UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tomaž Bevk

Sledenje s sprotnim izbiranjem značilnic za sočasno lokalizacijo in kartografiranje z eno samo kamero

DIPLOMSKO DELO NA
UNIVERZITETNEM ŠTUDIJU

Mentor:
prof. dr. Aleš Leonardis

Ljubljana 2008

Zahvala

Za mentorstvo se zahvaljujem prof. dr. Alešu Leonardisu. Za izkazano pomoč bi se prav tako zahvalil mag. Matjažu Joganu. Za prijazne odgovore na konkretna vprašanja v zvezi z obravnavano problematiko bi se rad zahvalil tudi dr. Helmutu Grabnerju iz Tehnološke Univerze v Grazu. Za pomoč pri slikovnih elementih bi se zahvalil mojemu bratu Tadeju Bevku. Zahvala gre tudi moji družini in pa vsem prijateljem, ki so mi v času pisanja diplomske naloge nudili še kako potrebno moralno podporo.

Kazalo

Povzetek

Razvoj algoritmov za uporabo v avtonomnih mobilnih robotih je ena glavnih tem, s katerimi se danes ukvarja področje računalniškega vida. Eden od pristopov je uporaba koncepta tako imenovane sočasne lokalizacije in kartografiranja (*ang.* simultaneous localization and mapping - SLAM). SLAM je proces, v katerem robot inkrementalno gradi zemljevid svoje okolice, obenem pa ta zemljevid uporablja za določanje svoje lokacije. Informacijo o okolici lahko robot pridobi z uporabo različnih vrst senzorjev. V tem delu se osredotočamo na sisteme, ki v ta namen uporabljajo eno samo kamero. Ena od nalog, ki jih mora rešiti sistem SLAM, je sledenje objektom v prostoru. Ker se robot po prostoru premika, lahko pričakujemo, da se bo izgled posameznih objektov z vidika robota spreminjal. V ta namen smo implementirali sledilnik, ki je sposoben prilagajanja spremembam videza sledenega objekta. To smo dosegli s prevedbo problema sledenja na klasifikacijski problem in uporabo nekaterih metod strojnega učenja. Sledilnik smo integrirali v že obstoječo aplikacijo za SLAM, sistem MonoSLAM. Rezultati preizkusov so pokazali, da takšna kombinacija v nekaterih primerih daje boljše rezultate, kot osnovni sistem MonoSLAM, v drugih pa je prav sposobnost prilagajanja lastnost, zaradi katere sistem odpove. V zaključnem delu so podani nekateri predlogi za nadaljnje izboljšave in raziskovanje.

Ključne besede

avtonomni mobilni roboti, boosting, računalniški vid, sledenje, strojno učenje

Abstract

Development of algorithms for autonomous mobile robots is one of the main subjects that concerns the field of computer vision. One of the approaches introduces a concept called simultaneous localization and mapping (SLAM), by which a mobile robot can build a map of an environment, and at the same time, use this map to compute its own location. The robot can gather the information about the environment with the use of many different sensors. In our work we focus on systems that use a single camera for this purpose. One of the tasks that has to be tackled by a SLAM system, is tracking of objects that are present in the environment. Because we expect the robot to be constantly moving, we can presume that, from the robot's point of view, the appearance of the tracked objects will change. In order to resolve this problem we implemented a tracker that is capable of adapting to the changing appearance of the tracked object. We achieved this by translating the problem of tracking into a classification problem and by using some of the methods provided by the field of machine learning. We integrated the obtained tracker into an allready existent application of SLAM, called MonoSLAM. Results of our tests showed that such combination performs better in some situations than does the basic MonoSLAM, although in other situations the very ability of adapting makes the system fail. In the concluding section we point out some possible improvements and give suggestions for further research.

Keywords

autonomous mobile robots, boosting, computer vision, machine learning, tracking

1. Uvod

Računalniški vid je veja računalništva, ki se ukvarja z interpretacijo slik. Osnovni cilj računalniškega vida je z obdelavo slike ali zaporedja slik pridobiti informacijo o svetu okoli nas, oziroma o objektih v naši okolici. Računalniški vid sega v različna področja, kot so umetna inteligenca, robotika, procesiranje signalov, razpoznavanje vzorcev pa tudi psihologija. Tako se kot multidisciplinarna disciplina razvija že prek trideset let in obstaja že vrsta sistemov, ki s pridom uporabljajo dognanja s tega področja. Kljub temu pa problem posnemanja vida, kot ga pojmujemo ljudje, še zdaleč ni rešen.

Eno od področij, ki je danes v vzponu, je uporaba računalniškega vida v avtonomnih robotih. Zgodovina avtonomnih robotov sega nekam v drugo polovico prejšnjega stoletja. Ena prvih uspešnih aplikacij se je odvila v laboratoriju za mehansko inženirstvo Tsukuba (Tsukuba Mechanical Engineering lab). Na posebej pripravljeni stezi je avtonomno vozilo s sledenjem belim označbam ob progi doseglo hitrosti do trideset kilometrov na uro. Leta 1980 je robot iz tovarne avtomobilov Mercedes-Benz, ki ga je zasnoval Ernst Dickmans, dosegel prepričljivih sto kilometrov na uro. Upravljal ga je sistem, ki je deloval zgolj na podlagi vida, sposoben pa je bil vožnje po ulicah brez prometa. Ob tem dosežku je Evropska Komisija podprla projekt EUREKA Prometheus (EUREKA Prometheus Project), vreden osemsto milijonov dolarjev.

Danes se avtonomni roboti uporabljajo predvsem v vesoljski in vojaški industriji, ter v medicini, avtomobilski industriji in proizvodnji. Ideja za prihodnost pa je recimo osebni avtomobil, ki bi bil sposoben samostojno pripeljati od ene točke na zemljevidu do druge.

Eden od najbolj obetavnih pristopov za uporabo v avtonomnih mobilnih robotih je sočasna lokalizacija in kartografiranje (*ang.* Simultaneous Localization and Mapping - SLAM). V njem robot inkrementalno gradi zemljevid svoje okolice, obenem pa ta zemljevid uporablja za določanje svoje lokacije. Sam koncept sočasne lokalizacije in kartografiranja je bil razvit že v devetdesetih letih prejšnjega stoletja [1], [2]. V splošnem lahko robot v sistemu SLAM za zaznavanje okolice uporablja različne senzorje, kot je na primer laserski merilec razdalje

(*ang.* laser range finder), ultrazvočni detektor (*ang.* sonar) ali kamera. Morda se zdi presenetljivo, da uporaba kamer v SLAM do zdaj ni bila glavna točka interesa v razvoju tega sistema, saj se prav uporaba vida zdi najbolj naravna izbira. Vid je namreč glavno čutilo, ki ga ljudje in nekatere živali uporabljamo za navigacijo. Problem je bil predvsem v pomanjkanju ustreznih algoritmov, ki bi omogočali obdelavo velike količine podatkov, ki jih dobimo iz kamere. S hitrim povečanjem zmogljivosti računalnikov in pocenitvijo visoko kakovostnih kamer, ter novimi dosežki na področju računalniškega vida, katerim smo priča v zadnjih letih, pa se je le razvila ideja o sistemu SLAM, kjer bi za edino čutilo, ki ga premore robot, uporabili kamero oziroma računalniški vid.

S problemom SLAM, kot enim ključnih problemov v področju mobilne robotike, se stroka ukvarja že več kot 20 let [1], [2]. V zadnjih desetih letih pa se razvija predvsem ideja, da bi, kot edini senzor robota, uporabili eno samo kamero. V sledečem odstavku podajamo kratek pregled del s področja sočasne lokalizacije in kartografiranja, od sistemov, ki uporabljajo stereoskopski vid v kombinaciji z notranjimi meritvami[1], pa vse do sistemov, ki realizirajo zgoraj omenjeno idejo (SLAM z eno samo kamero).

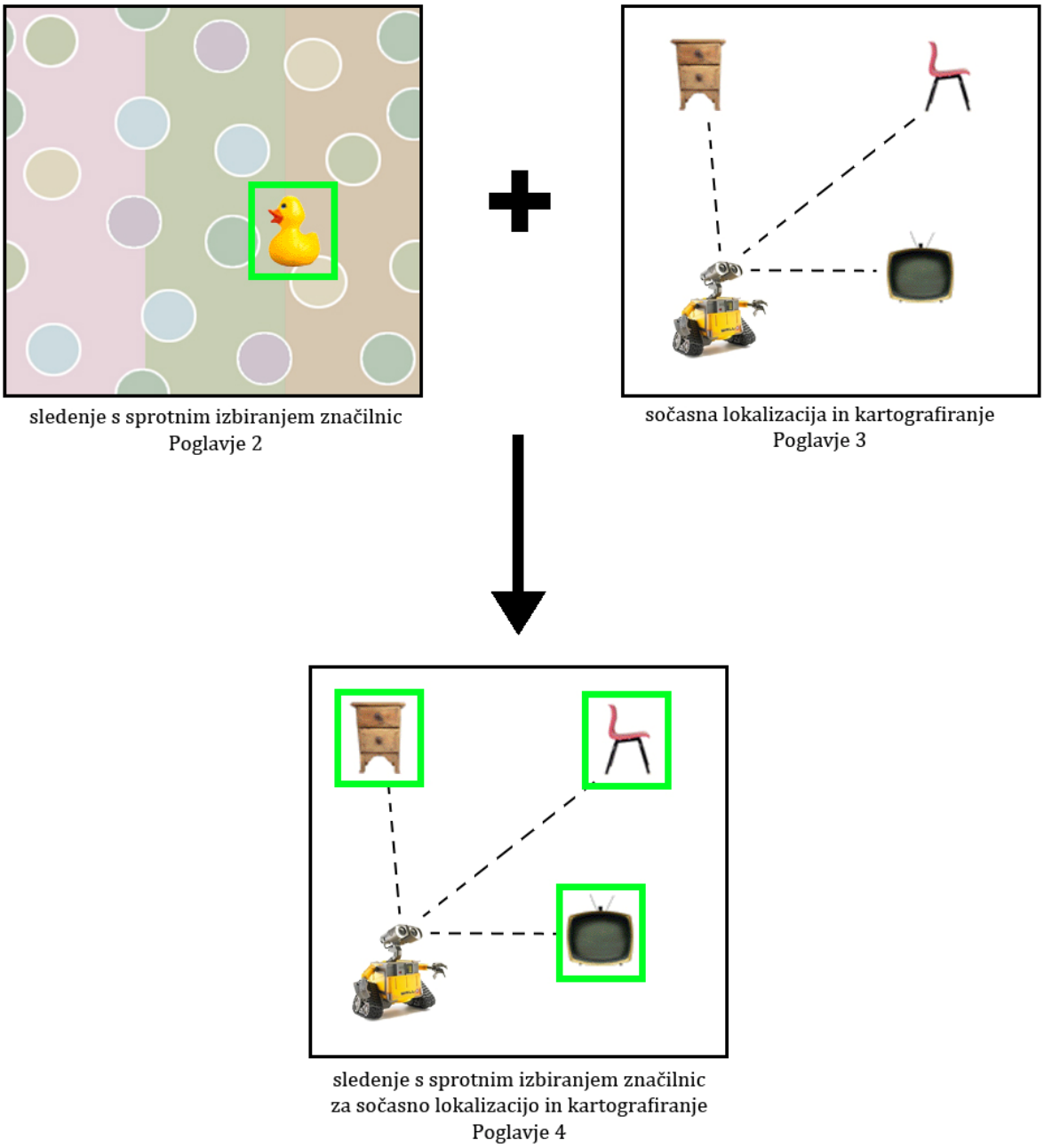
Eden prvih sistemov SLAM na podlagi vida, ki je deloval v realnem času (5 Hz), je uporabljal aktivni stereoskopski vid v povezavi z notranjimi meritvami robota [3]. Gibljiva kamera je omogočala široko vidno polje, zaznavanje značilnih točk iz okolice pa je služilo popravljanju ocene položaja robota, dobljene z notranjimi meritvami. Jung in Lacroix [4] sta v svojem delu predstavila uporabo stereoskopskega vida za lokalizacijo zrakoplova in grajenje podrobnega zemljevida terena nad katerim je zrakoplov letel. Njun algoritem je slike sicer obdeloval zaporedno, vendar ni deloval v realnem času. Eustice in sodelavci [5] so v svojem delu uporabili eno samo kamero v povezavi z vztrajnostnimi senzorji za lokalizacijo podvodnega plovila in grajenje natančnega zemljevida dna. Sim in sodelavci [6] so z uporabo algoritma FastSLAM [7] v kombinaciji z značilnicami SIFT [8] razvili sistem SLAM, ki je deloval le na podlagi vida in je bil sposoben grajenja velikih zemljevidov. Izračun značilnic SIFT je računsko zelo zahteven in s hitrostmi 0,1 slike na sekundo sistem ni bil primeren za aplikacije, ki zahtevajo delovanje v realnem času. Foxlin [9] je z vnaprej postavljenimi označbami v prostoru in z uporabo visoko zmogljivih notranjih senzorjev v kombinaciji z eno samo kamero dosegel prepričljivo natančnost lokalizacije robota. Slabost njegovega pristopa pa je v tem, da je sistem omejen na delovanje v vnaprej pripravljenem prostoru. Davison in sodelavci [10] so prvi uspeli realizirati sistem SLAM z uporabo ene same kamere, ki je deloval v realnem času. Njihov pristop je odprl vrata nadaljnjim raziskavam [11], [12], [13].

V nadaljevanju se osredotočamo predvsem na sistem SLAM, ki ga so ga razvili Davison in sodelavci [14] in omogoča sočasno lokalizacijo in kartografiranje z uporabo ene same kamere, brez uporabe notranjih meritev in deluje v realnem času.

[1]Notranje meritve so meritve, ki direktno ocenijo premike robota. (npr. števec prevožene razdalje).

V sistemu SLAM robot svoje premike ocenjuje na podlagi relativnih razdalj do različnih objektov v okolici v različnih časovnih trenutkih. To pa pomeni, da mora biti sposoben slediti različnim objektom skozi neko zaporedje slik. To nas pripelje na drugo področje računalniškega vida, ki se ukvarja s sledenjem objektom.

V pričujočem delu bomo podrobneje spoznali eno od konkretnih metod sledenja in jo tudi v celoti implementirali. Gre za sledenje s sprotnim izbiranjem značilnic (*ang.* on-line tracking). Ideja sledenja s sprotnim izbiranjem značilnic je v tem, da se med samim sledenjem znamo prilagoditi možnim spremembam v izgledu objekta, ki mu sledimo. Očitno je, da v zgoraj omenjenem sistemu SLAM do teh sprememb prihaja, saj robot svojimi premiki spreminja kot, pod katerim zaznava prostor, s tem pa se z vidika robota spremeni tudi videz objektov, ki se nahajajo v prostoru. Sledenje s sprotnim izbiranjem značilnic bi lahko torej uporabili za določanje položajev objektov v sistemu SLAM. V drugem delu diplomske naloge bomo tako naš sledilnik s sprotnim izbiranjem značilnic združili z eno od konkretnih izvedb sistema SLAM, Davisonovo aplikacijo MonoSLAM [14]. Idejo ponazarja *Slika 1*.



Slika 1. Shematska ponazoritev združitve sistema SLAM in sledilnika s sprotnim izbiranjem značilnic.

V sledečem poglavju je podrobneje razložen princip sledenja s sprotnim izbiranjem značilnic. V tretjem poglavju je opisan sam koncept sistema SLAM, kot tudi konkretna različica MonoSLAM, kjer gre za uporabo ene same kamere. Četrto poglavje podaja podrobnosti o sami implementaciji učenja s sprotnim izbiranjem značilnic in integraciji slednjega v MonoSLAM, dotaknemo pa se tudi same implementacije programa MonoSLAM. V petem poglavju so predstavljeni nekateri preizkusi delovanja tako razširjenega programa MonoSLAM. Šesto poglavje vsebuje razmislek o opravljenem delu in podaja nekaj smernic za nadaljnji razvoj, sedmo poglavje pa vsebuje nekatere priloge.

2. Sledenje s sprotnim izbiranjem značilnic

Sledenje objektom je pomembna naloga v domeni računalniškega vida. Ob vse večji dostopnosti zmogljivih računalnikov in poceni kamer visoke kvalitete, ter ob vse večjih potrebah po avtomatskem analiziranju video posnetkov, se je povečalo tudi zanimanje za razvoj različnih algoritmov za sledenje. Danes se sledenje objektom v praksi uporablja na področjih kot so video nadzor, kontrola kvalitete izdelkov v proizvodnji, sistemi za interakcijo med računalnikom in človekom, sistemi za pomoč vozniku, itd. [15]. V najosnovnejši obliki lahko sledenje definiramo kot problem določanja poti objekta v ravnini slike, ki ponazarja gibanje objekta v posnetem kadru. Povedano drugače, sledenje je določitev pozicije sledenega objekta v vsaki od slik nekega video posnetka. Pregled metod in algoritmov, ki se danes uporabljajo v sledenju podaja [15].

Eden od pristopov k problemu sledenja objektom, je sledenje kot detekcija. To pomeni, da v vsaki sliki zaporedja za neka podpodročja trenutne slike (ponavadi gre za okolico zadnjega znanega položaja) ugotavljamo ali predstavljajo objekt, ki mu sledimo, ali ne. Lahko si predstavljamo, da v vsakem koraku generiramo neko množico slik, nato pa za vsako od njih ugotavljamo ali predstavlja naš objekt ali ne. To pa je tipičen primer klasifikacijskega problema, ki ga znamo reševati z različnimi metodami strojnega učenja. Tako lahko z nekim učnim algoritmom sledilnik naučimo razpoznave objekta v sliki. Večina metod, ki za opis objekta uporablja učenje (oziroma z učenjem pridobljeni klasifikator), pa uporablja tako imenovano paketno (*ang.* off-line) učenje, kar pomeni, da klasifikator najprej naučimo na neki množici pozitivnih in negativnih primerov, nato pa ga za sledenje objekta nespremenjenega uporabljamo skozi celotno zaporedje slik. Slabosti takšnega pristopa so očitne. V naravnem okolju je objekt podvržen številnim dejavnikom, ki lahko vplivajo na njegov videz. Zaradi samih premikov kamere se v veliki meri lahko spremeni ozadje in kot, pod katerim opazujemo objekt. Objekt lahko čez čas spremeni svojo obliko, spremeni se lahko osvetlitev (tako v smislu spremembe ambientne svetlobe, kot v smislu pojavljanja odbojev na svetlečih površinah). Objekt lahko delno zakrije kak drug predmet iz okolja, itd..

Vse to slej ko prej pripelje do odpovedi sledenja. Potrebujemo torej algoritem, ki bo sposoben med samim sledenjem spreminjati model klasifikatorja in se tako prilagoditi na spremembe v videzu sledenega objekta. Sledenje z uporabo takšnega učnega algoritma bomo poimenovali sledenje s sprotnim izbiranjem značilnic. Uporabili bomo algoritem AdaBoost [16] z manjšo razširitvijo, ki omogoča uporabo v sledenju [17], [18], [19].

2.1. Sledenje kot klasifikacijski problem

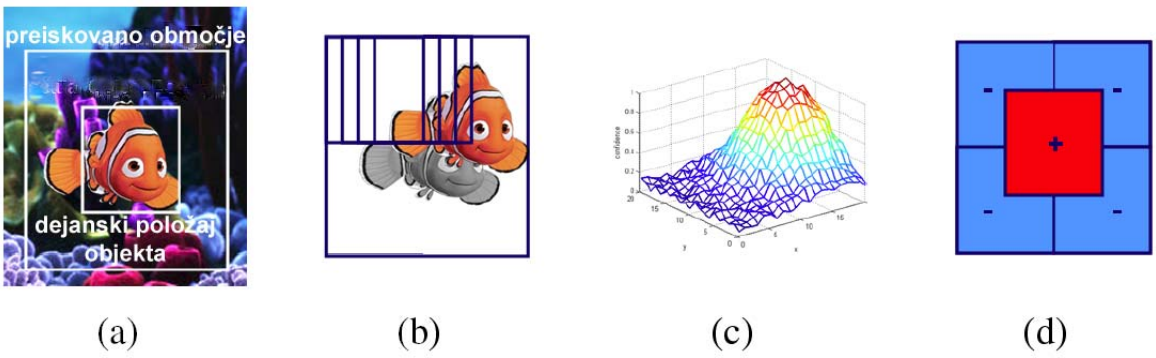
Sledenje objektov lahko zelo enostavno formaliziramo kot binarni klasifikacijski problem. Pozitivne primere predstavljajo slike objekta, negativne primere pa slike na katerih objekta ni, oziroma slike ozadja. Nad tako dobljeno množico primerov lahko uporabimo učni algoritem, ki vrne klasifikator[2], s katerim znamo za poljubno območje v sliki napovedati razred. Znamo torej napovedati, ali gre za naš objekt (pozitivna klasifikacija), ali gre za ozadje (negativna klasifikacija).

Če pa imamo na voljo učni algoritem, ki je sposoben sprotnega učenja, lahko vsakič, ko uspešno zaznamo objekt, klasifikator posodobimo, ter ga tako prilagodimo morebitnim spremembam v izgledu sledenega objekta. Tudi če objekt skozi celotno zaporedje slik ostaja nespremenjen, ima tak pristop številne prednosti. Izpustimo namreč fazo učenja, kar pomeni, da s sledenjem lahko začnemo takoj, za poljuben objekt v poljubnem zaporedju slik. To pa pomeni, da nam ni potrebno predhodno priskrbeti množice primerov za učenje. Dovolj je že, da nekajkrat izvedemo postopek posodobitve klasifikatorja nad prvo sliko zaporedja.

Brez izgube splošnosti lahko predpostavimo, da poznamo položaj objekta v prvi sliki nekega zaporedja slik (problem določanja pozicije objekta v prvi sliki sodi v domeno detekcije objektov, s čimer se v tem delu ne ukvarjamo). To območje v sliki označimo kot pozitiven primer, okoliška območja enakih velikosti pa kot negativne. Množico tako pridobljenih primerov podamo učnemu algoritmu, da posodobi klasifikator. Prva slika je izjema, saj je naš klasifikator še »prazen«, oziroma nenaučen. Da bi dobili začeten stabilen sledilnik (klasifikator), postopek posodobitve, s primeri pridobljenimi iz prve slike, nekajkrat ponovimo. Na naslednji sliki za vsako možno pozicijo objekta (omejimo se na neko regijo okrog zadnjega znanega položaja) pridobimo vrednost zaupanja, ki jo vrne klasifikator, ter jo vnesemo v sliko zaupanja. Položaj objekta postavimo v točko, v kateri ima slika zaupanja največjo vrednost. To točko lahko učinkovito določimo z uporabo algoritma mean shift [20]. Po uspešno določenem novem položaju objekta, območje, ki predstavlja objekt, zopet uporabimo kot pozitiven primer, območja, ki ga obkrožajo, pa kot negativne primere in z

[2] V kontekstu sledenja z uporabo učenja je klasifikator tisti, ki določi ali nek del slike predstavlja objekt, ki mu sledimo ali ne. Zato se v tem delu pojma klasifikator in sledilnik uporabljata izmenično.

njimi posodobimo klasifikator. Ta postopek se ponavlja za vsako sliko in klasifikator se je na ta način sposoben prilagoditi spremembam objekta, kot tudi spremembam območja, ki obkroža objekt, torej ozadja. Postopek je ponazorjen s *Sliko 2*.



