SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3891

NEKONVENCIONALNI NAČINI UPRAVLJANJA VOZILIMA

Oleg Jakovljev

Zagreb, Lipanj 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 6. ožujka 2015.

ZAVRŠNI ZADATAK br. 3891

Pristupnik: **Oleg Jakovljev (0036475226)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Upravljanja modelom vozila pomoću nekonvencionalnog sučelja**

Opis zadatka:

Proučiti način izrade virtualnog modela vozila te ostvariti upravljane vozilom pomoću konvencionalnog načina, odnosno korištenjem tipkovnice i miša. Proučiti i načiniti pregled različitih tehnologija izrade nekonvencionalnih sučelja prema korisniku a posebice prirodna korisnička sučelja. Razmotriti mogućnosti upravljanja vozilom temeljem postavljenih znakova u virtualnom prostoru. Razraditi primjere koji će demonstrirati upravljanje modelom automobila primjenom proučenih tehnologija nekonvencionalnih sučelja. Načiniti ocjenu i usporedbu ostvarenih rezultata.
Izraditi odgovarajući programski proizvod. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 13. ožujka 2015.
Rok za predaju rada: 12. lipnja 2015.

Mentor:

Prof. dr. sc. Željka Mihajlović

Predsjednik odbora za završni rad modula:

Prof. dr. sc. Siniša Srbljić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

SADRŽAJ

1. Uvod

Konvencionalni načini upravljanja vozilima uključuju, u slučaju motornih vozila, upravljački volan i kontrole gasa te kočnice kao osnovni način promjene brzine i smjera kretanja vozila. U slučaju neizravnog sučelja vozila preko računala, ponekad je prikladniji način prilagođen računalima, koristeći tipkovnicu, miš, igraće palice te ostale ulazne periferne jedinice računala. Nekonvencionalni načini upravljanja vozilima odstupaju od tih normi, te ovisno o namjeni, uključuju kontroliranje vozila putem sučelja mozak-računalo, kamera, inercijskih senzora, glasovnih naredbi itd. Nekonvencionalni načini upravljanja vozilima imaju prednosti nad konvencionalnim upravljačkim sustavima u nekim primjenama i specifičnim situacijama ili kada je korisnik onemogućen koristiti standardan načinu unosa naredbi.

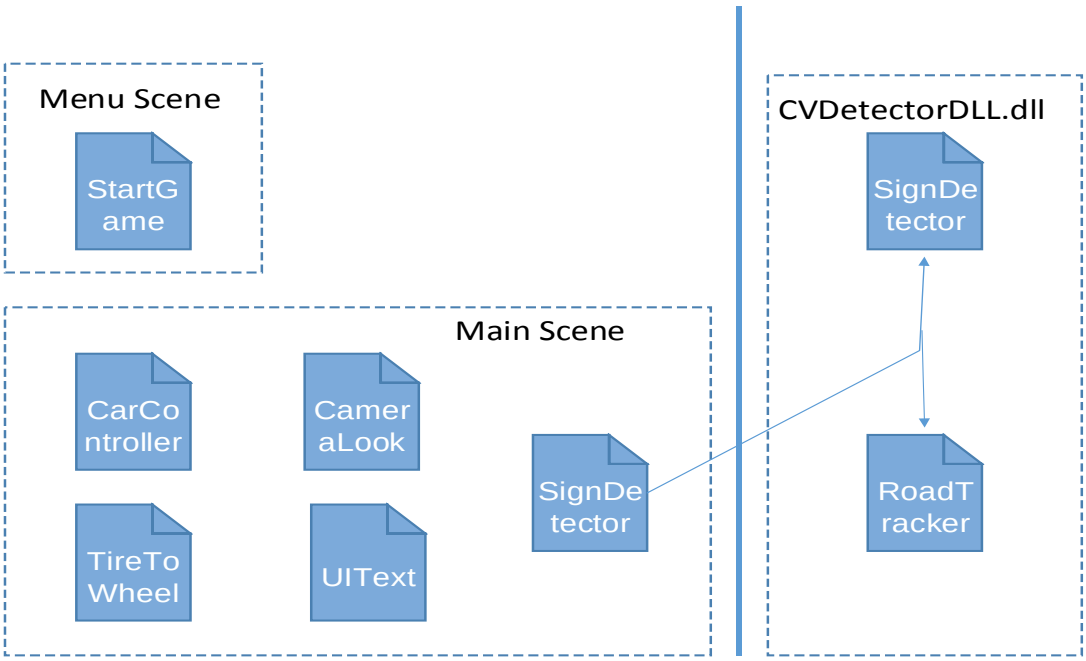
Ovaj rad obrađuje upravljanje virtualnim vozilom u simuliranom okolišu pomoću računalnog vida, prepoznavanja elemenata simuliranog svijeta i demonstrira autonomno kretanje vozila usmjereno ulazom iz virtualne kamere. Simulirano vozilo je na mapi kontrolirano od strane korisnika do trenutka kad je vidljiv znak za uključivanje autonomnog načina kada kontrolu preuzima računalo. Algoritam za praćenje obojenih oznaka na cesti se tada uključuje a računalo pazi da je vozilo usmjereno pravilno te da je njegova brzina unutar zadanog intervala.

