Windows aplikacija za nadzornu kameru

Kolar, Marin

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJAOSIJEK

Sveučilišni studij

Windows aplikacija za nadzornu kameru

Diplomski rad

Marin Kolar

Osijek, 2023.

SADRŽAJ

1. UVOD

Prema podacima statista za 2023. godinu, trenutno je na svijetu 6.92 milijarde korisnika pametnih mobitela (engl. *smartphone*) [1]. Samo na temelju tih podataka, može se primijetiti sveprisutnost kamera u svijetu, ne uzimajući u obzir ostale uređaje kao što su prijenosna računala i stolna računala. U današnjem vremenu i tehnološkom okruženju u kojem se svijet nalazi, javlja se sve veća potreba za aplikacijama koje detektiraju i analiziraju pokrete kako bi se odgovorilo na zahtjeve u različitim područjima kao što su sigurnost, medicina, zabava i industrija. Detekcija pokreta omogućava sustavima da prepoznaju promjene u okruženju i reagiraju na njih, što može rezultirati poboljšanom interakcijom s korisnicima, povećanom sigurnošću i boljim upravljanjem procesima. Ovaj rad istražuje implementaciju aplikacije za detekciju različitih vrsta pokreta te analizira uspješnost i brzinu obrade takve aplikacije. Aplikacija je testirana uspješno, a za rad je korišteno računalo (Intel i5 2300, 8GB RAM-a) starosti 12 godina, te je samim time dokazano da za rad ove aplikacije nisu potrebne napredne komponente, već samo prosječno računalo s web kamerom

Detekcija pokreta u stvarnom vremenu predstavlja tehnički izazov zbog raznih varijacija u okruženju, brzini i obliku pokreta, kao i kvaliteti kamere koja se koristi. Cilj ovog rada je razviti aplikaciju koja može pouzdano detektirati različite vrste pokreta u stvarnom vremenu, dok istovremeno optimizira brzinu obrade.

1.1. Pregled područja

Postoje mnoga istraživanja o detekciji pokreta preko kamere i to istraživačko područje je već dugo vremena aktivno te je primjena jako široka. Radovi koji se bave tom tematikom su [2], [3] i [4].
U radu [2] opisana je nova metoda detekcije objekata u videu koja koristi i kontekst kretanja objekta i prostorno-vremenske agregirane značajke kako bi poboljšala performanse detekcije objekata u videu. Rad [3] predlaže softverski sustav za detekciju pokreta koji omogućuje nadzor (promatranje) kretanja oko objekta ili vizualnog područja. Rad [4] predlaže definiranje pokretnih objekata u videu pomoću vektora gibanja poput MPEG-a (engl. *Moving Picture Experts Group*).

Uz istraživački dio, postoji i segment praktične primjene. Prije svega, tu su mobilne i desktop aplikacije koje se koriste u svakodnevnoj primjeni.

Aplikacija Security Camera CZ je namijenjena za roditeljski nadzor, ili nadzor kućnih ljubimaca [5].

Video Surveillance Ivideon aplikacija za video nadzor, daljinski video nadzor i video snimanje za sigurnosne kamere. Glavne značajke su joj obavijest u slučaju da se nešto dogodi, te snimanje videa lokalno ili na oblaku (engl. *cloud*) [6].

VicoHome: Security Camera App koja omogućuje spajanje na postojeću kameru, snimanje i video nadzor, te pregledavanje postojećih prethodno snimljenih video zapisa [7].

2. KORIŠTENE TEHNOLOGIJE

U ovom poglavlju opisane su tehnologije koje su ključne za razvoj aplikacije koja ima mogućnost detekcije pokreta. To su programski jezik C# i biblioteka EMGU CV.C# je korišten kao osnovni programski jezik za razvoj aplikacije, dok je EMGU CV biblioteka omogućila pristup naprednim algoritmima za obradu slika u realnom vremenu te njihovu daljnju obradu.

2.1. C# programski jezik

C# je višenamjenski objektno orijentirani programski jezik koji je razvijen od strane Microsofta 2000. godine unutar .NET platforme [8]. Nastao je kao odgovor na nedostatke tada postojećih jezika poput C-a, C++-a i Visual Basica, dok je istovremeno zadržao njihove pozitivne karakteristike. Kao višenamjenski jezik, C# se koristi u različitim područjima, uključujući izradu Windows aplikacija, baza podataka, XML web servisa i klijent-server aplikacija. Također je izuzetno pogodan za razvoj softverskih komponenti pogodnih za primjenu u distribuiranim okruženjima. C# je potpuno objektno orijentiran programski jezik, te svi elementi unutar njega predstavljaju objekte. Objekte možemo definirati kao određene strukture koje sadrže podatkovne elemente i metode, te opisuju njihovu međusobnu interakciju. Sintaksa C#-a slična je sintaksama koje se nalaze u jezicima poput C-a, C++-a i Jave, ali istovremeno pojednostavljuje određene kompleksnosti i složenosti prisutne u C++, dok pruža određene funkcionalnosti koje nedostaju u Javi, kao što je izravno pristupanje memoriji. U C#-u, točka sa zarezom označava kraj naredbe, dok se grupiranje naredbi vrši putem zagrada. Budući da je C# objektno orijentiran jezik, podržava koncepte enkapsulacije, nasljeđivanja i polimorfizma. Sve varijable i metode, uključujući i Main metodu, su učahurene (enkapsulirane) unutar definicije klase. Enkapsulacijom se postiže sakrivanje unutarnjih detalja klase, te objekti postaju neovisniji i interne promjene jednog objekta ne utječu na rad drugog objekta. C# također podržava nasljeđivanje, što znači da jedna klasa može naslijediti sve ili dio atributa i metoda koji su definirani u nadređenoj klasi (klasa roditelj) sa mogućnošću implementacije više sučelja. Jedna od glavnih značajki C#-a je da ne zahtijeva deklariranje metoda i tipova određenim redoslijedom. U C#-u je moguće definirati neograničen broj klasa, sučelja i struktura, koje pružaju i fleksibilnost u organizaciji koda i prilagodbu različitim potrebama. C# je također poznat po svojoj podršci za upravljanje memorijom putem sustava sakupljanja smeća (engl. *Garbage collection*).Sustav sakupljanja smeća automatski oslobađa memoriju koju više ne koriste objekti u program, te se ne mora ručno pratiti i oslobađati memorija, što omogućava veću razinu sigurnosti pri razvoju sigurnih i stabilnih aplikacija te se smanjuju greške uzrokovane curenjem memorije (engl. *memory leak*).

Još jedna važna značajka C#-a je podrška za paralelno programiranje. Uz pomoć klase "Task" i "async/await" sintakse, mogu se lako izvršavati asinkrone operacije te obrađivati više zadataka paralelno. Ovo je posebno korisno za poboljšanje performansi aplikacija koje zahtijevaju obradu velikih količina podataka ili komunikaciju s vanjskim resursima poput mrežnih poziva ili baza podataka. Sustavski dio C#-a, poznat kao .NET Framework, također pruža širok spektar biblioteka i okvira koji olakšavaju razvoj aplikacija. Biblioteke poput Windows Forms, WPF (Windows Presentation Foundation) i ASP.NET omogućavaju brzo izrađivanje grafičkih korisničkih sučelja, web aplikacija i još mnogo toga. Osim toga, C# je interoperabilan s drugim jezicima koji se izvode na .NET platformi, što omogućava integraciju s postojećim kodom napisanim na jezicima poput Visual Basic i F#.

2.2. EMGU CV

EmguCV je iznimno koristan .NET omotač za OpenCV biblioteku, koji omogućuje programerima da iskoriste računalnu obradu slike u svojim .NET aplikacijama [9]. OpenCV je biblioteka otvorenog koda (engl. *open-source*) koja pruža bogat skup alata i tehnika za analizu, manipulaciju i obradu slika. Jedna od glavnih prednosti EmguCV-a je njegova višestruka platformska podrška. S obzirom na mogućnost sastavljanja (engl. *compile*) u Mono okruženju, EmguCV može se koristiti na različitim platformama kao što su Linux, Windows, Mac OSX, pa čak i Android. To znači da programeri mogu stvarati aplikacije za obradu slika koje će funkcionirati na različitim uređajima i operacijskim sustavima. Jedna od ključnih značajki EmguCV-a je njegova podrška za različite programski jezike. Aplikacije razvijene u C#, C++, Iron Pythonu ili nekom drugom .NET kompatibilnom jeziku, mogu pozvati OpenCV funkcije putem EmguCV-a. To omogućuje developerima pristup i korištenje funkcija i metoda koje su već napisane u C i C++ jeziku unutar .NET okruženja.

U radu s EmguCV-om, koriste se različiti imenici (engl. *namespace*) kako bi se olakšalo korištenje različitih funkcionalnosti:

1. Emgu.CV.UI – Imenik koji sadrži korisničko sučelje (ImageBox) za prikaz Image objekata. Iznimno korisno za vizualizaciju i pregled slika unutar aplikacije.

2. Emgu.CV.Util–Sadrži zbirku uslužnih projekata koje koriste EmguCV projekti. Ove uslužne funkcije olakšavaju rad s slikama, omogućujući programerima da brzo i učinkovito manipuliraju podacima.

3. Emgu.Util–Imenik koji sadrži dodatne uslužne projekte koji podržavaju funkcionalnosti Emgu projekata. Pružajući dodatnu potporu, ovaj imenik doprinosi glatkom i učinkovitom razvoju aplikacija.

4. Emgu.CV.Tracking–Imenik u kojem se nalazi se API za dugoročno optičko praćenje. Ova značajka omogućuje praćenje objekata kroz više kadrova slike, što je korisno u različitim aplikacijama za analizu videozapisa.

5. Emgu.CV.Structure – Imenik koji predstavlja omotač za OpenCV strukture. Kroz ovaj omotač, omogućen je lakši pristup i manipulacija strukturama podataka koje su temeljne za obradu slike.

6. Emgu.CV.CvEnum - Ovdje se nalazi nabrajanje (enumeracija) za OpenCV. To olakšava korištenje različitih konstanti i postavki unutar EmguCV-a, što dovodi do jasnijeg i čitljivijeg koda.

3. IMPLEMENTACIJA

Kako bi aplikacija uspješno radila, potrebno je provesti niz koraka čija će implementacija rezultirati uspješno detektiranim pokretom. Prvi korak je učitavanje unaprijed snimljenog video zapisa ili dohvaćanje slike direktno s kamere, ovisno o izboru koji se može definirati u samome kodu. Sljedeći korak je obrada videa računanjem optičkog toka, primjenom Farnebackove metode. Nakon izračuna optičkog toka, traži se detekcija pokreta u slici, te ako je pokret uočen kao zadnji korak započinje snimanje video isječaka i njegovo pohranjivanje.

3.1. Učitavanje videa

Učitavanje video isječaka ili slike s kamere ključno je za započinjanje procesa računalne obrade slike pomoću EmguCV biblioteke. U nastavku je detaljan opis dijela koda koji se odnosi na tu funkcionalnost.

U svrhu ovog rada, omogućeno je učitavanje video isječaka koji su ranije snimljeni u nekom video formatu, ili su učitane slike direktno s kamere, što olakšava kasnije testiranje i provjeru algoritama računalne obrade slike.

```
private void btnStart_Click(object sender, EventArgs e)

   {

    ibVideo.Enabled = false;
    if (capture == null)

    {
        try
        {

            //capture = new VideoCapture("parking_example_1.avi");
            capture = new VideoCapture(0);
            prevGrayImage = new Image<Gray, byte>(capture.Width, capture.Height, new Gray(0));

            lbResolution.Text = "Resolution: " + capture.Width + "x" + capture.Height + "px";

            fourcc = Convert.ToInt32(capture.GetCaptureProperty(Emgu.CV.CvEnum.CapProp.FourCC));



        }
        catch (NullReferenceException excpt)
        {
            MessageBox.Show(excpt.Message);
        }
    }
```

Sl.3.1.Učitavanje videa - VideoCapture klasa

Za dohvaćanje slike sa kamere ili videa koristi se VideoCapture klasa (Slika 3.1.). VideoCapture() stvara snimku pomoću zadane kamere. Prilikom pokretanja aplikacije, kamera ili video isječak se inicijalizira samo jednom. Ako je varijabla "capture" (tipa VideoCapture) null, tada se provjerava želimo li učitati video isječak ili sliku s kamere.

Ako se želi učitati (unaprijed snimljeni) video isječak, koristi se “capture = newVideoCapture("parking_example_1.avi");”, gdje "parking_example_1.avi" predstavlja ime video isječka koji se nalazi u debug folderu. Ovdje se može koristiti bilo koji drugi video format koji želimo analizirati.

Za učitavanje slike s kamere, koristi se `capture = newVideoCapture(0);`, gdje broj 0 predstavlja vanjsku kameru. Ako postoji više kamera, može se koristiti broj 1(za sljedeću kameru) kako bi se označilo kojoj kameri se pristupa. Primjer “capture = newVideoCapture(1);”. Budući da se slika definira po svojim generičkim parametrima boji i dubini, za stvaranje slike u boji veličine 640x360 koristi se sljedeća metoda:

Image<Bgr, Byte> image1 = newImage<Bgr ,Byte>(640,360);

Nakon što se video isječak ili slika s kamere uspješno učitaju, stvara se nova slika "prevGrayImage" u sivim tonovima (engl. *Grayscale*), koja će služiti za praćenje promjena između pojedinih okvira (engl. *frame*) videa. Budući da rezolucija ovisi o kameri, kako se rezolucija ne bi morala svaki puta ručno mijenjati, umjesto unaprijed zadanih vrijednosti, definira se dohvaćanje visine i širine direktno s kamere:

prevGrayImage = newImage<Gray, byte>(capture.Width, capture.Height, new Gray(0));

Definiranje visine i širine na gore navedeni način može se koristiti u daljnjoj obradi jer daje uvid u kojoj rezoluciji snima kamera, te je pogodno za dobivanje određenih podataka koji se mogu koristiti u daljnjoj obradi. Rezolucija kamere ili video isječka se prikazuje na grafičkom sučelju pomoću label elementa (lbResolution). Primjer korištenja širine i visine za dobivanje rezolucije koja će biti ispisana na grafičkom sučelju:

lbResolution.Text = „Resolution:“ + capture.Width + „x“ + capture.Height + „px“;

Sljedeći korak je priprema formata u kojem će se snimati. To se radi kako bi osiguralo da snimak bude u istom formatu kao ulazni video. U tu svrhu koristi se metoda: *`fourcc = Convert.ToInt32(capture.GetCaptureProperty(Emgu.CV.CvEnum.CapProp.FourCC));`*.