Slika 2. Štirje koraki sledenja s klasifikatorjem. Recimo, da poznamo položaj objekta v sliki v času t (a). V času $t+1$ ovrednotimo klasifikator nad različnimi deli slike v okolici objekta (b). Dobljene vrednosti vnesemo v sliko zaupanja in iz nje ocenimo nov položaj objekta (c). Določimo pozitivne in negativne primere in z njimi posodobimo klasifikator (d). (povzeto po [17])

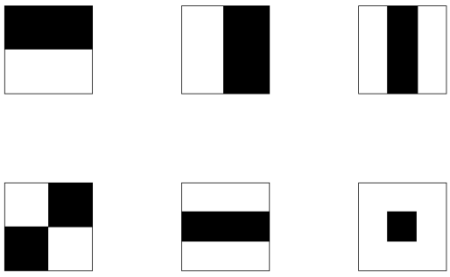
2.2. Značilnice

Značilnica je neka globalna lastnost slike ali dela slike (lahko tudi enega samega slikovnega elementa). Vrednost značilnice je lahko zelo enostavno izračunljiva, kot na primer povprečna intenziteta sivin v sliki ali pa, še bolj enostavno, kar vrednost ene izmed komponent v RGB barvnem modelu (ko je značilnica določena iz posameznega slikovnega elementa, kot na primer v [19]). Lahko pa gre za bolj kompleksne strukture, kot so na primer značilnice, ki temeljijo na lokalnih binarnih vzorcih (*ang.* local binary patterns - LBP) [21] ali značilnice dobljene na podlagi histogramov orientiranih gradientov (*ang.* histograms of oriented gradients - HOG) [22]. Vsem značilnicam pa je skupno to, da z njihovim izračunom povzamemo neko informacijo o sliki in jo predstavimo v obliki, primerni za obdelavo v nadaljnjih procesih računalniškega vida (v našem primeru sledenja). Za naš konkreten primer sledenja želimo izbrati predvsem značilnice, ki bodo hitro izračunljive, saj želimo realizirati sistem, ki bo deloval v realnem času.

2.2.1. Haarove značilnice

Haarove značilnice so enostavne značilnice, ki spominjajo na Haarove bazne funkcije, katere je za detekcijo objektov v svojem delu predlagal Papageorgiou [23]. Uporabnost Haarovih značilnic se je izkazala za primerno predvsem v domeni detekcije obrazov [24].

Vrednosti značilnic izračunamo kot razliko intenzitet sosednjih pravokotnih področij v sliki. Prototipe značilnic podaja *Slika 3*. Vrednost značilnice izračunamo tako, da od seštevka intenzitet slikovnih elementov v belem območju odštejemo seštevek intenzitet slikovnih elementov v črnem območju. Celotno množico značilnic dobimo tako, da iz vsakega prototipa tvorimo značilnice vseh možnih dimenzij in pozicij. Na ta način dobimo ogromno množico značilnic, izmed katerih bo lahko izbiral naš učni algoritem. Iz slike velikosti 40x40 slikovnih elementov lahko tako izračunamo 1.310.000 različnih Haarovih značilnic.



Slika 3. Prototipi uporabljenih Haarovih značilnic. Da dobimo celotno množico Haarovih značilnic, vsakega od prototipov raztegnemo v horizontalni in/ali vertikalni smeri in ga postavimo na vsa možna mesta v sliki. Vrednost značilnice v sliki dobimo tako, da od vsote slikovnih elementov, ki jih pokriva beli del, odštejemo vsoto slikovnih elementov, ki jih pokriva črni del značilnice.

Glavni namen uporabe značilnic namesto samih vrednosti slikovnih elementov, kot vhod v učni algoritem, je zmanjšanje variabilnost znotraj nekega razreda objektov in povečanje variabilnosti objektov zunaj tega razreda. Tako so na primer vrednosti neke značilnice izračunane iz različnih slik obrazov podobne, medtem ko je vrednost te iste značilnice, izračunana nad sliko, ki ni obraz, povsem nepredvidljiva (*Slika 4*). Značilnice običajno vsebujejo neko znanje o domeni (v primeru tretjega obraza iz *Slike 4*, na primer to, da je predel obraza pod očesom ponavadi svetlejši od očesa), ki se ga je težko naučiti iz surovih podatkov o vrednostih posameznih slikovnih elementov. Zelo velika množica enostavnih značilnic v povezavi z mehanizmom za izbiro značilnic (opisanem v *Razdelku 2.3*) pa močno poveča zmogljivost klasifikatorja. Uporaba pravilne kombinacije takih enostavnih značilnic je tista, ki nam omogoči uspešno ločevanje objekta od ozadja.

Slika 4. Nekaj značilnic, ki dobro predstavljajo razred obrazov. (povzeto po [25])

2.2.2. Značilnice na osnovi histograma orientacij robov

Značilnice na osnovi histograma orientacij robov (*ang.* edge orientation histogram feature – EOH značilnica), so definirali in uspešno uporabili v [26] v povezavi z zgoraj opisanimi Haarovimi značilnicami. Značilnice EOH so v nasprotju s Haarovimi značilnicami (ki jih lahko interpretiramo kot nekakšne detektorje linearnih robov), zmožne povzeti zapletenejše geometrijske lastnosti objekta (*Slika 5*).

Slika 5. Strukture, iz katerih izračunamo značilnice EOH (desno), v primerjavi s strukturami, ki jih povzamemo s Haarovo značilnico (levo). (povzeto po [26])

Za izračun vrednosti značilnic tipa EOH nad sliko najprej izvedemo detekcijo robov z uporabo Sobelovih jeder. Sliki vodoravnih in navpičnih robov, G_x in G_y, dobimo s konvoluiranjem Sobelovih jeder S_x in S_y s sliko I:

$$G_x = S_x * I \tag{1}$$

in

$$G_y = S_y * I\,, \tag{2}$$

kjer je

$$S_x = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad \text{in} \quad S_y = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}. \tag{3}$$

Moč roba v točki (x, y) izračunamo kot

$$G(x, y) = \sqrt{G_x(x, y)^2 + G_y(x, y)^2}\,. \tag{4}$$

Orientacija roba v točki (x, y) je

$$\theta(x, y) = \arctan\left(\frac{G_y(x,y)}{G_x(x,y)}\right). \tag{5}$$

Nato glede na njihovo orientacijo, moči robov razporedimo v K košov (*ang.* bin). Vrednost k-tega koša izračunamo kot

$$\psi_k(x, y) = \begin{cases} G(x, y) & \text{če } \theta(x, y) \in bin_k \\ 0 & sicer \end{cases}. \tag{6}$$

Nadalje definiramo

$$E_k(R) = \sum_{(x,y)\in R} \psi_k(x, y)\,, \tag{7}$$

kjer R predstavlja neko pravokotno področje v sliki.

Sedaj lahko definiramo prvo množico značilnic, A, kot

$$A_{k_1,k_2}(R) = \frac{E_{k_1}(R)+\varepsilon}{E_{k_2}(R)+\varepsilon}, \tag{8}$$

kjer je ε neko zelo majhno število. Za vsak R imamo $\binom{K}{2}$ značilnic, kar pomeni, da za slike dimenzije $h * w$ množica A vsebuje $\binom{K}{2} \frac{h(h+1)}{2} \frac{w(w+1)}{2}$ značilnic. Značilnice v tej množici lahko interpretiramo kot različna razmerja med vsemi možnimi pari orientacij robov. Če se zopet vrnemo na domeno obrazov (*Slika 6a*), lahko na primer ugotovimo, da je v predelu ust vodoravnih robov veliko več kot navpičnih in lahko pričakujemo ustrezno visoko vrednost pripadajoče značilnice.

(a) (b) (c)

Slika 6. (a) V predelu ust je vodoravnih robov več kot navpičnih. (b) v označenem delu obraza navpični robovi prevladujejo nad vsemi ostalimi orientacijami robov. (c) spodnji in zgornji del obraza nista simetrična. (povzeto po [26])

Drugo množico značilnic, ki jih lahko izračunamo iz histogramov orientacij robov, lahko pojmujemo, kot zaznavanje dominante orientacije robov na nekem območju. Torej, kadar na nekem območju v sliki neka orientacija prevladuje nad drugimi, bo vrednost pripadajoče značilnice ustrezno visoka (*Slika 6b*). Te značilnice so definirane kot

$$B_k(R) = \frac{E_k(R)+\varepsilon}{\sum_i E_i(R)+\varepsilon}. \tag{9}$$

Moč množice teh značilnic je $K\frac{h(h+1)}{2}\frac{w(w+1)}{2}$.

Simetrija je ena od lastnosti, ki si jo lastijo skoraj vsi umetno ustvarjeni predmeti, kot tudi večina predmetov v naravi, zato definiramo še tretjo množico značilnic:

$$C(R_1,R_2) = \frac{\sum_{k\in K}|E_k(R_1)-E_k(R_2)|}{sizeof(R_1)}. \tag{10}$$

R_1 in R_2 sta pravokotnika enakih velikosti, stojita pa eden nasproti drugemu, glede na eno izmed osi simetrije. Osi simetrije sta postavljeni v center slike (navpična in vodoravna os). Velikost pravokotnika $sizeof(R_1)$ izračunamo kot zmnožek višine in širine pravokotnika R_1. Tako kot to velja za značilnice iz ravnokar definiranih množic A in B, lahko tudi z značilnico iz množice C ugotovimo, da lastnosti, katero značilnica opisuje, v nekem predelu slike ni (*Slika 6c*). Vrednost značilnice označene na *Sliki 6c* bo majhna, saj spodnji in zgornji del obraza nista simetrična.

Moč te množice značilnic, je

$$\left\lfloor\frac{w}{2}\right\rfloor\frac{\left\lfloor\frac{w}{2}+1\right\rfloor}{2}\frac{h(h+1)}{2} + \left\lfloor\frac{h}{2}\right\rfloor\frac{\left\lfloor\frac{h}{2}+1\right\rfloor}{2}\frac{w(w+1)}{2}. \tag{11}$$

S procesom normalizacije lahko tako za značilnice EOH, kot tudi za Haarove značilnice dosežemo invarianco na spremembe v globalni osvetlitvi, kar jih naredi še posebej primerne za uporabo v našem sistemu. Sprememba globalne osvetlitve pomeni množenje vsake od vrednosti intenzitet slikovnih elementov z neko konstanto. Očitno je, da se po spremenjeni globalni osvetlitvi, spremenijo tudi vrednosti značilnic, ki so izračunane kot razlika vsot

intenzitet slikovnih elementov na določenih območjih (njihova vrednost se prav tako pomnoži z zgoraj omenjeno konstanto). Z normalizacijo slike dosežemo, da vrednosti značilnic ostanejo neodvisne od globalne osvetlitve. Normalizacija pomeni razširitev razpona intenzitet slikovnih elementov slike na nek ciljni razpon. Ciljni razpon običajno zavzema najnižjo vrednost intenzitete 0 in najvišjo vrednost intenzitete 255 (osem bitov). Denimo, da je najnižja vrednost intenzitete neke nenormalizirane slike int_{min}, najvišja pa int_{max}. Da bi dosegli ciljni razpon vrednosti med 0 in 255, moramo intenziteto int_i vsakega slikovnega elementa popraviti v skladu s sledečo enačbo

$$int_i^{norm} = \frac{(int_i - int_{min}) \times 255}{int_{max}}. \quad (12)$$

Po drugi strani pa niti Haarove značilnice niti značilnice EOH niso invariantne na rotacije v slikovni ravnini, kar pa bi znalo predstavljati problem le v pristopih, kjer je nabor izbranih značilnic fiksen (kot v [24], [26]).

2.3. Algoritem Adaboost in sprotno izbiranje značilnic

Pojem učnega algoritma, kot ga uporabljamo v tem delu, se nanaša na področje tako imenovanega nadzorovanega učenja (*ang.* supervised learning). Problem, ki ga rešuje nadzorovano učenje, je iz množice znanih parov $\langle vhod, izhod \rangle$ izračunati funkcijo, ki bo ustrezno določila izhode novim, še ne videnim vhodom. Tipična naloga nadzorovanega učenja je klasifikacija. Učni algoritem poišče funkcijo (hipotezo), ki na podlagi nekega vhodnega vektorja $\boldsymbol{x} = [x_1, \dots, x_n]$ napove enega izmed izhodnih razredov $y \epsilon \{c_1, \dots, c_m\}$. Za učenje ima učni algoritem na voljo množico znanih parov $\{\langle \boldsymbol{x}_1, y_1 \rangle, \dots, \langle \boldsymbol{x}_N y_N \rangle\}$, ki jim pravimo učna množica primerov. Kadar imamo le dva izhodna razreda (kot v našem primeru), govorimo o binarni klasifikaciji.

Boosting je splošna metoda za izboljšanje točnosti zgoraj omenjene množice učnih algoritmov [27], [28]. Sama ideja metode je združevanje enostavnih pravil (ponavadi klasifikatorjev), ki bodo kot celota delovali bolje kot pa vsako pravilo samo zase. Metoda temelji na uteževanju učnih primerov glede na njihovo težavnost. Na začetku vsem primerom učne množice priredimo utež enako 1. V prvi iteraciji z izbranim učnim algoritmom (ki mora znati obravnavati utežene učne primere) zgradimo prvo hipotezo. S to hipotezo napovemo razred vsakemu od primerov učne množice. Primerom, ki so klasificirani napačno, utež povečamo, primerom, ki so klasificirani pravilno, pa utež zmanjšamo. Tako bo učni algoritem v naslednji iteraciji usmerjen v obravnavanje težavnejših primerov. Postopek ponavljamo, da dobimo množico hipotez, ki jih nato z uteženim glasovanjem združimo v eno.

Za boljše razumevanje zgoraj opisanega pristopa in postopkov opisanih v nadaljevanju definirajmo:

Šibek klasifikator: Šibek klasifikator je hipoteza, ki jo dobimo z nekim enostavnim učnim algoritmom, njena točnost pa mora biti le nekoliko boljša od naključja. V primeru binarne klasifikacije to pomeni, da mora šibek klasifikator pravilno klasificirati le nekaj več kot polovico primerov. Označimo ga s h^{weak}.

Močen klasifikator: Z združevanjem množice šibkih klasifikatorjev dobimo močen klasifikator h^{strong}. Napoved močnega klasifikatorja dobimo z uteženim glasovanjem N šibkih klasifikatorjev:

$$h^{strong}(\boldsymbol{x}) = sign(conf(\boldsymbol{x})) \,, \tag{13}$$

$$conf(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_n \cdot h^{weak}(\boldsymbol{x}) \,, \tag{14}$$

kjer α_n predstavlja glasovalno utež n-tega šibkega klasifikatorja.

Vrednost $conf(\boldsymbol{x})$ obravnavamo kot mero zaupanja, oziroma kot vrednost, ki nam pove kako prepričan je klasifikator o svoji odločitvi. Te vrednosti določajo tako imenovano sliko zaupanja omenjeno v *Razdelku 2.1*.

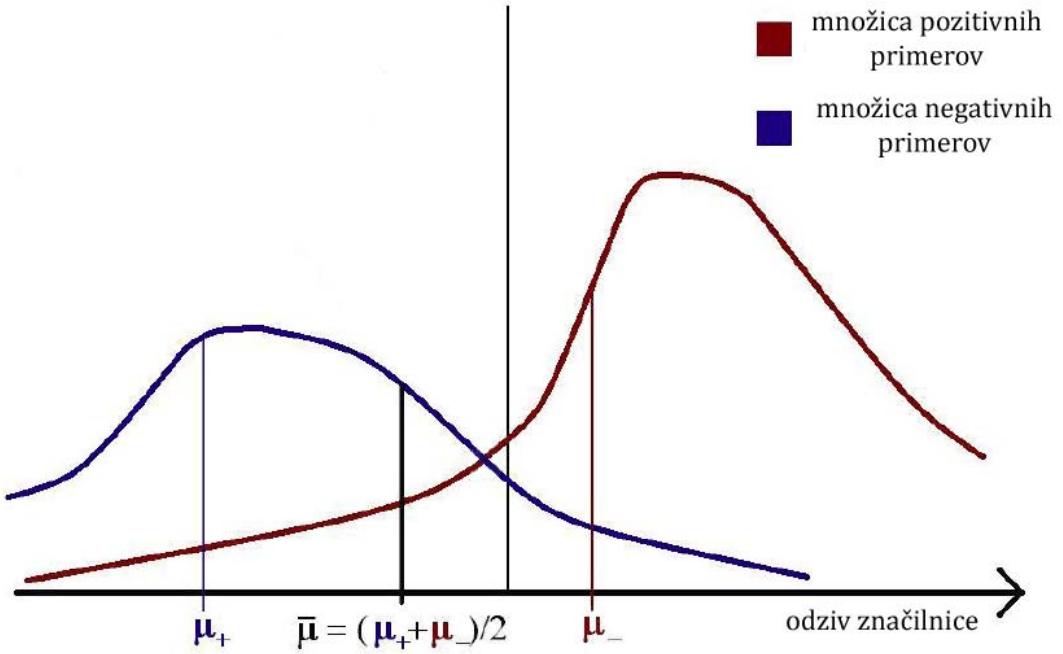
V nadaljevanju se bomo osredotočili na konkreten algoritem, ki se imenuje AdaBoost (Adaptive Boosting) [17]. Na učni množici primerov $X = \{\langle\{\boldsymbol{x_1}, y_1\rangle, \dots, \langle\{\boldsymbol{x_L}, y_L\rangle \,|\boldsymbol{x_i} \in \mathbb{R}^{\boldsymbol{m}}, y_i \in \{1, -1\}\}$ z začetno uniformno porazdelitvijo primerov $p(\boldsymbol{x_i}) = \frac{1}{L}$ uporabimo nek učni algoritem, da pridobimo šibek klasifikator h^{weak}. V skladu z njegovo napako e mu dodelimo glasovalno utež $\alpha = \frac{1}{2} \cdot \ln(\frac{1-e}{e})$. Porazdelitev $p(\boldsymbol{x})$ spremenimo tako, da se verjetnost izbire napačno klasificiranih primerov poveča, verjetnost izbire pravilno klasificiranih primerov pa zmanjša. Postopek ponavljamo, dokler ne dosežemo nekega ustavitvenega pogoja (npr. dosežena zahtevana klasifikacijska točnost na testni množici). Freund in Schapire [16] sta v svojem delu pokazala, da v primeru binarnega klasifikacijskega problema z uporabo algoritma AdaBoost napaka na učni množici pada eksponentno glede na število boosting iteracij, torej glede na število šibkih klasifikatorjev.

2.3.1. Boosting za izbiro značilnic

Ideja uporabe boostinga za izbiro značilnic je v tem, da vsakemu šibkemu klasifikatorju priredimo eno značilnico [29]. Šibek klasifikator na podlagi vrednosti te značilnice poskuša čim bolje ločiti množico primerov na pozitivne in negativne primere. Naj bo $f_i(\boldsymbol{x})$ vrednost i-te značilnice izračunane nad primerom $\boldsymbol{x}$. Šibek klasifikator bo za ta primer napovedal razred (1 ali -1) po naslednji shemi:

$$h_i^{weak}(\boldsymbol{x}) = \begin{cases} 1, & če\ p_i f_i(\boldsymbol{x}) < p_i \theta_i \\ -1, & sicer \end{cases}, \tag{15}$$

kjer je θ_i prag, p_i pa pariteta šibkega klasifikatorja h_i^{weak}. To sta parametra, ki jih šibkemu klasifikatorju določimo na podlagi odziva značilnice na celotni množici učnih primerov. Določimo ju z oceno dveh verjetnostnih porazdelitev – porazdelitve vrednosti značilnice ovrednotene na pozitivnih učnih primerih in porazdelitve vrednosti značilnice ovrednotene na negativnih učnih primerih (*Slika 7*). Prag θ je določen kot $\frac{\mu_+ + \mu_-}{2}$, kjer sta μ_+ in μ_- aritmetični sredini porazdelitev vrednosti značilnice pozitivnih oziroma negativnih primerov. Pariteto p enostavno določimo kot $|\mu_+ - \mu_-|$.



Slika 7. Primer porazdelitve odziva neke konkretne značilnice čez celotno učno množico primerov (razdeljeno na pozitivne in negativne primere). Odziv značilnice je njena vrednost izračunana nad nekim primerom (sliko). (povzeto po [25])

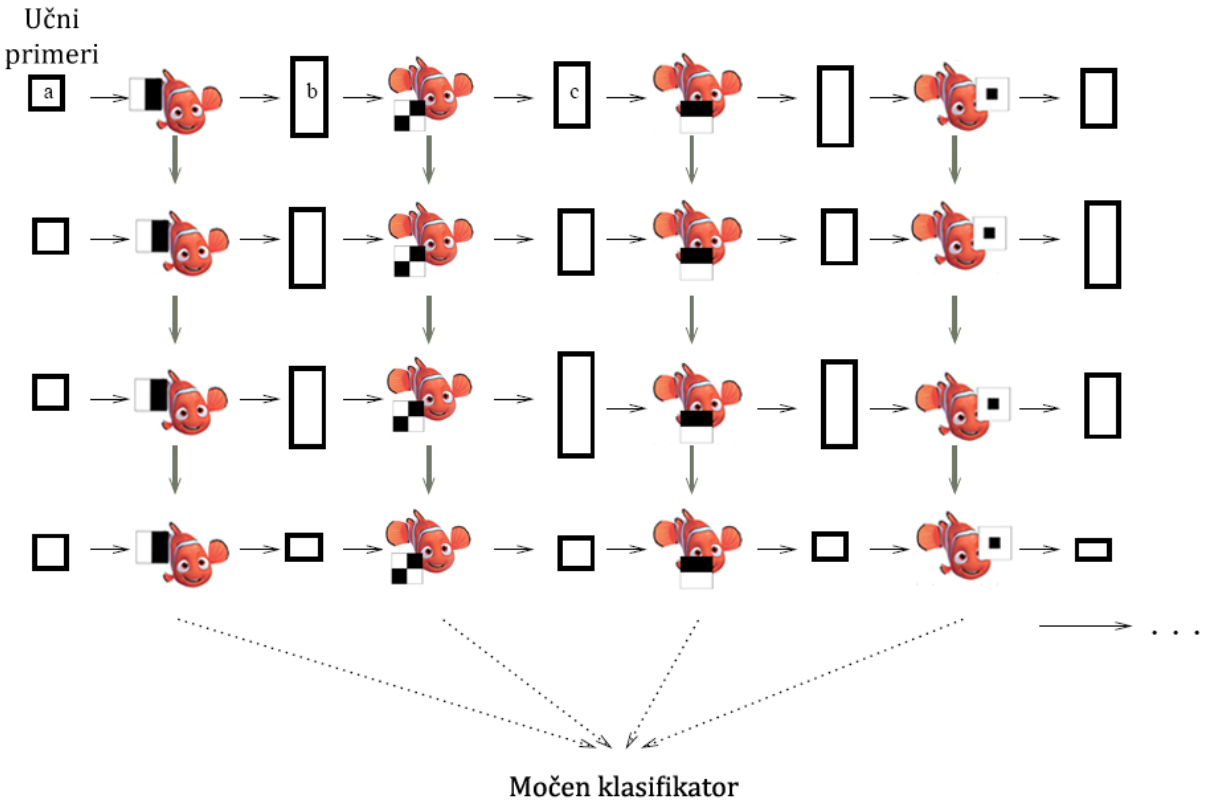
Ker je naša množica značilnic ogromna (glej *Razdelek 2.2*), se raje omejimo na neko podmnožico značilnic. V vsakem koraku metode boosting iz te podmnožice izberemo značilnico, iz katere lahko tvorimo najboljši (glede na njegovo predvideno napako) šibek klasifikator h^{weak}. Z ustrezno glasovalno utežjo ga dodamo naboru šibkih klasifikatorjev, ki tako skupaj tvorijo močen klasifikator h^{strong}.