2. Virtualna simulacija vozila

2.1 Modeliranje mape i vozila

2.1.1 Dizajn sustava

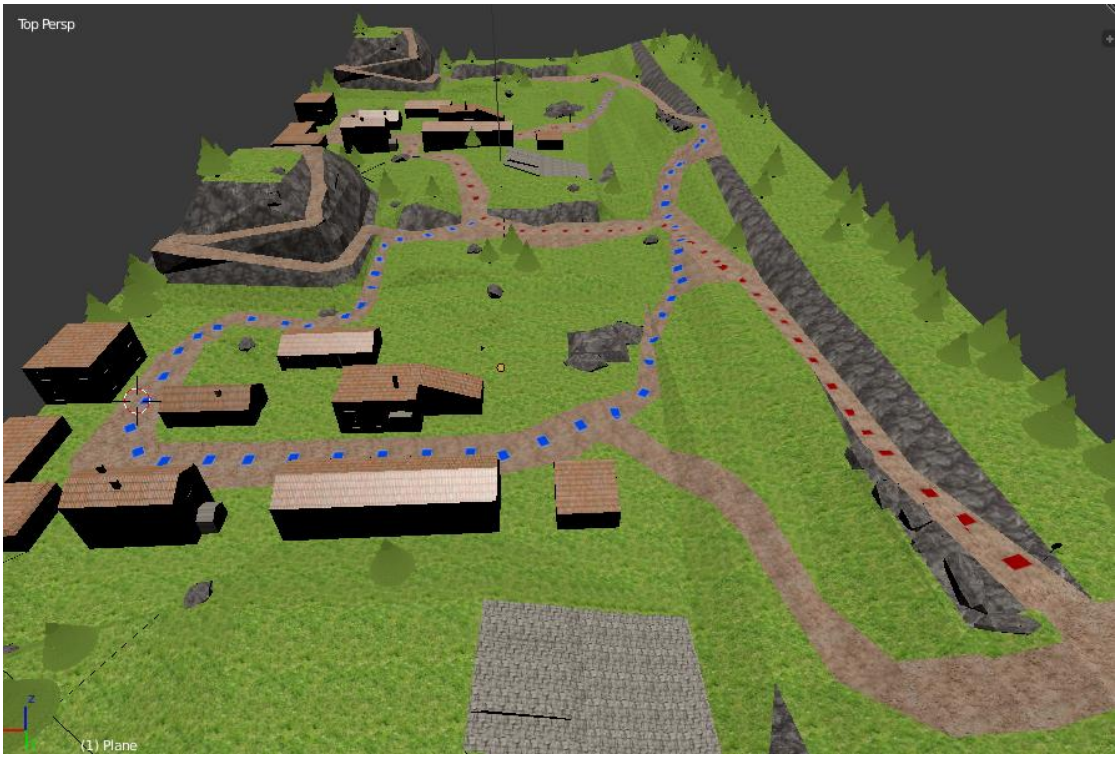
Simulacija je izvedena pomoću grafičkog pogona Unity3D a algoritmi za računalni vid pomoću kojih se izvodi pokretanje vozila su implementirani neovisno, u vanjskoj biblioteci. Unity3D simulacija se sastoji od dvije scene: glavnog izbornika i scene simulacije. Izbornik ima jednu skriptu koja izvršava funkcionalnosti vezane uz pritisak na tipke za započinjanje simulacije ili izlazak iz programa. Scena simulacije objedinjuje logiku potrebnu za rad s vozilom i simulaciju njegovog ponašanja te skripte za komunikaciju s vanjskom bibliotekom i prikaz podataka na korisničko sučelje. Vanjska je biblioteka napisana u programskom jeziku C++ i koristi elemente biblioteke otvorenog koda za računalni vid OpenCV. Ona se sastoji od dva dijela: jedan za detektiranje znakova za započinjanje samostalnog pokretanja vozila, a drugi za praćenje oznaka na cesti i usmjeravanje vozila. Dva dijela sustava povezuje skripta *SignDetector.cs* koja uvozi funkcije iz biblioteke i poziva ih po potrebi, ovisno o tijeku simulacije. Kako bi se poboljšale performanse simulacije, koristi se višedretvenost te se sama simulacija pokreće u glavnoj dretvi a svaki dio vanjske biblioteke u posebnoj dretvi. Slika 1 prikazuje dijagram sustava.



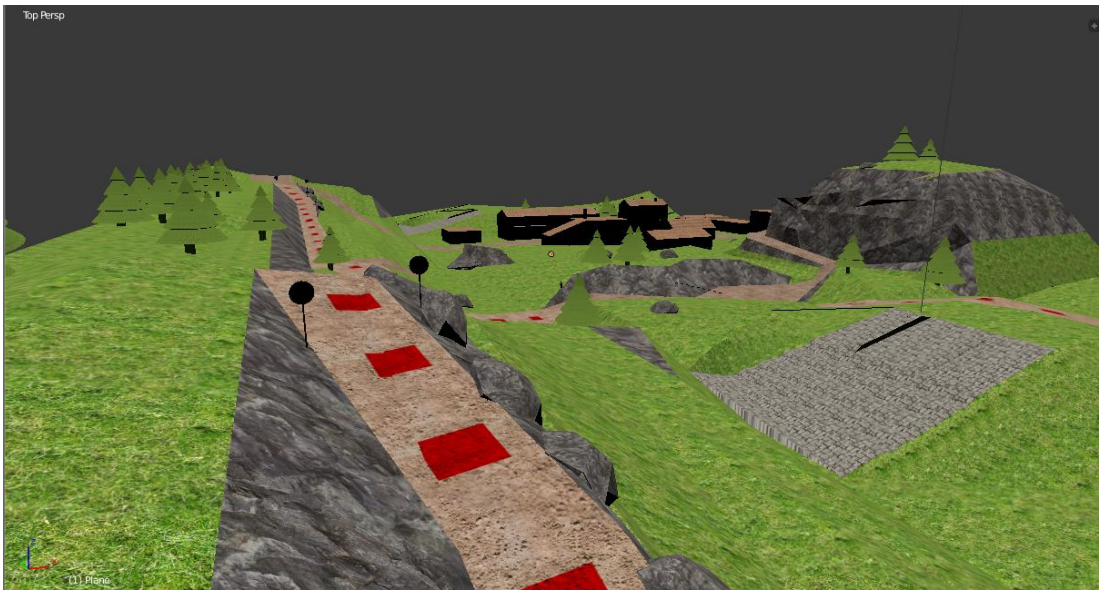
Slika 1. *Dijagram arhitekture sustava*

2.1.2 Modeliranje mape

Model mape izrađen je u alatu Blender 2.72. Model se sastoji od 20 795 vrhova i 40 996 poligona te je u Unity3D razvojnu okolinu unesen u nekoliko dijelova, zbog internih ograničenja izvedbe programa. Model sadrži neravan teren, dvije skupine stambenih objekata, dva brežuljka te cestu koja se prostire širom mape. Na nekim su dijelovima ceste crvene ili plave oznake koje služe kako bi algoritam za praćenje mogao usmjeriti vozilo dok je ono u stanju autonomnog kretanja. Na mapi također postoje i prometni znakovi koje vozilo prepoznaje te na temelju njih prelazi u stanje autonomnog kretanja. Slike 1. i 2. prikazuju izgled mape.