Kada je inicijalizacija završena, ako je video isječak ili slika s kamere uspješno učitana, gumb "Start/Stop" (Slika 3.2.) mijenja svoj tekst ovisno o stanju snimanja (captureInProgress). Ako snimanje već traje (captureInProgress je *true*), pritiskom na gumb zaustavlja se snimanje i uklanja se događaj "ProcessFrame" iz događajnog lanca "Application.Idle", što znači da se funkcija za obradu slike (ProcessFrame) više neće izvršavati.

Ako snimanje nije u tijeku (captureInProgress je *false*), pritiskom na gumb pokreće se snimanje (engl. *Capturing*) i dodaje se događaj "ProcessFrame" u događajni lanac "Application.Idle". To znači da će se funkcija "ProcessFrame" izvršavati za svaki novi okvir videa ili kad se promijeni stanje prikaza na kameri.

Gore navedeni kod omogućuje jednostavno pokretanje i zaustavljanje snimanja video isječaka ili slika s kamere, što je korisno za daljnju obradu tih podataka u algoritmima računalne obrade slike.

```
if (capture != null)
{
    if (captureInProgress)
    {
        btnStart.Text = "Start"; //

        Application.Idle -= ProcessFrame;
    }
    else
    {

        btnStart.Text = "Stop";

        Application.Idle += ProcessFrame;
    }

    captureInProgress = !captureInProgress;
}
```

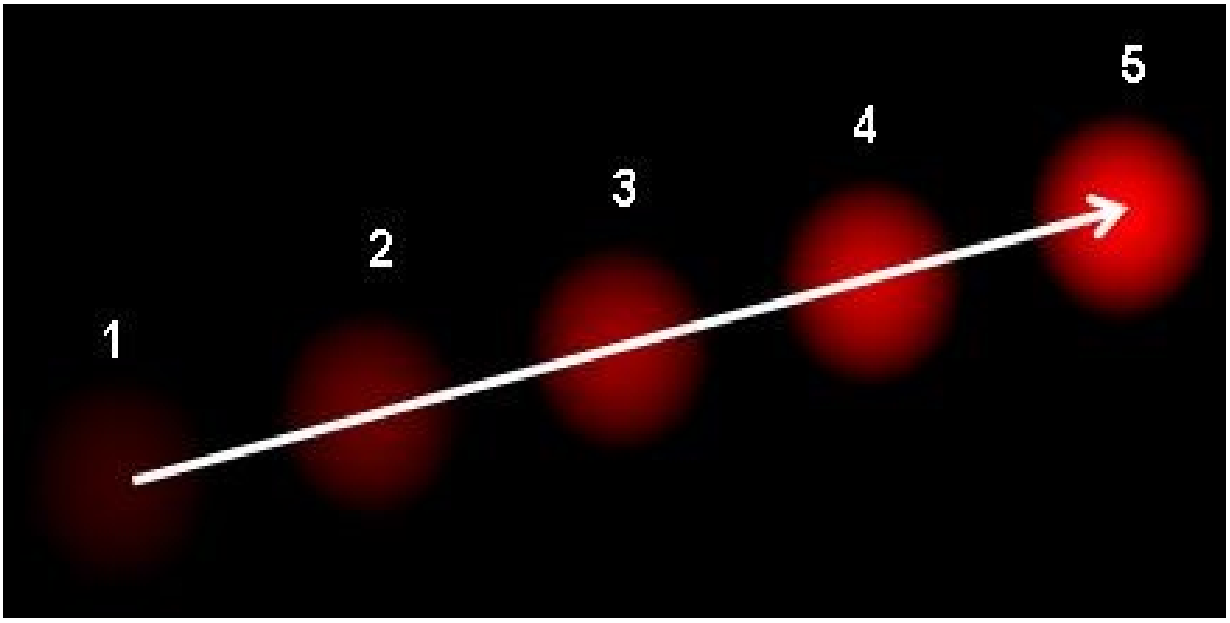
Sl. 3.2. Funkcija "Start/Stop"

3.2. Optički tok

Optički tok (engl. *optic flow*) je obrazac prividnog kretanja objekata, površina i rubova u vizualnom prizoru koji je uzrokovan relativnim kretanjem između promatrača i okoline. Optički tok se također može definirati kao distribucija prividnih brzina kretanja svjetlosnih obrazaca na

slici. Često se koristi u robotici za obradu slike, kontrolu navigacije, detekciju pokreta, informacije o vremenu do kontakta, svjetlinu.

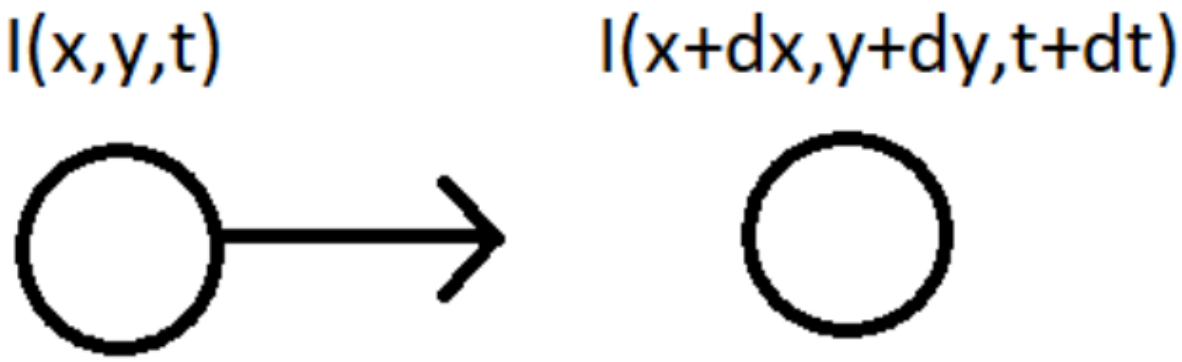
Primjer optičkog toka opisan je kroz scenarij gdje promatrač vozi automobil. Promatrač koji vozi automobil primijetio bi kako se znak sa strane ceste kreće s centra njegovog vidnog polja prema strani. Kada bi imao vidno polje od 360 stupnjeva, taj bi se znak brzo pomicao s njegove strane prema natrag, gdje bi se smanjivao. To kretanje znaka je njegov optički tok. Budući da optički tok ovisi samo o relativnom kretanju, on ostaje isti kada se promatrač kreće, a okolina ostaje mirna, te isto tako kada promatrač miruje, a sva okolina oko njega se kreće. Optički tok je obrazac prividnog kretanja objekata na slikama između dvije uzastopne slike (engl. *frame*) uzrokovan kretanjem objekta ili kamere (Slika 3.3.) [10]. To je 2D vektorsko polje gdje svaki vektor predstavlja vektor pomaka koji pokazuje kretanje točaka sa prve slike na drugu.



Sl.3.3. Obrazac prividnog kretanja objekata na slikama između dvije uzastopne slike

U digitalnoj obradi slike, svaki piksel je povezan s brojčanom vrijednošću koja predstavlja njegovu svjetlinu ili boju. Za slike u nijansama sive (engl. *grayscale*), intenzitet obično predstavlja jednu vrijednost koja se kreće od crne (minimalni intenzitet, često 0) do bijele (maksimalni intenzitet, često 255 u 8-bitnoj slici). Kako bi se što točnije procijenilo koliki je pomak napravljen promatrajući njegovo prividno gibanje na dvije slike, bitno je što veće očuvanje svjetline. Budući da optički tok izračunava procjenu kretanja intenziteta svjetlosti pojedinog piksela između dvije slike, intenzitet svjetla (svjetlina) promatranog piksela bi trebao

biti što konstantniji. Uzimajući u obzir objekt sa intenzitetom *I (x, y, t)*, nakon vremena *dt* on se pomiče za *dx* i*dy,* te je njegov novi intenzitet *I (x+dx, y+dy, t+dt)* (Slika 3.4.).



Sl. 3.4. Pomak objekta sa intenzitetom *I (x, y, t)*, nakon vremena *dt*

Prvo se pretpostavlja da je intenzitet piksela samoga objekta konstantan između dva uzastopna okvira (formula 3-1).

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t) \tag{3-1}$$

Primjenom Taylorove aproksimacije na desnoj strani jednadžbe i uklanjanjem zajedničkih članova dobije se (formula 3-2):

$$\begin{aligned} &I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t + \cdots \\ &=> \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0 \end{aligned} \tag{3-2}$$

Kako bi se dobila jednadžba optičkog toka, cijeli izraz se dijeli sa *dt* (formula 3-3.):

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}\delta v + \frac{\partial I}{\partial t} = 0 \tag{3-3}$$

U gore navedenoj jednadžbi u predstavlja $u = dx/dt$ dok v predstavlja $v = dy/dt$. dI/dx je gradijent slike duž horizontalne osi, dI/dyje gradijent slike duž vertikalne osi a dI/dti vrijeme. Kako bi se riješila ova jednadžba sa dvije nepoznanice (u, v), koriste se različite metode. Odabir metode ovisi o tome hoće li se implementirati rijetki (engl. *sparse*) ili gusti (engl. *dense*) optički tok.

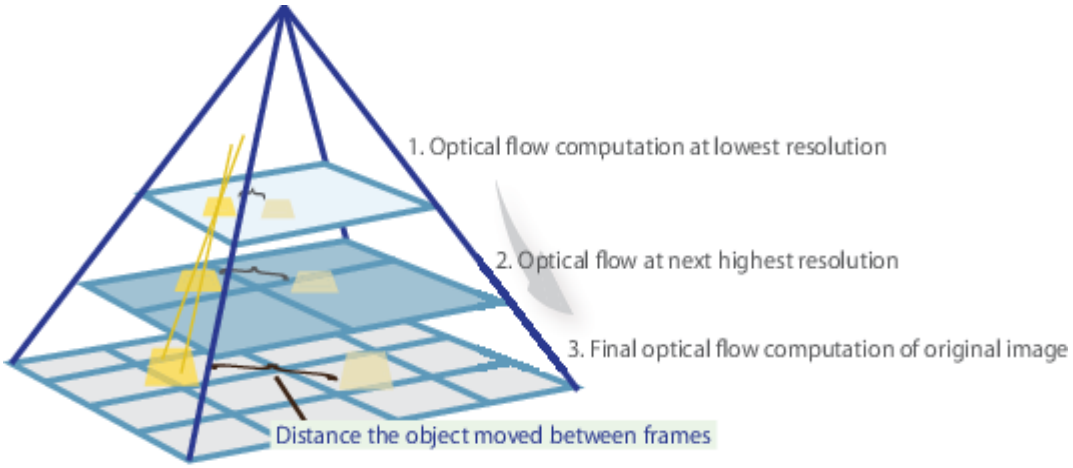
Rijetki optički tok daje vektor protoka samo na određenim točkama ili pikselima u kadru. Ove točke se često biraju unaprijed, i obično su postavljena na mjestima gdje se očekuje da će promjena biti značajna (rubovi i kutovi objekta).

Gusti optički tok daje vektor protoka za svaki piksel u kadru. Omogućava dobivanje detaljnog prikaza kretanja piksela među uzastopnim kadrovima. Gusti optički tok je precizniji u odnosu na rijetki optički tok, ali zahtjeva više računalnih resursa i dulje vrijeme za izračun.

3.2.1. Gunnar Farneback metoda

Kako bi se dobili što točniji izračuni koji prikazuju kretanje piksela u videu, izračun je napravljen za gusti optički tok i to Gunnar Farneback metodom. Gunnar Farneback je švedski istraživač koji je razvio algoritam za izračunavanje optičkog toka između slika (dva uzastopna okvira) i predstavio ga u svom radu: "Procjena gibanja dva okvira na temelju polinomske ekspanzije" (engl. *"Two-Frame Motion Estimation Based on Polynomial Expansion"*) [11]. Farnebackov algoritam koristi tehniku temeljenu na distribuciji intenziteta piksela u okolini kako bi pronašao odgovarajuće točke između dvije slike i izračunao vektor brzine za svaku točku. Algoritam koristi polinom drugog stupnja za aproksimaciju distribucije intenziteta i pronalaženje karakterističnih točaka. Ova metoda radi na način da približava određene detalje ili manje dijelove slike tako da ih modelira pomoću kvadratnih polinoma putem transformacije polinomne ekspanzije. Polinomna ekspanzija transformira slike tako da se koriste polinomi drugog stupnja kako bi se bolje uhvatile promjene intenziteta piksela unutar prozora. Ova transformacija omogućava da se slika izrazi kao zbir polinoma različitih stupnjeva, a točnije, koristi se kvadratna polinomna ekspanzija jer se koriste polinomi drugog stupnja (kvadratni članovi).Zatim se koristi piramidalna struktura kako bi se što točnije izračunala derivacija intenziteta svjetlosti u odnosu na x i y smjerove. Prilikom pomicanja objekta s jednog okvira (engl. *frame*) na sljedeći, kretanja piksela je vrlo malo, te je moguće precizno izračunati derivacije svjetlosnog intenziteta za smjerove x i y. Međutim ako dođe do značajnog pomicanja objekta za nekoliko piksela, te derivacije više nisu točne jer je došlo do nagle promjene intenziteta svjetlosti uslijed većeg pomaka, što za posljedicu ima netočan iznos optičkog toka. Kako bi se izračunao optički tok, originalne slike se smanjuju kako bi se dobila niža rezolucija, a zatim se izračunava optički tok za par slika na nižim razinama piramide (obično se koriste dvije slike). Te dvije slike su uzete iz uzastopnih okvira (engl. *frame*), gdje je jedna slika prethodna, a druga trenutna slika, i na njima se računa optički tok na način da se prate pomaci svjetlosnih intenziteta piksela između te dvije slike (ovo je ključni korak, zato što je daljnja obrada rezultata dobivenih izračunom optičkog toka, rezultirala binarnom slikom, slika 3.8.). Struktura piramide se sastoji od više razina, gdje

najviša razina predstavlja slike sa najmanjom rezolucijom, a najniža razina sliku sa najvišom rezolucijom. Izračun optičkog toka kreće od najviše razine prema najnižoj, jer je udaljenost i pomak između piksela na slikama najmanji. Dobiveni optički tok na višim razinama (slikama sa nižom rezolucijom) koristi se kao inicijalni optički tok prilikom izračuna optičkog toga na sljedećoj nižoj razini (slike sa višom rezolucijom) (Slika 3.5.).