2.3.2. Boosting za sprotno izbiranje značilnic

Problem AdaBoost algoritma v svoji osnovni obliki je, da ne omogoča sprotnega učenja. Povedano z drugimi besedami, algoritem ne omogoča posodobitve šibkih klasifikatorjev na podlagi novo prispelih učnih primerov. Samo po sebi posodabljanje šibkih klasifikatorjev ni problematično, saj lahko uporabimo številne algoritme, ki omogočajo sprotno učenje. Tudi glasovalne uteži znamo izračunati, potrebujemo le podatek o predvideni napaki šibkih klasifikatorjev, ki pa jo znamo oceniti iz dosedanjih napovedi. Problem je, da za novo prispeli primer ne poznamo njegove uteži, oziroma ne vemo, kako težavna bo njegova klasifikacija.

Uporabimo pristop, ki ga je v svojem delu predlagal Oza [30]. Vsakemu na novo prispelemu primeru priredimo neko utež λ (z začetno vrednostjo 1), ki predstavlja težavnost primera. Predpostavimo, da že imamo nek močen klasifikator, sestavljen iz množice šibkih klasifikatorjev, katere želimo posodobiti v skladu z novo prispelim učnim primerom. Denimo,

da se nahajamo pri i-tem šibkem klasifikatorju h_i^{weak}. h_i^{weak} z uporabo Kalmanovega filtra najprej k-krat posodobimo (podrobneje opisano v *Razdelku 4.1.2*), kjer je k verjetnostna spremenljivka porazdeljena po Poissonovi porazdelitvi $k \sim Poisson(\lambda)$, nato pa z njim klasificiramo novi primer. V kolikor je klasifikacija pravilna, λ zmanjšamo, sicer λ povečamo. S primerom, ki ima zdaj spremenjeno utež, posodobimo naslednji šibek klasifikator h_{i+1}^{weak}. i-ti šibek klasifikator h_i^{weak} tako posodobimo v skladu s težavnostjo primera, ki jo ocenimo skozi napovedi klasifikatorjev h_1^{weak} … h_{i-1}^{weak}. Postopek prikazuje *Slika 8*.



Slika 8. Ilustracija, ki prikazuje delovanje metode boosting s sprotnim posodabljanjem. Vsaka vrstica predstavlja prihod enega učnega primera, ki ga pošljemo skozi vse šibke klasifikatorje, da se z njim posodobijo. Vsak šibek klasifikator (predstavljen s sličico ribe) je generiran iz šibkega klasifikatorja nad njim prek posodobitve z uteženim učnim primerom. Kvadratki predstavljajo učne primere, višina kvadratka pa ponazarja velikost uteži primera. (prirejeno po [30])

Ta pristop zahteva, da je število šibkih klasifikatorjev določeno vnaprej, medtem ko število učnih primerov vnaprej ni znano. Osnovna (paketna) različica metode boosting pa po drugi strani predpostavlja fiksno velikost učne množice, število šibkih klasifikatorjev pa ni nujno določeno vnaprej. Oza je tudi empirično dokazal [30], da se s povečevanjem števila učnih primerov, z različico metode s sprotnim posodabljanjem (on-line) po učinkovitosti dobljenega klasifikatorja približujemo paketni (off-line) različici algoritma.

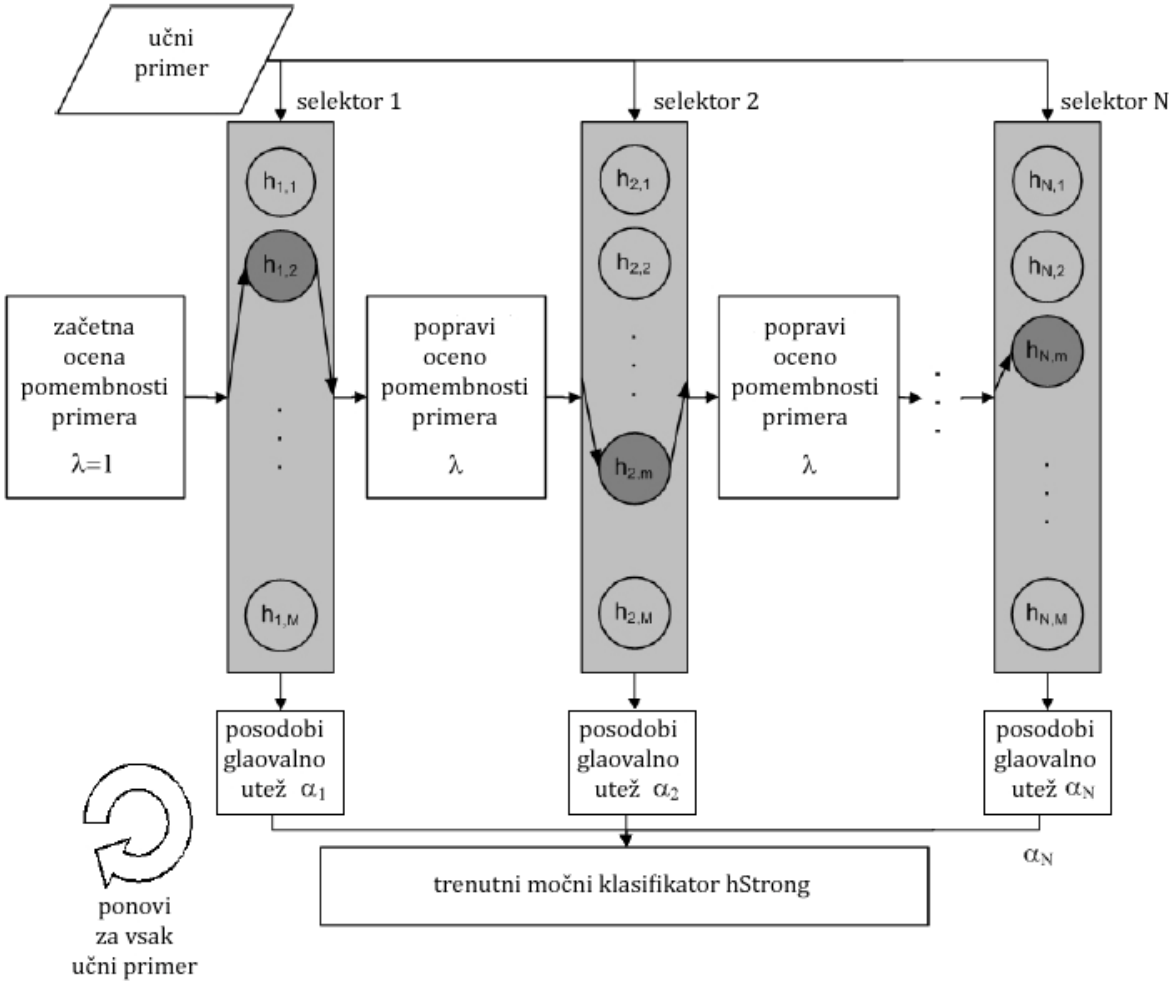
V želji, da bi zgoraj opisani algoritem uporabili za izbiro značilnic, pa potrebujemo sledeči mehanizem [18]:

Selektor: Iz dane množice M šibkih klasifikatorjev $\mathcal{H}^{weak} = \{h_1^{weak}, \dots, h_M^{weak}\}$, selektor izbere natanko enega

$$h^{sel} = h_m^{weak} , \tag{16}$$

kjer je m izbran v skladu z nekim optimizacijskim kriterijem. V našem primeru je to ocenjena napaka e_i posameznega šibkega klasifikatorja h_i^{weak}. Iz množice šibkih klasifikatorjev $\mathcal{H}^{weak}$ selektor h^{sel} torej izbere klasifikator z najnižjo ocenjeno napako e_i.

Tako je dejansko učenje razdeljeno na dva nivoja. Na višjem nivoju lahko že sam selektor obravnavamo kot klasifikator, ki se nauči izbrati pravilen šibek klasifikator iz množice šibkih klasifikatorjev, ki so mu na voljo. Na nižjem nivoju pa se posamezen šibek klasifikator, kateremu je pridružena konkretna značilnica, uči na podlagi vrednosti te značilnice, pravilno klasificirati primere. Učenje selektorja tako pomeni posodobitev vseh šibkih klasifikatorjev v pripadajoči množici in izbor šibkega klasifikatorja z najnižjo predvideno napako. Boosting s sprotnim posodabljanjem torej izvajamo nad selektorji (posodabljamo celotne podmnožice šibkih klasifikatorjev), ne pa neposredno nad samimi šibkimi klasifikatorji. Postopek je ponazorjen s *Sliko 9* in *Algoritmom 1*.



Slika 9. Boosting s sprotnim posodabljanjem za izbiro značilnic. $h_{i,j}$ predstavlja $j-ti$ šibek klasifikator v $i-ti$ množici šibkih klasifikatorjev. Iz vsake od teh množic izbira $i-ti$ selektor. (povzeto po [18])

Vhod: učni primer $\langle \boldsymbol{x}, y \rangle$, $y \in \{-1, +1\}$
močen klasifikator h^{strong} (inicializiran naključno)
uteži $\lambda_{n,m}^{corr}$, $\lambda_{n,m}^{wrong}$ (inicializirane na 1)

inicializiraj utež pomembnosti $\lambda = 1$
// za vsak selektor
for $n = 1,2, \ldots, N$ **do**
  // posodobi trenutni selektor h_n^{sel}
  for $m = 1,2, \ldots, M$
    // posodobi vse šibke klasifikatorje
    $h_{n,m}^{weak} = posodobi(h_{n,m}^{weak}, \langle \boldsymbol{x}, y \rangle, \lambda)$

    // oceni napake
    if $h_{n,m}^{weak}(\boldsymbol{x}) = y$ **then**
      $\lambda_{n,m}^{corr} = \lambda_{n,m}^{corr} + \lambda$
    else
      $\lambda_{n,m}^{wrong} = \lambda_{n,m}^{wrong} + \lambda$
    end if
    $e_{n,m} = \frac{\lambda_{n,m}^{wrong}}{\lambda_{n,m}^{corr} + \lambda_{n,m}^{wrong}}$
  end for
    // izberi šibek klasifikator z najmanjšo napako
    $m^{+} = \arg\min_{\mathrm{m}}(e_{n,m})$
    $e_n = e_{n,m^{+}};\ h_n^{sel} = h_{n,m^{+}}^{weak}$
    if $e_n = 0$ **or** $e_n > \frac{1}{2}$ **then**
      continue
    end if

    //izračunaj glasovalno utež
    $\alpha_n = \frac{1}{2} \cdot \ln\left(\frac{1 - e_n}{e_n}\right)$

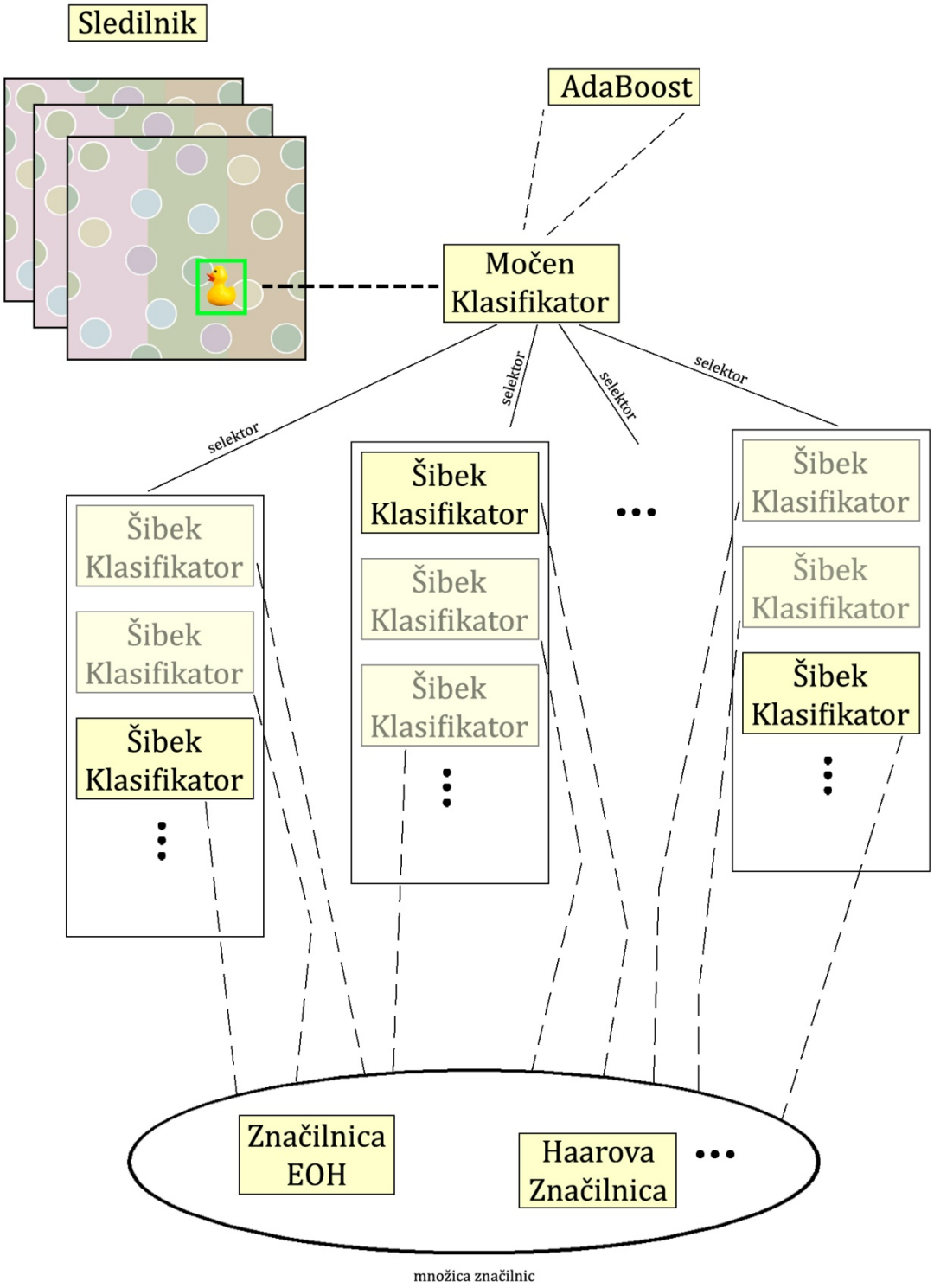
    // posodobi utež pomembnosti
    if $h_n^{sel}(\boldsymbol{x}) = y$ **then**
      $\lambda = \lambda \cdot \frac{1}{2 \cdot (1 - e_n)}$
    else
      $\lambda = \lambda \cdot \frac{1}{2 \cdot e_n}$
    end if

  // zamenjaj najslabši šibek klasifikator z novim
  $m^{-} = \arg\max_{\mathrm{m}}(e_{n,m})$
  $\lambda_{n,m^{-}}^{corr} = 1;\ \lambda_{n,m^{-}}^{wrong} = 1$
  pridobi nov $h_{n,m^{-}}^{weak}$
end for

Algoritem 1. Boosting s sprotnim posodabljanjem za izbiro značilnic

Vsak selektor ima svojo množico šibkih klasifikatorjev, katerim pripadajo značilnice, ki so na začetku naključno izbrane iz globalne množice vseh značilnic. Ker so značilnice na začetku izbrane naključno, pa tudi zato, da se bolje prilagodimo možnim spremembam ciljnega razreda (v našem primeru videzu objekta, ki mu sledimo), pa na koncu vsakega koraka glavne zanke najslabši šibek klasifikator skupaj s pripadajočo značilnico zamenjamo z novim, naključno izbranim šibkim klasifikatorjem iz globalne množice šibkih klasifikatorjev. Tako se bo skozi proces sprotnega učenja model izpopolnjeval in bodo čez čas v vsaki množici šibkih klasifikatorjev le tisti šibki klasifikatorji, ki predstavljajo »dobre« značilnice, in z uporabo katerih lahko uspešno klasificiramo nove primere.

V tem poglavju smo spoznali, kaj je to sledenje in kako ga lahko prevedemo na klasifikacijski problem. Predstavljeni sta bili množica Haarovih značilnic in množica značilnic EOH, s kombinacijo katerih lahko sestavimo model sledenega objekta (klasifikator). Za izbiro značilnic in njihovo združevanje v končni klasifikator smo uporabili metodo boosting, konkretno algoritem AdaBoost. Ugotovili smo, da bi želeli imeti sledilnik, ki bi bil sposoben prilagajanja spremembam videza objekta, ki mu sledimo, zato smo algoritem AdaBoost prilagodili tako, da omogoča sprotno izbiranje značilnic. *Slika 10* podaja shematski prikaz glavnih modulov, ki jih potrebujemo za realizacijo sledenja s sprotnim izbiranjem značilnic. V poglavju, ki sledi, bomo podrobneje spoznali splošen koncept sočasne lokalizacije in kartografiranja, ter konkretno izpeljavo, ki uporablja eno samo kamero. V četrtem poglavju pa bomo ugotovitve *Poglavja 2* in *Poglavja 3* združili v sistem, ki za sočasno lokalizacijo in kartografiranje z eno samo kamero uporablja v tem poglavju opisano sledenje s sprotnim izbiranjem značilnic.

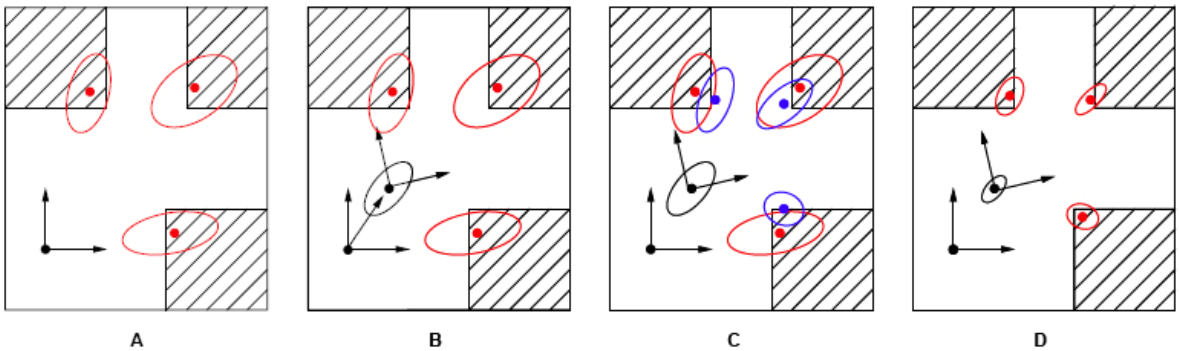


Slika 10. Shema prikazuje glavne module (rumeni kvadratki), ki jih potrebujemo za realizacijo sledenja s sprotnim izbiranjem značilnic. *Sledilnik* skrbi za glavno zanko, to je sprejema slike iz zaporedja in premika sledilno okno. *Močen klasifikator* predstavlja glavni mehanizem za sledenje in vsebuje prirejeno metodo *AdaBoost*, ki skrbi za sprotno izbiro značilnic (oziroma šibkih klasifikatorjev). Vsakemu *šibkemu klasifikatorju* je prirejena ena od značilnic iz množice vseh značilnic

3. Sočasna lokalizacija in kartografiranje

3.1. Splošno o sočasni lokalizaciji in kartografiranju

Avtonomni mobilni robot postavimo na neznano lokacijo v neko neznano okolje. S premikanjem po prostoru in zaznavanjem različnih značilnosti okolice robot inkrementalno gradi zemljevid prostora in z uporabo tega zemljevida sproti določa svoje stanje (lego in orientacijo glede na nek referenčni koordinatni sistem). Problem grajenja zemljevida in problem določanja stanja robota sta tesno povezana. Rešitev enega ponavadi zahteva tudi rešitev drugega, kar nas pripelje do ideje sočasne lokalizacije in kartografiranja. Ta koncept je pravzaprav precej intuitiven. Le z dobrim zemljevidom lahko namreč natančno določimo svojo lokacijo, in le če natančno poznamo svojo lokacijo, lahko zgradimo dober zemljevid. Dober pregled različnih rešitev za implementacijo sistema SLAM podaja [31]. Osnovni princip delovanja sistema sočasne lokalizacije in kartografiranja opisuje *Slika 11.* Postopek je tako v grobem razdeljen na štiri glavne korake, ki so podrobneje opisani v nadaljevanju.



Slika 11. Glavni koraki v procesu SLAM, ki so podrobneje opisani v sledečem razdelku. Robot zazna 3 nove značilnice v prostoru in jih doda v zemljevid (A). Robot se premakne in oceni svoj novi položaj (B). Iz novega položaja znova zazna značilnice (modre elipse), ki so že v zemljevidu (C). Popravi oceno svojega položaja in ocene položajev vseh značilnic v zemljevidu (D). (povzeto po [32])

3.1.1. Glavni koraki procesa SLAM

3.1.1.1. Detekcija in izbor značilnic iz okolja

Robot s svojimi senzorji posname okolico, v kateri se nahaja. Senzorji so naprave, s katerimi robot sprejema informacije iz okolja. V preteklosti so se v ta namen uporabljali predvsem laserski čitalci ali ultrazvočni detektorji. Ob množici algoritmov, ki jih danes ponuja področje računalniškega vida, pa te senzorje danes zamenjujejo lahke in poceni kamere [32]. Iz tako pridobljenih meritev nato izluščimo neke značilne točke v prostoru (značilnice), izmed katerih po nekem kriteriju izberemo najboljše. To pomeni, da izberemo take značilnice, za katere ocenimo, da jih bomo lahko uspešno zaznavali tudi kasneje, iz različnih položajev robota v prostoru. Ocenimo položaje izbranih značilnic in jih dodamo v zemljevid prostora.

Naj na tem mestu omenimo, da se pojem značilnice uporabljen v tem razdelku, razlikuje od tistega v *Razdelku 2.2*. Značilnica tu predstavlja neko konkretno točko (lahko pa tudi kakšno drugo geometrijsko strukturo) v prostoru, katere 3D koordinate določamo skozi proces SLAM, medtem ko v *Razdelku 2.2* govorimo o značilnici kot o neki globalni lastnosti slike ali dela slike.

3.1.1.2. Premik robota

V naslednjem koraku se robot premakne in izračuna oceno svojega novega položaja. Svoj položaj lahko oceni bodisi na podlagi vhodnih podatkov, s katerimi ga krmilimo (robot dobi na primer ukaz: »premakni se naprej za en meter«), bodisi na podlagi tako imenovanih notranjih meritev, ki direktno ocenjujejo premike robota (na primer merilec obratov kolesa robota), ali pa na podlagi nekega modela gibanja (kadar nimamo kontrole nad robotom).