Slika 2. *Izgled mape, visoki pogled*



Slika 3. *Izgled mape, niski pogled*

2.1.3 Modeliranje vozila

Model vozila također je izrađen u alatu Blender 2.72. Model vozila ima 6 772 vrhova i 9 853 poligona a izrađen je prema nacrtima stvarnog automobila. Model se sastoji od nekoliko pod-objekata, no posebice je bitno istaknuti kotače jer su oni korišteni animaciji simulacije kao posebni objekti kako bi omogućili realistično ponašanje opruga i ovjesa vozila.Vozilo također ima osnovno modeliran interijer, sjedala te volan no kamera je u simulaciji postavljena ispred vjetrobrana vozila te malo nagnuta prema dolje kako bi se bolje snimila cesta ispred (pri praćenju obojenih oznaka na cesti). Zbog pojednostavljenja, pojedine su teksture interijera zamijenjene jednostavnim bojama. Slike 3. i 4. prikazuju model vozila s postavljenim osnovnim teksturama i materijalima.

Slika 4. *Prednji perspektivni pogled - vozilo*

Slika 5. *Stražnji perspektivni pogled - vozilo*

2.2 Simulacija okoliša u Unity3D

Za implementaciju funkcionalnosti korišten je Unity3D Engine jer pruža jednostavno i fleksibilno sučelje te omogućava implementaciju traženih svojstava u manje vremena zahvaljujući postojećem grafičkom pogonu i sustavu skripti koje dopuštaju korisniku laku promjenu i dodavanje funkcionalnosti.

Prvi korak je postavljanje vozila i pripremanje modela za implementaciju kontrolera koji na korisnički unos putem tipkovnice pokreće vozilo u traženom smjeru. Svaki kotač je poseban pod-objekt modela automobila te mu se dodaju *Wheel Collider* svojstva koja osiguravaju interakciju s modelom mape koja ima dodan *Mesh Collider*. Kao pod-objekt automobila dodana je i glavna kamera scene: time se osigurava da kamera bude "fiksirana" na vozilo. Naposljetku, na prednji i stražnji kraj vozila dodana su po dva svjetla različitih intenziteta koja osiguravaju osvjetljavanje ceste ispred i dodaju vizualnoj autentičnosti simulacije.

Kontroler ponašanja vozila opisuju 3 skripte. Prva je *TireToWheel* koja transformira poziciju i rotaciju kotača s obzirom na roditeljski objekt (automobil) i na unos korisnika. Druga je skripta *CarController:* ona se brine o fizički realnoj simulaciji kretanja i ponašanja vozila, prati naginjanje (*roll*) vozila, poziciju i kretanje s obzirom na djelovanje opruga i kotača. Također se brine o okretanju kotača s obzirom na šasiju vozila i izračun brzine vozila. Treća je skripta *SignDetector* koja komunicira s vanjskom dinamičkom bibliotekom napisanom u C++u koja sadrži OpenCV funkcije za prepoznavanje znakova i praćenje oznaka. O ovoj će skripti više govora biti u dijelu 3.

Još valja spomenuti skriptu *CameraLook* koja je poprilično jednostavna i brine se o rotaciji kamere oko Z osi. Rotacija je omogućena uvijek no tijekom stanja autonomnog kretanja potrebno je kameru usmjeriti točno ispred inače su moguće neželjene posljedice rada algoritma.

Posljednja skripta koja radi kao dio simulacije svijeta u Unity3D *Engine*-u je skripta *UIText* koja služi za manipulaciju prikaza teksta na grafičkom sučelju. O skriptama vezanim uz simulaciju okoliša biti će više riječi kasnije, u poglavlju 4.

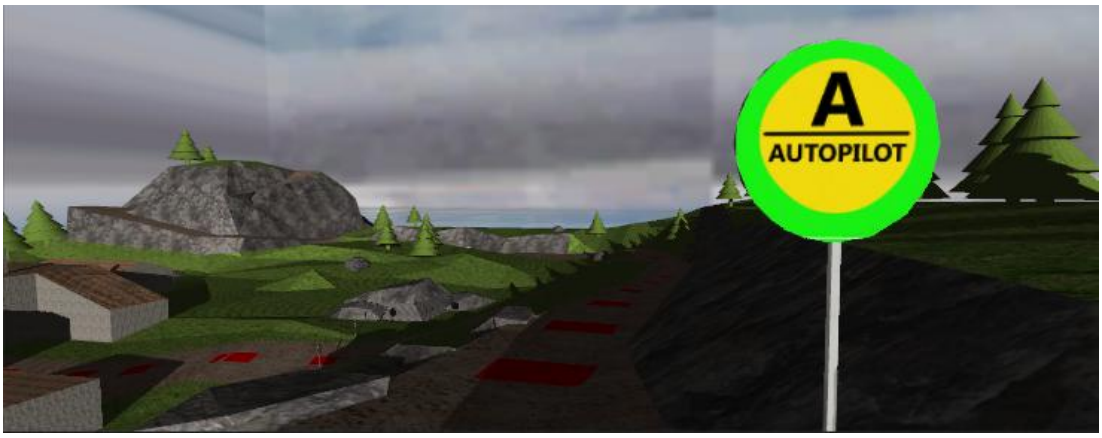
3. Funkcionalnost vanjske biblioteke CVDetectorDLL

Vanjska biblioteka je napisana u C++u a njene se funkcije pozivaju iz skripte *SignDetector* koja predstavlja sučelje između funkcionalnosti simuliranog okoliša u Unity3D i funkcionalnosti računalnog vida ostvarenih pomoću biblioteke OpenCV. Simulacija je izvedena kao stroj stanja gdje vozilo poprima stanje upravljanja korisnika i traženja znaka ili stanje autonomnog upravljanja i traženja znaka za isključivanje. Oba dijela biblioteke se oslanjaju na funkciju *hwnd2mat* koja dohvaća video prikaz glavnog ekrana računala u stvarnom vremenu i pretvara ga u oblik matrice s kojom OpenCV radi. Tako se simulira virtualna kamera koja snima ono što korisnik vidi u simulaciji. Zbog načina na koji je algoritam praćenja ceste implementiran, nužno je u načinu autonomnog pokretanja centrirati kameru tako da gleda ravno ispred vozila.