Sl. 3.5. Piramidalna struktura za izračun optičkog toka

Korištenjem piramidalne strukture dobiva se uvid u smjer i brzinu kojom se objekti gibaju između uzastopnih slika, na osnovu pomicanja piksela (promjena intenziteta svjetlosti) [12].

Nakon što se izračuna optičku tok na svim razinama piramidalne strukture, analizira se promjena polinoma prilikom pomaka objekta između dva okvira. Kretanje objekta stvara promjene u polinomu i stvara se specifičan obrazac preko kojega se mogu pratiti koliko se svaka točka pomakla između dvije slike (polje pomaka). Preko polja pomaka, tj. preko skupa uzoraka promjena polinoma može se napraviti precizna procjena stvarnih pomaka piksela između slika.

3.2.2. Optički tok i Farnebackova metoda u radu

U ovom radu, korištena je Farnebackova metoda za izračun optičkog toka, i implementirana je putem OpenCV funkcija, kako bi se mogao detektirati pokret s kamere.

```
//FARNEBACK OPTIC_FLOW:
Image<Gray, float> FlowX = new Image<Gray, float>(grayImage.Size),
    FlowY = new Image<Gray, float>(grayImage.Size),
    FlowResult = new Image<Gray, float>(grayImage.Size);

CvInvoke.CalcOpticalFlowFarneback(prevGrayImage, grayImage, FlowX, FlowY, 0.5, 1, 50, 2, 5, 1.1,
  Emgu.CV.CvEnum.OpticalflowFarnebackFlag.UseInitialFlow);

FlowY = FlowY.Mul(FlowY);
FlowX = FlowX.Mul(FlowX);
FlowResult = FlowX + FlowY;
CvInvoke.Sqrt(FlowResult, FlowResult);
```

Sl. 3.6. Optički tok primjenom Farnebackove metode.

CvInvoke.CalcOpticalFlowFarneback – funkcija koja izračunava optički tok između dviju uzastopnih sivih slika koristeći Farnebackov algoritam (Slika 3.6.).

prevGrayImage - prethodni okvir sivih nijansi. Predstavlja početni okvir koji se uspoređuje s idućim okvirom kako bi se izračunao optički tok.

grayImage - trenutni okvir sivih nijansi. Predstavlja idući (sljedeći) okvir koji se uspoređuje s prethodnim okvirom kako bi se izračunao optički tok.

FlowX i FlowY - Ovo su izlazni nizovi koji sadrže izračunate vektore optičkog toka za x i y smjerove.

pyr_scale - parametar koji kontrolira i određuje razmjeran omjer slike za izgradnju piramidalne strukture slika. Označava koliko će se slika smanjiti u odnosu na svoju originalnu veličinu. Obično je postavljen između 0 i 1. Manja vrijednost stvara kvalitetniju slikovnu piramidu i detektira manje pokrete. Klasična piramida je u mjerilu 0.5, što znači da svaki novi sloj koji se doda je upola manji u odnosu na prethodni.

levels – parameter koji određuje broj razina piramide (uključujući i inicijalnu sliku) koje se koriste za izračun. Ako je parametar postavljen na 1 to znači da se ne stvaraju dodatni slojevi, te se koristi samo originalna slika.

winsize - veličina prozora koja se koristi za procjenu toka na svakoj razini piramide. To je veličina prozora za procjenu toka polja. Što je veća vrijednost, to je algoritam robusniji i otporniji na smetnje, te je u mogućnosti otkriti brze pokrete, ali kao rezultat ima zamućenije polje kretanja.

iterations - broj iteracija korištenih za svaku razinu piramide. Veća vrijednost može dovesti do preciznijeg procjenjivanja toka, ali zahtijeva više računanja.

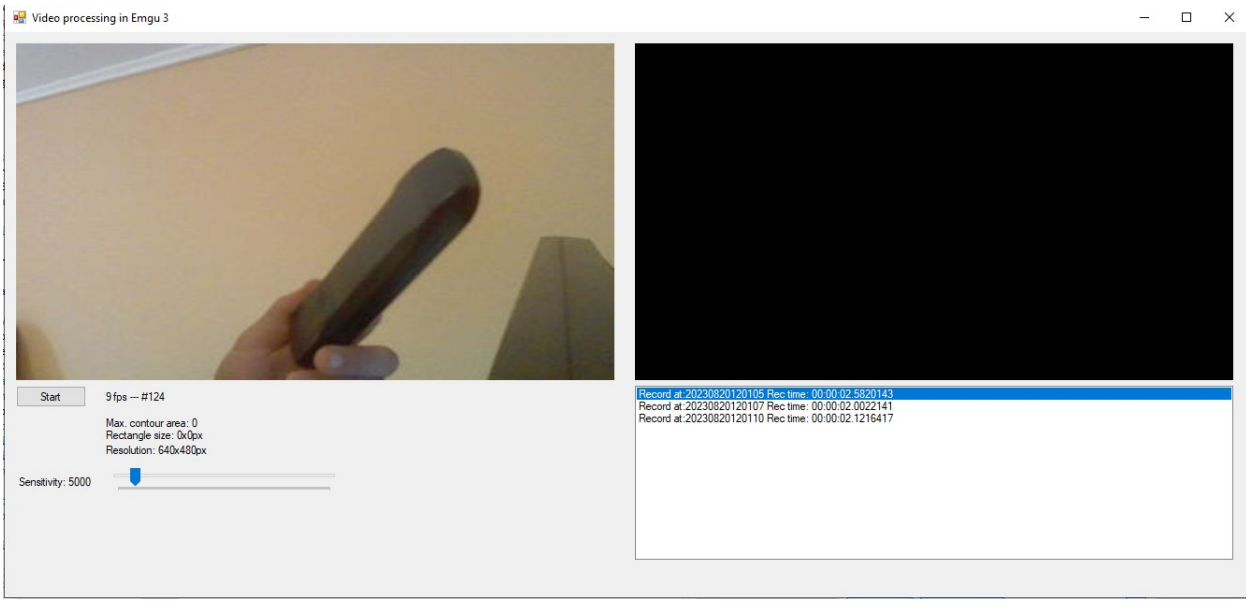
poly_n - parametar koji određuje i dohvaća veličinu susjednih piksela (engl.*size of the pixel neighborhood*) koje se koristi za polinomsku ekspanziju u svakom pikselu za procjenu toka. Veća vrijednost znači da će slika biti približno određena (aproksimirana) sa glađom površinom, što daje robusniji algoritam i zamućenije polje kretanja. Tipična vrijednost poly_n je 5 ili 7.

poly_sigma - standardna devijacija Gaussove funkcije koja se koristi za uglađivanje deriviranih vrijednosti za polinomsku ekspanziju. Ako je poly_n = 5 , onda se definira poly_sigma =1.1, a ako je poly_n = 7 , vrijednost poly_sigma se definira kako poly_sigma = 1.5.

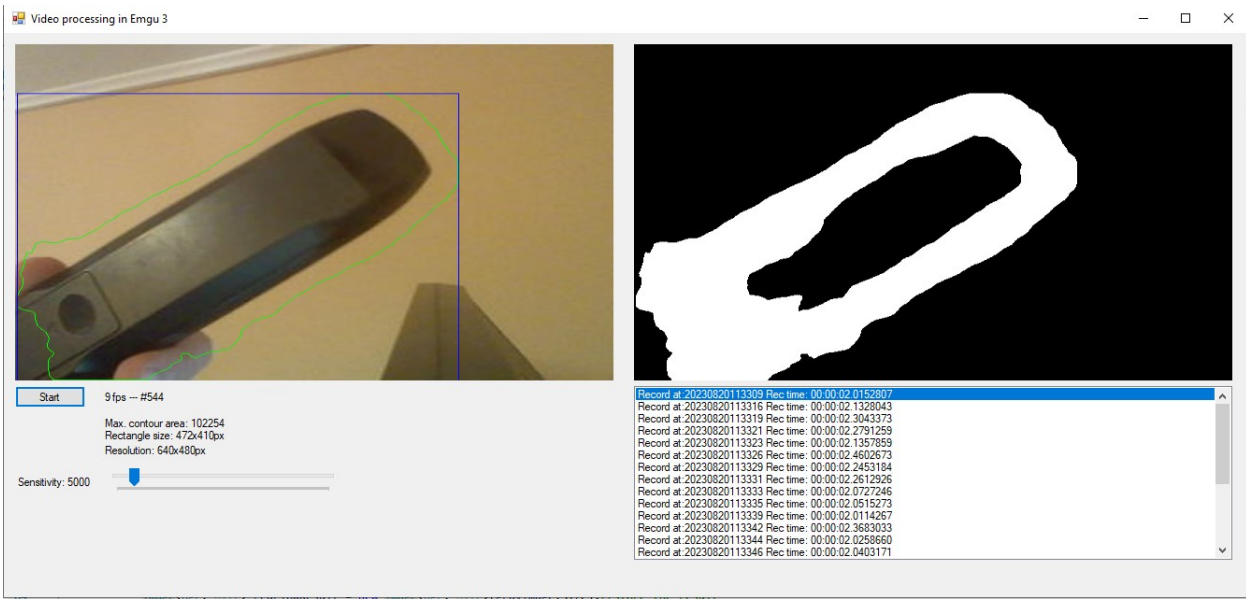
Flags – Zastavice s kojim se kontroliraju (obilježavaju) različiti aspekti algoritma. 'Emgu.CV.CvEnum.OpticalflowFarnebackFlag.UseInitialFlow' označava da se početni tok (dobiven iz drugih metoda, npr. oskudnog optičkog toka) treba koristiti kao početno stanje za algoritam.

3.3. Detekcija pokreta

Kako bi se odredilo je li pokret detektiran, tj. je li došlo do pokreta, potrebno je usporediti dva susjedna okvira (okvira koji dolaze jedan za drugim). Slika se prvo obrađuje metodom *Grayscale*, koja sliku u boji pretvara u crno bijelu sliku (binarna slika), kako bi se što lakše detektirale konture a samim time i pokret. Kod binarne slike, pozadina je u potpunosti crna, a svaki zabilježeni pokret od strane kamere je prikazan kao niz potpuno bijelih piksela omeđenih sa konturom. Kada sustav uspoređuje dvije uzastopne slike (ili dva uzastopna okvira), prva slika na kojoj nema gibanja ili pomaka, biti će u potpunosti crna (Slika 3.7.), dok na drugoj slici, u trenutku pomaka kada dolazi do gibanja, detektirati (ocrtati) će se konture te će prostor unutar konture postati popunjen pikselima bijele boje (Slika 3.8.). Budući da na prvoj slici (okviru) nije bilo bijelih piksela, a na drugoj slici (okviru) postoje bijeli pikseli, sustav će na osnovu te usporedbe zaključiti da je došlo do pokreta, i započeti će snimanje.



Sl. 3.7. Binarizacija - pomak nije detektiran, podloga je potpuno crna.



Sl. 3.8. Binarizacija - podloga je ispunjena bijelim pikselima na mjestu gdje je detektirana kontura.

Nakon što je izvršena obrada slike metodom *Grayscale*, a konture su pronađene, sljedeći korak je odrediti najveću konturu koja predstavlja pokret. S obzirom na to da svaka detektirana kontura ne znači prisutnost pokreta (mogu postojati smetnje i šumovi), potrebno je filtrirati najveću i najrelevantniju konturu, koja će predstavljati pokret i biti označena na izlaznoj slici.

U nastavku je prikazan kod koji pronalazi konture i označava ih određenom bojom (Slika 3.9.). Pronalaženje svih kontura putem metode CvInvoke.FindContours:

```
Emgu.CV.Util.VectorOfVectorOfPoint countours = new Emgu.CV.Util.VectorOfVectorOfPoint();
Mat hier = new Mat();

CvInvoke.FindContours(resultImage, countours, hier, Emgu.CV.CvEnum.RetrType.External,
  Emgu.CV.CvEnum.ChainApproxMethod.ChainApproxSimple);
CvInvoke.DrawContours(colImage, countours, -1, new MCvScalar(0, 0, 255));
```

Sl. 3.9. Pronalazak i označavanje kontura

CvInvoke.FindContours(resultImage, countours, hier, Emgu.CV.CvEnum.RetrType.External, Emgu.CV.CvEnum.ChainApproxMethod.ChainApproxSimple);

- resultImage – izvorna 8-bitna jednokanalna slika
- contours – otkrivene konture, svaka se kontura pohranjuje kao vektor točaka
- hier – vektor koji sadrži informacije o topologiji slike
- Emgu.CV.CvEnum.RetrType.External – način dohvaćanja, External označava da će se dohvatiti samo vanjske konture
- Emgu.CV.CvEnum.ChainApproxMethod.ChainApproxSimple – Aproksimacijska metoda, Simple označava da će prilikom sabijanja vodoravnih, okomitih i dijagonalnih segmenata ostati samo njihove završne točke.