3.1.1.3. Asociacija podatkov

Iz novega položaja robota zopet posnamemo okolico, iz dobljenih podatkov izluščimo značilnice in poiščemo ujemanja z značilnicami, ki so že v zemljevidu. Pri tem seveda upoštevamo podatke o ocenjenih relativnih položajih značilnic glede na ocenjeni položaj robota, kot tudi podatke o oceni relativnih položajev samih značilnic med seboj.

3.1.1.4. Posodobitev stanja robota in položajev značilnic

Ta korak predstavlja jedro procesa SLAM. Z upoštevanjem različnih, do zdaj opravljenih napovedi in meritev, posodobimo oceno stanja robota in ocene položajev vseh značilnic v zemljevidu. Danes najbolj razširjena metoda za posodabljanje zemljevida in stanja robota je uporaba razširjenega Kalmanovega filtra [33], ki poleg samih položajev objektov v zemljevidu hrani tudi nedoločenost za vsakega od teh položajev (vsakemu objektu je pridružena neka vrednost, ki pove, kako prepričani smo v oceno njegovega položaja).

3.2. SLAM z eno samo kamero

V tem razdelku bomo podrobneje opisali konkretno rešitev problema sočasne lokalizacije in kartografiranja, ki jo predlagajo Davison in sodelavci [14]. Ideja je v realizaciji sistema SLAM z eno samo kamero, kot edino napravo, s katero zaznavamo okolje. Poleg tega ne predvidevamo nikakršnega nadzora nad gibanjem robota, ki bi nam omogočal direktne napovedi robotovih premikov. To pomeni, da lahko le ugibamo (oziroma z neko verjetnostjo napovemo), kam se bo robot premaknil v naslednjem časovnem koraku. V ta namen uporabimo model gibanja, ki je opisan v *Razdelku 3.2.3*. V želji, da bi sistem deloval v realnem času, je uporabljen princip aktivnega zaznavanja značilnic, ki je podrobneje opisan v R*azdelku 3.2.4*.

3.2.1. Verjetnostni zemljevid prostora

Verjetnostni zemljevid prostora vsebuje ocene prostorskih relacij med objekti v prostoru skupaj z njihovimi nedoločenostmi (*ang.* uncertainty). Objekti v našem primeru so vse izbrane značilnice in robot. Z gibanjem robota po prostoru in zaznavanjem značilnic iz različnih položajev se zemljevid stalno posodablja z uporabo razširjenega Kalmanovega filtra (posodabljajo se tako sami položaji objektov kot tudi pripadajoče nedoločenosti). Matematično je zemljevid predstavljen z vektorjem stanja $\hat{x}$ in kovariančno matriko P. Prva vrstica vektorja $\hat{x}$ predstavlja stanje robota, vse naslednje vrstice pa predstavljajo stanja značilnic:

$$\hat{x} = \begin{pmatrix} \hat{x}_v \\ \hat{y}_1 \\ \hat{y}_2 \\ \vdots \end{pmatrix}, \; P = \begin{bmatrix} P_{xx} & P_{xy_1} & P_{xy_2} & \cdots \\ P_{y_1x} & P_{y_1y_1} & P_{y_1y_2} & \cdots \\ P_{y_2x} & P_{y_2y_1} & P_{y_2y_2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{17}$$

S tako predstavitvijo aproksimiramo verjetnostno porazdelitev vseh parametrov zemljevida z eno samo večrazsežno normalno porazdelitvijo, katere dimenzija je enaka številu značilnic v zemljevidu + 1 (robot).

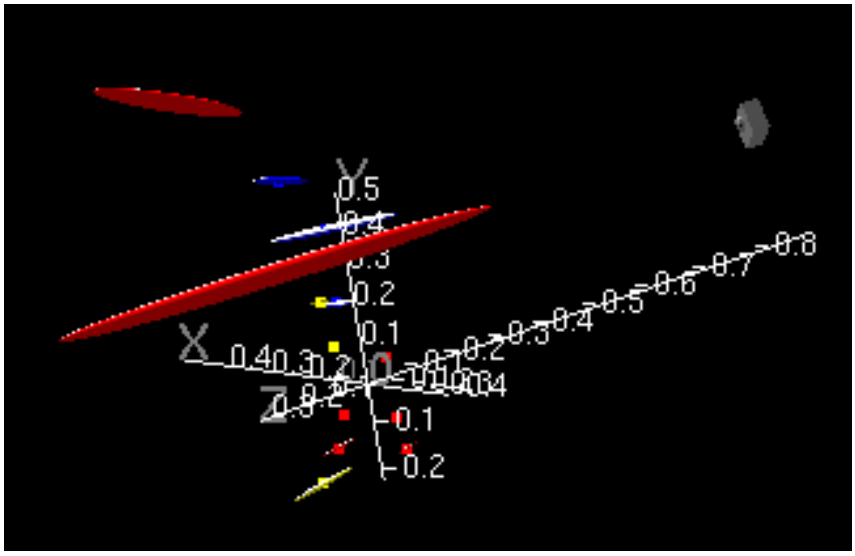
Vektor stanja robota $\hat{x}_v$ je sestavljen iz vektorja robotove pozicije v prostoru r^W, kvaterniona q^{WR}, ki določa orientacijo robota, vektorja hitrosti v^W in vektorja kotne hitrosti ω^R. Nadpisa W in R ponazarjata globalni oziroma robotov referenčni koordinatni sistem:

$$\hat{x}_v = \begin{pmatrix} r^W \\ q^{WR} \\ v^W \\ \omega^R \end{pmatrix}. \tag{18}$$

V tem kontekstu zemljevid ne služi izčrpni predstavitvi prostora, v katerem se nahaja robot, temveč hrambi (redke) množice čimbolj enakomerno[3] posejanih značilnic, s katerimi znamo v vsakem trenutku hitro in natančno določiti položaj robota. Z ustreznimi kriteriji za izbiro značilnic iz okolja, lahko že z zelo majhno množico značilnic uspešno obvladujemo lokalizacijo robota po prostoru velikosti ene sobe.

Ko v vidnem polju robota začne primanjkovati značilnic, ki so že v našem zemljevidu prostora, moramo iz okolice zaznati novo značilnico in jo umestiti v zemljevid. V želji, da bi bile značilnice enakomerno razporejene po prostoru, najprej v sliki naključno izberemo območje neke velikosti (npr. 80x60 slikovnih elementov), v katerem še ni nobene značilnice. Z upoštevanjem modela gibanja robota, opisanim v nadaljevanju, se izognemo izbiri območij, za katera ocenimo, da bodo kmalu zapustila vidno polje kamere. Nato preiščemo tako izbrano področje slike in izberemo najboljšo točko v skladu s kriterijem, opisanim v [34].

Slika 12 predstavlja 3D posnetek zemljevida prostora, na katerem je označena predvidena lokacija robota (sivi kvader), značilnice pa so ponazorjene z elipsoidi, ki predstavljajo nedoločenosti njihovih lokacij.



Slika 12. Daljša elipsa v smeri določene koordinate pomeni večjo nedoločenost te koordinate v oceni položaja značilnice. Iz slike je lepo razvidno, da je »globina« značilnice tista, ki jo je najtežje oceniti, kar pa je razumljivo, saj z eno samo kamero globine ne moremo direktno zaznavati. Slika pa ne prikazuje dejstva, da so posamezne značilnice med seboj korelirane. To informacijo nosijo neničelni elementi kovariančne matrike *P*.

3.2.2. Problem zagona sistema SLAM

Pri sistemu SLAM ob uporabi ene same kamere se ob zagonu pojavita dva problema. Brez znanih značilnic v prvi sliki zaporedja slik ne moremo takoj vstopiti v glavno zanko procesa SLAM (detekcija, premik, asociacija, posodobitev), saj nimamo nobene značilnice, na podlagi katere bi se lahko orientirali. Ker iz slike zajete z eno samo kamero ne moremo direktno

[3] V praksi velikokrat ne moremo doseči enakomerne posejanosti značilnic po prostoru, saj nekateri deli prostora ne vsebujejo primernih struktur, ki bi nam lahko služile kot značilnice (npr. prazna bela stena).

oceniti globine, na kateri leži neka nova značilnica, moramo značilnico zaznavati skozi zaporedje več slik (iz več različnih položajev), šele nato pa jo lahko umestimo v zemljevid prostora. Drug problem, ki je prav tako povezan z uporabo ene same kamere, pa je določitev merila. V večini realnih aplikacij želimo poznati absolutne razdalje med objekti v prostoru, ki jih brez neke znane reference ne moremo določiti. V ta namen Davison in sodelavci [14] predlagajo zagon sistema s postavitvijo štirih orientacijskih točk, katerih pozicija je znana že vnaprej. Te značilnice umestimo v zemljevid z ničelno nedoločenostjo in te nedoločenosti kasneje ne spreminjamo, saj smo popolnoma prepričani v njihov položaj.

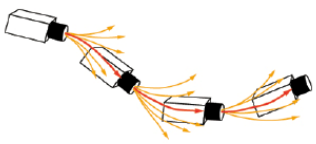
3.2.3. Model gibanja

Čas med dvema posnetkoma, ki jih dobimo iz kamere, je obdobje, ko ne prejemamo nobene informacije o gibanju robota. Ker tudi nimamo nikakršnega podatka o namenih robota (nimamo kontrole nad robotom), lahko le z neko verjetnostjo napovemo, kje se bo robot nahajal v naslednji časovni enoti. Definirati moramo torej nek verjetnostni model gibanja.

Do uporabljenega modela gibanja pridemo s sledečim razmislekom. Recimo, da se robot trenutno giblje z neko hitrostjo v^W in neko kotno hitrostjo ω^R. V vsakem časovnem koraku robot prejme pospešek a^W in kotni pospešek α^R. Ti dve vrednosti si lahko predstavljamo tudi kot šum procesa, njuni vrednosti pa naj bosta porazdeljeni v skladu z normalno porazdelitvijo z aritmetično sredino nič:

$$n = \begin{pmatrix} V^W \\ \Omega^R \end{pmatrix} = \begin{pmatrix} a^W \Delta t \\ \alpha^R \Delta t \end{pmatrix}. \tag{19}$$

Tako lahko govorimo o modelu konstantne hitrosti in konstantne kotne hitrosti, ki ju v vsaki časovni enoti popači šum n. Vizualizacijo uporabljenega modela daje *Slika 13.*



Slika 13. Vizualizacija modela konstantne hitrosti in konstantne kotne hitrosti. V vsakem koraku predvidimo najverjetnejšo pot kamere, skupaj z alternativami majhnega odklona. (povzeto po [14])

Sedaj lahko ocenimo novo stanje robota kot:

$$x_v = \begin{pmatrix} r^W_{new} \\ q^{WR}_{new} \\ v^W_{new} \\ \omega^R_{new} \end{pmatrix} = \begin{pmatrix} r^W + (v^W + V^W)\Delta t \\ q^{WR} \times q((\omega^R + \Omega^R)\Delta t \\ v^W + V^W \\ \omega^R + \Omega^R \end{pmatrix}. \tag{20}$$

Pri modeliranju stanja z razširjenim Kalmanovim filtrom moramo v vsakem koraku izračunati tudi kovarianco procesnega šuma Q_v, ki predstavlja nedoločenost stanja robota. Ta je

posredno določena s P_n, kovarianco vektorja šuma n. S P_n je torej določeno, kako bo rasla nedoločenost v modelu gibanja in z nastavitvijo njenih vrednosti lahko kontroliramo gladkost gibanja robota. Majhne vrednosti v P_n pomenijo bolj enakomerno gibanje s počasnimi spremembami hitrosti, velike pa manj predvidljivo gibanje, kjer pričakujemo bolj sunkovite spremembe hitrosti.

3.2.4. Napovedovanje pozicij značilnic in iskanje ujemanj

V novo prispeli sliki želimo določiti položaj neke značilnice (poiskati ujemanje), ki je že v našem zemljevidu prostora. Najenostavnejši pristop bi bil izčrpno preiskovanje celotne slike. Ker pa naša predstavitev prostora vsebuje informacijo o najverjetnejši točki v prostoru, na kateri se nahaja značilnica, kot tudi verjetnost, da značilnica res leži v tej točki (nedoločenost), lahko ta dva podatka uporabimo za določitev lokacije in velikosti območja v sliki, v katerem iščemo ujemanje. Tako zožimo območje, ki ga pregledujemo, in s tem pohitrimo postopek iskanja ujemanj, ki je sam po sebi ponavadi računsko potraten. Z uporabo ocene položaja robota x_v, in ocene položaja neke značilnice y_i izračunamo 3D položaj značilnice glede na položaj robota:

$$h_{i,3D}^R = R^{RW}(y_i^W - r^W)\,, \tag{21}$$

kjer R^{RW} predstavlja premik iz globalnega koordinatnega sistema v koordinatni sistem robota. Položaj značilnice v sami sliki $\boldsymbol{u} = (u, v)$ dobimo z uporabo modela kamere (*ang.* pinhole camera model):

$$h_i = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} - \frac{1}{h_{i,3D_z}^R} \begin{pmatrix} f_u \cdot h_{i,3D_x}^R \\ f_v \cdot h_{i,3D_y}^R \end{pmatrix}, \tag{22}$$

kjer f_u in f_v predstavljata goriščni razdalji, u_0 in v_0 pa center slike (parametri kamere). Pri kamerah s širokim kotom zajema ponavadi pride do občutne radialne popačenosti[4], ki jo moramo upoštevati pri izračunavanju položajev značilnic v sliki. V tem primeru je položaj značilnice v sliki $\boldsymbol{u}_d = (u_d, v_d)$ določen z dodatno transformacijo zgoraj izračunanih koordinat:

$$u_d - u_0 = \frac{u - u_0}{\sqrt{1 + 2Kr^2}}\,, \tag{23}$$

$$v_d - v_0 = \frac{v - v_0}{\sqrt{1 + 2Kr^2}}\,, \tag{24}$$

kjer je

$$r = \sqrt{(u - u_0)^2 + (v - v_0)^2}\,, \tag{25}$$

[4] Radialno popačenje pomeni spreminjanje (ponavadi zmanjševanje) povečave slike, z oddaljevanjem od optične osi leče. (Ravne linije v prostoru niso predstavljene z ravnimi linijami v sliki.)

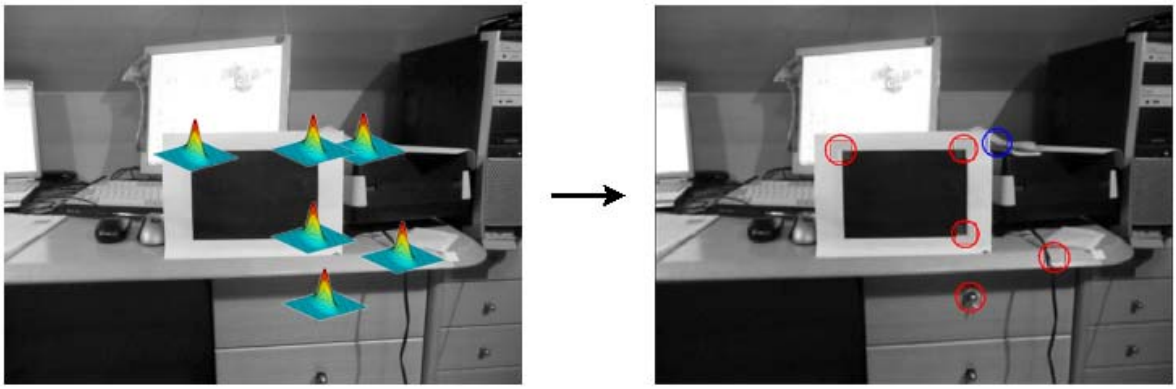
K pa je koeficient popačenosti (parameter kamere).

Z uporabo kovariančne matrike P (ta je del našega zemljevida) in Jacobijeve matrike zgornje projekcijske funkcije, lahko izračunamo tako imenovano inovacijsko kovariančno matriko S_i, ki določa nedoločenost napovedanega položaja značilnice v sliki:

$$S_i = \frac{\partial \boldsymbol{u}_{di}}{\partial x_v} P_{xx} \frac{\partial \boldsymbol{u}_{di}}{\partial x_v}^T + \frac{\partial \boldsymbol{u}_{di}}{\partial x_v} P_{xy_i} \frac{\partial \boldsymbol{u}_{di}}{\partial y_i}^T + \frac{\partial \boldsymbol{u}_{di}}{\partial y_i} P_{y_i x} \frac{\partial \boldsymbol{u}_{di}}{\partial x_v}^T + \frac{\partial \boldsymbol{u}_{di}}{\partial y_i} P_{y_i y_i} \frac{\partial \boldsymbol{u}_{di}}{\partial y_i}^T + R \,, \qquad (26)$$

kjer matrika R predstavlja meritveni šum.

Matrika S_i dimenzije 2x2 predstavlja dvorazsežno normalno verjetnostno porazdelitev prek vrednosti koordinat značilnice. Ob postavitvi nekega praga na to verjetnost dobimo eliptično področje v sliki, v katerem z določeno verjetnostjo leži iskana značilnica. Ujemanje iščemo torej samo znotraj tako določene elipse. Ideja je ponazorjena s *Sliko 14.*



Slika 14. Določanje velikosti preiskovanih območij v okolici ocenjenih položajev značilnic v sliki, prek dvorazsežnih normalnih porazdelitev slikovnih koordinat značilnic, določenih z inovacijskimi kovariančnimi matrikami S_i.

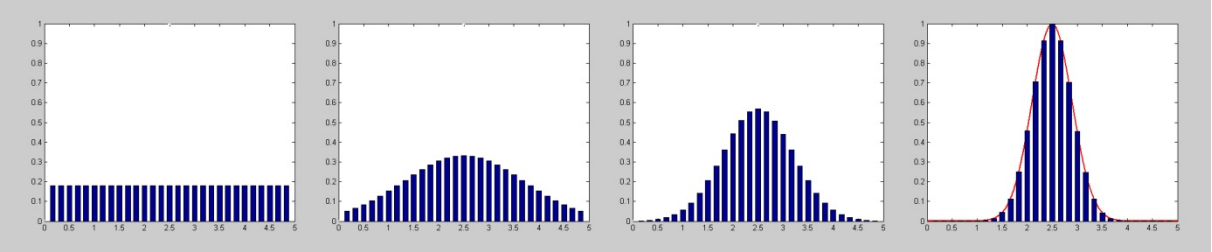
Matrika S_i pa ima še dodatno vlogo v našem sistemu. Določa namreč količino informacije, ki jo pridobimo ob uspešnem lociranju značilnice. Značilnice z veliko S_i so značilnice, katerih položaj je slabo znan in ga je težje napovedati (pripadajoča elipsa je velika). Če bomo torej našli ujemanje s tako značilnico, bomo pridobili več informacije za ocenjevanje položajev objektov v prostoru, kot če bi uspešno locirali neko značilnico z nizko S_i, katere položaj že tako ali tako precej natančno poznamo (pripadajoča elipsa je majhna). Kadar imamo »premalo časa«, da bi poiskali ujemanja z vsemi značilnicami, ki jih imamo v zemljevidu, nam torej S_i služi kot dober kriterij za izbor značilnic, katerih ujemanja bomo iskali. V vsakem koraku se torej lahko omejimo le na množico značilnic z dovolj visokimi vrednostmi S_i. S tako vodenim izbiranjem značilnic tudi preprečimo, da bi za določeno značilnico nedoločenost postala previsoka.

3.2.5. Določanje globine pri novih značilnicah

Ker gre za sistem z eno samo kamero, je ob novo zaznani značilnici nemogoče neposredno določiti njeno globino. Potrebni so premiki robota in več meritev iste značilnice preden lahko povsem določimo položaj značilnice v prostoru. Vemo pa, da značilnica leži nekje na poltraku, ki ima izhodišče v trenutnem položaju robota in se razteza v neskončnost v smeri zaznane značilnice. V zemljevidu tako (delno inicializirano) značilnico predstavimo kot:

$$y_{pi} = \begin{pmatrix} r_i^W \\ \hat{h}_i^W \end{pmatrix}, \tag{27}$$

kjer je r_i^W izhodišče poltraka, na katerem leži značilnica, $\hat{h}_i^W$ pa je enotski vektor, ki določa smer poltraka. Po poltraku nato enakomerno porazdelimo množico delcev, ki na začetku tvorijo uniformno verjetnostno porazdelitev globine (vse globine so enako verjetne). V vsakem od naslednjih korakov vsakega od delcev projiciramo v sliko, ter s postopkom iz *Razdelka 3.2.4* določimo elipse znotraj katerih naj bi se posamezen delec nahajal. Z iskanjem ujemanja znotraj vsake od elips dobimo verjetje (ang. likelihood), da delec dejansko predstavlja lego značilnice. Verjetje dobimo kot vrednost dvorazsežne verjetnostne porazdelitve, ki določa to elipso (glej *Razdelek 3.2.4*), v točki, ki smo jo dobili s postopkom iskanja ujemanja. Po Bayesovem pravilu popravimo verjetnostno porazdelitev globine in postopek ponavljamo, dokler standardni odklon porazdelitve ne postane dovolj majhen, da jo lahko aproksimiramo kot normalno. Njena aritmetična sredina določa globino in značilnico lahko iz delno inicializirane pretvorimo v popolnoma inicializirano. Njene koordinate v prostoru dobimo kot $y_i = r_i^W + \lambda \hat{h}_i^W$, kjer je λ pravkar ocenjena globina. *Slika 15* nazorneje prikazuje opisani postopek.



Slika 15. Spreminjanje verjetnosti različnih hipotetičnih globin značilnice skozi zaporedje slik. Ko lahko verjetnostno porazdelitev globine aproksimiramo kot normalno, oziroma ko razmerje med standardnim odklonom globine in njeno aritmetično sredino (to je dejansko oceno globine) pade pod neko vrednost, značilnico spremenimo iz delno inicializirane v popolnoma inicializirano.

V tem poglavju smo spoznali splošen pristop k reševanju problema sočasne lokalizacije in kartografiranja, ter glavne korake, ki sestavljajo rešitev. Opisana je bila konkretna izvedba sistema SLAM, ki kot edini senzor uporablja eno samo kamero. Uporaba ene same kamere vpelje določene težave (problem zagona sistema in problem določanja globine novim značilnicam), katerih rešitve so bile podrobneje razložene. Pojasnjeno je bilo tudi, kako je

predstavljen zemljevid prostora in kako robot zemljevid prostora, skupaj z oceno svojega zadnjega premika (v skladu s predlaganim modelom gibanja), uporabi za določanje svojega trenutnega stanja. V sledečem poglavju je opisana sama implementacija sledilnika iz *Poglavja 2*, ter njegova integracija v zgoraj opisani sistem SLAM.

4. Implementacija sledenja s sprotnim izbiranjem značilnic in integracija v sistem SLAM

4.1. Implementacija sledenja s sprotnim izbiranjem značilnic v programskem jeziku C++

Algoritem sledenja s sprotnim izbiranjem značilnic smo implementirali na sistemu Linux (distribucija Ubuntu 8.04) v programskem jeziku C++. Za samo delo s slikami smo uporabili paket knjižnic VW34 [35], ki jih je razvila Skupina za aktivni vid (Active vision group) iz Univerze v Oxfordu. Čeprav so knjižnice zelo slabo dokumentirane in nam je to v samem procesu programiranja povzročalo veliko preglavic, smo se kljub temu odločili za njihovo uporabo. Uporabljene so namreč tudi v Davisonovem paketu za realizacijo sistema sočasne lokalizacije in kartografiranja, ki je opisan v *Razdelku 4.2*. Uporaba enakih struktur, kot v omenjenem paketu, nam je omogočila bolj učinkovito združitev našega sledilnika z Davisonovim sistemom za SLAM.