3.1 Detektor znakova

Detektor znakova je implementiran pomoću biblioteke otvorenog koda OpenCV (Open Computer Vision) 2.4.10 u programskom jeziku C++. Detektor znakova koristi metodu kaskadnih klasifikatora s Haar-svojstvima. Klasifikator (algoritam za prepoznavanje) je potrebno ‘trenirati’ pomoću slika negativa (koje *ne* sadržavaju objekt koji se prepoznaje) i slika pozitiva (koje sadržavaju objekt koji se prepoznaje). Rezultat postupka poznatog kao treniranje klasifikatora je vektorska datoteka koja omogućava traženje uzoraka tj. objekata na slici u stvarnom vremenu. Prvi dio biblioteke zadužen je za prepoznavanje znakova na video nizu čiji je izvor sam prikaz simulacije

na zaslonu. Slika 5. prikazuje znak u simuliranom okolišu koji algoritam treba prepoznati.



Slika 6. *Znak za uključivanje autonomnog pokretanja vozila*

Program detektora ima dva osnovna dijela koji se pozivaju iz koda u jeziku C# pripadnih skripti Unity3D pokretačkog okoliša. Prvi dio služi za inicijalizaciju prikaznih matrica koje čuvaju podatke o ekranu i varijabli koje se koriste u detekciji znakova. Dio za inicijalizaciju se zove iz Start() metode gore navedene Unity3D skripte koja se poziva jednom, pri inicijalizaciji objekta i vezane skripte. Drugi je dio funkcija za detekciju znakova koja vraća vrijednost 0 ako znak nije pronađen ili vrijednosti 1 tj. 2 za znak za uključivanje to jest isključivanje autonomnog stanja vozila. Ta se vrijednost obrađuje izvan biblioteke, u skriptama Unity3D.

Koraci rada algoritma za detekciju znakova jesu:

1. Učitaj pripadajuće klasifikatore koji su u XML obliku pomoću naredbe *CascadeClassifier.load( )*

2. Dohvat bitmapu trenutne slike glavnog prozora (simulacije)

3. Pretvori bitmapu iz RGB oblika u *grayscale,* crno-bijelu sliku. Rad s neobojenim slikama je brži zbog manjeg zauzeća memorije.

4. Izvrši ujednačavanje histograma – ovaj korak pojašnjava sliku ‘rastežući’ njezinu distribuciju boja na širi raspon. Ovo je potrebno kako bi klasifikator lakše pronašao sliku.

5. Za kaskadne klasifikatore izvrši funkciju OpenCV-a *detectMultiScale* koja preko argumenta vraća niz (vektor) detektiranih oblika.

6. Vrati prikladnu vrijednost s obzirom na pronađeni znak. Ova se vrijednost čita u skripti simulacije te se u slučaju prepoznatog znaka za aktivaciju pokreće dretva te prelazi u način autonomnog upravljanja.

3.1.1 Učenje klasifikatora za prepoznavanje znakova

Za prepoznavanje znakova koristi se kaskadni klasifikator, poznat na engleskom jeziku kao *cascade classifier with Haar-like features*. Koristi se neuralna mreža kako bi se klasifikator istrenirao i kako bi mogao prepoznavati tražene objekte na slici. Klasifikator je vektorski zapis u XML obliku kojega funkcija *detectMultiScale* spomenuta iznad koristi kao ulaz za algoritam prepoznavanja svojstava na slici. Klasifikator se priprema kroz niz koraka koji uključuju obradu ulaznih podataka kroz nekoliko programa. Potreban materijal za obradu podataka preuzet je s [1].

Ulaz za proces učenja klasifikatora za prepoznavanje pojedinog predmeta su dva skupa slika: negativi i pozitivi. Negativi su slike koje ne sadrže traženi predmet i često se nazivaju pozadinskim slikama. Pozitivi su slike koje sadrže traženi predmet pod različitim kutevima, s različitim osvjetljenjem, iz različitih gledišta. U procesu učenja, program kombinira pozitivne i negativne slike kako bi proizveo što veću bazu slika. Poželjno je imati što više slika jer će tako algoritam biti precizniji i otporniji na različite faktore poput osvjetljenja, smetnji, atmosferskih pojava pa je za ovaj klasifikator korišteno oko 200 pozitiva i 800 negativa. Za klasifikatore u stvarnim primjenama, gdje okoliš nije kontroliran, uobičajeno je imati broj pozitiva i negativa reda 10 000, kako bi se osiguralo da algoritam radi bez obzira na smetnje. Nužno je napomenuti kako je čak i za manje količine slika proces učenja veoma

zahtijevan i može trajati satima, pa čak i tjednima u nekim slučajevima. Učenje klasifikatora sadrži nekoliko koraka:

1. Dokumentiranje pozitiva i negativa u zapis koji čita daljnji korak učenja

2. Određivanje pozicije objekta na slikama pozitiva, okvir se unosi ručno putem programa koji koordinate zapisuje u tekstualnu datoteku.

3. Kreiranje vektorskog zapisa podataka koji se izvode iz dviju skupina slika

4. Pokretanje treniranja kojemu je ulaz vektorski zapis, broj pozitiva i negativa te veličina slikovnog elementa koji se koristi za učenje klasifikatora. Specificira se i broj razina (kaskada) klasifikatora a izlaz programa su mape koje odgovaraju razinama klasifikatora.

5. Posljednji korak je pretvaranje treniranog klasifikatora u XML oblik pogodan za korištenje u programu, u gore navedenoj funkciji.

Ova metoda prepoznavanja svojstava na slici je među najpoznatijima te se zbog svoje brzine izračuna može koristiti za prepoznavanje objekata na slici u stvarnom vremenu. Algoritam koristi *integralne slike*, tablice koje odgovaraju veličini slika koje su se koristile za učenje, koje u matričnom zapisu omogućavaju brz rad algoritma.