Metoda koja se koristi za pronalaženje kontura na crno-bijeloj slici (resultImage) koja je prethodno obrađena metodom Grayscale. Rezultat metode pohranjuje se u vektor "countours", gdje svaka kontura predstavlja vektor točaka koje čine konturu.

Pronalazak najveće konture (Slika 3.10.):

Kao prvi korak provjerava se postoje li uopće neke konture:

if (countours.Size> 0).

Nakon što su pronađene konture, prolazi se kroz sve konture u petlji kako bi se pronašla najveća kontura.

for (int i = 0; i <countours.Size; i++)

To se postiže usporedbom površina pojedinih kontura pomoću metode *CvInvoke.ContourArea(countours[i])*. Varijabla *maxArea* sadrži površinu trenutno najveće

konture, a *maxContourI* pamti indeks te konture. Na kraju petlje, dobiva se indeks najveće konture koju treba označiti.

```
double maxArea = 0;
int maxContourI = 0;
Rectangle rect = new Rectangle(0, 0, 0, 0);
if (countours.Size > 0)
{
    for (int i = 0; i < countours.Size; i++)
    {
        double area = CvInvoke.ContourArea(countours[i]);
        if (area > maxArea)
        {
            maxArea = area;
            maxContourI = i;
        }

    }
    CvInvoke.DrawContours(colImage, countours, maxContourI, new MCvScalar(0, 255, 0));
    rect = CvInvoke.BoundingRectangle(countours[maxContourI]);
    CvInvoke.Rectangle(colImage, rect, new MCvScalar(255, 0, 0));
}
```

Sl. 3.10. Pronalazak najveće konture

Na osnovu indeksa najveće konture moguće je pozvati metodu pomoću koje će se najveća kontura označiti određenom bojom, u konkretnom primjeru je to zelenom bojom:

CvInvoke.DrawContours(colImage, contours, maxContourI, newMCvScalar(0, 255, 0))

- colImage – slika na kojoj će konture biti ocrtane
- contours – sve ulazne konture, i svaka se pohranjuje kao vektor točke
- maxContourI – parametar koji određuje koja kontura će biti ocrtana. Ukoliko je negativan sve konture će biti prikazane.
- MCvScalar – boje konture.

Najveća kontura se zatim crta na izlaznoj slici (`colImage`) zelenom bojom koristeći metodu *CvInvoke.DrawContours*. Također, određuju se granice najveće konture u obliku pravokutnika (*rect*) putem metode *CvInvoke.BoundingRectangle*, a pravokutnik se crta oko konture plavom bojom pomoću metode *CvInvoke.Rectangle*.

Na taj način, metoda pronalaženja kontura osigurava prepoznavanje najveće konture koja predstavlja pokret, što je ključno za detekciju pokreta na obrađenoj slici i daljnje akcije koje će se poduzeti, poput snimanja ili obrade dodatnih podataka [13].

3.4. Snimanje isječaka

Konture dobivene u prethodnim koracima ključne su za snimanje isječaka, jer, kako je već prije spomenuto, snimanje neće započeti ako nije detektiran pokret.

```
if (maxArea > sensitivity && recording == false)
{
   Video = new VideoWriter(@"temp" + DateTime.Now.ToString("yyyyMMddHHmmss") +
     ".avi", VideoWriter.Fourcc('I', 'Y', 'U', 'V'), 24, new Size(capture.Width,
     capture.Height), true);
    recording = true;
    recTime.Start();
}
if (maxArea > sensitivity && recording == true)
{
    Video.Write(colImageOrig.Mat);
}
else if (maxArea <=sensitivity && recording == true &&
  recTime.ElapsedMilliseconds>2000)
{
    Video.Dispose();
    recording = false;
    recTime.Stop();
    recordingLog.Add("Record at:" + DateTime.Now.ToString("yyyyMMddHHmmss") + "
      Rec time: " + recTime.Elapsed);
    lbInfo.DataSource = null;
    lbInfo.DataSource = recordingLog;

    recTime.Reset();
}
```

Sl. 3.11. Snimanje započinje u trenutku kad je najveća kontura veća od zadane vrijednosti praga.

Kao uvjet se uzima da veličina najveće konture mora biti veća od varijable osjetljivost (engl. *sensitivity*) i da snimanje nije u tijeku (slika 3.11.).

if (maxArea>sensitivity&&recording == false)

Kada se provjeri da je veličina najveće konture (maxArea) veća od vrijednosti *sensitivity* i snimanje trenutno nije u tijeku (recording == false), tada se pokreće snimanje video isječka. U tom trenutku stvara se objekt klase VideoWriter koji je odgovoran za snimanje i zapisivanje slika

u video formatu. Kod stvaranja VideoWriter objekta koriste se specifične informacije kako bi se postavile potrebne opcije snimanja.

MaxArea predstavlja veličinu najveće zabilježene konture, dok je varijabla osjetljivost postavljena na početnu vrijednost od 5000 piksela. Funkcionalnost ove aplikacije omogućuje promjenu osjetljivosti po potrebi na grafičkom sučelju, kako se ona ne bi morala prilagođavati direktno u kodu.

Video = newVideoWriter(@"temp" + DateTime.Now.ToString("yyyyMMddHHmmss") + ".avi", VideoWriter.Fourcc('I', 'Y', 'U', 'V'), 24, newSize(capture.Width, capture.Height), true);

Priprema i stvaranje snimača koristeći specifične informacije:

- VideoWriter – klasa za stvaranje video snimača koji zapisuje slike u video formatu
- (@"temp" + DateTime.Now.ToString("yyyyMMddHHmmss") – ime video datoteke u koju se zapisuje. U gore navedenom slučaju ime se sastoji od riječi temp i datuma kada je zabilježena u formatu godina, mjesec, sat, minuta, sekunda i ekstenzijom avi.
- VideoWriter.Fourcc('I', 'Y', 'U', 'V') – kompresijski kod.To je u stvari kod koji generira 4 znaka kodeka koji se koristi za kompresiju okvira. To je slijed od 4 bajta koji se koriste za jednoznačnu identifikaciju podatkovnih formata.
- 24 – broj sličica u sekundi. (fps -frame rate per second)
- newSize(capture.Width, capture.Height) – veličina frame-a , u ovom slučaju je namješteno da dohvaća širinu i visinu sa kamere.
- True – boolean vrijednost koja označava da je video u boji.

Ako je veličina najveće konture veća od varijable osjetljivosti, dakle praga koji mu je zadan, i snimanje je u tijeku, aplikacija započinje snimanje video zapisa u boji. Snimanje se odvija dokle god je veličina najveće konture veća od zadane vrijednosti praga. Onog trenutka kada veličina konture padne ispod zadanog praga, snimanje se zaustavlja. Dodatno, postavljen je uvjet od 2 sekunde prije zaustavljanja snimanja kako bi se izbjeglo stvaranje puno malih videa iznimno kratkog roka trajanja, što olakšava preglednost i štedi resurse.

U gore navedenom kodu za naziv videa je odabran format datuma, kako bi svaki zapis bio jedinstven i jednoznačan. U primjeru je naziv datoteke definiran kao "temp" + trenutno vrijeme u formatu "yyyyMMddHHmmss". Na taj način će svaki snimljeni video imati različito ime, temeljeno na trenutku snimanja. U prijašnjim pokušajima korištena je numerička vrijednost za naziv videa, no ona se nakon nekog vremena pokazala neefikasnom, jer bi se nakon ponovnog pokretanja aplikacije počeli koristiti ponovno brojevi od 0. To je za posljedicu imalo da novi zapis prebriše stari zapis pod istim imenom.

Na kraju snimanja, video datoteka se zatvara i snimanje se označava kao završeno. Detaljni zapisi o snimanju i vremenu trajanja zapisuju se u log listu (recordingLog), a zatim se prikazuju na grafičkom sučelju kako bi korisnik mogao pratiti informacije o svakom snimanju.

Ovaj cjeloviti postupak omogućuje automatsko snimanje video isječaka samo kada se detektira pokret, čime se štedi prostor za pohranu i olakšava kasnija analiza snimljenih materijala. Također, korištenje vremenskog žiga u imenu datoteke osigurava jedinstvenost svakog snimka i sprječava brisanje postojećih zapisa.

3.5. Grafičko sučelje

Grafičko sučelje pruža korisniku vizualni pregled aplikacije te omogućuje praćenje i kontrolu snimanja video isječaka (slika 3.12).

Sl. 3.12. Prikaz grafičkog sučelja

U gornjem lijevom prozoru prikazuje se ulazna slika u boji dobivena s kamere ili unaprijed

spremljenog videa. Ta slika se prikazuje putem ImageBox klase kojoj je pridružena varijabla colImage, u kojoj su spremljeni dohvaćeni okviri u boji.

ibVideo.Image=colImage;

Nadalje, ta slika se koristi za daljnju obradu, kako bi se dobila binarna slika na kojoj će se tražiti pokret. Binarizacija slike izvodi se pretvaranjem optičkog toka u crno-bijeli prikaz pomoću funkcije:

resultImage = redFlowResult.Convert<Gray, byte>().ThresholdBinary(new Gray(10), new Gray(255));

Dobivena binarna slika se zatim sprema i prikazuje u desnom prozoru pomoću ImageBox klase:

ibProcessedVideo.Image = resultImage;

Metoda *ThresholdBinary* određuje prag osjetljivosti iznad kojeg se intenzitet piksela postavlja na vrijednost 0 (crno) ili 255 (bijelo). U gore navedenom slučaju, prag intenziteta postavljen je na 10. Svaki piksel iznad vrijednosti 10 će poprimiti maksimalnu vrijednost 255 (potpuno bijelo), dok će svaki piksel ispod vrijednosti 10 poprimiti minimalnu vrijednost 0 (potpuno crno). Ovime se stvaraju jasni kontrasti između pokretnih i nepokretnih dijelova slike, što olakšava detekciju pokreta na temelju promjena u intenzitetu piksela.

Na grafičkom sučelju također se nalaze i sljedeće funkcionalnosti:

1. Gumb*Start/Stop*: Omogućuje korisniku pokretanje i zaustavljanje aplikacije. Kada je aplikacija u stanju snimanja, prikazuje se tekst "Stop", a kada je zaustavljena, prikazuje se tekst "Start".

2. Brojač sličica u sekundi (engl. *FPS - Frames Per Second*): Prikazuje broj sličica koji se prikazuju u sekundi. Ovo je korisno za praćenje brzine rada aplikacije i osigurava da snimljeni materijal bude što uglađeniji i bez trzanja.

3. Pokazivač najveće konture (engl. *Max.contourarea*): Prikazuje veličinu najveće detektirane konture na binarnoj slici. Ovo je važan podatak za odlučivanje o snimanju isječka, jer snimanje će započeti samo ako je detektirana kontura veća od određenog praga osjetljivosti.

4. Pokazivač veličine pravokutnog područja (engl. *Rectanglesize*): Prikazuje veličinu pravokutnog područja oko najveće detektirane konture. Ovo je korisno za analizu snimljenih isječaka i može pružiti dodatne informacije o pokretnim objektima na slici.

5. Upravljač osjetljivosti (engl. *Sensitivity*): Omogućuje korisniku podešavanje osjetljivosti detekcije pokreta. Postavljanjem odgovarajuće vrijednosti osjetljivosti može se prilagoditi aplikacija okolini i smanjiti broj lažnih detekcija pokreta ili definirati ispod koje razine ako se pokret dogodi, da se smatra nebitnim.

6. Prozor za ispis videa zabilježenih tijekom snimanja: Ovaj prozor služi za praćenje i pregled snimljenih isječaka. Kada se snima, svaki isječak će biti prikazan u ovom prozoru s vremenskim žigom koji označava vrijeme snimanja.

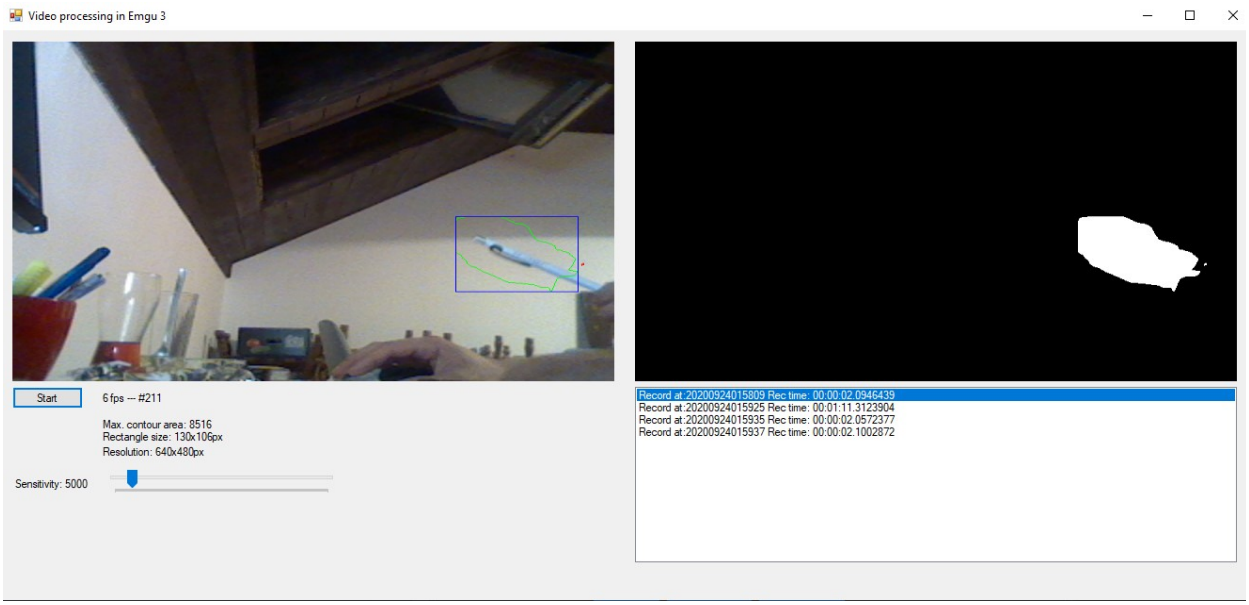
Ovakvo grafičko sučelje omogućuje korisniku jednostavno upravljanje aplikacijom, praćenje detekcije pokreta i pregled snimljenih materijala, čime se olakšava proces analize i nadzora nad područjem koje se prati.