Sama implementacija algoritma sledenja s sprotnim izbiranjem značilnic je ob razumevanju vseh zgoraj navedenih konceptov precej enostavna, zato bom v nadaljevanju podrobneje opisal le nekatere dele. Sledeči razdelek se nanaša na izračunavanje vrednosti značilnic. Gre za opis vmesne strukture, ki nam služi za predstavitev trenutne slike. Uporaba te strukture zmanjša število dostopov v polje slike, ki so potrebni za izračun vrednosti značilnice in tako predstavlja občutno pohitritev delovanja sledilnika. V *Razdelku 4.1.2* je opisan postopek s katerim ob novo prispelem primeru posodobimo šibke klasifikatorje, *Razdelek 4.1.3* pa vpelje dodatno pohitritev, ki jo predstavlja uporaba skupne množice šibkih klasifikatorjev iz katere izbirajo vsi selektorji.

Podrobnosti opisane v sledečih treh razdelkih so v samo implementacijo umeščene v *razdelku 4.1.4*, kjer so tudi opisani glavni moduli (razredi), ki implementirajo sledenje s sprotnim izbiranjem značilnic.

4.1.1. Integralna slika (Integral Image)

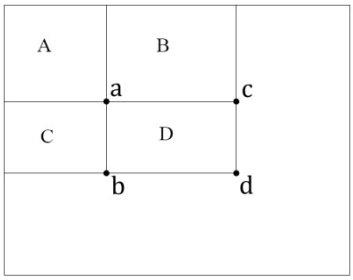
Koncept integralne slike, kot učinkovit pristop k izračunu vrednosti značilnic, sta v svojem delu predstavila Viola in Jones [24]. Z uporabo te strukture lahko v veliki meri pohitrimo postopek izračunavanja vsot intenzitet na pravokotnih področjih slike, ki so osnova za izračun vrednosti mnogih značilnic.

Integralna slika Int slike I je definirana kot:

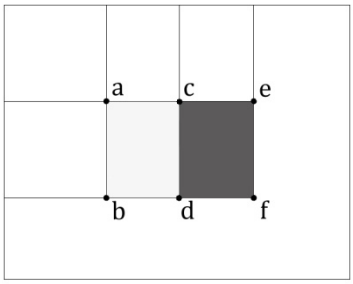
$$Int(x,y) = \sum_{x'=0}^{x} \sum_{y'=0}^{y} I(x',y') \,. \tag{28}$$

Vrednost integralne slike v točki (x,y) je torej vsota točke (x,y) in vseh točk, ki ležijo nad in levo od točke (x,y).

Z uporabo integralne slike lahko vsoto poljubnega pravokotnega območja znotraj slike izračunamo z le štirimi dostopi v integralno sliko (*Slika 16*). V konkretnem primeru Haarovih značilnic, lahko tako razliko med vsotami dveh pravokotnikov izračunamo z osmimi dostopi v integralno sliko. Ker pa pravokotnika v taki značilnici ležita en zraven drugega, pravzaprav zadostuje že šest dostopov (*Slika 17*). Podobno za izračun značilnice s tremi pravokotniki zadostuje osem dostopov in za izračun značilnice s štirimi pravokotniki devet dostopov.



Slika 16. Izračun vsote intenzitet slikovnih elementov v pravokotnem področju označenem z D zahteva štiri branja integralne slike. Vrednost integralne slike v točki a predstavlja vsoto slikovnih elementov v pravokotniku A. Vrednost v točki c je $A+B$, vrednost v točki b je $A+C$, vrednost v točki d pa je $A+B+C+D$. Vsoto slikovnih elementov v D lahko torej izračunamo kot $d-c-b+a = A+B+C+D-(A+B)-(A+C)+A = D$ (štirje dostopi).



Slika 17. Za izračun vrednosti značilnice, to je razlike med vsotami slikovnih elementov v osenčenih pravokotnikih, zadostuje 6 dostopov v integralno sliko. Vsoto slikovnih elementov v svetlo osenčenem pravokotniku izračunamo kot $d-c-b+a$, vsoto v temno osenčenem pravokotniku pa kot $f-e-d+c$. Vrednost značilnice, kot razliko med vsoto v temnem pravokotniku in vsoto v svetlem pravokotniku tako dobimo z $f-e-d+c-(d-c-b+a) = f-e-2d+2c-b+a$ (šest dostopov).

Koncept integralne slike lahko uporabimo na poljubnih poljih nenegativnih števil. Tako lahko tudi enačbo (7) iz *Razdelka 2.2.2* izračunamo le s štirimi dostopi v ustrezno integralno sliko in na ta način pohitrimo tudi izračun značilnic tipa EOH.

4.1.2. Posodobitev šibkih klasifikatorjev z uporabo Kalmanovega filtra

Naj $f_i(\boldsymbol{x})$ predstavlja vrednost značilnice ovrednotene nad primerom (sliko) $\boldsymbol{x}$. Verjetnostni porazdelitvi vrednosti značilnic za množici do sedaj videnih pozitivnih in negativnih primerov modeliramo z Normalnima porazdelitvama $\mathcal{N}(\mu^+, \sigma^+)$ oziroma $\mathcal{N}(\mu^-, \sigma^-)$. Aritmetični sredini μ^+ in μ^- ter standardna odklona σ^+ in σ^- porazdelitev določamo z inkrementalnim posodabljanjem z uporabo Kalmanovega filtra [33]. Ob vsakem novo prispelem primeru (v kontekstu Kalmanovega filtra govorimo o opravljeni meritvi), popravimo ti dve vrednosti v skladu s sledečimi enačbami:

$$K_t = \frac{P_{t-1}}{P_{t-1}+R}\,, \tag{29}$$

$$\mu_t = K_t \cdot f_i(\boldsymbol{x}) + (1 - K_t) \cdot \mu_{t-1}\,, \tag{30}$$

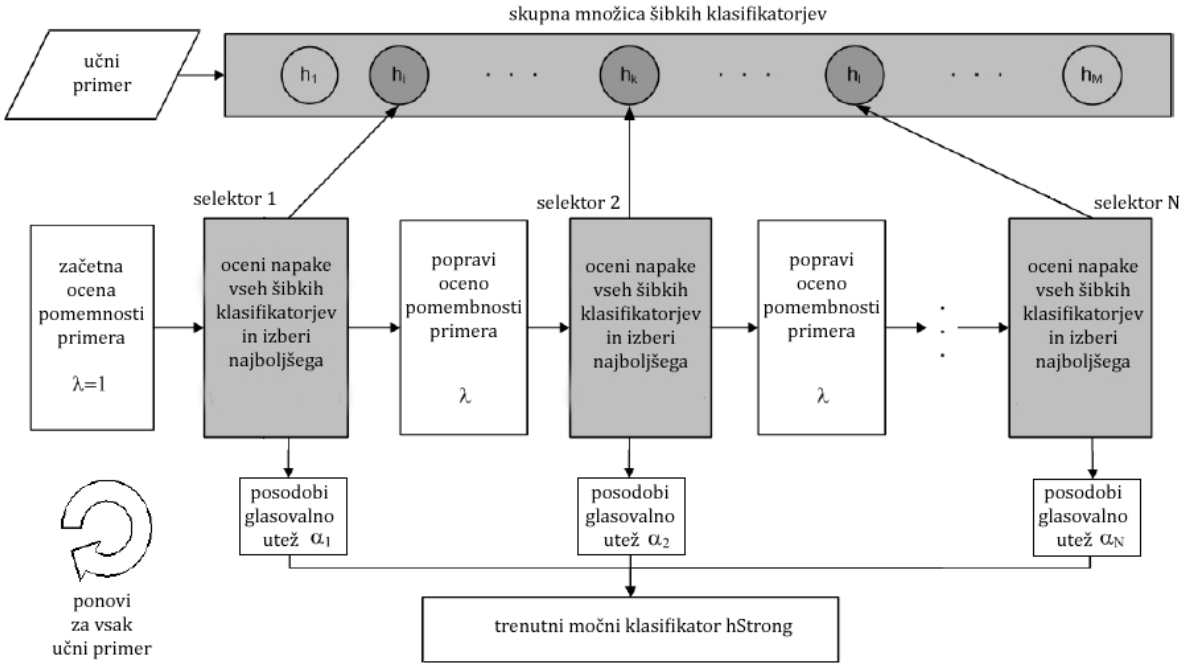
$$\sigma_t^2 = K_t \cdot (f_i(\boldsymbol{x}) - \mu_t)^2 + (1 - K_t) \cdot \sigma_{t-1}^2\,, \tag{31}$$

$$P_t = (1 - K_t) \cdot P_{t-1}\,. \tag{32}$$

Začetne vrednosti P_0, μ_0, σ_0^2 in vrednost R, ki določa šumnost meritev (novih primerov), nastavimo kot je to storjeno v [18], to je $P_0 = 1000$, $\mu_0 = 0$, $\sigma_0^2 = 0$ in $R = 0{,}01$.

4.1.3. Pohitritev z uporabo skupne množice šibkih klasifikatorjev

Recimo, da za učenje v našem algoritmu uporabljamo 100 selektorjev, od katerih vsak izbira iz množice 500 šibkih klasifikatorjev (značilnic). V tem primeru bi morali za vsako sliko posodobiti 50000 šibkih klasifikatorjev. Prav posodabljanje šibkih klasifikatorjev je najbolj potraten del algoritma, kar nas pripelje do ideje, da vsem selektorjem ponudimo isto množico šibkih klasifikatorjev. Tako moramo v vsakem koraku posodobiti le eno množico šibkih klasifikatorjev (v našem primeru 500). S tem se sicer nekoliko oddaljimo od koncepta modeliranja težavnosti novega primera s parametrom λ, kot je bilo opisano zgoraj, saj sedaj posodobimo celotno množico šibkih klasifikatorjev naenkrat, neodvisno od parametra λ. Težavnost primera λ uporabimo le za za oceno napak klasifikatorjev, na podlagi katerih selektorji izberejo najboljši klasifikator. Ker sedaj vsi selektorji izbirajo iz iste množice šibkih klasifikatorjev, se lahko zgodi, da želi več selektorjev izbrati isti šibek klasifikator. Temu se enostavno izognemo tako, da implicitno prepovemo izbiro šibkega klasifikatorja, ki že pripada nekemu selektorju. Tako poenostavljen postopek ponuja precejšno pohitritev ob še vedno zadovoljivih rezultatih. Princip je ponazorjen s *Sliko 18 in Algoritmom 2*.



Slika 18. Princip uporabe metode boosting za sprotno izbiranje značilnic. Po posodobitvi celotne skupne množice šibkih klasifikatorjev, uporabimo oceno pomembnosti učnega primera λ le za izbiro najboljšega šibkega klasifikatorja in za izračun ocene njegove napake. (povzeto po [17])

Vhod: učni primer $\langle \boldsymbol{x}, y \rangle$, $y \in \{-1, +1\}$
močen klasifikator h^{strong} (inicializiran naključno)
uteži λ_m^{corr}, λ_m^{wrong} (inicializirane na 1)

inicializiraj utež pomembnosti $\lambda = 1$
// posodobi vse šibke klasifikatorje
for $m = 1,2, \ldots, M$
$h_m^{weak} = posodobi(h_m^{weak}, \langle \boldsymbol{x}, y \rangle)$
end for

// za vsak selektor
for $n = 1,2, \ldots, N$ **do**
// oceni napake šibkih klasifikatorjev
for $m = 1,2, \ldots, M$
if $h_m^{weak}(\boldsymbol{x}) = y$ **then**
$\lambda_m^{corr} = \lambda_m^{corr} + \lambda$
else
$\lambda_m^{wrong} = \lambda_m^{wrong} + \lambda$
end if
$e_m = \frac{\lambda_m^{wrong}}{\lambda_m^{corr} + \lambda_m^{wrong}}$
end for
// izberi šibek klasifikator z najmanjšo napako, ki še ni bil izbran
$m^+ = \arg\min_m(e_m)$
$e_n = e_{m^+}$; $h_n^{sel} = h_{m^+}^{weak}$
if $e_n = 0$ **or** $e_n > \frac{1}{2}$ **then**
continue
end if

//izračunaj glasovalno utež
$\alpha_n = \frac{1}{2} \cdot \ln\left(\frac{1-e_n}{e_n}\right)$

// posodobi utež pomembnosti
if $h_n^{sel}(\boldsymbol{x}) = y$ **then**
$\lambda = \lambda \cdot \frac{1}{2 \cdot (1-e_n)}$
else
$\lambda = \lambda \cdot \frac{1}{2 \cdot e_n}$
end if

// zamenjaj najslabši šibek klasifikator z novim
$m^- = \arg\max_m(e_m)$
$\lambda_{m^-}^{corr} = 1$; $\lambda_{m^-}^{wrong} = 1$
pridobi nov $h_{m^-}^{weak}$
end for

Algoritem 2. Metoda boosting za sprotno izbiranje značilnic iz skupne množice šibkih klasifikatorjev.

4.1.4. Kratek opis glavnih razredov, ki implementirajo sledenje

V tem razdelku so opisani glavni moduli (razredi), ki implementirajo sledenje s sprotnim izbiranjem značilnic. V *Prilogi A* so vključene glave (*ang*. header) v nadaljevanju opisanih razredov, ki vsebujejo prototipe vseh metod, ki jih posamezen razred implementira.

Razred `OnlineAdaboostTracker`

V tem razredu se odvija glavna zanka sledenja. V grobem lahko razred uporabimo na dva načina, zato je tudi večina metod v njem preobloženih (*ang.* overloaded). V prvem načinu razred interno hrani vse potrebne podatkovne strukture (množica značilnic, integralne slike). Tako mora uporabnik ob vsakem klicanju metod za premik sledilnega okna le priskrbeti naslednjo od slik v zaporedju. V drugem načinu pa moramo vsako od teh struktur podati razredu od zunaj. Razlog za implementacijo drugega načina, je v tem, da kadar hočemo na nekem zaporedju slik slediti več kot samo enemu objektu, bi bilo nesmiselno te strukture izračunavati za vsak sledilnik posebej. Tako na primer, integralno sliko izračunamo globalno, vsakemu od inicializiranih sledilnikov pa podamo le referenco nanjo. Enako velja tudi za množico značilnic, v primeru, ko vsi sledilniki uporabljajo sledilno okno enake velikosti. Množico generiramo globalno in vsakemu od sledilnikov podamo referenco nanjo. Za sam premik okna imamo na voljo tri metode: prva enostavno preišče okolico prejšnje pozicije objekta, ki je dvakrat večja od velikosti sledilnega okna, druga omogoča izbiro poljubne velikosti preiskovane okolice, tretja pa išče po eliptičnih območjih v okolici poljubno izbrane točke.

Razred `StrongClassifier`

V tem razredu se nahaja jedro Adaboost algoritma. Razred vsebuje polje šibkih klasifikatorjev in polje selektorjev. Slednji so v resnici le kazalci v polje šibkih klasifikatorjev. V metodi `OnlineAdaboostUpdate(Example &example)` je implementiran *Algoritem 2* iz *Razdelka 4.1.3*.

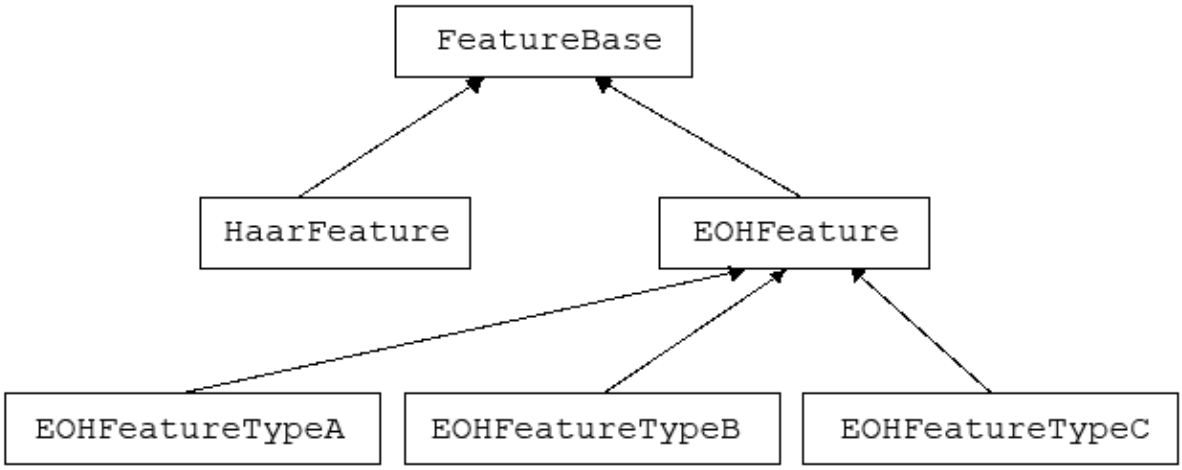
Razred `WeakClassifier`

Razred `WeakClassifier` vsebuje strukture, ki hranijo podatke o značilnici, ki jo objekt tega tipa uporablja za klasifikacijo in vse parametre opisane v *Razdelku 2.3.1*. Glavno funkcionalnost razreda predstavljata metodi `Update(Example &example)`, in `Classify(Example &example)`. Prva posodobi šibek klasifikator v skladu s postopkom, opisanim v *Razdelku 4.1.2*, druga pa vrne napovedani razred v skladu s postopkom, opisanim v *Razdelku 2.3.1*.

Razredi značilnic

Razred `FeatureBase` je abstraktni nadrazred vseh razredov značilnic (značilnice EOH ter Haarove značilnice). Vsebuje virtualno metodo `CalculateFeatureValue(Example &example, OnlineAdaboostTracker *tracker)`, ki vrne vrednost značilnice ovrednotene nad podanim primerom `example`. Metoda kot vhod zahteva tudi referenco na sledilnik `tracker`. Ta je potrebna zato, ker strukture (različne integralne slike opisane v

Razdelku 4.1.1), ki jih potrebujemo za dejanski izračun vrednosti značilnice prebivajo v razredu `OnlineAdaboostTracker`. Primer `example` nam namreč zgolj pove nad katerim območjem v sliki moramo aplicirati značilnico. S tem se izognemo nepotrebnemu prepisovanju podatkov, do katerega bi prišlo, če bi struktura `example` vsebovala dejanski izsek slike (*ang.* image patch). Na tem mestu bi se morda lahko vprašali, zakaj metodi `CalculateFeatureValues` ne podamo kar reference na strukturo, ki jo potrebujemo za izračun vrednosti značilnice. Problem je v tem, da različni tipi značilnic za izračun svojih vrednosti potrebujejo različne strukture. S tem, ko bi v vsakem tipu značilnice deklarirali drugačno metodo za izračun vrednosti, bi izgubili koncept polimorfizma in s tem možnosti za enostavno dodajanje novih tipov značilnic. *Slika 19* podaja diagram dedovanja med uporabljenimi razredi značilnic.



Slika 19. Diagram dedovanja med razredi značilnic.

4.2. Davisonov paket SceneLib

Knjižnice, vsebovane v paketu SceneLib (dostopen na [36]), predstavljajo ogrodje za realizacijo sistema za sočasno lokalizacijo in kartografiranje, s konkretno implementacijo za sistem SLAM z eno samo kamero, ki deluje v realnem času. Vse so napisane v programskem jeziku C++.

V nadaljevanju so na kratko opisane tri knjižnice, ki jih vsebuje paket SceneLib in njihovi glavni razredi:

- Scene je knjižnica, ki nudi osnovno funkcionalnost sistema SLAM. Sestavljajo jo razredi:
  - `Scene_Single`
    V tem razredu se hranijo ocene položajev robota in značilnic, skupaj z njihovimi nedoločenostmi. Objekt tega razreda si lahko predstavljamo, kot

posnetek stanja sveta kot ga pozna robot v določenem trenutku. Vse značilnice, ki so trenutno v zemljevidu, so shranjene v obliki povezanega seznama objektov tipa `Feature`. Celotna kovariančna matrika pa je shranjena v objektu tipa `VNL::Matrix`[5].

- `Feature`
  Vsebuje stanje značilnice in njeno kovarianco, kovarianco med značilnico in robotom, njen zadnji izmerjeni položaj, ter njen napovedani položaj. Razred je uporabljen tako za delno inicializirane značilnice, kot tudi za popolnoma inicializirane, omogoča pa tudi pretvorbo iz prve oblike značilnice v drugo.
- `Kalman`
  Implementacija razširjenega Kalmanovega filtra. Uporablja strukture, ki se hranijo v razredu `Scene_Single`.

- `SceneImProc` je paket metod za obdelavo slik. Vsebuje metode za iskanje korelacije med dvema slikama, metode za iskanje značilnice v sliki in metode za odkrivanje novih dobrih značilnic v sliki.

- `MonoSLAM` je konkretna implementacija sistema SLAM z eno samo kamero. Vsebuje sledeče razrede:
  - `ImagemonoExtraData`
    Vsebuje podatke, ki določajo značilnico v sliki. V konkretni implementaciji gre kar za izsek slike velikosti 11x11 slikovnih elementov.
  - `MonoSLAM`
    Ta razred ustvari in hrani instance objektov vseh glavnih razredov, potrebnih za delovanje sistema SLAM. Glavni potek dogodkov v izvajanju sistema SLAM se dogaja znotraj tega razreda.
  - `Robot`
    V tem razredu je implementiran vmesnik, ki skrbi za komunikacijo z robotom in posreduje meritve, ki jih je le ta opravil, razredom iz knjižnice `Scene`
  - `threeddraw`
    Skupek funkcij za izrisovanje stanja robota in značilnic. Omogoča izrisovanje skozi »oči« robota in izrisovanje 3D prostora iz zunanjega pogleda. Vse funkcije temeljijo na knjižnici `OpenGL`.