3.2 Praćenje oznaka

Kad simulacija pređe u stanju autonomnog kretanja, ključan je drugi dio biblioteke. Koristeći metodu prepoznavanja po boji, iz video *stream*-a se u stvarnom vremenu izdvajaju nijanse crvene boje (boja oznaka na cesti) pomoću metode pretvorbe RGB (*Red Green Blue* – crveno, zeleno, plavo) slike u HSV (*Hue Saturation Value* - boja, zasićenost, vrijednost) oblik. Nakon inicijalizacije koja je dijeljena s inicijalizacijom prvoga dijela, u Update() metodi C# skripte *SignDetector* se poziva funkcija praćenja pri svakom osvježavanju scene, u posebnoj dretvi. Ta funkcija računa koordinatu i veličinu najvećeg oblika prepoznatog na ekranu pomoću metode

prepoznavanja boje. Koordinata se tada preračunava u os prema kojoj vozilo autonomno odlučuje kamo je potrebno skrenuti. Ideja je nadasve jednostavna: ako je oblik detektiran na desnoj strani ekrana, vozilo mora skrenuti desno, ako je na lijevoj strani – obrnuto. Pri prepoznavanju filtriraju se objekti čija je površina manja od zadane granice da se isključe eventualne smetnje koje bi narušile funkcionalnost algoritma. Efikasnost algoritma za praćenje oznaka na cesti je određena i rasporedom oznaka na mapi (na cesti) pa preširok ili pregust raspored može uzrokovati gubitak mogućnosti praćenja putanje.

Koraci rada algoritma za praćenje oznaka jesu:

1. Dohvati bitmapu trenutne slike glavnog prozora (simulacije)

2. Pretvori bitmapu u HSV oblik

3. Filtriraj raspon boja koji odgovara crvenoj boji: (0,156,0) do (21,256,256) ili plavoj boji: (90, 196,0) do (120, 256, 256)

4. *Erodiraj* bitmapu slikovnim elementima veličine 2x2 piksela – ovo umanjuje smetnje na slici i briše manje oblike koji bi usporili ili omeli algoritam

5. Proširivanje (*dilation*) filtriranih dijelova slike elementom veličine 4x4 piksela – ovaj korak izražava filtrirane oblike kako bi ih algoritam lakše pronašao

6. Izračunaj interpoliranu izlaznu vrijednost koju simulacija interpretira i pretvara u signal za vozilo (skretanje lijevo ili desno). U simulaciji je u stanju autonomnog upravljanja bitno centrirati kameru jer se os skretanja interpolira prema X koordinati pronađenog elementa na ekranu. Različite implementacije se različito odražavaju na način rada simulacije – pregrubo interpoliranje vodi do krivudanja vozila. Naposlijetku, vrati vrijednost iz intervala [-1, 1] ako je oznaka pronađena negdje na ekranu, ili -2 inače.

Slika 7. *Princip pretvorbe koordinate detektiranog objekta u naredbu za vozilo*

Slika 6. prikazuje jednostavnu metodu kako se X-koordinata detektiranog oblika crvene boje pretvara u analognu naredbu za vozilo. Zeleni križić označava centar detektiranog objekta a vertikalne bijele linije odvajaju područja interpretacije koordinate. Naglasak je na analognoj naredbi za vozilo jer se u skripti *CarController.cs* povratna vrijednost funkcije *getHorizontalAxis* iz dinamičke biblioteke CVDetectorDLL.dll pretvara u os skretanja u rasponu od -1.0 do 1.0, gdje prva granica predstavlja puno skretanje ulijevo a druga skretanje udesno. Algoritam za praćenje radi efikasno s punom linijom na cesti no performanse se ne smanjuju jako niti u slučaju isprekidane linije. Simulacija raskršća i grananja puta na više dijelova je teško izvediva s linijom u jednoj boji no može se zaobići konvencijom bojanja linije različitim bojama. Tako bi, na primjer, prije raskršća gdje se cesta grana na lijevo i desno, korisnik odabrao plavu boju za skretanje ulijevo i crvenu za skretanje udesno. Primjenjivost ovakvog načina upravljanja vozilom je svakako diskutabilna no sličan se sustav može primijeniti u većim skladištima gdje automatizirani roboti dohvaćaju pojedine predmete na policama a po prostoru putuju prateći obojene linije na podu skladišta. Sustav nije prikladan za stvarna okruženja gdje bi boje okolnih predmeta i objekata mogle zasmetati algoritmu ili gdje postoje nepredvidivi faktori poput drugih vozila ili pješaka.

3.3 Druge metode upravljanja

Implementacija ove simulacije je, zbog pojedinih dijelova, ograničena na operacijski sustav Windows. Snimanje radnog prozora korišteno kao ulaz za algoritme praćenja i prepoznavanja je specifične implementacije za Windows sustave, kao i rad s dinamičkim bibliotekama. Zbog ovoga, bilo bi otežano izvesti komunikaciju s mobilnim uređajima ili programima pripremljenim za druge platforme. No, valja teoretski razmotriti ove mogućnosti: mobilni uređaji imaju žiroskope i inercijske senzore koji bi se mogli iskoristiti za usmjeravanje vozila, BCI (*Brain – computer interface*) uređaji isto se mogu pripremiti da šalju naredbe za usmjeravanje vozila, a mogu se koristiti i druge metode orijentacije u prostoru, poput sonara koji se može simulirati u Unity3D

okruženju. Teško je izdvojiti one metode koji bi se mogle smatrati korisnima ili primjenjivima jer je ova simulacija samo jedan specifičan slučaj upravljanja vozilom u određenom okolišu, no te se metode mogu koristiti za druge vrste vozila poput plovila ili vlakova. Također je, naravno, presudno napraviti puno pouzdaniji sustav u slučaju upravljanja pravog vozila gdje nema mjesta za pogreške i rizik je stvaran faktor. Jedan od mogućih smjerova poboljšanja performanse sustava za autonomno upravljanje jest implementacija učenja i neuralnih mreža kako bi vozilo prikupljalo znanje i izbjegavalo prepreke i dodavanje senzora za blizinu te dinamička prilagodba brzine kretanja. Slične se implementacije koriste u stvarnim vozilima koje koriste autonomno kretanje.