4. TESTIRANJE APLIKACIJE

Cilj testiranja je bio odrediti koliko uspješno će aplikacija detektirati pokrete i isto tako kojom brzinom će to napraviti.

4.1. Uspješnost

Aplikacija je uspješno detektirala različite vrste pokreta, kao što su pomicanje ruke, podizanje objekta i ulazak u prostoriju. Međutim, nisu bili uspješno detektirani objekti koji su padali ili se gibali velikom brzinom. To je uobičajena situacija kod ovakvih aplikacija jer brzi pokreti mogu rezultirati slabo uočljivim ili nejasnim konturama, što otežava njihovu detekciju.

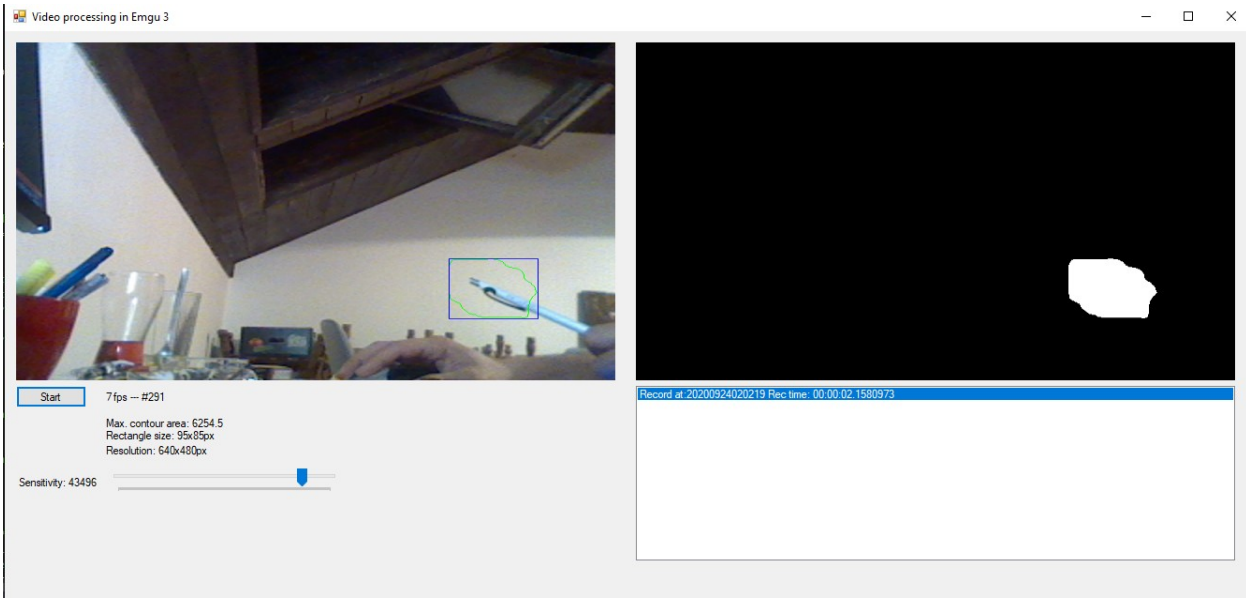


Sl. 4.1. Primjer uspješno detektiranog pokreta.

Na slici 4.1 prikazuje se primjer uspješnog otkrivanja pokreta, gdje je detektirana kontura kemijske olovke veličine 10x1 cm, a snimanje je uspješno započelo jer je veličina konture premašila prag osjetljivosti od 5000 piksela. Podešavanje osjetljivosti praga je bitna stavka koja utječe na snimanje video zapisa. U trenutku kada aplikacija detektira konturu veličine veće od 5000 piksela započinje snimanje, te ukoliko nema pokreta čija je veličina veća od 5000 piksela, snimanje se prekida. Na primjeru sa slike je vidljivo kako je detektirana kontura veličine 8516 piksela, a budući da je ona veća od praga, pokrenuto je snimanje.

Kako bi se istražila pouzdanost detekcije, testirane su različite vrijednosti praga osjetljivosti. Slike 4.1 i 4.2 su snimane u tamnijem okruženju (noćni uvjeti, bez prisustva dnevnog svjetla). Na

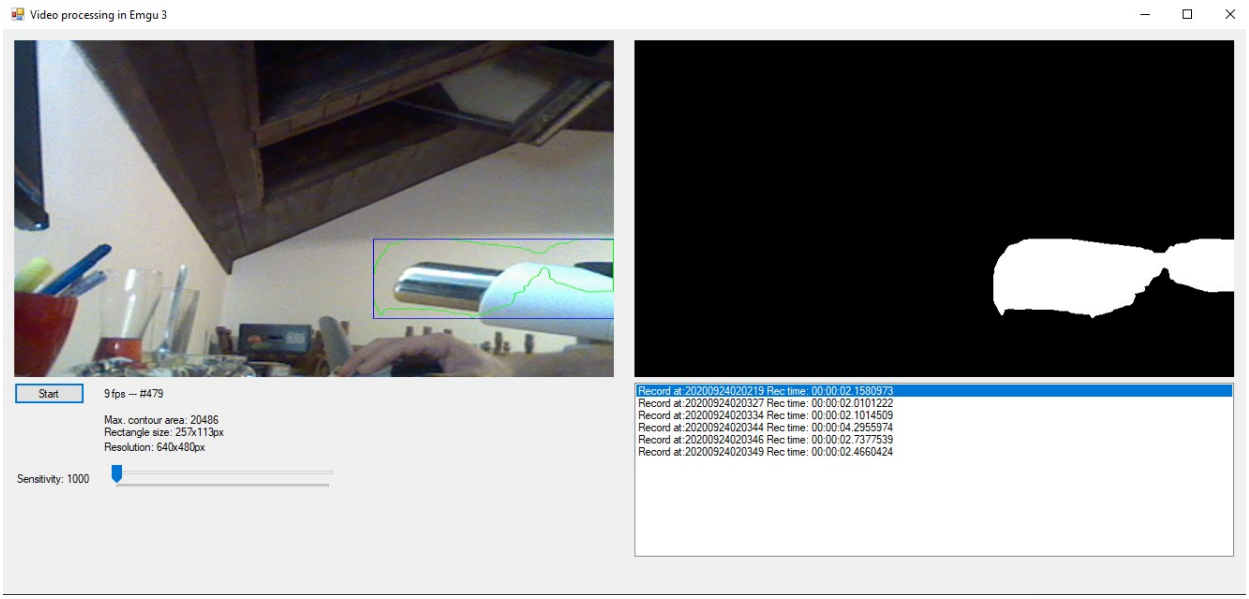
slici 4.2 primijenjen je prag od 43496 piksela, te iako je detektiran pokret i kontura veličine 6454.5 piksela, snimanje nije započelo jer je prag znatno veći od veličine konture.



Sl. 4.2. Postavljanje praga osjetljivosti na visoku vrijednost (43496 piksela).

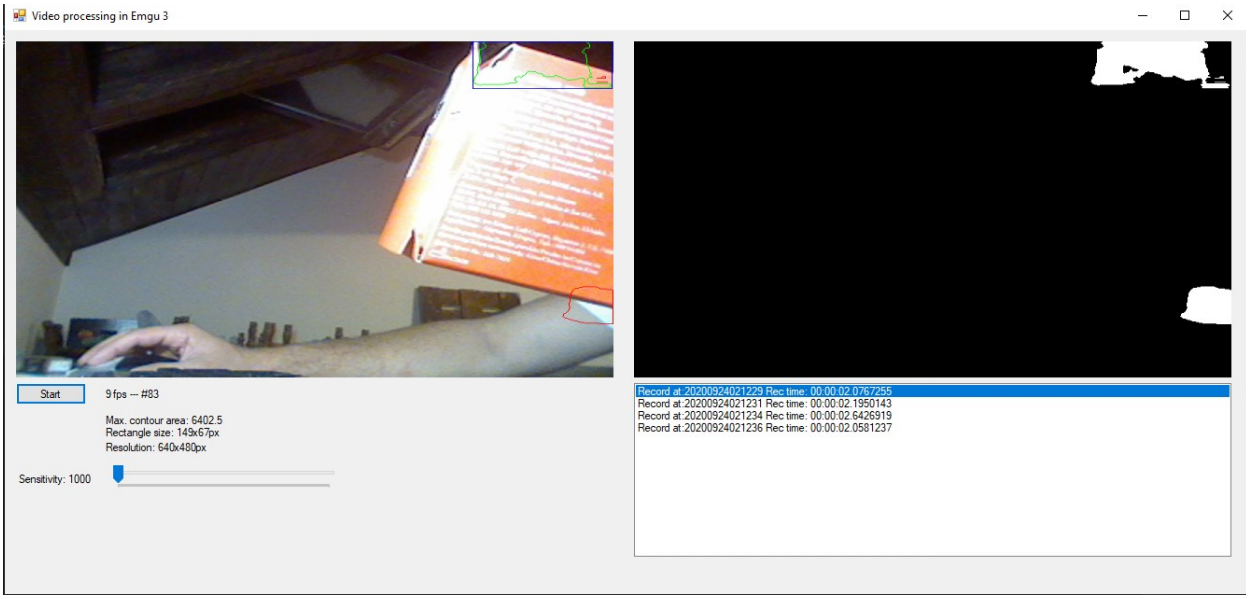
Može se zaključiti da je aplikacija dosta uspješna u tom segmentu. Na već spomenutoj slici 4.2 može se primijetiti da je napravljen određeni video zapis, no on je nastao u nekom ranijem trenutku. Isto tako trebalo bi napomenuti da zbog različitih vanjskih utjecaja , te zbog gibanja predmeta i kvalitete kamere, veličine kontura dosta osciliraju, što je za posljedicu imalo znatno povećan broj pokretanja i zaustavljanja snimanja, te je rezultiralo sa jako puno snimljenih video zapisa (Slika 4.3). Kako bi se korigiralo ovakvo ponašanje, potrebno je za pojedini prostor i kameru podesiti osjetljivost (engl. *sensitivity*) na različite vrijednosti kako bi se zanemarile manje konture nastale zbog šuma ili nekih drugih smetnji.

Slika 4.3 prikazuje testiranje s minimalnim pragom od 1000 piksela, što je omogućilo detekciju svih pokreta čija je veličina prelazila navedenu vrijednost. Međutim, brzi pokreti nisu bili pouzdano detektirani čak ni pri minimalnom pragu. Prilikom testiranja (na određenim) slikama doveden je dodatni izvor svijetlosti no to nije utjecalo na povećanje ili smanjenje šuma ili nekih drugih smetnji, ali je poboljšalo brzinu obrade koja se u prosjeku povećala za 1 fps. U svrhu daljnjeg testiranja, bila bi potrebna kvalitetnija kamera.



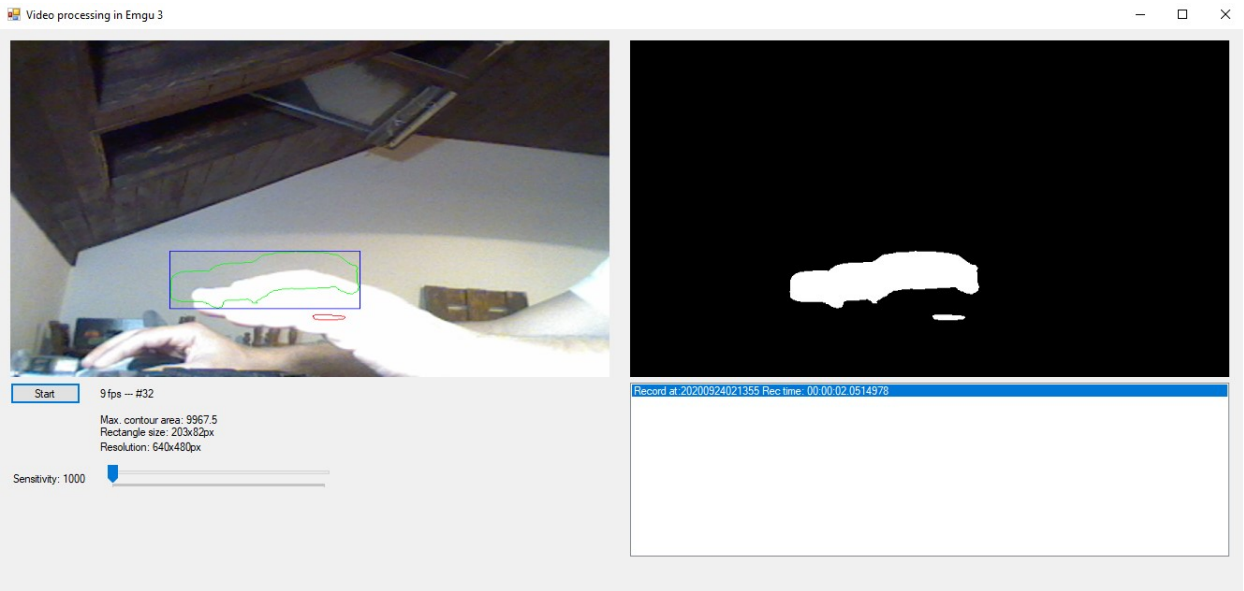
Sl. 4.3. Prag postavljen na 1000 piksela (po noći).

