selle funktsionaalsuse realiseerimiseks läbi katsetatud, paralleelselt kliendiga suheldes. Esimene versioon oli meetod, mis jagas taustapildi sektsioonideks ning seejärel joonistas värvitud pildid vastavalt punktide tihedusele nendes sektsioonides. Piltide loomiseks ja *Canvas* elemendile joonistamiseks kulus aga liiga palju ressurssi, ning väga väikeseid pilte on praktiliselt võimatu teha ilma *tkinterit* kokku jooksutamata. Seetõttu ei saanud selle variandiga luua piisava täpsusega *heatmappi*.

Teine katsetus oli genereerida üks suur pilt kasutades pildi rotatsiooni, pikslite ja meetrite vahelist suhet ning alguspunkti asukohta põrandaplaani peal. Tegin koopia põrandaplaanist *Image.copy()* meetodiga ning sarnaselt esimesele katsetusele jagasin selle sektsioonideks. Sektsioonide, nullpunkti asukoha ja *BeaconPath.points* järgi sain leida igas sektsioonis vastuvõtja poolt veedetud aeg ning salvestada see maatriksisse. Järgnevalt värvisin sektsioonide külgede pikkuste järgi ja sektsioonis veedetud aja järgi põrandaplaanile läbipaistva värviga ristkülikud *ImageDraw.rectangle()* meetodiga. Tulemuseks tuli rahuldava täpsusega *heatmap* (vt Joonis 9).



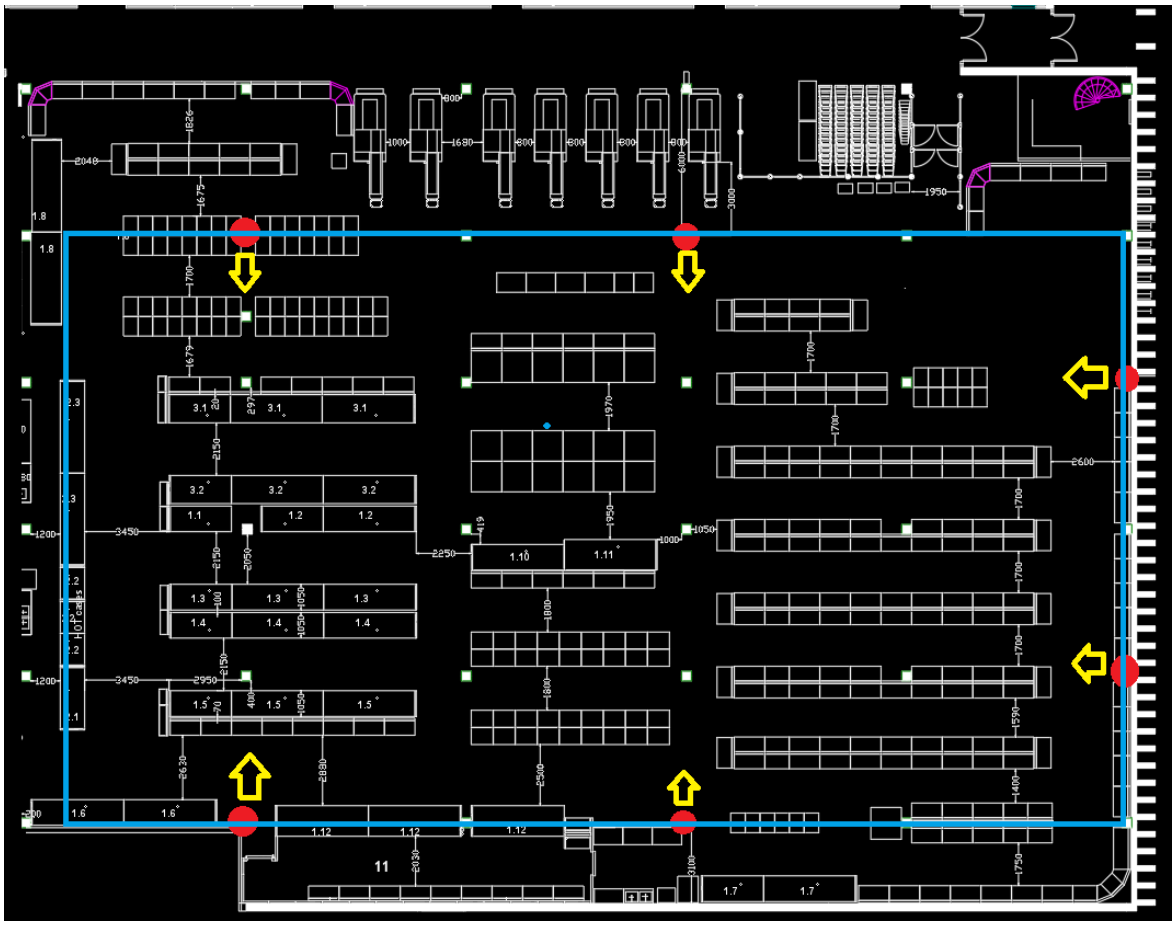
Joonis 9. Vaade rakendusest koos genereeritud *heatmapiga*.