Implementacija pokretanja vozila pomoću senzorskih podataka s uređaja s operacijskim sustavom Android nije implementirana u simulaciji no razrađena je u nastavku. Gotovo svi mobilni uređaju sadrže dvo-osni akcelerometar a neki imaju i žiroskop kako bi još preciznije utvrdili položaj u prostoru .Android SDK (*Software development kit*) nudi jednostavan način rada s senzorima uređaja i dohvaćanja podataka s istih. Podatke nekako treba poslati s Android uređaja na računalo kako bi ih simulacija koristila: za to je prikladan prijenos preko mreže, pa je vanjski uređaj tako TCP klijent a skripta unutar simulacije je poslužitelj. Prikladnije nego pisati skriptu u biblioteci u jeziku C++ je izmijeniti skriptu *SignDetector.cs* i implementirati server u programskom jeziku C#.

4. Funkcionalnost Unity3D skripti

Sljedeća ključna komponenta je skripta *SignDetector.cs* koja rukuje s vanjskom bibliotekom CVDetectorDLL.dll te poziva njene funkcije i obrađuje povratne vrijednosti. Unity3D internu funkcionalnost za renderiranje scene i manipulaciju objektima izvršava u vlastitoj dretvi. Dodatno, ova simulacija koristi još dvije korisničke dretve kako bi operacija algoritama za prepoznavanje i praćenje bila optimizirana. Jedna je dretva posvećena funkcijama za prepoznavanje znakova a druga funkcijama za praćenje

oznaka na cesti. Skripta *SignDetector* se brine za pozivanje ovih dretvi i rad s njima.

```
public class SignDetector : MonoBehaviour
{
    Thread detectionThread, trackingThread;

    void Start(){
        ...
        detectionThread = new Thread(new ThreadStart(doDetection));
        trackingThread = new Thread(new ThreadStart(doColorTracking));
             detectionThread.Start ();
        ...
    }

    void doDetection(){

    void doColorTracking(){ ... }

    void Update(){ ... }

    void OnApplicationQuit(){ ... }
}
```

Metoda Start() se poziva jednom, pri instanciranju skripte, pri pokretanju scene u Unity3D. Tamo se instanciraju dretve i pripremaju za pokretanje, a također se poziva i vanjska funkcija za inicijalizaciju biblioteke CVDetectorDLL. Dretva *detectionThread* ima registriranu metodu *doDetection* i poziva se odmah pri inicijalizaciji. Njena je namjena pozivanje algoritma iz prvog dijela vanjske biblioteke opisanog u prethodnom poglavlju. Ako je znak za uključivanje autonomnog stanja pronađen na slici, povratna je vrijednost 1 i metoda *doDetection* pokreće drugu dretvu, čija je namjena praćenje obojenih simbola na cesti. Registrirana funkcija druge dretve, *trackerThread*, jest *doColorTracking.* Jednostavna funkcionalnost te metode je usmjeravati vozilo u pravilnom smjeru, s obzirom na povratnu vrijednost funkcije iz vanjske biblioteke. Pritiskom na tipku *C* na tipkovnici tijekom simulacije mijenja boju koju vozilo prati na cesti u načinu autonomnog praćenja. Početno, boja je postavljena na crvenu, no može se promijeniti na plavu te natrag na crvenu uzastopnim pritiskanjem tipke *C.* Ovako korisnik može pri grananju puta odabrati kamo će vozilo skrenuti.

Update() metoda, koju Unity3D poziva pri svakom osvježavanju scene, ima namjenu da simulira pritisak tipke s obzirom na željeno kretanje vozila, ako je autonomno stanje aktualno. Program se brine da brzina vozila u autonomnom načinu bude u intervalu između 17 i 20 km/h (početno) te usmjerava kretanje vozila s obzirom na linearnu interpolaciju koordinate iz funkcije *doColorTracking.* Prednost ovog načina je što simulira vanjski unos korisnika te je uz manje promjene algoritam prenosiv na druge sustave, primjerice na robotska vozila koja šalju signale aktuatorima. Posljednji dio skripte *SignDetector* koju razmatramo je funkcija *OnApplicationQuit()* koja prekida dretve pri izlasku iz aplikacije, kako ne bi bilo neželjenih učinaka višedretvenosti.

Skripta *CarController.cs* je zadužena za kontroliranje vozila i njegovog ponašanja. Ovdje se prate položaji kotača i njihovih objekata, brzina, specifikacije vozila bitne za simulaciju i sile koje na vozilo djeluju pri kretanju. Vozilo ima definiran broj okretaja kotača, radijus skretanja, moment sile na osovinu i moment sile kočnice, točku centra mase, način pogona (prednji, stražnji ili pogon na sva četiri kotača) i koeficijente vezane uz otpor opruga i trenje po osi vertikalnoj i horizontalnoj na vozilo. Sve ove značajke omogućavaju realističnu simulaciju ponašanja vozila te doprinose dojmu pri simulaciji. Definicija maksimalnog i optimalnog broja okretaja kotača u minuti omogućava nelinearno ubrzanje vozila, poput stvarnog ubrzanja. Radi jednostavnosti simulacije, prijenos i mjenjač nisu izvedeni, već se vozilo ponaša kao da ima kontinuirani ili izravni prijenos. Metoda *doRollBar* uzimajući u obzir inercijske sile pri skretanju i vožnji vozila te koeficijente opruga pruža silu u suprotnom smjeru i tako simulira fizički utemeljeno ponašanje šasije vozila. Sila na opruge se računa na temelju faktora *AntiRoll* i promjene puta opruge po vertikalnoj osi. Pozivom

```
rigidbody.AddForceAtPosition(WheelL.transform.up * -antiRollForce,
                                        WheelL.transform.position);
```

grafički pogon Unity dodaje silu opruzi prema izračunatim vrijednostima. Nadalje, skripta usmjerava rotaciju prednjih kotača oko Z osi (prema gore, okomito na vozilo) s obzirom na unos korisnika ili, u načinu autonomnog