Tijekom testiranja, aplikacija je uspješno zabilježila pokrete na različitim objektima poput kutija (slika 4.4) i pomicanje ruke (slika 4.5). Također, testiranje je provedeno i u uvjetima danjeg svjetla (slike 4.6, 4.7 i 4.8).

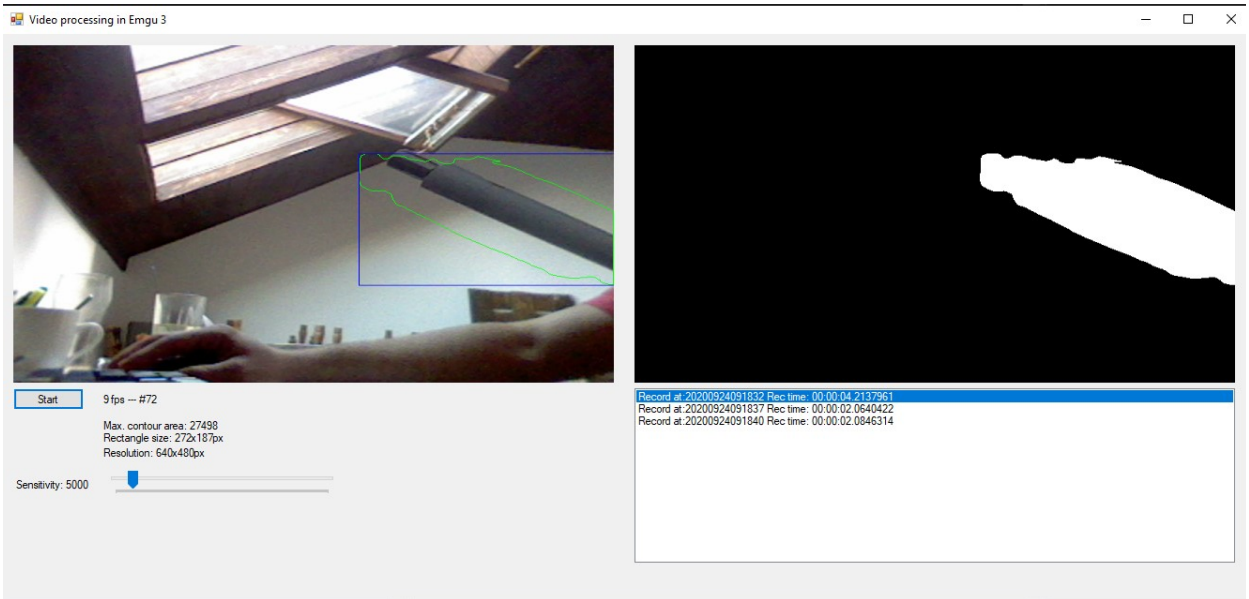


Sl. 4.4. Pomicanje kutije

Na slici 4.6 je prag postavljen na 5000 piksela, a najveća zabilježena kontura bila je veličine 27498 piksela.



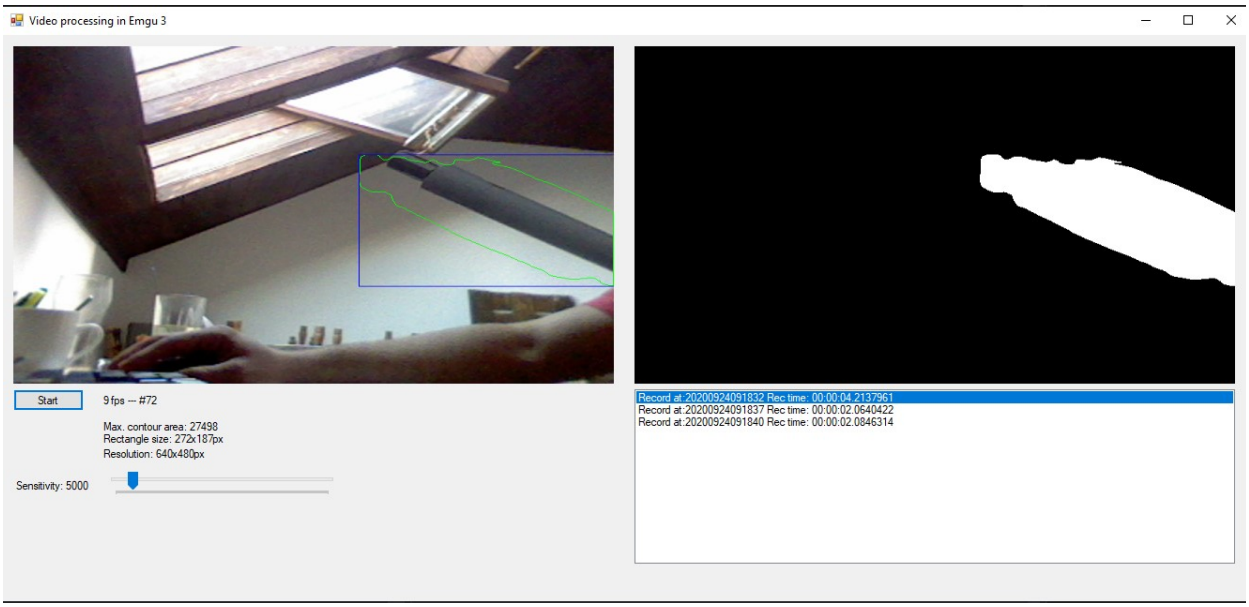
Sl. 4.5. Pomicanje ruke



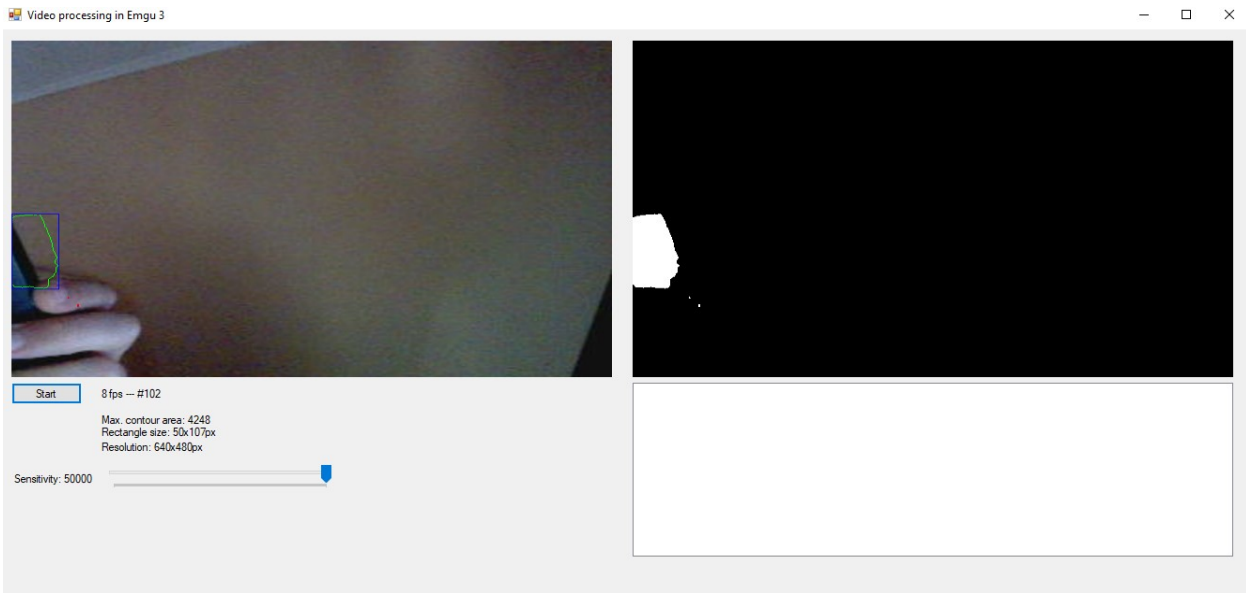
Sl. 4.6. Testiranje aplikacije pri pragu od 5000 piksela.

Na slici 4.7, prag od 1000 piksela omogućio je bilježenje pokreta uz najveću konturu od 7097.5 piksela.

Na slikama 4.8 i 4.9., prag je postavljen na maksimalnih 50000 piksela. Iako je sustav detektirao pokret, nije ga spremio u video zapis jer je veličina konture bila 4284 piksela (slika 4.8), te 19687 piksela (slika 4.9) što je bilo ispod postavljenog praga.



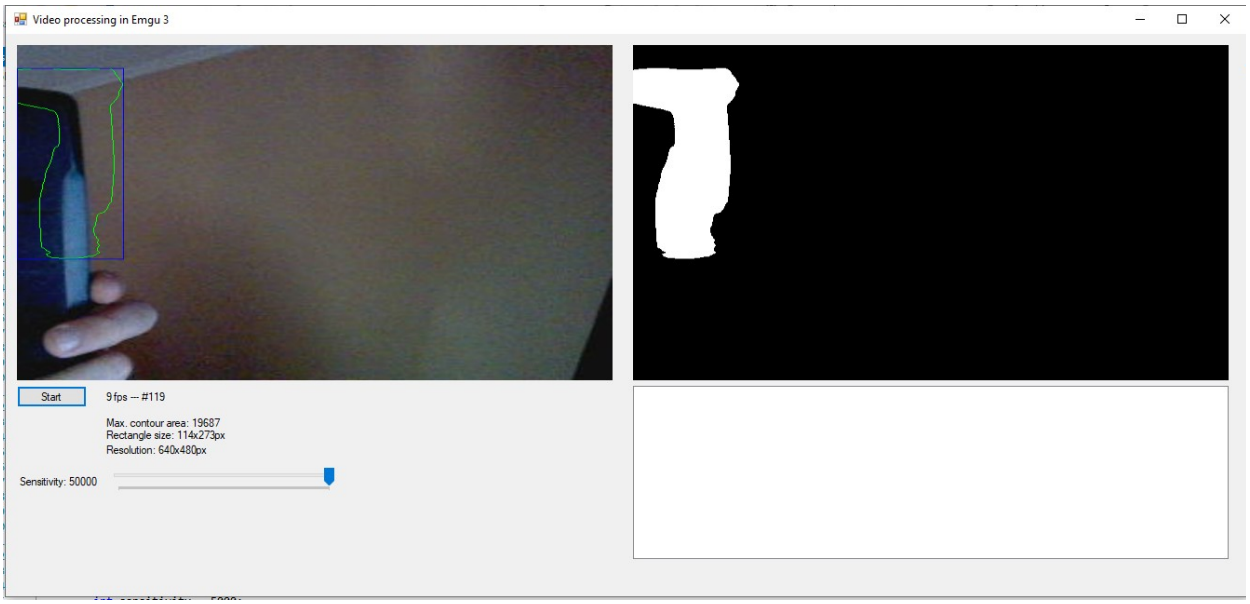
Sl. 4.7. Testiranje aplikacije kada je prag postavljen na 1000 piksela (po danu).



Sl. 4.8. Detekcija pokreta kada je prag postavljen na 50000 piksela (najveća kontura 4284).

Sustav je bio osjetljiv na različite postavke praga osjetljivosti, a brzina obrade se povećala kada je prag postavljen na minimalnu vrijednost. Sličice s prikazima snimanja omogućuju vizualno praćenje uspješnosti detekcije na različitim objektima i uvjetima osvjetljenja.

U zaključku, aplikacija je postigla određenu razinu uspješnosti u detekciji pokreta, no brzi pokreti i kvaliteta kamere mogu utjecati na rezultate. Kontinuirano prilagođavanje pragova i samoga osvjetljenja može poboljšati performanse aplikacije.



Sl. 4.9. Detekcija pokreta kada je prag postavljen na 50000 piksela (najveća kontura 19687)

4.2. Brzina obrade

Brzina obrade u ovom radu ovisi o više čimbenika, uključujući jačinu procesorske jedinice i kvalitetu kamere koja je korištena. Na raspolaganju je bio računalni sustav s Intel I5 2300 procesorom, 8 GB DDR3 memorije i Logitech QuickCam E3500.

Kao što se može vidjeti na gore navedenim slikama (4.1 -4.10), prilikom snimanja videa u realnom vremenu, brzina obrade, odnosno broj sličica u sekundi (fps), nije prelazio 10 bez obzira na uvjete osvjetljenja. Najviša zabilježena brzina bila je 10 fps, dok je najniža iznosila 6 fps. Kako bi se eliminirao vanjski utjecaj na brzinu obrade videa, te kako bi se dobili što točniji podaci, aplikacija je testirana i sa prethodno snimljenim video zapisom, a primjeri su prikazani na slikama 4.11 – 4.15.