5 Rakenduse testimine

Algselt oli plaanis rakendust testida Tartus Ujula Konsumis, ning kuni veebruarikuuni oli see võimalik. Riigis kehtestatud eriolukorra tõttu pidime aprillikuus osad saatjad demonteerima, et saada kodutingimustes rakenduse funktsionaalsust testida.

5.1 Tartu Ujula Konsum

Kogusin 2 tekstifaili *hedge.position()* väljastatud väärtusi algse testimise jooksul, sest enamus ajast läks Marvelmind'i enda süsteemi sätestamisele. Lõplik versioon kasutas ainult 6 saatjat algsest plaanitud 10st (vt Joonis 10). Põhjus vähemate saatjate kasutamiseks tulenes sellest, et teoorias planeeritud 10 saatjaga süsteem ei töötanud ja hakkasime töötavast kahe saatjaga süsteemist aeglaselt suurema arvu saatjatega süsteemi üles ehitama. Selle käigus õppisime kasutama TDMA (*Time Division Multiple Access*) järgnevuste süsteemi, et koordineerida ultraheli saatmise aegu saatjate vahel interferentsi vähendamiseks (Manual, 2019). Enne koroonaviiruse pandeemiat oli meil 6 saatjaga süsteemiga enamus poest kaetud ning seetõttu tahtsime suurema vastuvõtjate arvuga testida olemasolevat süsteemi.



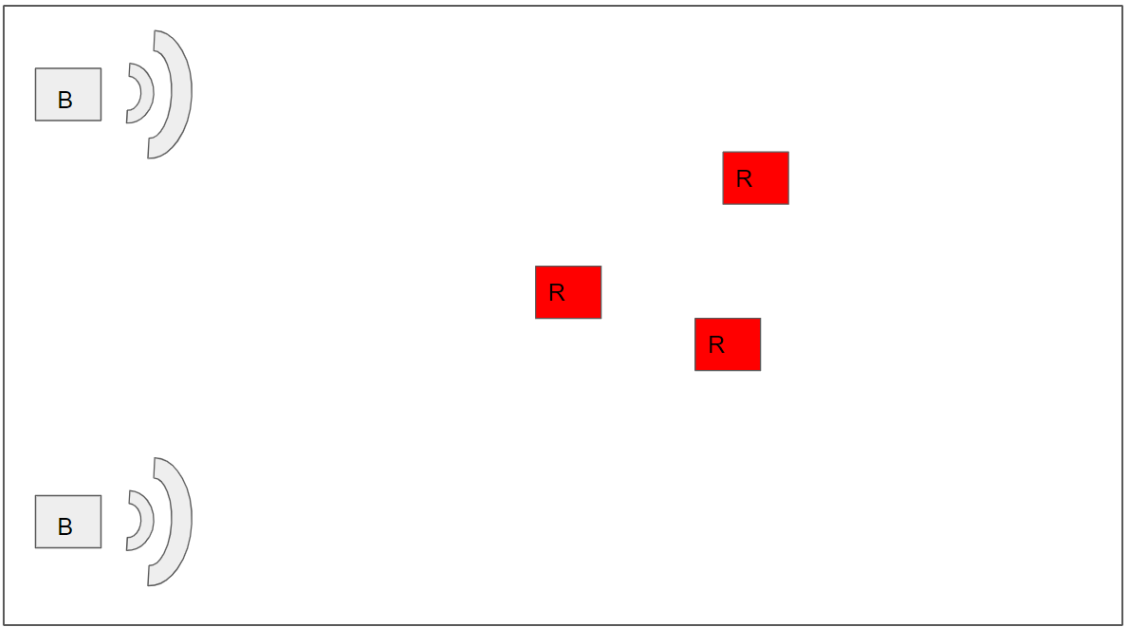
Joonis 10. Ujula Konsumi põrandaplaan koos ülesseatud 6 saatjaga süsteemiga.