[5] VNL (Vision Numerics Library) je ena od knjižnic, ki jih vsebuje v *Razdelku 4.1* omenjeni paket VW34. Vsebuje strukture in metode za delo z matrikami in vektorji ter nekatere numerične algoritme

4.3. Integracija sledilnika s sprotnim izbiranjem značilnic v MonoSLAM

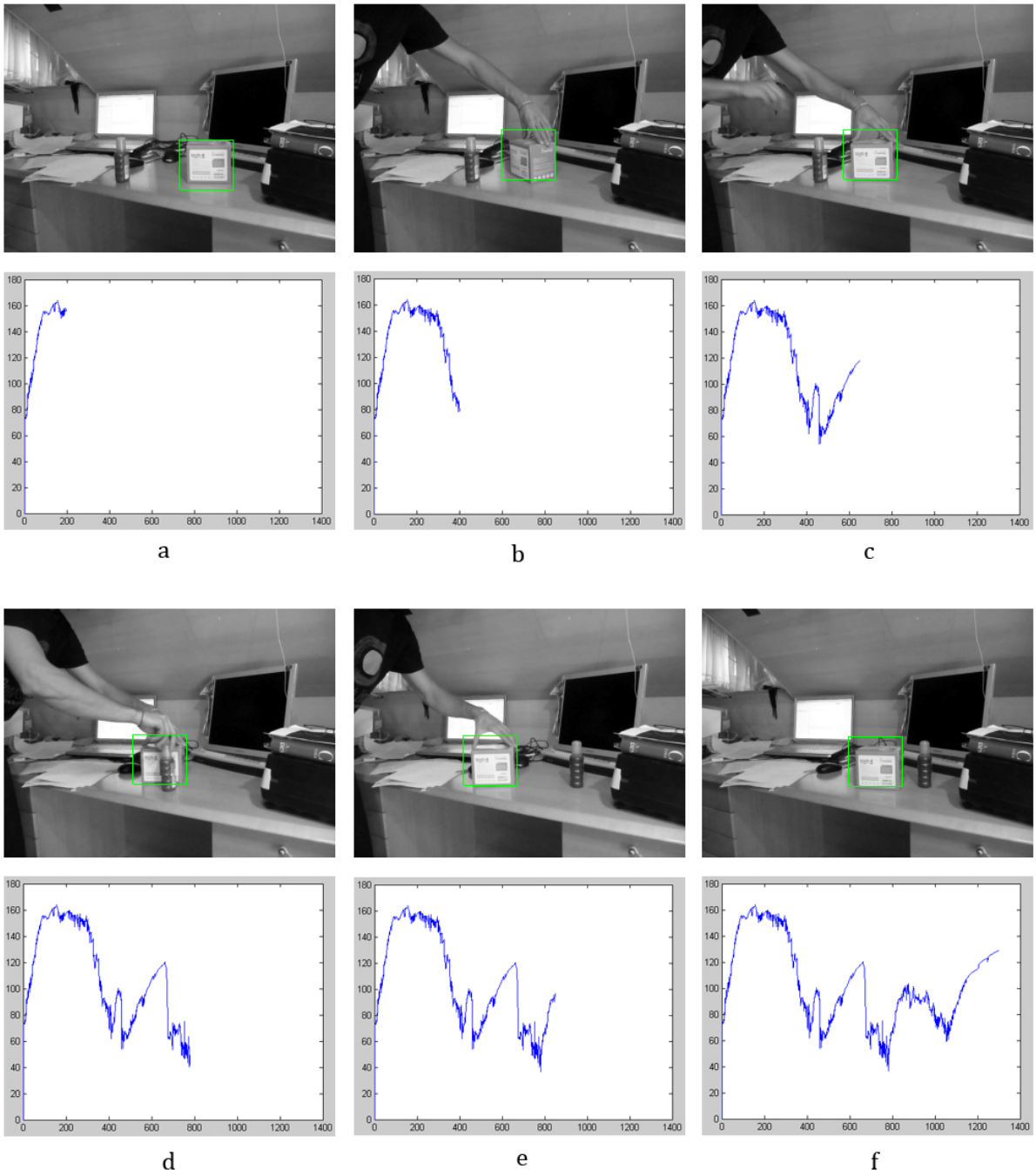
Davisonov program MonoSLAM za vsako značilnico v prostoru shrani izsek slike velikosti 11x11 slikovnih elementov v okolici točke, ki predstavlja značilnico v sliki v trenutku, ko je bila značilnica prvič zaznana in umeščena v zemljevid. Za iskanje položaja te značilnice v novi sliki je uporabljena enostavna metoda korelacije. Ker pa robot prostor zaznava iz različnih pogledov, to pa pomeni, da se izgled značilnih točk v prostoru lahko zelo spremeni, ima tak pristop le omejeno uporabo. Kar želimo je to, da bi se predstavitev neke značilnice stalno posodabljala, kar bi omogočilo uspešno iskanje ujemanj skozi celotno zaporedje slik. Tu pride na vrsto naše sledenje s sprotnim izbiranjem značilnic. Ko v zemljevid dodamo novo značilnico, namesto da si zapomnimo izsek slike v njeni okolici, nad tem območjem enostavno inicializiramo nov sledilnik. Le-tega čez celotno zaporedje slik posodabljamo in ga tako prilagajamo spreminjajočemu se izgledu značilnice.

Za uspešno integracijo sledilnika je bilo potrebnih zelo malo sprememb osnovnega programa `MonoSLAM`. V razred `ImagemonoExtraData`, ki je do sedaj vseboval izsek slike v okolici značilnice, smo dodali objekt tipa `OnlineAdaboostTracker`, ki predstavlja naš sledilnik. V samem sledilniku smo tudi implementirali metodo `MoveTrackingWindow-EllipticalSearch`, ki omogoča iskanje ujemanja v eliptičnem območju določenem z nedoločenostjo značilnice. Ker pa sledilnik ob vsakem premiku okna vrne tako položaj novega okna (nov položaj sledenega objekta), kot tudi maksimalno vrednost slike zaupanja (ki jo je zgradil med določanjem nove pozicije sledenega objekta), lahko slednjo vrednost uporabimo za odločanje o tem ali je ujemanje uspešno ali ne. `MonoSLAM` namreč dopušča možnost neuspešnega lociranja neke značilnice v sliki. Takrat se poveča njena nedoločenost in v naslednji sliki bo preiskovano eliptično območje temu primerno večje. S postavitvijo praga na zaupanje, ocenimo ali je prišlo do uspešnega ujemanja ali ne. Seveda sledilnik posodobimo samo v primeru, da je do uspešnega ujemanja res prišlo, saj ne želimo, da se prilagodi na napačno področje v sliki.

5. Preizkusi in rezultati

Vse preizkuse smo opravili na računalniku z 2,5 gigaherčnim procesorjem in 3,5 gigabajti delovnega spomina, v operacijskem sistemu Linux Ubuntu 8.04.

V sledilnikih smo uporabili množico petdesetih selektorjev, ki izbirajo iz skupne množice dvestopetdesetih šibkih klasifikatorjev. Preizkusi samega sledilnika so pokazali, da taka izbira zagotavlja zadostno natančnost sledenja ob dokaj hitrem delovanju. Pri izračunih značilnic EOH smo uporabili 4 koše orientacij robov. Delovanje enega samega sledilnika na zaporedju slik, v katerem pride do precejšnih sprememb v videzu sledenega objekta prikazuje *Slika 20*.

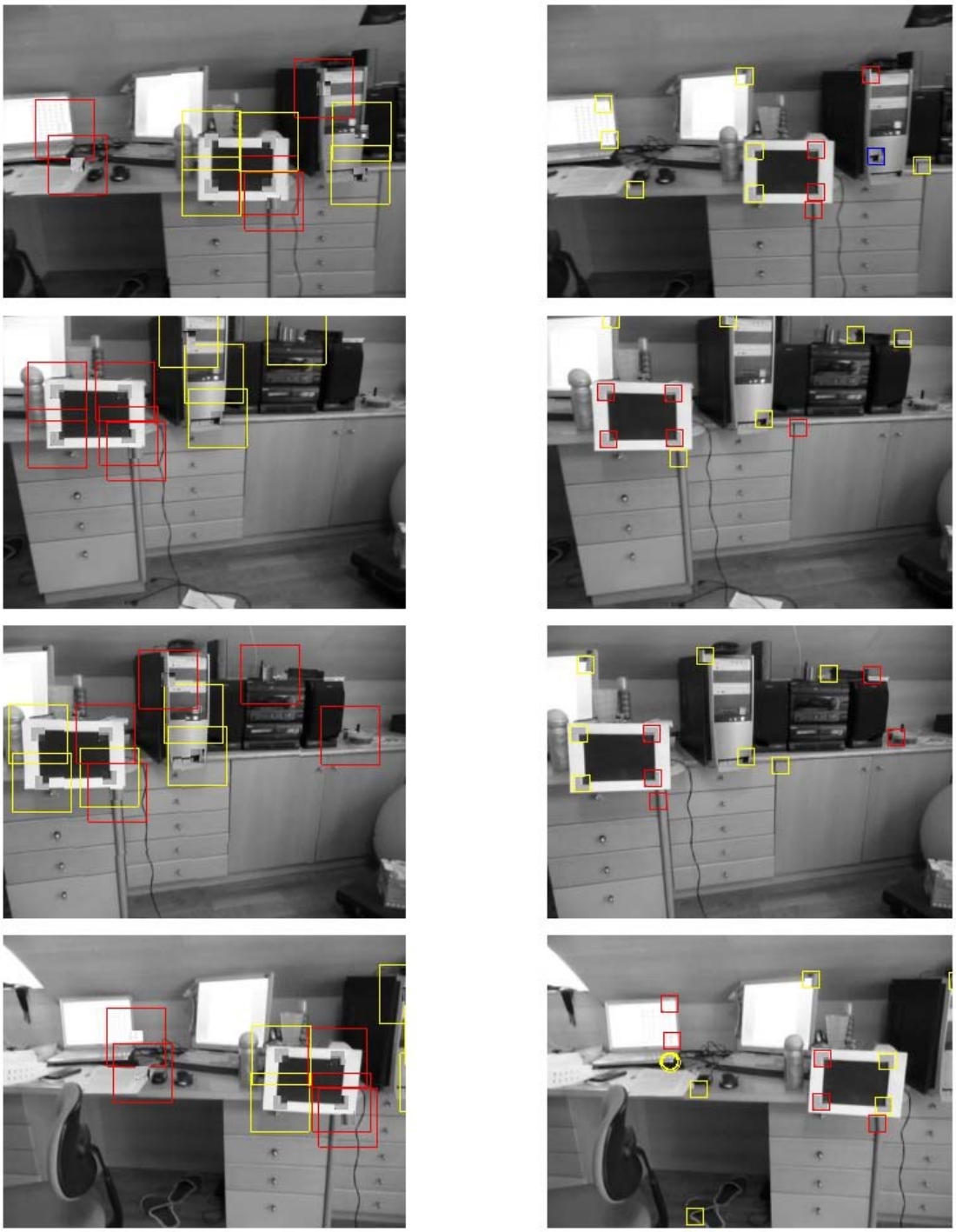


Slika 20. Prikaz delovanja sledenja s sprotnim izbiranjem značilnic. Zgornje od parov slik prikazujejo posamezne slike zaporedja, spodnje pa kako se spreminja največja vrednost slike zaupanja, ki jo uporablja sledilnik, da določi nov položaj sledenega objekta. Ko se videz objekta ne spreminja, se največja vrednost slike zaupanja veča (a). Nato se videz objekta spremeni (objekt zarotiramo) in vrednost se zmanjša (b), vendar se novemu izgledu objekta sledilnik prilagodi in največja vrednost slike zaupanja se zopet poveča (c). Ko sledeni objekt zakrije nek drug predmet iz prostora, največja vrednost slike zaupanja spet pade (d), vendar sledilnik uspe objektu uspešno slediti še naprej (e, f).

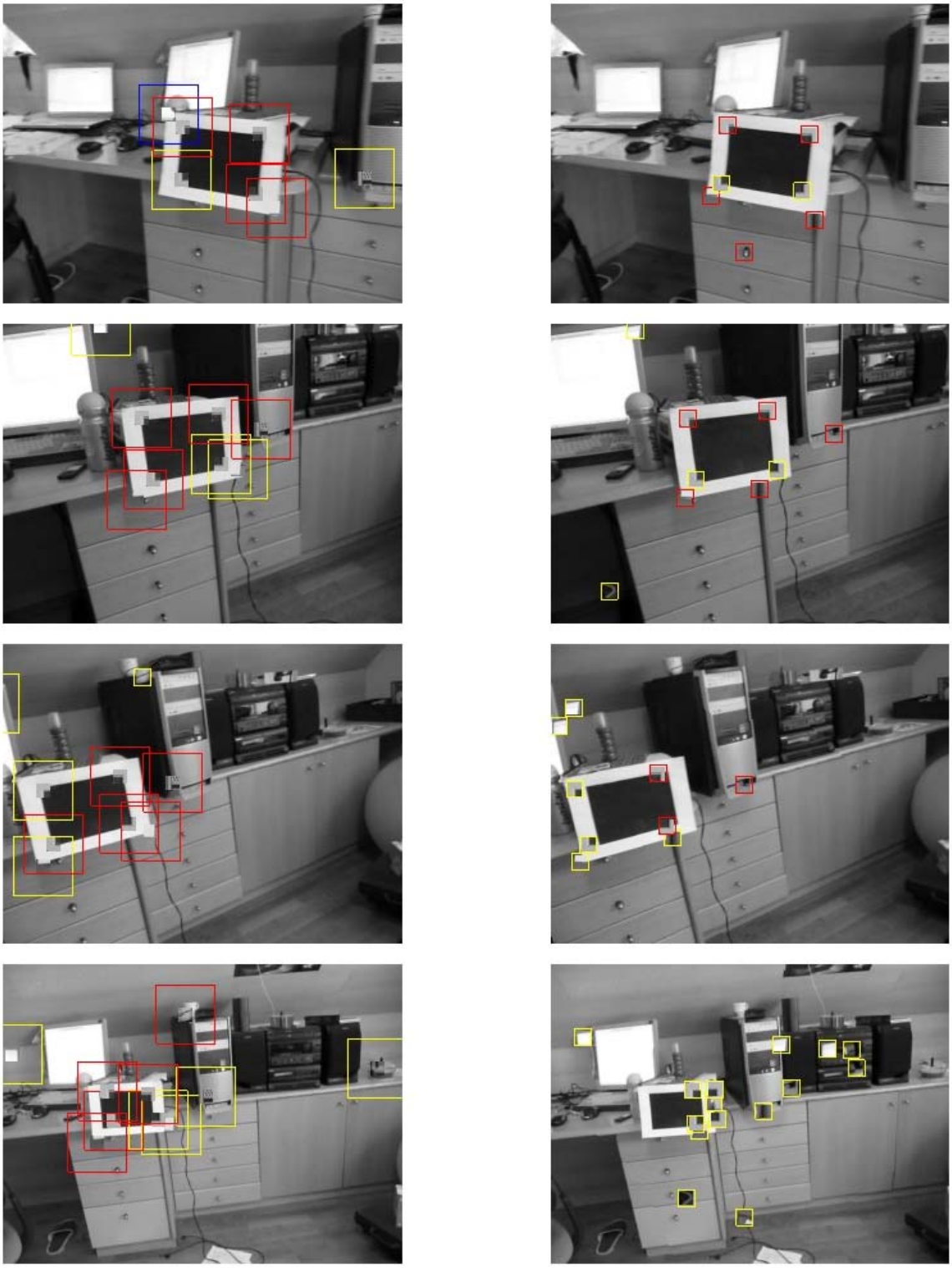
V prvem preizkusu smo primerjali delovanje osnovnega sistema MonoSLAM in sistema MonoSLAM z uporabo sledilnikov. Zaporedja slik so bila posneta s kamero *Panasonic DMC-TZ3* (kompakten fotografski aparat, ki omogoča snemanje videa). Za določanje parametrov kamere (*Razdelek 3.2.4*) smo uporabili Matlabovo orodje za umerjanje kamere [37]. Uporabili smo sledeče vrednosti: $f_u = f_v = 269$, $u_0 = 162$, $v_0 = 121$, ter $K = 0$ (radialne popačenosti nismo upoštevali, saj je bila zanemarljivo majhna). V vsakem koraku je bilo za določanje položaja kamere (robota) uporabljenih pet značilnic. V konfiguraciji s sledilniki smo uporabili sledilnike z okni velikosti 44x44 slikovnih elementov (sledili smo območjem velikosti 44x44 slikovnih elementov).

V prvem zaporedju slik se kamera giblje predvsem v stranski smeri, brez kakšnih večjih rotacij, oziroma se giblje bolj ali manj po ravnini, ki je vzporedna z inicializacijsko tarčo. Tako so spremembe v izgledu značilnic majhne in obe konfiguraciji sistema delujeta skoraj enako (*Slika 21*). Pri drugem posnetku pa so prisotne večje rotacije kamere, ter več gibanja po sami globini slike (gibanje naprej in nazaj) (*Slika 22*), kar pomeni, da prihaja do večjih sprememb v izgledu značilnic katerim sledimo. Izkaže se, da v drugem primeru uporaba sledilnikov izboljša delovanje sistema, saj se sledilniki spremenjenim izgledom značilnic prilagodijo. Osnovni sistem pa za določanje položaja značilnice v novi sliki uporablja enostavno metodo korelacije, ki pa pri tako spremenjenem izgledu značilnic odpove.

Seveda pa je potrebno omeniti tudi, da je sistem z uporabo sledilnikov deloval veliko počasneje od osnovnega sistema (z uporabo sledilnikov smo v povprečju dosegli hitrost delovanja 3,6 slik na sekundo, z osnovnim sistemom pa 50,9 slik na sekundo).



Slika 21. Levi stolpec prikazuje delovanje sistema z uporabo sledilnikov z velikostmi okna 44x44 slikovnih elementov. Desni stolpec prikazuje delovanje osnovnega sistema. Ker gre za zaporedje slik, kjer se izgledi značilnic ne spreminjajo veliko, delujeta oba sistema enako dobro. Rumeni okvirčki predstavljajo napovedane položaje značilnic, ki jih v trenutni sliki nismo izbrali za določanje stanja robota. Rdeči okvirčki predstavljajo značilnice, ki smo jih v trenutni sliki izbrali in tudi uspešno določili njihov položaj. Modri okvirčki pa predstavljajo v trenutni sliki izbrane značilnice, katerih določitev položaja ni uspela. Znotraj okvirčkov so izrisani tudi deli slik, nad katerimi so bile značilnice inicializirane. To omogoča primerjavo dejanskega položaja značilnic z ocenami položajev, ki jih vrača sistem, ki ga preizkušamo. Zaradi preglednosti so v obeh primerih izrisani »inicializacijski« deli slik veliki le 11x11 slikovnih elementov, čeprav sledilniki sledijo območjem velikim 44x44 slikovnih elementov.



Slika 22. Levi stolpec prikazuje delovanje sistema z uporabo sledilnikov z velikostmi okna 44x44 slikovnih elementov. Desni stolpec prikazuje delovanje osnovnega sistema. V zadnji sliki vidimo, da se je kamera pomaknila daleč nazaj, kar je imelo za posledico precejšne spremembe v izgledu značilnic. Sledilniki so se na te spremembe uspešno prilagodili, osnovni sistem pa je v tej situaciji odpovedal. Iz slike spodaj desno je lepo razvidno, da so vsi napovedani položaji značilnic v sliki napačni.

V drugem preizkusu smo sistem MonoSLAM z uporabo našega sledenja preizkusili na zaporedju slik posnetih s kamero s širokim kotom zajema (testni posnetek, ki je vsebovan v paketu `SceneLib`). Širok kot namreč pripomore k temu, da določena značilnica ostane dalj časa v vidnem polju kamere, tako da ji lahko sledimo skozi daljše obdobje v zaporedju. Parametri kamere so bili nastavljeni na sledeče vrednosti: $f_u = f_v = 195,\ u_0 = 162,$ $v_0 = 125$, ter $K = 6 \times 10^{-6}$.

V prvem delu posnetka je vselej vidna inicializacijska tarča, v drugem delu pa le-ta zapusti vidno polje kamere. Gibanje je predvsem stransko brez večjih premikov po sami globini prostora. Kamera se sprva giblje počasi, nato pa (nekje okrog 700. slike) pride do nekaj hitrejših premikov. Gre torej za nekakšno stopnjevanje zahtevnosti situacije v kateri preizkušamo sistem.

Delovanje smo preizkušali glede na nastavitev sledečih parametrov:

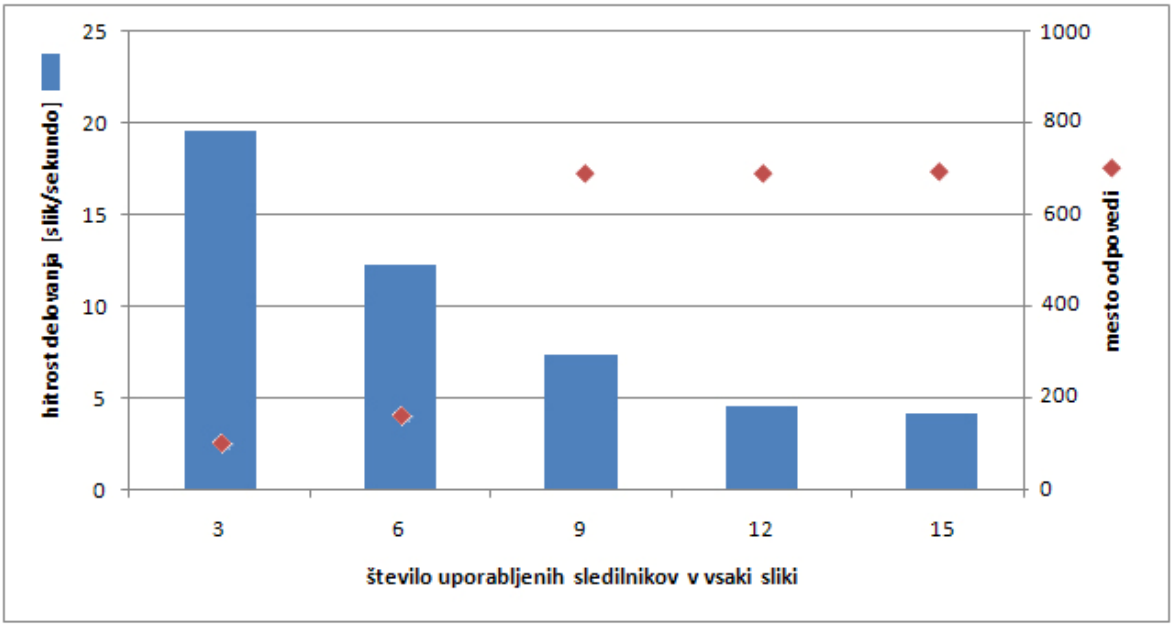
- Katere značilnice uporabljamo pri sledenju. (Uporabimo samo Haarove značilnice ali Haarove značilnice v kombinaciji z značilnicami EOH.)

- Koliko sledilnikov uporabimo v vsaki sliki za sledenje značilnim točkam v prostoru.

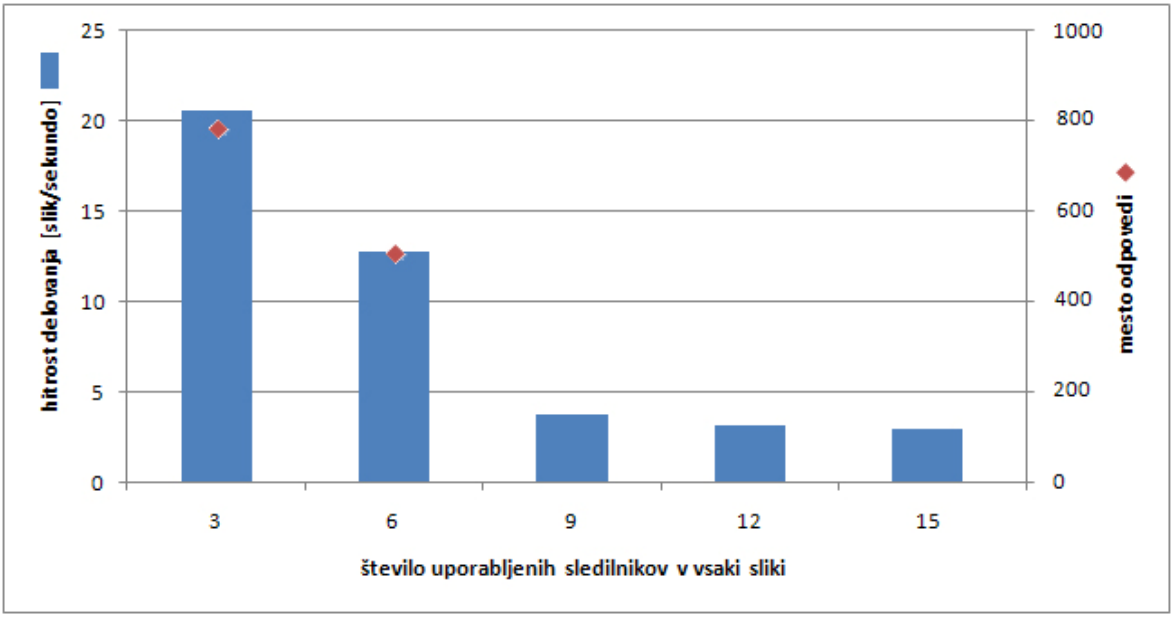
- Kakšna naj bo velikost sledilnega okna sledilnika, oziroma kako velika naj bodo območja v sliki katerim sledimo.