pokretanja, povratnoj vrijednosti funkcije *getHorizontalAxis* iz dinamičke biblioteke CVDetectorDLL. Posljednja stvar koju ova skripta radi jest okretanje svih kotača oko svoje Y osi, tj. izvršava kotrljanje. Slika 7. prikazuje programski kod metode *FixedUpdate* koju Unity3D pogon poziva periodički.

```
49      void FixedUpdate () {
50          // Calculate and display speed to GUI
51          speed =  -(wheelRR.radius * Mathf.PI * wheelRR.rpm * 60f / 1000f);
52          TextField.text = "Speed: " +speed.ToString("0.00") + "km/h    RPM: " + (-wheelRL.rpm);
53
54          float scaledTorque =-Input.GetAxis("Vertical") * torque;
55
56          if(wheelRL.rpm < idealRPM)
57              scaledTorque = Mathf.Lerp(scaledTorque/10f, scaledTorque, wheelRL.rpm / idealRPM );
58          else
59              scaledTorque = Mathf.Lerp(scaledTorque, 0,  (wheelRL.rpm-idealRPM) / (maxRPM-idealRPM) );
60
61          // Counter the inertial forces on the vehicle by adding force to springs
62          DoRollBar(wheelFR, wheelFL);
63          DoRollBar(wheelRR, wheelRL);
64
65          // Get user input for turning or, if autopilot is active, axis value from DLL
66          if (SignDetector.getAxis () == -2.0) {
67              wheelFR.steerAngle = Input.GetAxis ("Horizontal") * turnRadius;
68              wheelFL.steerAngle = Input.GetAxis ("Horizontal") * turnRadius;
69          } else {
70              wheelFR.steerAngle = (float)SignDetector.getAxis() * turnRadius;
71              wheelFL.steerAngle = (float)SignDetector.getAxis() * turnRadius;
72          }
73
74          // Set torque to drive-wheels
75          wheelFR.motorTorque = driveMode==DriveMode.Rear  ? 0 : scaledTorque;
76          wheelFL.motorTorque = driveMode==DriveMode.Rear  ? 0 : scaledTorque;
77          wheelRR.motorTorque = driveMode==DriveMode.Front ? 0 : scaledTorque;
78          wheelRL.motorTorque = driveMode==DriveMode.Front ? 0 : scaledTorque;
79          // If brakes are active, apply brakeTorque
80          if(Input.GetButton("Fire1")) {
81              wheelFR.brakeTorque = brakeTorque;
82              wheelFL.brakeTorque = brakeTorque;
83              wheelRR.brakeTorque = brakeTorque;
84              wheelRL.brakeTorque = brakeTorque;
85          }
86          else {
87              wheelFR.brakeTorque = 0;
88              wheelFL.brakeTorque = 0;
89              wheelRR.brakeTorque = 0;
90              wheelRL.brakeTorque = 0;
91          }
92
93          rotateWheels();
94      }
95
```

Slika 8. *Metoda* FixedUpdate *skripte* CarController.cs

Druga skripta vezana uz pokretanje vozila je *TireToWheel.cs* koja prati položaj i svojstva kotača i računa podatke korištene od strane prethodno pokazane skripte *CarController.cs*. Svaki objekt razreda TireToWheel posjeduje referencu na pripadajući *WheelCollider* tj. entitet koji služi za detekciju sudara kotača i terena. Na temelju pozicije vozila i terena, objekt vezan uz pojedini objekt kotača podešava njegov položaj s obzirom na šasiju

vozila, što omogućava simulaciju neovisnog ovjesa vozila. Trenje između kotača i podloge je definirano po dvije osi i služi za određivanje maksimalnog kuta pod kojim se vozilo može kretati u odnosu na teren. Važno je ograničiti kut zbog položaja centra mase vozila: ako ove vrijednosti nisu dobro usuglašene, vozilo će se prevrtati previše često.

Važno je spomenuti i skriptu *UIText.cs* koja vrši ispisivanje u tekstualne elemente grafičkog korisničkog sučelja koji služe za obavještavanje korisnika o događanjima tijekom simulacije. Skripta koristi listu od tri tekstualna niza koji se ispisuju u pripadajućim tekstualnim elementima pozivom metode *ShowText(string text)* koju poziva skripta *SignDetector.cs* pri određenim događajima koji su važni za korisnika.

Skripta *CameraLook.cs* je veoma sažeta i omogućava okretanje kamere pomoću miša. Da bi se to postiglo, prvo se zaključa kursor miša kako bi bio centriran u simulaciji, a onda se s obzirom na promjenu horizontalne osi kamera okreće oko svoje Z osi. Skripta *StartGame.cs* je dio scene glavnog izbornika i izvršava željenu funkcionalnost od obje tipke prisutne na izborniku: promjena na glavnu scenu simulacije ili izlazak iz programa.

4.1 Komunikacija simulacije i vanjske biblioteke

Komunikacija vanjske dinamičke biblioteke CVDetectorDLL i Unity3D skripti zaduženih za funkcionalnost simulacije je ključan segment. Algoritmi u biblioteci napisani u programskom jeziku C++ računaju potrebne vrijednosti s obzirom na trenutno stanje simulacije a skripte u jeziku C# rade s tim traženim vrijednostima. Za komunikaciju između ta dva modula potrebno je prevesti dio koji računa vrijednosti kao dinamičku biblioteku kako bi se njegove funkcije mogle pozivati neovisno. Vanjska je biblioteka izrađena u razvojnom okolišu (IDE) *Visual Studio 2010 Professional.* U zaglavlju *CVDetectorDLL.h* pomoću pretprocesorske naredbe definirano je da biblioteka izvozi (*export*) funkcije:

```
#ifdef SIGNDETECTORSDLL_EXPORTS
#define SIGNDETECTORSDLL_API __declspec(dllexport)
```

Pri prevođenju definiran je simbol “SIGNDETECTORSDLL_EXPORTS” pa pri prevođenju biblioteka izvozi funkcije a pri korištenju, kad simbol nije definiran, biblioteka uvozi funkcije. U izvornom kodu, funkcije su definirane u bloku extern "C" koji prevoditelju nalaže da funkcije prevodi uz konvenciju jezika C, kako se ime funkcije ne bi dekoriralo. Funkcije su deklarirane na sljedeći način:

```
__declspec(dllexport) int __cdecl SignDetector::getFoundSign(){}
```

Prvi dio specificira da se funkcija izvozi iz biblioteke, a dio *__cdecl* ponovno specificira da funkcija treba biti prevedena sukladno s konvencijom jezika C. Ako takvo prevođenje nije moguće, jer su npr. funkcije podijeljene u prostore imena (*namespace*), ime funkcije će biti dekorirano kako bi se zadržala kompatibilnost s C programima koji bi mogli koristiti biblioteku te ne podržavaju objektno - orijentirane specifičnosti.