Põhjus selliseks saatjate asetuseks on poes olevad seinad ja riiulid, kuna nad blokeerivad saatjate poolt välja lastud ultrahelisignaali. Joonisel paremal seinas olevad 2 saatjat töötavad praktiliselt 0% signaali kaotusega, samas ülejäänutel tekivad järsud hüpped asukohas, mis ei ole kooskõlas reaalse vastuvõtja teekonnaga. Järsud hüpped on põhjustatud signaali puudumise, põrkumise või tõkestamise tõttu nendes kohtades. Olukorda saaks parandada saatjate kõrgemale tõstmisega, et suurendada ala, kus on suhtlus tõkestamata, aga see on päris keeruline tavalise redeliga ning sealhulgas peavad saatjad jääma samale kõrgusele üksteise suhtes, tehes paigutamise veelgi keerukamaks. Põhjus saatjate samal kõrgusel hoidmiseks on täpsus.

5.2 Koroonaviirus ja kodus testimine

Märtsis oli plaanitud minna Tartu Ujula Konsumisse rakendust katsetama, kuid riigis kehtestatud eriolukorra tõttu jäi see ära. Selle asemel võtsin Tartu Ujula Konsumist ülejäänud saatjad ja vastuvõtjad ning seadsin enda elukohas üles väga primitiivse süsteemi, et katsetada ja edasi arendada rakenduse reaalajas asukoha näitamist.

Seadsin üles 2 saatjaga ja 3 vastuvõtjaga süsteemi. Kahest saatjast piisas, sest kasutatav ruum oli väike ja nähtavus saatja ja vastuvõtja vahel oli alati garanteeritud. Saatjad olid paigutatud ühte ruumi seina umbes 3 meetri kõrgusele (vt Joonis 11).



Joonis 11. Kodus ülesseatud süsteem 2 saatja ja 3 vastuvõtjaga

Selle viimase katsetamise jooksul sai peamiselt viimistletud reaalajas jälgimise võimalust.

6 Edasiarendused tulevikus

Pärast valminud tööd läheb programmi arendamine edasi ning edasiarenduste jaoks on juba plaane tehtud. Eesmärgiks on teha lihtsasti ülesseatav süsteem, mida ettevõtetele müüa, ning toode on suuresti veel algfaasis. Edasiarendused fokuseerivad pigem programmi laiendamise lihtsustamisele ja töökindluse parandamisele.

6.1 Lähiaja laiendused

Hetkel on andmete kogumise tingimuseks see, et modem on ühendatud arvutiga. Selle parandamiseks oleks vaja luua viis hoida modem kogu aeg ühenduses ning andmeid salvestamas. Variant selleks oleks modem Raspberry Pi külge ühendada ja see asetada koos modemiga kuskile poodi. Raspberry Pi salvestaks sellele kirjutatud rakenduse abil ööpäevringi andmeid ning neid saaks hiljem pärida suhtlemisliidese abil arvutisse analüüsimiseks.