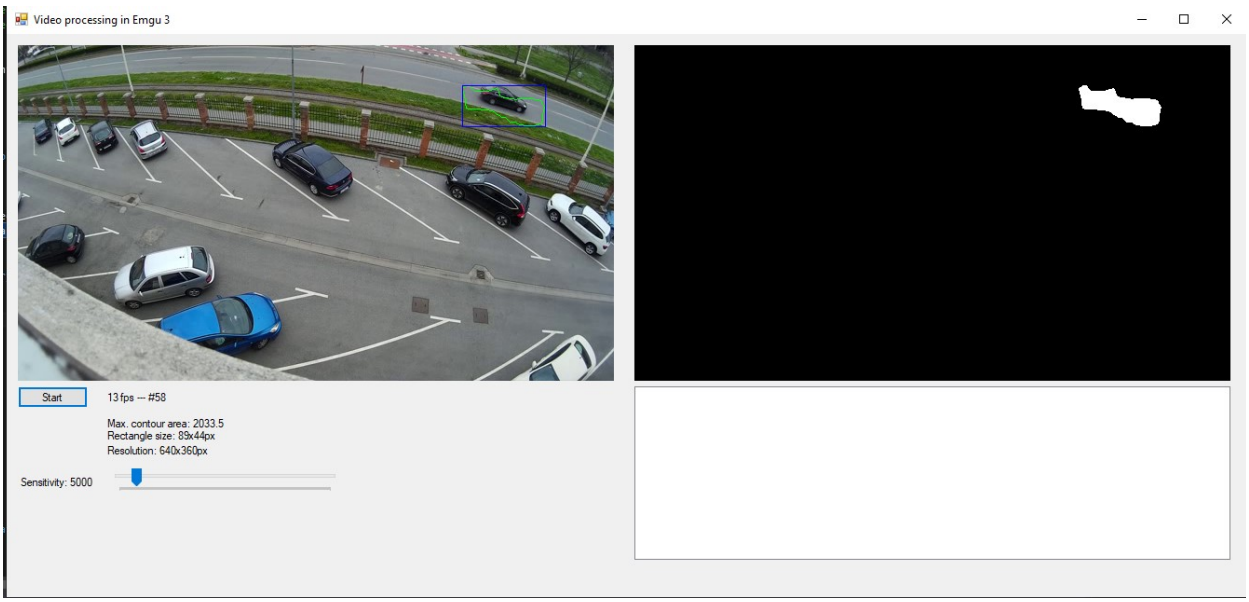
Testiranje je napravljeno na 3 računala i5 2300 [14], i5 3570k [15] i i7 8700k [16], a brzine obrade su vidljive u tablici 4.1.

Uspoređujući rezultate dobivene obradom videa koji je snimljen u realnom vremenu sa rezultatima obrade dobivenih sa unaprijed snimljenim videom, broj sličica u sekundi (fps) je značajno veći kod unaprijed snimljenog videa.

Testiranje je isto tako pokazalo da broj sličica u sekundi (tj. obrada) uvelike ovisi i o procesorskoj moći računala (Tablica 4.1.). Najmanje obrađenih sličica je imalo računalo sa i5 2300 procesorom, dok je najbolji rezultat imalo računalo sa i7 8700k procesorom.

Tablica 4.1.Brzina obrade na različitim računalima.

CPU	Intel i5 2300	Intel i5 3570k	Intel i7 8700k
RAM	8 GB	8 GB	32 GB
KAMERA	Logitech QuickCam E3500	Logitech QuickCam E3500	Logitech c270
FPS	14	17	25



Sl. 4.10. Kontura je niža od razine praga (snimanje nije pokrenuto).

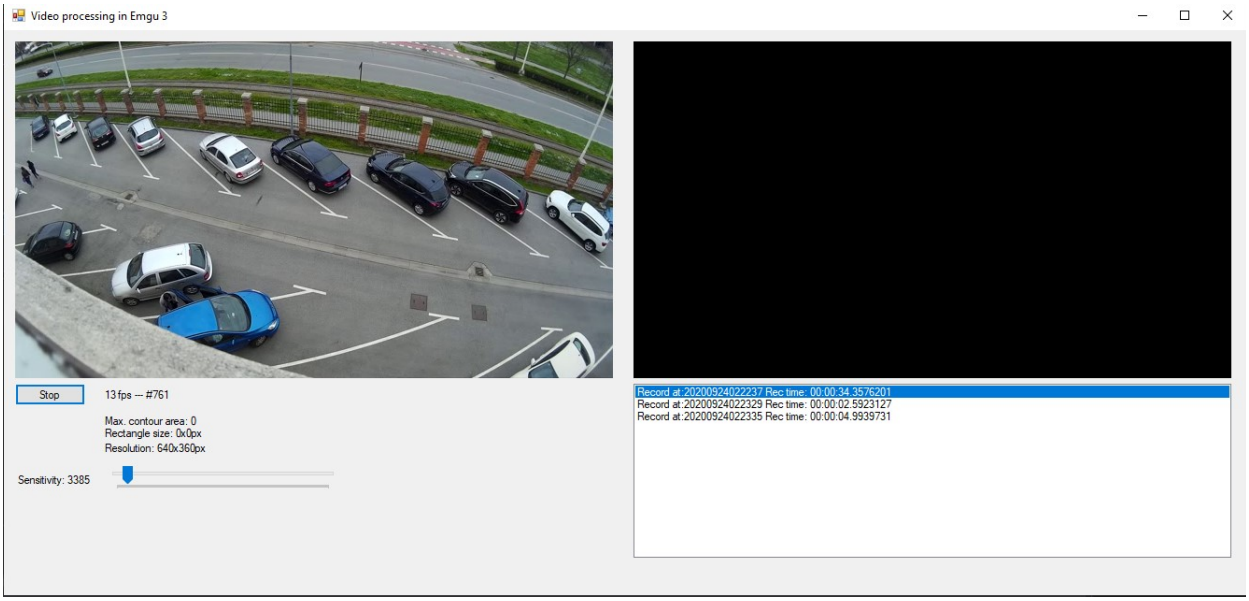
Na slici 4.10, prag osjetljivosti postavljen je na 5000 piksela, ali kontura koja je detektirana bila je veličine 2033.5 piksela, što je ispod praga, pa snimanje nije izvršeno.

Slika 4.11 prikazuje situaciju u kojoj nije detektiran nikakav pokret, što se može dogoditi ako je scena vrlo statična ili ako su pokreti vrlo suptilni i nejasni za detekciju.

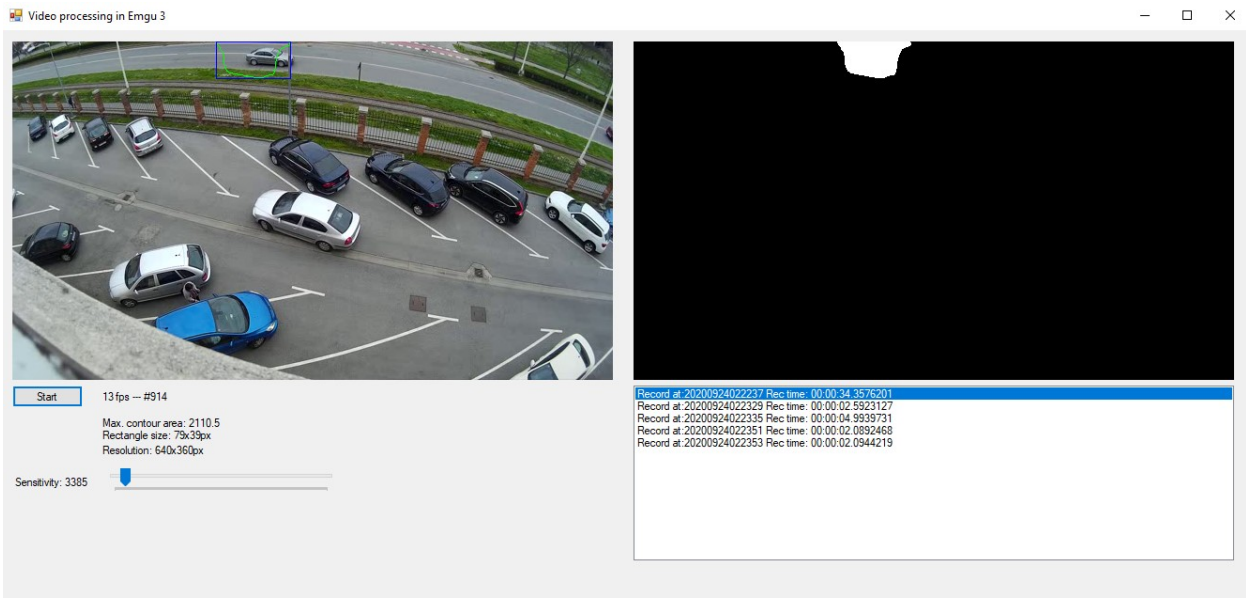
U trenutku kada je zabilježeno značajno gibanje, najveća zabilježena kontura 21105 je bila veća od praga, te je započet proces snimanja (Slika 4.13).

Na slikama 4.12, 4.13 i 4.14, detektirani su i zabilježeni pokreti. Na slici 4.13 prag osjetljivosti je postavljen na minimalnu vrijednost od 1000. Ovdje je postignuta najveća brzina obrade od 14 slika u sekundi te su snimljena 24 video isječka. To je očekivan rezultat budući da je osjetljivost

postavljena na 1000, te ukoliko je u bilo kojem trenutku detektirana kontura koja je bila veća od 1000 (min. prag) započeto je snimanje.

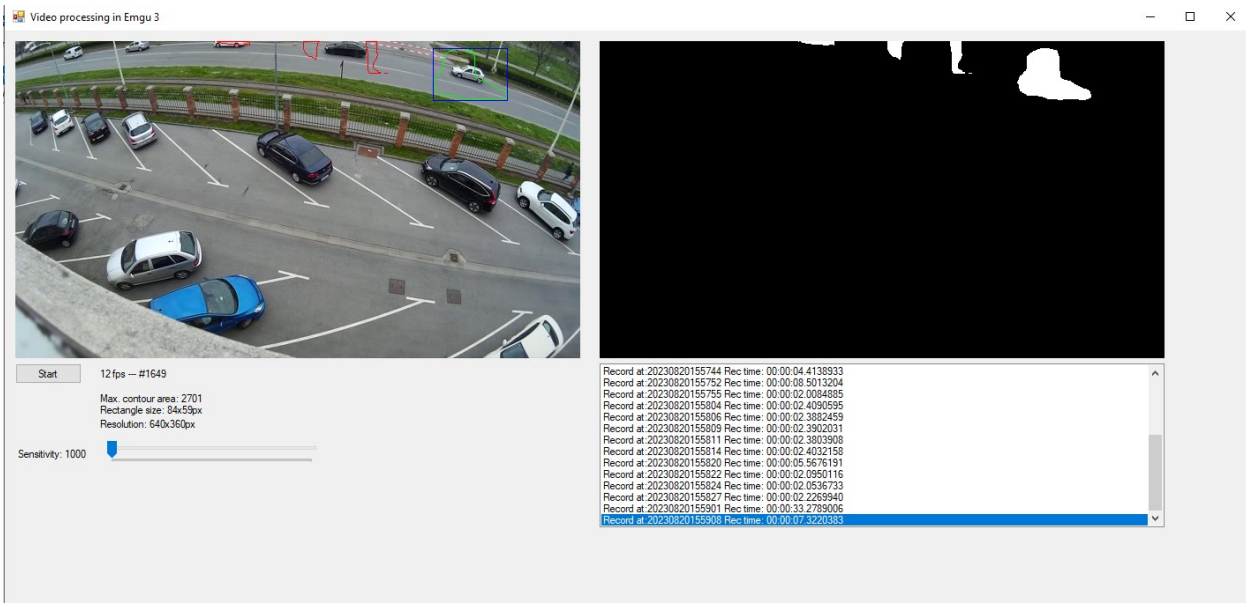


Sl. 4.11. Statična scena(pokret nije detektiran).

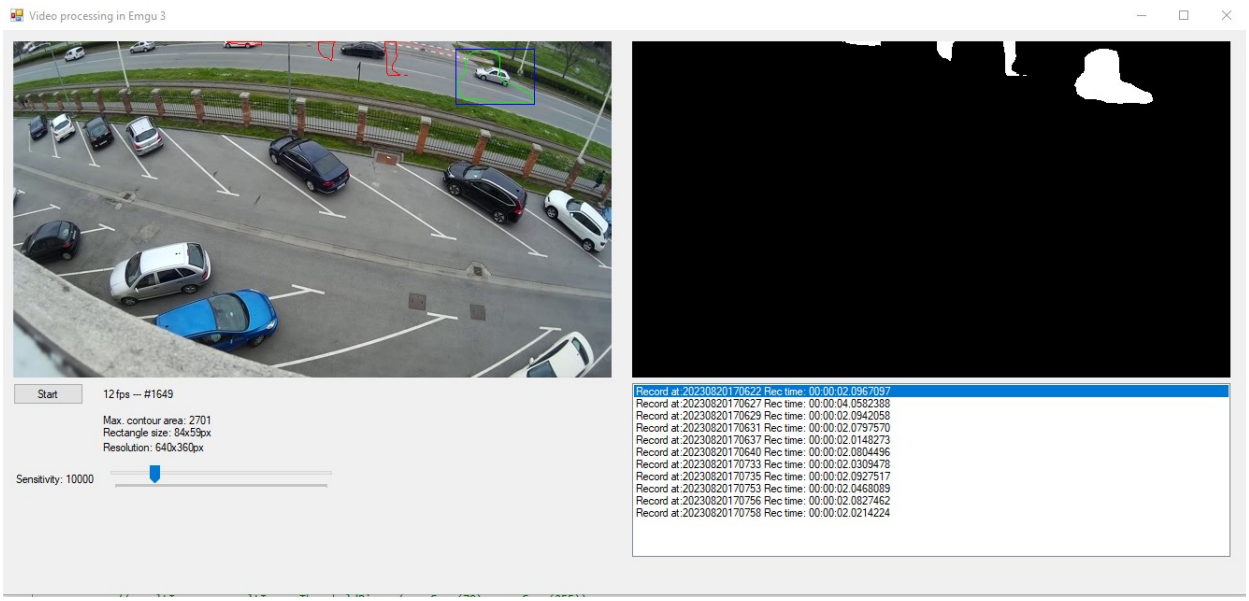


Sl. 4.12. Započet proces snimanja (detektirana kontura je veća od zadanog praga).

Na slici 4.14 prag osjetljivosti je povećan na 10000. Najveća brzina obrade se nije mijenjala i iznosila je 14 slika u sekundi te je snimljeno 10 video isječaka.