Skripta *SignDetector.cs* napisana u programskom jeziku C# koju koristi Unity3D simulacija je odgovorna za drugu stranu povezivanja. Ona prvo otvara dinamičku biblioteku te uvozi njezine funkcije, a onda pozivajući te funkcije dobiva potrebne vrijednosti.

```
[DllImport("CVDetectorDLL", CallingConvention = CallingConvention.Cdecl, EntryPoint = @"?initialiseDetector@SignDetector@CVDetector@@SAHXZ")]
    private static extern int initialiseDetector();
```

Iznad je naveden dio koda u programskom jeziku C# koji učitava biblioteku pozivom funkcije *DLLImport* te ispod definira prototip uvezene funkcije *initialiseDetector.* Kao ulazna točka u biblioteku zadano je dekorirano ime navedene funkcije. Ovdje vidimo da je dekorirano ime složeno od imena funkcije, imena razreda u kojem je smještena i imena prostora imena u kojemu je razred smješten. U skripti pozivi su funkcije sasvim normalni, kao da radimo s običnom funkcijom unutar skripte. Za svaku funkciju koju uvozimo potrebno je ponoviti ovaj postupak.

Dodirne točke Unity3D simulacije i vanjske biblioteke su ograničene na nekoliko zahtijeva skripte *SignDetector.cs*. Za skriptu je potrebno da funkcija

getFoundSign vraća 0 ako znak nije pronađen, 1 ako je pronađen znak za početak autonomnog upravljanja i 2 ako je pronađen znak za zaustavljanje autonomnog upravljanja. Također je potrebno da funkcija *getHorizontalAxis* vraća vrijednost između -1.0 koja označava skretanje ulijevo i 1.0 koja označava skretanje udesno. Ovo dovodi do svojstva da se način rada u vanjskoj biblioteci može zamijeniti bez izmjena u skriptama simulacije. Moguće je na primjer umetnuti vanjsku biblioteku koja radi s podacima iz žiroskopskog senzora ili uređaja za interpretaciju moždanih signala.

Zaključak

Cilj ovog rada je izrada i implementacija simuliranog okoliša te sučelja za jedan od načina nekonvencionalnog upravljanja vozilima, tj. upravljanje simuliranim vozilom putem kamere i obojenih elemenata na mapi. Unity3D grafički pogon nudi velik stupanj fleksibilnosti i druge metode pokretanja vozila moguće je implementirati koristeći istu simulaciju, koja je dizajnirana da bude jednostavna i otvorena za nadogradnje. Program traženja i praćenja znakova oslanja se na računalni vid i biblioteku otvorenog koda OpenCV, a njegovi su ključni elementi implementirani u programskom jeziku C++. Za nadogradnju tih algoritama nije potrebno uopće mijenjati Unity3D scenu ili skripte, jer se funkcije vanjske biblioteke CVDetectorDLL pozivaju transparentno te je za nadogradnju samo potrebno zamijeniti navedenu biblioteku. Također je jednostavno i promijeniti model vozila ili model mape te utvrditi ponašanje algoritma u drugačijem okolišu, npr. urbana scena s mnogo detalja koji bi mogli omesti algoritam praćenja. Trenutna je mapa implementirana tako da do izražaja dolazi i teren te je moguće demonstrirati da algoritam funkcionira i na neravnom terenu.

Sažetak

Upravljanja modelom vozila pomoću nekonvencionalnog sučelja

U ovom radu implementirana je simulacija pokretanja vozila u virtualnom okolišu pomoću algoritama računalnog vida. U programskom alatu Blender modelirano je vozilo i okoliš koji sadrži ceste s obojenim oznakama, prometne znakove i jednostavne objekte okoline. Korisnik može kontrolirati vozilo konvencionalnim načinom, putem tipkovnice i miša, a kad vozilo prepozna prometni znak, ono prelazi u autonomni način pokretanja i slijedi obojene oznake na cesti. Za prepoznavanje znakova implementiran je algoritam prepoznavanja klasifikatorom svojstava tipa Haar a za praćenje oznaka na cesti koristi se metoda prepoznavanja i praćenja boje pretvorbom iz RGB(*Red Green Blue*) modela slika u HSV(*Hue Saturation Value*) model slike. Odnos vozila i okoliša te fizički točan sustav simulacije vozila su implementirani pomoću Unity3D grafičkog pogona.

Ključne riječi: simulacija vozila, računalni vid, virtualno okruženje, autonomna vožnja, nekonvencionalno sučelje, Unity3D, Blender, C#, C++

Abstract

Driving of a vehicle model using an unconventional user interface

This thesis implements a simulation of a vehicle movement method in a virtual environment using computer vision algorithms. Using Blender modeling software, a vehicle has been modeled along with terrain consisting of simple environment objects, traffic signs and colored marks on the road. The user can control the vehicle using conventional mouse and keyboard input. When the vehicle detects a traffic sign, it switches to an autonomous movement state where it follows the colored markers on the road surface. The sign detector algorithm uses a cascade classifier with Haar-like features and the road tracking algorithm utilises a method of color tracking with RGB(*Red Green Blue*) to HSV(*Hue Saturation Value*) conversion. The relationship between the vehicle and the environment as well as a physically

sound system of vehicle dynamics has been implemented in the Unity3D graphics engine.

Keywords: vehicle simulation, computer vision, virtual environment, autonomous driving, unconvenctional interface, Unity3D, Blender, C#, C++

Literatura

[1] OpenCV 2.4.11.0 Documentation,

http://docs.opencv.org/index.html, 2015

[2] Microsoft Developer Network,

https://msdn.microsoft.com/, 2015

[3] Unity3D Documentation,

http://docs.unity3d.com/Manual/index.html, 2015

[4] Shehan's blog – opencv haartraining,

http://www.tectute.com/2011/06/opencv-haartraining.html, 2011

[5] Blender 2.74 Reference Manual,

https://www.blender.org/manual/, 2015