Samuti ei pruugi *tkinteri* abil tehtud graafiline liides tulevikus toetada kõike nõutud funktsionaalsust. Ülemineku variandiks oleks teha Python'i rakendusest *back-end* leheküljele, mis võimaldaks erinevaid veebiarenduses kasutatavaid liideseid integreerida. *Canvas* elemendi asemel saaks kasutada mõnda WebGL liidest (nt *three.js*), mis protsessoriaja asemel kasutaks graafikakaarti, parandades rakenduse võimekust. Samuti oleks kujunduse modifitseerimine oluliselt kergem HTML-i ja CSS-i kasutades.

6.2 Kaugem tulevik

Kui positsioneerimissüsteem muutub piisavalt töökindlaks ja saab integreeritud näiteks Coopi nutikäppadesse, siis oleks võimalik näidata kliendile ostupuldi peal reklaami vastavalt kliendi asukohale poes. Sealhulgas saaks näidata ka allahindlusi või muud konkreetse letiga seotud infot.

Kui poes oleks kõik riiulid süsteemis ära märgistatud, oleks võimalik läbi ostupuldi inimesi juhatada soovitud tooteid hoidva riiulini. See tuleks kasuks suuremates poodides, kus on palju erinevaid tootekategooriaid ja sellest tingituna ka riiulite arv suurem.

7 Viidatud kirjandus

Bader, D. (4. 4 2020. a.). *schedule*. Allikas: readthedocs: https://schedule.readthedocs.io/en/stable/

Cominelli, M., Patras, P., & Gringoli, F. (2019). *Dead on Arrival: An Empirical Study of the Bluetooth 5.1 Positioning System.* EBSCOhost. Tsiteeritud 1. 12 2019. a.

effbot.org. (13. 3 2020. a.). Allikas: tkinter events and bindings: https://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

FooBar167. (2. 1 2018. a.). *Move and zoom a tkinter canvas with mouse*. Allikas: stackoverflow: https://stackoverflow.com/questions/25787523/move-and-zoom-a-tkinter-canvas-with-mouse/48069295#48069295

infsoft. (2. 12 2019. a.). Kasutamise kuupäev: 2. 12 2019. a., allikas Indoor Positioning, Tracking and Indoor Navigation with Wi-Fi: https://www.infsoft.com/technology/positioning-technologies/wi-fi

Kumar, N. (13. 4 2020. a.). *GeeksforGeeks*. Allikas: Working with zip files in Python: https://www.geeksforgeeks.org/working-zip-files-python/

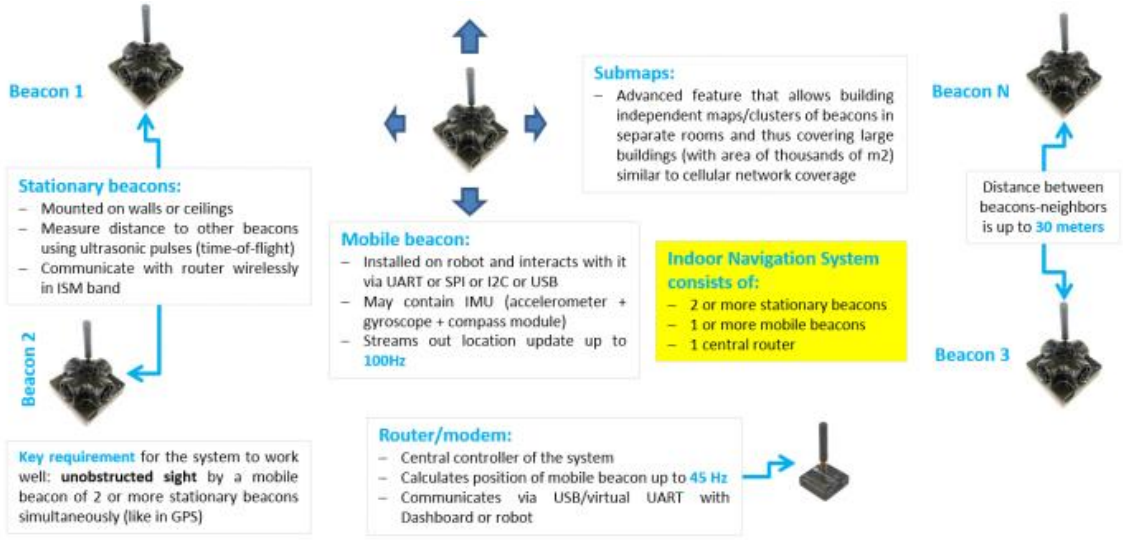
Manual. (29. 9 2019. a.). Allikas: www.marvelmind.com: https://marvelmind.com/pics/marvelmind_navigation_system_manual.pdf