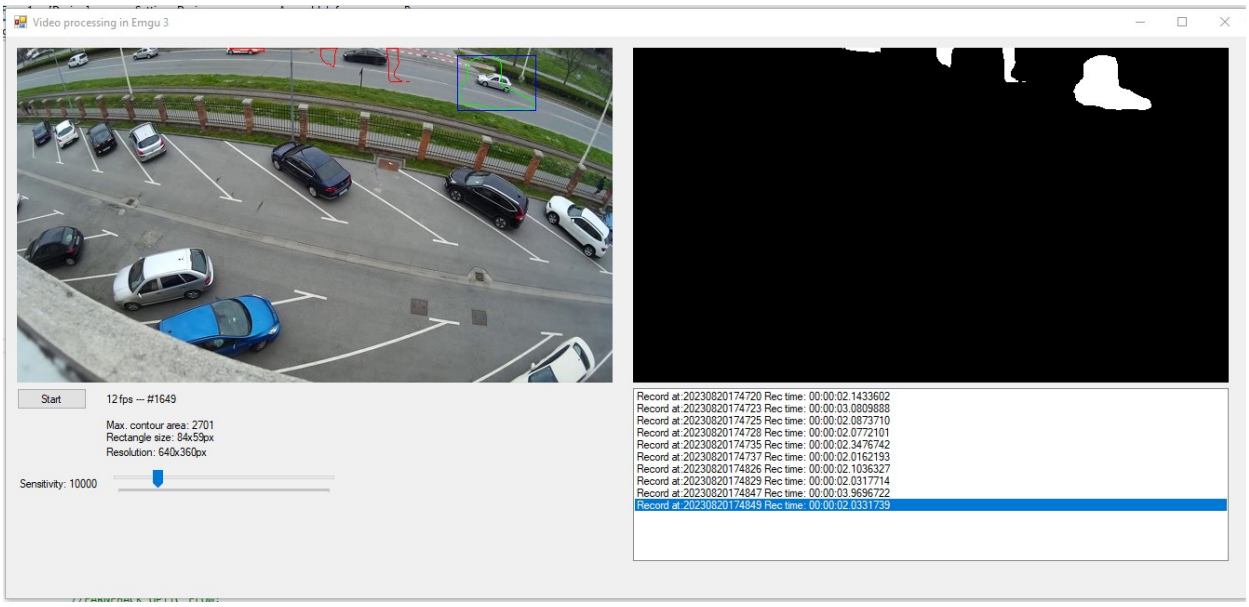


Sl. 4.13. Prag osjetljivosti postavljen na 1000 (veliki broj snimljenih isječaka).

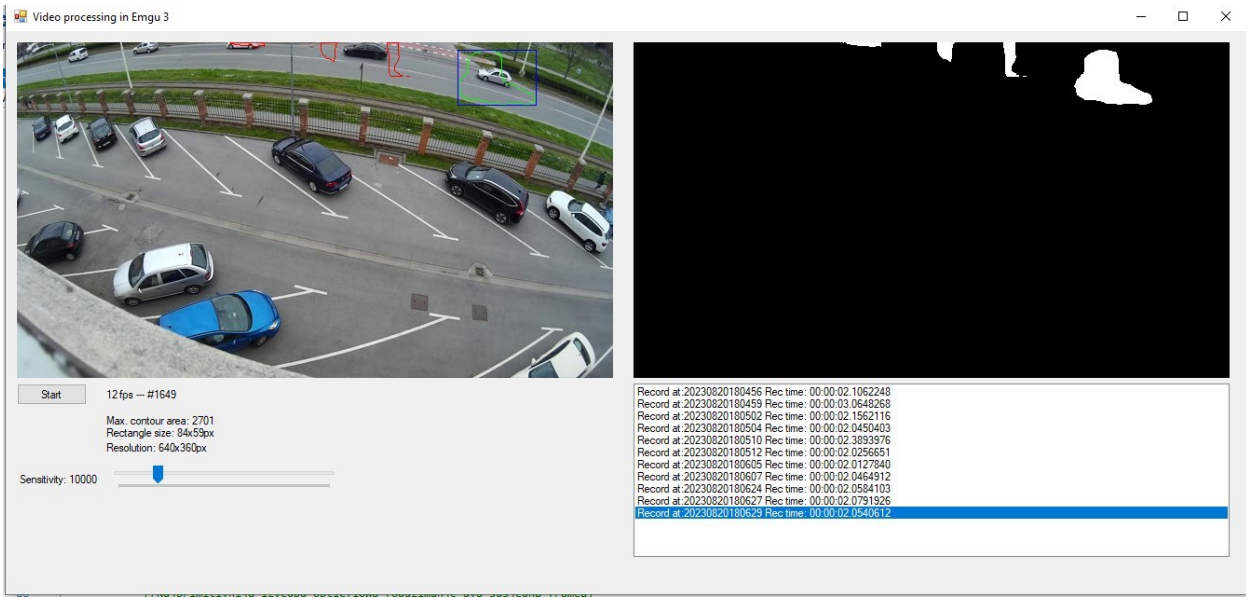


Sl. 4.14. Prag osjetljivosti je postavljen na 10000 (pokreti su detektirani uz manji broj video isječaka).

Promatrajući broj video isječaka snimljenih pri osjetljivosti od 10000 (slike 4.14,4.15 i 4.16) kroz više iteracija može se primijetiti da njihov broj varira i nije konstantan. Budući da je riječ o video isječku koji je unaprijed snimljen, te samim time nema dodatnih vanjskih utjecaja kao što su svjetlost (osvjetljenje prostorije) ili kvaliteta kamere (korišten je isti snimak u svakoj iteraciji), nedvojbeno je da brzina obrade slike ovisi o procesorskoj moći računala.



Sl. 4.15. Prag osjetljivosti je postavljen na 10000 (10 snimljenih video isječaka).



Sl. 4.16. Prag osjetljivosti je postavljen na 10000 (11 snimljenih video isječaka).

Iz navedenih rezultata može se zaključiti da, osim procesorske snage, brzina obrade također ovisi o kvaliteti kamere koja je korištena. Rezultati su bili zadovoljavajući za rad aplikacije i detekciju pokreta, s brzinama od 6 do 9 fps. Ovakva brzina je dovoljna za većinu primjena u kojima je potrebno otkrivanje i snimanje pokreta, ali bi se, u slučaju potrebe, mogla optimizirati primjenom jačih procesorskih jedinica i boljih kamera za još bolje performanse.

5. ZAKLJUČAK

U ovom istraživanju uspješno je razvijena aplikacija za detekciju različitih vrsta pokreta u stvarnom vremenu. Aplikacija je pokazala sposobnost detekcije pomicanja ruke, podizanja objekta te ulaska u prostoriju. Međutim, brzi pokreti i objekti koji padaju ili se gibaju velikom brzinom nisu uvijek pouzdano detektirani, što je uobičajeno za ovakve vrste aplikacija.

Jedan od ključnih faktora uspješnosti aplikacije jest pravilno podešavanje osjetljivosti praga detekcije. Testiranje različitih vrijednosti praga pokazalo je da se aplikacija može prilagoditi različitim uvjetima osvjetljenja i objektima. Također, kvaliteta kamere ima utjecaj na brzinu obrade i uspješnost detekcije.

Brzina obrade je važan faktor za aplikacije ovog tipa. Testiranje je pokazalo da brzina obrade ovisi o procesorskoj snazi računala te kvaliteti kamere. Postignuta brzina od 6 do 9 sličica u sekundi pokazala se zadovoljavajućom za većinu primjena, ali postoji potencijal za daljnje optimizacije primjenom jačih procesorskih jedinica i boljih kamera.

U zaključku, ovaj rad pruža dublji uvid u razvoj i performanse aplikacije za detekciju pokreta. Iako postoje izazovi u detekciji brzih pokreta i objekata u određenim uvjetima, aplikacija je postigla zadovoljavajuće rezultate. Daljnje istraživanje moglo bi se usmjeriti prema optimizaciji algoritama detekcije, testiranju na različitim hardverskim konfiguracijama te implementaciji naprednih tehnika za bolju detekciju i praćenje pokreta.

LITERATURA

[1] T. text provides general information S. assumes no liability for the information given being complete or correct D., to varying update cycles, S. C. D. M. up-to-D. D. T. R., in the Text, „Topic: Smartphones“ [online]. Dostupno na: https://www.statista.com/topics/840/smartphones/. [Pristupljeno: 15.9.2023.].

[2] J., Kim, J., Koh, J. W., Choi, „Video Object Detection Using Motion Context and Feature Aggregation“, u *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, str. 269–272, 2020.

[3] S., Parveen, J., Shah, „A Motion Detection System in Python and Opencv“, u *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, str. 1378–1382, 2021.

[4] K., Takaya, „Detection and Segmentation of Moving Objects in Video“, u *2006 Canadian Conference on Electrical and Computer Engineering*, str. 2069–2073, 2006.

[5] „Security Camera CZ - Apps on Google Play“ [online]. Dostupno na: https://play.google.com/store/apps/details?id=cz.scamera.securitycamera&hl=en_US. [Pristupljeno: 19.9.2023.].

[6] „Video Surveillance Ivideon - Apps on Google Play“ [online]. Dostupno na: https://play.google.com/store/apps/details?id=com.ivideon.client&hl=en_US. [Pristupljeno: 19.9.2023.].

[7] „VicoHome: Security Camera App - Apps on Google Play“ [online]. Dostupno na: https://play.google.com/store/apps/details?id=com.smartaddx.vicohome&hl=en_US. [Pristupljeno: 19.9.2023.].

[8] „C Sharp (programming language)“, *Wikipedia*. 14-ruj-2023.

[9] „Emgu CV: OpenCV in .NET (C#, VB, C++ and more)“ [online]. Dostupno na: https://emgu.com/wiki/index.php/Main_Page. [Pristupljeno: 15.9.2023.].

[10] „Optical flow“, *Wikipedia*. 29-tra-2023.

[11] G., Farnebäck, „Two-Frame Motion Estimation Based on Polynomial Expansion“, u *Image Analysis*, sv. 2749, J. Bigun i T. Gustavsson, Ur. Springer Berlin Heidelberg: Berlin, Heidelberg, 2003, str. 363–370.

[12] „Introduction to Motion Estimation with Optical Flow“ [online], 24-tra-2019. Dostupno na: https://nanonets.com/blog/optical-flow/. [Pristupljeno: 15.9.2023.].

[13] P. A. S., Mendes, M., Mendes, A. P., Coimbra, M. M., Crisostomo, „Movement Detection and Moving Object Distinction Based on Optical Flow“, 2019.

[14] „Intel® Core™ i5-2300 Processor (6M Cache, up to 3.10 GHz) - Product Specifications“ [online]. Dostupno na: https://www.intel.com/content/www/us/en/products/sku/52206/intel-core-i52300-processor-6m-cache-up-to-3-10-ghz/specifications.html. [Pristupljeno: 15.9.2023.].

[15] „Intel® Core™ i5-3570K Processor (6M Cache, up to 3.80 GHz) - Product Specifications“ [online]. Dostupno na: https://www.intel.com/content/www/us/en/products/sku/65520/intel-core-i53570k-processor-6m-cache-up-to-3-80-ghz/specifications.html. [Pristupljeno: 15.9.2023.].

[16] „Intel® Core™ i7-8700K Processor (12M Cache, up to 4.70 GHz) - Product Specifications“ [online]. Dostupno na: https://www.intel.com/content/www/us/en/products/sku/126684/intel-core-i78700k-processor-12m-cache-up-to-4-70-ghz/specifications.html. [Pristupljeno: 15.9.2023.].

SAŽETAK

Cilj ovoga rada je bio istražiti je li moguće napraviti Windows aplikaciju koja će uspješno detektirati pokrete i na odgovarajući način ih spremiti. U radu su korišteni C# programski jezik i EmguCV .Net omotač (omotač u OpenCV biblioteci za obradu slike). Rad ove aplikacije uvjetovan je nizom međusobno povezanih koraka: učitavanje videa, izračun optičkog toka Farneback metodom, detekcijom pokreta i u konačnici snimanjem video isječaka. Aplikacija je uspješno detektirala pokrete i spremila ih lokalno u obliku video zapisa. Ovim radom je također potvrđeno, da je za rad ovakve vrste aplikacije dovoljno starije računalo (i5 2300) uz najosnovniju kameru koja je dostupna u svakoj trgovini informatičke opreme.

Ključne riječi: C#, EmguCV, Farneback metoda, nadzorna kamera, optički tok

ABSTRACT

Windows Application for Surveillance Camera

The aim of this work was to explore whether it is possible to create a Windows application that can successfully detect movements and save them in an appropriate manner. The work used the C# programming language and the EmguCV .Net wrapper (a wrapper in the OpenCV library for image processing). The development of this application involved a series of interconnected steps: loading the video, calculating optical flow using the Farneback method, motion detection, and ultimately recording video clips. The application successfully detected movements and saved them locally in the form of video recordings. This work also confirmed that for the operation of such an application, an older computer (i5 2300) with the most basic camera available in any computer hardware store is sufficient.

Keywords: C#, EmguCV, Farneback method, optical flow, surveillance camera

ŽIVOTOPIS

Marin Kolar rođen je 01.04.1987. godine u Osijeku. Osnovnoškolsko obrazovanje stječe u Osnovnoj školi Mladost. Godine 2006. završava Isusovačku klasičnu gimnaziju s pravom javnosti u Osijeku i upisuje Elektrotehnički fakultet, preddiplomski studij računarstva u Osijeku. Preddiplomski studij završava 2011. godine i stječe akademski naziv sveučilišni prvostupnik. Iste te godine upisuje nastavak obrazovanja, diplomski studij računarstva u Osijeku. Od 2014. do 2018. godine radi u Transcomu, a od 2019. do 2023. godine u TTTech-u.

_____

Potpis autora