Prvi preizkusi so nemudoma pokazali, da pri uporabi Haarovih značilnic skupaj z značilnicami EOH, kar se tiče natančnosti sledenja značilnim točkam v prostoru, sistem ne deluje nič bolje, kot če uporabimo samo Haarove značilnice. Značilnice EOH so namreč zelo uporabne v problemih, kjer je potrebno določiti ali nek objekt spada v neko skupino objektov ali ne. V [26] gre na primer za odločanje o tem, ali je na sliki obraz ali ne. V omenjenem primeru moramo z značilnicami nekako posplošiti celoten razred obrazov, oziroma z njimi povzeti neke zakonitosti, ki veljajo za vse slike obrazov. Ker lahko z značilnicami EOH zajamemo kompleksnejše geometrijske strukture, ki veljajo za celoten razred obrazov, se v zgornjem primeru izkažejo bolje, kot enostavne Haarove značilnice. V našem primeru pa gre za sledenje točno določenim objektom v prostoru (recimo zgornjemu desnemu robu ekrana), ki jih lahko dovolj natančno predstavimo z enostavnejšimi Haarovimi značilnicami. Uporaba značilnic EOH tudi močno upočasni delovanje sistema, kar gre pripisati večji računski zahtevnosti njihovega izračuna. Z uporabo obeh vrst značilnic sistem v povprečju deluje štirikrat počasneje, kot z uporabo le Haarovih značilnic (ob uporabi sledilnikov z okni velikosti 22x22 slikovnih elementov je bila povprečna hitrost pri uporabi obeh vrst značilnic 2,1 slike na sekundo, pri uporabi le Haarovih značilnic pa 8,6 slik na sekundo). Zato smo v vseh nadaljnjih preizkusih v sledilnikih uporabili le množico Haarovih značilnic.

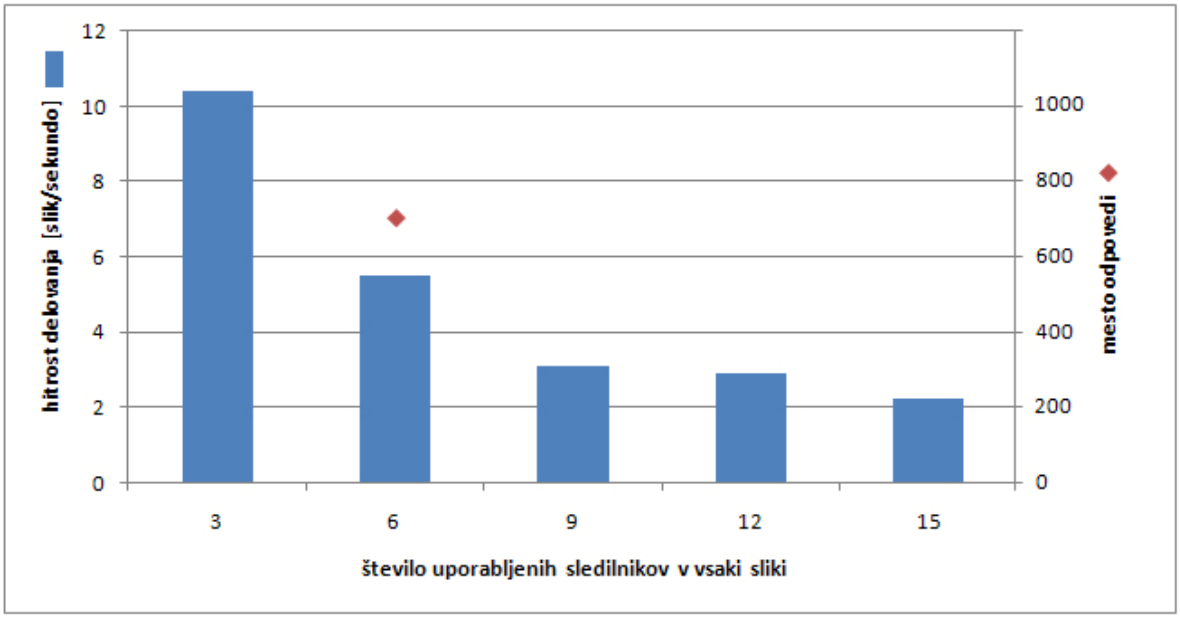
Grafi 1, 2 in 3 povzemajo delovanje sistema pri spreminjanju števila sledilnikov in velikosti sledilnega okna. Uporabili smo 3, 6, 9, 12 oziroma 15 sledilnikov, ter okna velikosti 11x11 (*Graf 1*), 22x22 (*Graf 2*) in 44x44 (*Graf 3*) slikovnih elementov.



Graf 1. SLAM s sledilniki z velikostjo sledilnega okna 11x11 slikovnih elementov. Graf prikazuje hitrost delovanja sistema SLAM v odvisnosti od števila sledilnikov, ki jih uporabimo za določitev položaja značilnice v vsaki sliki zaporedja. Z rdečimi pikami je označeno v kateri sliki (desna skala) je sistem v posameznih konfiguracijah odpovedal.

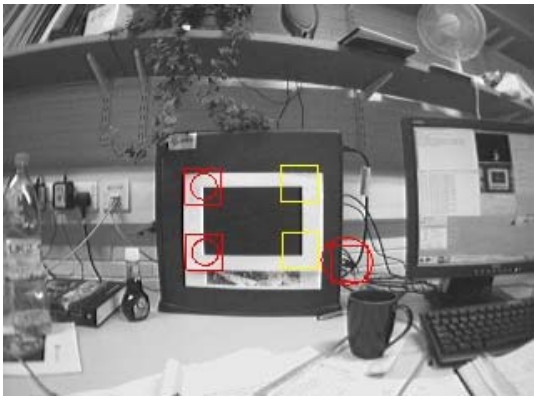


Graf 2. SLAM s sledilniki z velikostjo sledilnega okna 22x22 slikovnih elementov. Graf prikazuje hitrost delovanja sistema SLAM v odvisnosti od števila sledilnikov, ki jih uporabimo za določitev položaja značilnice v vsaki sliki zaporedja. Z rdečima pikama je označeno v kateri sliki (desna skala) je sistem v posameznih konfiguracijah odpovedal. Kjer pike ni, je sistem uspešno deloval skozi celotno zaporedje 1000 slik.

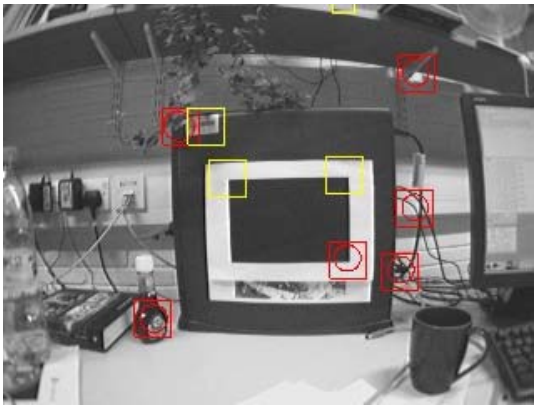


Graf 3. SLAM s sledilniki z velikostjo sledilnega okna 44x44 slikovnih elementov. Graf prikazuje hitrost delovanja sistema SLAM v odvisnosti od števila sledilnikov, ki jih uporabimo za določitev položaja značilnice v vsaki sliki zaporedja. V konfiguraciji s šestimi sledilniki je sistem odpovedal nekje v 700. sliki (rdeča pika). V ostalih primerih sistem deluje uspešno skozi celotno zaporedje 1000 slik.

Presenetljiva ugotovitev, ki jo lahko razberemo iz *Grafa 2* in *Grafa 3* je, da ob uporabi šestih sledilnikov sistem odpove prej, kot če uporabimo le tri sledilnike. Na tem mestu naj še pojasnimo, da gre pri določanju števila uporabljenih sledilnikov pravzaprav za nastavitev dveh parametrov. Prvi je število sledilnikov, ki naj bodo vidni v vsaki sliki zaporedja, drugi pa je zgoraj omenjeno število sledilnikov, ki jih v vsaki sliki dejansko uporabimo za samo sledenje. Prvi parameter je bil v naših testiranjih nastavljen tako, da število uporabljenih sledilnikov predstavlja sedemdeset odstotkov števila vedno vidnih sledilnikov. Tako v primeru s tremi uporabljenimi sledilniki, zahtevamo, da je v sliki vselej vidnih pet sledilnikov, v primeru s šestimi uporabljenimi sledilniki pa, da je vselej vidnih devet sledilnikov. Razlog za slabše delovanje v drugem primeru je verjetno v tem, da je v prvem delu posnetka vedno vidna inicializacijska tarča. Ko uporabljamo samo tri sledilnike, za lokalizacijo vselej uporabimo vsaj dve točki inicializacijske tarče, katerih položaj v prostoru pa je natanko znan (od petih sledilnikov so štirje na inicializacijski tarči, in če izmed teh petih izberemo tri, sta vsaj dva od njiju na inicializacijski tarči). Ko pa uporabljamo šest sledilnikov, se ravno zaradi izbiranja značilnic z največjo nedoločenostjo (*Razdelek 3.2.4*) poveča možnost napačnega sledenja (značilnice, ki niso na inicializacijski tarči, imajo seveda višjo nedoločenost in izbran je le en sledilnik, ki leži na inicializacijski tarči). Te napake sčasoma privedejo do nekonsistentnosti zemljevida in sistem odpove. *Slika 23* prikazuje vidne in izbrane sledilnike v eni izmed slik zaporedja ob uporabi treh sledilnikov, *Slika 24* pa prikazuje vidne in izbrane sledilnike ob uporabi šestih sledilnikov.



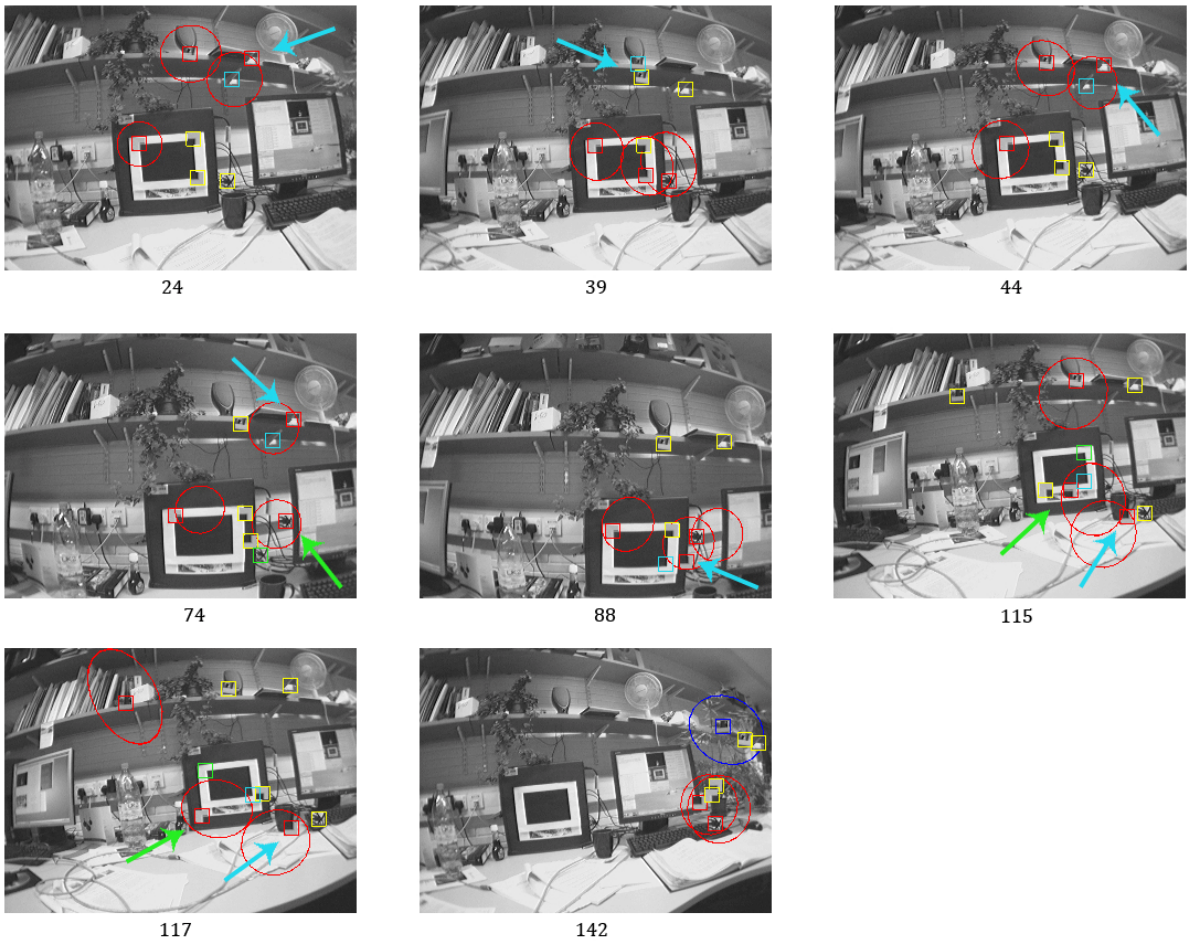
Slika 23. Z rdečo barvo so označeni sledilniki, ki smo jih izbrali v trenutni sliki. Z rumeno so označeni inicializirani sledilniki, ki v tej sliki niso bili izbrani. Od petih sledilnikov, ki so vidni v sliki, za posodobitev zemljevida in stanja robota uporabljamo tri. Od tega sta dva na inicializacijski tarči. To pomeni, da ima šestinšestdeset odstotkov (2/3) značilnic, ki jih v tej sliki uporabimo za določanje stanja robota in ostalih značilnic, nedoločenost enako nič.



Slika 24. Od devetih vidnih sledilnikov jih izberemo šest. Od tega je samo eden na inicializacijski tarči, kar pomeni, da sedaj značilnice z nedoločenostjo nič, predstavljajo le okrog sedemnajst (1/6) odstotkov značilnic, ki jih v tej sliki uporabimo za določanje stanja robota in ostalih značilnic.

Iz *Grafa 1* lahko razberemo, da zadnje tri konfiguracije z velikostmi okna 11x11 slikovnih elementov odpovejo nekje okrog sedemstote slike zaporedja. To sovpada s hitrejšimi premiki kamere v tistem delu videa, ki jim sledilnik s tako majhnim oknom ni sposoben uspešno slediti.

Očitno je, da velikost okna v večji meri vpliva na kvaliteto delovanja sistema, kot samo število uporabljenih sledilnikov. V primeru velikosti okna 44x44 slikovnih elementov so vse konfiguracije razen ene, uspešno delovale skozi celotno zaporedje slik. To pa pomeni, da je bolje imeti manj dobrih sledilnikov (sledilnikov z velikimi okni), kot pa veliko slabih. Največji problem namreč predstavljajo nepravilna ujemanja, ko sledilnik napačno določi položaj neke značilnice v sliki. Zaradi zmožnosti prilagajanja izgledu sledenega objekta, se nato sledilnik še posodobi nad tem območjem v sliki, kar ima za posledico, da počasi »oddrsi« od značilnice, ki naj bi ji sledil. To pa sproži nekakšno verižno reakcijo, saj sedaj postanejo tudi napovedi položajev vseh ostalih značilnic nekoliko popačene in enako se zgodi z ostalimi sledilniki. V končni fazi postane zemljevid prostora povsem nekonsistenten in sistem odpove. Primer odpovedi je nazorneje prikazan v *Sliki 25*.



Slika 25. Nekaj slik testnega zaporedja ob uporabi treh sledilnikov z velikostjo okna 11x11 slikovnih elementov. Pod sliko je označeno za katero sliko zaporedja gre. V 24. sliki lahko opazimo, da je sledilnik popolnoma napačno določil položaj označene značilnice (njen pravilni položaj je označen s kvadratkom v barvi puščice). Tako je v naslednjih slikah zaporedja prišlo do napačnih napovedi položajev nekaterih drugih značilnic (39. slika). Ker se je sledilnik ob napačno določenem položaju značilnice v 24. sliki zaporedja, nad tem položajem tudi posodobil in se s tem nekako »prilepil« na novo območje slike, je napačno odločitev o položaju te značilnice podal tudi v 44. in 74. sliki zaporedja. V 74. sliki zaporedja je tudi eden od drugih sledilnikov nepravilno določil položaj značilnice (zelena puščica), napovedi položajev ostalih značilnic (rumeni kvadratki) pa so postale že zelo napačne. Te napovedi so privedle do tega, da so skoraj vsi sledilniki iskali ujemanja v napačnih območjih slike (88, 115, 117), kjer so jih zaradi majhnosti sledilnega okna tudi našli in se nad najdenimi področji prilagodili. V končni fazi so postale same napovedi položajev značilnic tako napačne, da so predvidena mesta značilnic počasi zapustila vidno polje kamere in sistem je popolnoma odpovedal (142. slika).

Če primerjamo *Graf 1* in *Graf 2*, sprva izgleda, kot da pri uporabi treh oziroma šestih sledilnikov sistem deluje hitreje, ko uporabimo sledilnike z večjimi okni. Razlog je v tem, da moramo vsak na novo dodani sledilnik najprej inicializirati. To pa pomeni, da ga nekajkrat (v našem primeru štiridesetkrat) posodobimo nad območjem v sliki, kateremu naj bi sledil. Ob samem zagonu sistema je tako potrebno inicializirati štiri sledilnike, ki bodo sledili inicializacijski tarči. Pri uporabi okna velikosti 11x11 slikovnih elementov, ko sistem odpove že po nekaj slikah zaporedja, čas porabljen za inicializacijo sledilnikov predstavlja velik delež celotnega časa izvajanja, kar pomeni, da je število obdelanih slik v sekundi manjše, kot pri uporabi okna velikosti 22x22 slikovnih elementov, kjer sistem deluje dlje (inicializacijski čas sledilnikov se razdeli na daljše zaporedje slik).

6. Sklepne ugotovitve

Prvi pogoj, ki ga morajo izpolnjevati sistemi za uporabo v avtonomnih robotih, je delovanje v realnem času. Kot pa je očitno iz rezultatov testiranj predstavljenih v *Poglavju 5*, sistem MonoSLAM, razširjen z našim sledilnikom, deluje zelo počasi. Tri slike na sekundo, ki jih dosežemo pri konfiguraciji, ki omogoča zanesljivo delovanje, je občutno premalo, da bi ga bilo možno uporabiti v kakršnikoli realni aplikaciji, saj vemo, da je standardna hitrost zajema slike današnjih kamer 30 slik na sekundo. Čeprav Davison v svojem delu [14] navaja delovanje osnovnega sistema MonoSLAM s hitrostmi do 30 slik na sekundo, mi je ob poganjanju njegovega programa s podobnimi nastavitvami parametrov (število uporabljenih značilnic), kot pri uporabi z našim sledenjem, uspelo doseči le hitrosti do desetih slik na sekundo. V tem kontekstu pa postanejo naši rezultati nekoliko bolj primerljivi z rezultati, ki jih daje osnovni sistem MonoSLAM.

Čeprav sledenje s sprotnim izbiranjem značilnic, ki smo ga implementirali, samo po sebi deluje dobro, se je prav njegova zmožnost prilagajanja, ki se je sprva zdela velika prednost za uporabo v sistemu SLAM, v nekaterih primerih izkazala za veliko slabost. V samih preizkusih je večkrat prišlo do prilagoditve napačnemu delu slike, kar je privedlo do nekonsistentnosti v zemljevidu prostora in odpovedi sistema. Sistem MonoSLAM namreč deluje tako, da stalno izvaja nekakšno naravno selekcijo nad množico značilnic, ki so trenutno v zemljevidu. Značilnico, za katero ne najdemo ujemanja v določenem številu poizkusov, odstranimo iz zemljevida prostora in poiščemo drugo, ter jo dodamo v zemljevid. Tako po nekem času v zemljevidu ostanejo le značilnice, ki so do neke mere invariantne na spremembe kota s katerega jih opazujemo. Na ta način pa uporaba našega sledilnika z možnostjo prilagajanja izgledu sledenega objekta nekako izgubi svoj smisel. Njegova prednost tako postane njegova slabost.

Do zgornjih težav pride, ko sledilnik sicer uspešno določa položaj objekta skozi neko zaporedje slik, vendar v vsaki sliki zgreši le za nekaj slikovnih elementov. Tako se lahko zgodi, da se počasi oddalji od sledenega objekta, saj se znova in znova prilagaja na nekoliko napačna območja v sliki. Postavitev višjega praga na mero zaupanja, ki jo vrača sledilnik (na

podlagi te vrednosti sistem SLAM odloči ali gre za uspešno najdeno ujemanje ali ne), ne bi rešila problema, saj sledilnik v vsakem koraku postavimo v maksimum slike zaupanja, ki pa je še vedno v napačni točki.

Drug problem, ki povzroča preglavice pri uporabi našega sledilnika v sistemu SLAM, pa je sledeč. Ko se robot premika in se spreminja vidno polje, ki ga zajema kamera, okna nekaterih sledilnikov ostanejo zunaj slike. Po določenem času kamera zopet zajame del prostora, ki vsebuje prej »izgubljeno« okno sledilnika, vendar ga zajame pod nekim drugim kotom, kot ga je zapustila. Sledilnik se seveda ves čas, ko je bil zunaj vidnega polja kamere, ni mogel posodabljati in se tako prilagoditi spremenjenemu izgledu sledene točke v prostoru. Sledilnik bo tako najverjetneje poročal o neuspešnem iskanju ujemanja, ali kar je še slabše, našel bo ujemanje z napačnim delom slike, se nad tem področjem tudi posodobil, ter tako začel slediti napačni točki v prostoru.

Možna rešitev opisanega problema bi bila transformacija slike v skladu z razliko med kotom, pod katerim smo nazadnje opazovali sledeno območje in kotom, pod katerim ga opazujemo zdaj (ta podatek imamo, saj je ocena stanja robota, bistvena naloga sistema SLAM). Sledilnik bi na tako spremenjeni sliki našel ujemanje, nad dobljenim položajem pa bi ga nato ponovno inicializirali (to je nekajkrat pognali njegovo posodobitev nad tem delom slike).

Z opravljenimi preizkusi smo do neke mere tudi ovrgli tezo o superiornosti značilnic EOH, nad enostavnejšimi Haarovimi značilnicami. Ostaja sicer dejstvo, da so značilnice EOH sposobne povzeti kompleksnejše strukture slike, vendar se je izkazalo, da za naš problem niso primerne. Za samo sledenje nam zadostujejo Haarove značilnice, katerih velika prednost pa je hitra izračunljivost. Kot možnost izboljšave, bi na tem mestu podali idejo o uporabi še kakšnih drugih vrst značilnic v samem sledenju (npr. HOG [22], LBP [21]).