Marvelmind Robotics. (2. 12 2019. a.). Kasutamise kuupäev: 2. 12 2019. a., allikas Starter Set IA 2D TDMA: https://marvelmind.com/product/starter-set-ia-03-2d-tdma/

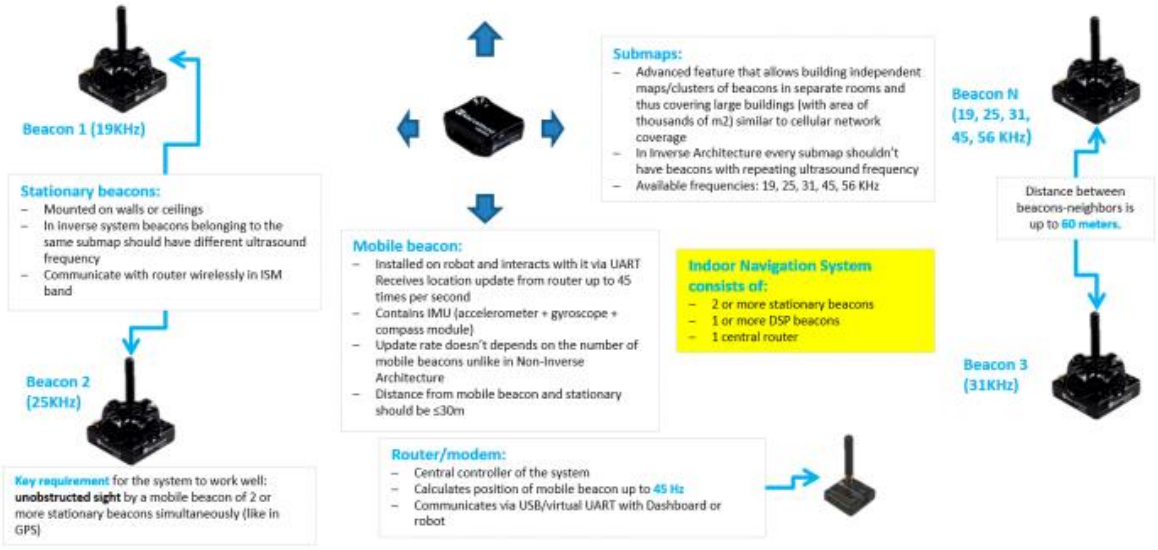
Rudykh, A. (2. 12 2019. a.). *GitHub.* Kasutamise kuupäev: 2. 12 2019. a., allikas marvelmind.py: https://github.com/MarvelmindRobotics/marvelmind.py

Tutorialspoint. (2 2020. a.). Allikas: Python GUI programming: https://www.tutorialspoint.com/python/python_gui_programming.htm

Lisad



Lisa 1. NIA (*Non-Inverse Architecture*) süsteemi tööpõhimõte ja ülesseadmise kirjeldus (Manual, 2019).



Lisa 2. IA (*Inverse Architecture*) süsteemi tööpõhimõte ja ülesseadmise kirjeldus (Manual, 2019).

I. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, _____Karl Kuusik_____,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose _____Siseruumides positsioneerimise tarkvara loomine_____, mille juhendaja on ____Alo Peets_____, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Karl Kuusik
07.05.2020