Pri sami integraciji sledilnika v sistem SLAM smo do neke mere ignorirali maksimalno vrednost slike zaupanja, ki jo vrne sledilnik ob določitvi novega položaja sledene značilnice. To vrednost smo uporabili le za sprejemanje odločitve o tem, ali je bilo sledenje v nekem koraku uspešno ali ne. Tu se morda odpira možnost za izboljšavo, saj maksimum slike zaupanja dobro ponazarja, kako prepričan je sledilnik v svojo odločitev. To vrednost bi lahko uporabili pri posodabljanju nedoločenosti položaja značilnice, ki ji sledimo. Ob visoki vrednosti zaupanja bi značilnici zmanjšali nedoločenost, ob nizki pa bi jo povečali (v smeri vzporedni z ravnino slike). Še vedno pa bi ohranili prag na vrednost zaupanja, ki ga moramo preseči, da določitev novega položaja sledene točke s strani sledilnika, sprejmemo kot uspešno.

Eden od problemov sistema SLAM z eno samo kamero, je določanje globine novo izbrane značilnice. Problem postane še toliko večji, kadar gre za zaporedja slik, ki predstavljajo predvsem gibanje naprej in nazaj, torej gibanje po sami globini slike. V tem primeru je določanje delca, ki predstavlja pravilno lego značilnice (*Razdelek 3.2.5*), še posebej težavno, saj se elipse znotraj katerih naj bi posamezni delci ležali povsem prekrivajo. Problem bi dokaj

enostavno rešili z uporabo stereoskopskega vida. Tako bi neko novo zaznano značilnico lahko takoj umestili v zemljevid prostora, saj bi njen 3D položaj lahko določili že iz ene same slike. Vendar pa bi s tem prekršili osnovno idejo, kateri je pri razvoju sistema SLAM sledil Davison, to je uporaba ene same kamere.

Za konkretne aplikacije sistema bi lahko uporabili tudi nek bolj specifičen model gibanja. Za osebni avtomobil na primer znamo napovedati, da kadar pelje naravnost, se giblje po premici, kadar zavija, se giblje po krožnici, nikoli pa se ne giblje vstran. Prav tako bi lahko pri sami lokalizaciji robota uporabili tako imenovane interne meritve (npr. merilec prevožene razdalje), katere bi lahko enostavno vključili v sistem MonoSLAM, saj le-ta v ta namen ponuja vmesnik v obliki razreda `Internal_Measurements`. S tem bi gotovo precej zmanjšali tako nedoločenost ocene položaja robota, kot tudi nedoločenosti vseh značilnic v zemljevidu.

Kljub temu, da je uporaba sledenja s sprotnim izbiranjem značilnic za sočasno lokalizacijo in kartografiranje v sam sistem vpeljala nekatere dodatne težave, smo uspeli dokazati, da je taka kombinacija možna in povsem izvedljiva. Kot smo omenili zgoraj, obstaja nemalo možnosti za nadaljnje raziskovanje. Uporaba sledenja sprotnim izbiranjem značilnic v sistemu SLAM je v času pisanja te naloge še vedno nekaj novega in ta koncept še ni bil preizkušen. Morda pa bo prav to delo navdihnilo in spodbudilo še koga, da bo nadalje raziskal možnosti uporabe sledenja s sprotnim izbiranjem značilnic v sistemih SLAM.

7. Priloge

A. Glave glavnih razredov, ki implementirajo sledenje s sprotnim izbiranjem značilnic

Razred `OnlineAdaboostTracker`

```
#ifndef ONLINEADABOOSTTRACKER_H_
#define ONLINEADABOOSTTRACKER_H_
#include "Rectangle.h"
#include "StrongClassifier.h"
#include "FeaturePool.h"
#include "EOHIntegralImages.h"
#include <VW/image.h>
#include <VW/Improc/integralimage.h>
#include <VNL/matrixfixed.h>
#include <VNL/vectorfixed.h>

class StrongClassifier;
class FeaturePool;

class OnlineAdaboostTracker
{
  protected:
    int _example_patch_height;
    int _example_patch_width;
    StrongClassifier *_strong_classifier;
    FeaturePool *_feature_pool;
    bool _feature_pool_is_internal;
    Rectangle _previous_rectangle;
    VW::IntegralImage<unsigned int> *_integral_image;
    bool _integral_image_is_internal;
    EOHIntegralImages *_EOH_integral_images;
    bool _EOH_integral_images_are_internal;
    bool _using_EOH_features;
```

```
public:
  /* Ustvari nov sledilnik in zgradi interno množico značilnic.
   */
  OnlineAdaboostTracker(
            int example_patch_height,
            int example_patch_width,
            int num_selectors,
            int num_weak_classifiers);

  /* Ustvari nov sledilnik. Uporabi zunanjo množico značilnic.
   */
  OnlineAdaboostTracker(
            int example_patch_height,
            int example_patch_width,
            int num_selectors,
            int num_weak_classifiers,
            FeaturePool *feature_pool,
            bool using_EOH_features);

  virtual ~OnlineAdaboostTracker();

  /* Inicializira sledilnik, tako da n-krat požene posodobitev
   *  sledilnika v začetni (znani) poziciji. Integralne slike se hranijo
   *  interno.
   */

  void InitializeOnFirstImage(
            const VW::ImageMono<unsigned char>& first_image,
            Rectangle first_rect, int n);

  /* Inicializira sledilnik, tako da n-krat požene njegovo posodobitev
   * v začetni (znani) poziciji. Integralne slike so podane od zunaj.
   */
  void InitializeOnFirstImage(
            VW::IntegralImage<unsigned int> *first_image,
            EOHIntegralImages *first_EOH_integral_images,
            Rectangle first_rect, int n);

  /* Določi pozicijo sledenega objekta na sliki next_image, tako da
   * preišče območje dvojne velikosti sledilnega okna v okolici zadnje
   * znane pozicije objekta.
   */
  Rectangle MoveTrackingWindow(
            const VW::ImageMono<unsigned char>& next_image);

  /* Določi pozicijo sledenega objekta na sliki next_image, tako da
   * preišče območje določeno s search_rectangle.
   */
  Rectangle MoveTrackingWindow(
            const VW::ImageMono<unsigned char>& next_image,
            Rectangle search_rectangle);
```

```
/* Določi pozicijo sledenega objekta na sliki next_image, tako da
 * preišče eliptično območje v okolici točke centre. Integralne slike
 * se hranijo interno.
 */

bool MoveTrackingWindowEllipticalSearch(
         const VW::ImageMono<unsigned char>& next_image,
         const VNL::VectorFixed<2, double> &centre,
         const VNL::MatrixFixed<2,2,double> &PuInv,
         unsigned int &row_found, unsigned int &col_found);

/* Določi pozicijo sledenega objekta na sliki next_image, tako da
 * preišče eliptično območje v okolici točke centre. Integralne
 * slike dobi od zunaj.
 */
bool MoveTrackingWindowEllipticalSearch(
         VW::IntegralImage<unsigned int> *next_integral_image,
         EOHIntegralImages *EOH_integral_images,
         const VNL::VectorFixed<2, double> &centre,
         const VNL::MatrixFixed<2,2,double> &PuInv,
         unsigned int &row_found, unsigned int &col_found);

Rectangle GetLastRectangle() {return _previous_rectangle;}
int GetExamplePatchHeight() {return _example_patch_height;}
int GetExamplePatchWidth() {return _example_patch_width;}

/* Izračuna integralno sliko slike image.
 */
void SetIntegralImage(const VW::ImageMono<unsigned char>& image);

/* Iz slike image izračuna integralne slike moči robov, za vse
 * orientacije robov.
 */
void SetEOHIntegralImages(const VW::ImageMono<unsigned char>& image);

/* Vrne vsoto slikovnih elementov pravokotnega področja v sliki.
 */
unsigned int GetIntegralImageRegionSum(
         int top, int left,int bottom,int right);

/* Vrne vsoto moči robov k-te orientacije v pravokotnem področju
 * slike.
 */
unsigned int GetEOHIntegralImageRegionSum(
         int k, int top, int left,
         int bottom, int right);

/* Vrne najboljšo novo pozicijo sledilnega okna glede na vrednosti, ki
 * jih vrača klasifikator. Išče po območju določenim s
 * search_rectangle.
 */
Rectangle GetBestNewRectanglePosition(Rectangle search_rectangle);
```

```
    /* Izbere 5 primerov (pozitivnega na delu slike, kjer se nahaja
     * sledeni objekt in 4 negativne v njegovi okolici) in jih uporabi za
     * posodobitev klasifikatorja.
     */
    void UpdateTracker(Rectangle object_position);
};
#endif /*ONLINEADABOOSTTRACKER_H_*/
```

Razred StrongClassifier

```
#ifndef STRONGCLASSIFIER_H_
#define STRONGCLASSIFIER_H_

#include "Example.h"
#include "WeakClassifier.h"
#include "FeaturePool.h"
class FeaturePool;
class WeakClassifier;

class StrongClassifier
{
  private:
    int _num_of_selectors;
    int _num_of_weak_classifiers;
    WeakClassifier **_weak_classifiers;
    WeakClassifier **_selectors;
    double *_alpha;

  public:
    StrongClassifier();

    /* Ustvari nov močen klasifikator, s podanim številom slektorjev, ki
     * bodo izbirali med podanim številom šibkih klasifikatorjev.
     */
    StrongClassifier(
              int num_of_selectors,
              int num_of_weak_classifiers,
              FeaturePool *feature_pool,
              OnlineAdaboostTracker *tracker);

    virtual ~StrongClassifier();

    /* Posodobi celotno množico šibkih klasifikatorjev s primerom e.
     */
    void OnLineAdaboostUpdate(Example &e);

    /* Klasificira primer e, z uporabo uteženega glasovanja šibkih
     * klasifikatorjev, ki so trenutno izbrani (s strani selektorjev).
     */
    double Classify(Example &example);

  private:
    /* Vrne najboljši šibek klasifikator (glede na njegovo ocenjeno
     * napako), ki še ni izbran v nobenem selektorju.
     */
    double GetBestNotSelectedWeakClassifier(
              WeakClassifier*& best_weak_classifier);
```

```
    /* V množici šibkih klasifikatorjev zamenja šibek klasifikator z
     * najvišje ocenjeno napako z naključno izbranim šibkim
     * klasifikatorjem iz globalne množice šibkih klasifikatorjev
     * (značilnic).
     */
    void ReplaceWorstWeakClassifier();
};
#endif /*STRONGCLASSIFIER_H_*/
```

Razred WeakClassifier

```
#ifndef WEAKCLASSIFIER_H_
#define WEAKCLASSIFIER_H_

#include "Example.h"
#include "FeatureBase.h"
#include "FeaturePool.h"

class FeaturePool;

class WeakClassifier
{
  private:
    static const float R = 0.01;
    double _lambda_corr;
    double _lambda_wrong;
    double _error;
    double _P_plus;
    double _P_minus;
    double _ni_plus;
    double _ni_minus;
    int _p;
    int _threshold;
    bool _selected;
    FeatureBase *_feature;
    FeaturePool *_feature_pool;
    OnlineAdaboostTracker *_tracker; //Referenca, ki jo šibek klasifikator
                                     //potrebuje za dostop do struktur, iz
                                     //katerih izračuna vrednost
                                     //značilnice.
  public:
    /* Ustvari nov šibek klasifikator, in mu pridruži naključno izbrano
     * značilnico iz množice značilnic feature_pool.
     */
    WeakClassifier(
              FeaturePool *feature_pool,
              OnlineAdaboostTracker *tracker);

    virtual ~WeakClassifier();

    /* Posodobi šibek klasifikator z uporabo Kalmanovega filtra.
    */
    void Update(Example &example);

    /* Določi razred primeru example.
     */
    int Classify(Example &example);
    void IncreaseLambdaCorr(double lambda);
    void IncreaseLambdaWrong(double lambda);
    void CalculateError();
```

```
    double GetError();
    bool Selected();
    void SetSelected(bool selected);
    void ResetValues();

    /* Izbere naključno značilnico iz množice značilnic feature_pool.
     */
    void SelectNewFeature();
};
#endif /*WEAKCLASSIFIER_H_*/
```

Razredi značilnic

Razred FeatureBase

```
/* Abstraktni nadrazred vseh razredov značilnic.
 * Vsak izpeljani razred značilnice naj bi imel dodatne parametre in svoj
 * način izračuna vrednosti značilnice.
 */
#ifndef FEATUREBASE_H_
#define FEATUREBASE_H_

#include "Example.h"

class OnlineAdaboostTracker;

class FeatureBase
{
  protected:
    int _top_left_row;
    int _top_left_col;
    int _height;
    int _width;

  public:
    FeatureBase(
              int height, int width, int top_left_row, int top_left_col);

    virtual ~FeatureBase();

    /* Abstraktna metoda za izračun vrednosti značilnice, ki jo mora
     * implementirati vsak od potomcev tega razreda.
     */
    virtual double CalculateFeatureValue(
              Example &example, OnlineAdaboostTracker *tracker) = 0;
};
#endif /*FEATUREBASE_H_*/
```

Razred EOHFeature

```
/* Nadrazred vseh razredov značilnic EOH (še vedno abstrakten), izpeljan iz
 * razreda FeatureBase.
 */
#ifndef EOHFEATURE_H_
#define EOHFEATURE_H_

#include "FeatureBase.h"

class EOHFeature : public FeatureBase
{
  public:
    virtual ~EOHFeature();
    EOHFeature(int height, int width, int top_left_row, int top_left_col);

    static const int NUM_OF_BINS = 4;
    static const double EOH_EPSILON = 0.000000000000001;
};
#endif /*EOHFEATURE_H_*/
```

Razred EOHFeatureTypeA

```
#ifndef EOHFEATURETYPEA_H_
#define EOHFEATURETYPEA_H_

#include "EOHFeature.h"
#include "OnlineAdaboostTracker.h"

class EOHFeatureTypeA : public EOHFeature
{
  protected:
    int _k1;
    int _k2;

  public:
    EOHFeatureTypeA(
              int height, int width, int top_left_row, int top_left_col,
              int k1, int k2);
    virtual ~EOHFeatureTypeA();
    double CalculateFeatureValue(
              Example &example, OnlineAdaboostTracker *tracker);
};
#endif /*EOHFEATURETYPEA_H_*/
```

Razred EOHFeatureTypeB

```
#ifndef EOHFEATURETYPEB_H_
#define EOHFEATURETYPEB_H_

#include "EOHFeature.h"
#include "OnlineAdaboostTracker.h"

class EOHFeatureTypeB : public EOHFeature
{
  protected:
    int _k;

  public:
    EOHFeatureTypeB(
              int height, int width, int top_left_row, int top_left_col,
              int k);
    virtual ~EOHFeatureTypeB();
    double CalculateFeatureValue(
              Example &example, OnlineAdaboostTracker *tracker);
};
#endif /*EOHFEATURETYPEB_H_*/
```

Razred EOHFeatureTypeC

```
#ifndef EOHFEATURETYPEC_H_
#define EOHFEATURETYPEC_H_

#include "EOHFeature.h"
#include "OnlineAdaboostTracker.h"

class EOHFeatureTypeC : public EOHFeature
{
  protected:
    /* _top_left_row in _top_left_col iz nadrazreda se uporabita za
     * pozicijo prvega pravokotnika, _top_left_row2 in _top_left_col2 pa
     * se uporabita za pozicijo drugega pravokotnika, ki določa
     * značilnico. _height in _width določata višino in širino obeh
     * pravokotnikov (obe višini sta enaki in obe širini sta enaki).
     */
    int _top_left_row2;
    int _top_left_col2;

public:
    EOHFeatureTypeC(
              int height, int width, int top_left_row, int top_left_col,
              int top_left_row2, int top_left_col2);
    virtual ~EOHFeatureTypeC();
    double CalculateFeatureValue(
```

```
                Example &example, OnlineAdaboostTracker *tracker);
};
#endif /*EOHFEATURETYPEC_H_*/
```

Razred HaarFeature

```
#ifndef HAARFEATURE_H_
#define HAARFEATURE_H_

#include "FeatureBase.h"
#include "OnlineAdaboostTracker.h"

class HaarFeature : public FeatureBase
{
  protected:
    char _feature_type;   //Pove za katero vrsto Haarove značilnice gre
                          //(kateri prototip določa to značilnico).
  public:
    HaarFeature(
              char feature_type, int height, int width, int top_left_row,
              int top_left_col);
    virtual ~HaarFeature();
    double CalculateFeatureValue(
              Example &example, OnlineAdaboostTracker *tracker);
};
#endif /*HAARFEATURE_H_*/
```

8. Seznam uporabljenih virov

[1] R. C. Smith, M. Self, P. Cheesman, "Estimating uncertain spatial relationships in robotics," *Proceedings of the Second Annual Conference on Uncertainty in Artificial ntelligence*, str. 435-462, Filadelfija, ZDA, 1986.

[2] J. J. Leonard, H. F. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," *Proc. IEEE Int. Workshop on Intelligent Robots and Systems*, str. 1442-1447, Osaka, Japonska, 1991.

[3] A. J. Davison, D. W. Murray, "Mobile robot localisation using active vision," *Proceedings of Fifth European Conference on Computer Vision*, str. 809-825, Freiburg, Nemčija, 1998.

[4] I. Jung, S. Lacroix, "High resolution terrain mapping using low altitude aerial imagery," *Proceedings of the NinthIEEE International Conference on Computer Vision, 2003.*

[5] R. M. Eustice, H. Singh, J. J. Leonard, M. Walter, R. Ballard, "Visually navigating the RMS Titanic with SLAM information filters," *Proceedings of Robotics: Science and Systems*, 2005.

[6] R. Sim, P. Elinas, M. Griffin, J.J. Little, "Vision-based SLAM using the Rao-Blackwellised particle filter," *IJCAI Workshop Reasoning with Uncertainty in Robotics*, Edinburg, Škotska, 2005.

[7] M. Montemerlo, S.Thrun, D. Koller, B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," *Eighteenth national conference on Artificial intelligence*, str. 593-598, Edmonton, Alberta, Kanada, 2002.

[8] D. G. Lowe, "Object recognition from local scale invariant features," *International Conference on Computer Vision*, str. 1150-1157, Krf, Grčija, 1999.

[9] E. M. Foxlin, "Generalized architecture for simultaneous localization, auto-calibration, and map-building," v zborniku *IEEE/RSJ International Conference on Intelligent Robots ans System*, str. 527-533, Lozana, Švica, 2002.

[10] A. Davison, N. Kita, "Sequential localisation and map-building for real-time computer vision and robotics," *Robotics and Autonomous Systems*, št. 36, str. 171-183, 2001.

[11] D. Nister, "An efficient solution to the five-point relative pose problem," *IEEE Conference on Computer Vision and Pattern Recognition*, Madison, ZDA, 2003.

[12] A. Davison, "Real-time simultaneous localisation and mapping with a single camera," *IEEE International Conference on Computer Vision*, str. 1403-1410, Nica, Francija, 2003.

[13] T. Lemaire, S. Lacroix, J. Sola, "A practical 3D bearing-only SLAM algorithm," *IEEE International Conference on Intelligent Robots and Systems*, Kanada, 2005.

[14] A. J. Davison, I. D. Reid, N. D. Molton, O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, št. 29, zv. 6, str 1052-1067, 2007.

[15] A. Yilmaz, O. Javed, M. Shah, "Object tracking: A survey," *ACM Computing Surveys (CSUR)*, št. 38, zv. 4, str. 1-45, 2006.

[16] Y. Freund, R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, št. 55, str. 119-139, 1997.

[17] H. Grabner, M. Grabner, H. Bischof, "Real-time tracking via on-line boosting," *Proceedings of the British Machine Vision Conference*, št. 1, str. 47-56, 2006.

[18] H. Grabner, H. Bischof, "On-line boosting and vision," *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, št. 1, str. 260-267, 2006.

[19] S. Avidan, "Ensemble tracking," *IEEE Transactions on Pattern Analysis andMachine Intelligence*, št. 29, zv. 2, str. 261-271, 2007.

[20] D. Comaniciu, P. Meer, "Mean shift analysis and applications," *Seventh IEEE International Conference on Computer Vision*, str. 1197-1203, Kerkyra, Grčija, 1999.

[21] T. Ojala, M. Pietikäinen, T. Mäenpää, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, št. 24, zv. 7, str. 971-987, 2002.

[22] N. Dalai, B. Triggs, "Histograms of oriented gradients for human detection," *IEEE Computer Society Conference on Computer Vision and Patterm Recognition*, št. 1, str. 886-893, 2005.

[23] C. P. Papageorgiou, M. Oren, T. Poggio, "A general framework for object detection," *Sixth International Conference on Computer Vision*, str. 555-562, Bombaj, Indija, 1998.

[24] P. Viola, M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, št. 1, str. 511-518, 2001.

[25] B. Rasolzadeh, "Lab manual – 2D1427 Image based recognition and classification," Dostopno na: http://www.csc.kth.se/utbildning/kth/kurser/DD2427/bik08/Lab/Manual.pdf (13.10.2008)

[26] K. Levi, Y. Weiss, "Learning object detection from a small number of examples: the importance of good features," *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, št. 2, str. 53-60, 2004

[27] Y . Freund, R. E. Schapire, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence*, št. 14, str. 771-780, 1999.

[28] I. Kononenko, *Strojno učenje*, Ljubljana: Fakulteta za računalništvo in informatiko, 2005, pogl. 3.

[29] K. Tieu, P. Viola, "Boosting image retrieval," *Proceedin*gs *of IEEE Conference on Computer Vision and Pattern Recognition*, št. 1, str. 228-235, 2000.

[30] N. Oza, S. Russel, "Online bagging and boosting," *Proceedings of Artificial Intelligence and Statistics*, str. 105-112, 2001.

[31] H. Durrant-Whyte, T. Bailey, "Simultaneous localisation and mapping (SLAM): Part I The essential algorithms," *Robotics and Automation Magazine*, št. 13, str. 99–110, 2006.

[32] S. Lacroix, T. Lemaire, C. Berger, "More vision for SLAM," *2007 IEEE International Conference on Robotics and Automation*, Rim, Italija, 2007.

[33] G. Welch, G. Bishop, "An introduction to the Kalman filter," Tehnično poročilo TR 95-041, Univerza Severne Karoline, Oddelek za računalništvo, 1995

[34] J. Shi, C. Tomasi, "Good features to track," *IEEE Conference on Computer Vision and Pattern Recognition*, str. 593-600, Seattle, ZDA, 1994.

[35] VW34 - Knjižnica za računalniški vid iz laboratorija za akvtivni vid, Oxford, Anglija. Dostopno na: http://www.doc.ic.ac.uk/~ajd/Scene/Release/vw34.tar.gz (13.10.2008)

[36] SceneLib and MonoSLAMGlow Requirements, Download and Installation. Dostopno na: http://www.doc.ic.ac.uk/~ajd/Scene/download.html (13.10.2008)

[37] Camera Calibration Toolbox for Matlab. Dostopno na: http://www.vision.caltech.edu/bouguetj/calib_doc (13.10.2008)

9. Izjava o avtorstvu

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom mentorja prof. dr. Aleša Leonardisa. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Tomaž Bevk