VISOKA TEHNIČKA ŠKOLA U BJELOVARU
STRUČNI STUDIJ MEHATRONIKE

Upravljanje obrambenom kupolom pomoću Arduino razvojnog okruženja

Završni rad br. 16/MEH/2017

Antonio Nikolić

Bjelovar, rujan 2017.

VISOKA TEHNIČKA ŠKOLA U BJELOVARU
STRUČNI STUDIJ MEHATRONIKE

Upravljanje obrambenom kupolom pomoću Arduino razvojnog okruženja

Završni rad br. 16/MEH/2017

Antonio Nikolić

Bjelovar, rujan 2017.

obrazac ZR - 001

Visoka tehnička škola u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Kandidat: **Nikolić Antonio** Datum: 19.06.2017. Matični broj: 000583

JMBAG: 0069055895

Kolegij: **MIKRORAČUNALA**

Naslov rada (tema): **Upravljanje obrambenom kupolom pomoću Arduino razvojnog okruženja**

Područje: **Tehničke znanosti** Polje: **Temeljne tehničke znanosti**

Grana: **Automatika**

Mentor: **Zoran Vrhovski, mag.ing.el.techn.inf.** zvanje: **viši predavač**

Članovi Povjerenstva za završni rad:

1. **dr.sc. Alan Mutka, predsjednik**
2. **Zoran Vrhovski, mag.ing.el.techn.inf., mentor**
3. **Tomislav Pavlic, mag.ing.mech., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 16/MEH/2017

U radu je potrebno:
1. opisati i izraditi obrambenu kupolu
2. nacrtati i simulirati model obrambene kupole u SolidWorks programskom alatu
3. izraditi i opisati podsustav za detektiranje pokreta mete
4. opisati i izraditi sustav upravljanja obrambenom kupolom koji se zasniva na Arduino razvojnom okruženju

Zadatak uručen: 19.06.2017. Mentor: **Zoran Vrhovski, mag.ing.el.techn.inf.**

Sadržaj

1 UVOD

U ovome radu biti će prikazana implementacija mehatroničkog sustava u svrhu unapređenja obrambenih sustava za obranu na moru i kopnu. Rad pokriva više područja koja spadaju pod mehatroniku, a to su: mikrokontroleri, računalnom potpomognuti dizajn, automatizacija i senzorika. Model automatske obrambene kupole izrađen je u programskom alatu*SolidWorks*, programu za izradu 3D modela. Pomoću razvojnog okruženja *Arduino Uno R3*i prijenosnog računala izvršava se automatizacija obrambene kupole. Za programiranje razvojnog okruženja*Arduino Uno R3* koristi se programsko okruženje *Arduino IDE*, a za upravljački program navođenja kupole na metu koristi se programsko okruženje *Processing*. Senzor pokreta je web kamera spojena putem USB priključka na prijenosno računalo. Upravljački program pretvara informacije dobivene s web kamere u naredbe razumljive razvojnom okruženju *Arduino,* te tako kontrolira pokrete servo motora. U radu su korišteni servo motori *TowerPro MG996*. Materijali korišteni za konstrukciju kupole su iverica i puno drvo. Zupčanici i osovine korišteni za izradu ovog rada su dijelovi neispravnih printera i skenera.

Drugo poglavlje sadrži povjesni razvoj obrambenih kupola. U trećem poglavlju rada navedeni su i opisani svi dijelovi koji su korišteni za izradu automatske obrambene kupole te kako ona mehanički funkcionira. Četvrto poglavlje ovog rada opisuje *Arduino* razvojno okruženje i programski kod automatizacije obrambene kupole, te programski kod pisan u Processing softveru. Nadalje se u ovom poglavlju opisuje izrada 3D modela obrambene kupole u programskom alatu za 3D modeliranje *SolidWorks*. Kratki zaključak dan je u poglavlju pet.

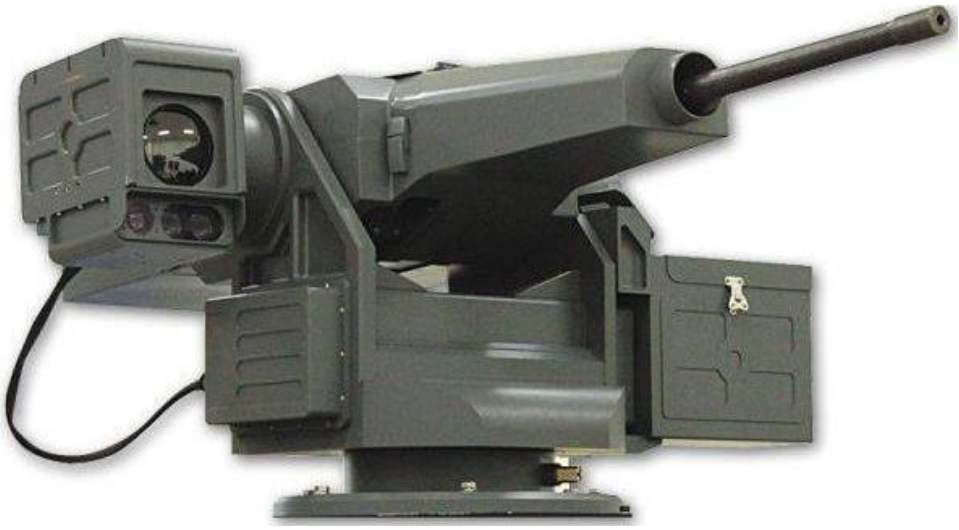
2 Povijesni razvoj obrambenih kupola

Obrambena kupola je oružje koje se automatski usmjerava i otvara vatru na pokretne ciljeve koje su otkrili senzori. Najranija funkcionalna vojna obrambena kupola bila je oružani sustav koji je vrlo sličan oružju za otkrivanje i uništavanje dolaznih raketa kratkog dometa i neprijateljskih zrakoplova. Koristili su se isključivo za pomorske bitke, a sada se koriste i kao obrambeni sustavi na kopnu.Jedna od prvih zemalja koja je počela koristiti automatske obrambene kupole je Južna Korea. *Samsung SGR-A1* (slika2.1) je vrsta kupole koju su zajednički razbili *Samsung Techwin* (sada *Hanwha Techwin*) i Sveučilište Korea kako bi pomogli južnokorejskim postrojbama u korejskoj demilitariziranoj zoni. Ovaj model kupole se ujedno smatra i najstarijim komercijaliziranim robotom s autonomnim sposobnostima i prvom takvom jedinicom koja ima integrirani sustav koji uključuje nadzor, praćenje, pucanje i prepoznavanje glasa. Broj ovakvih kupola koji je postavljen u zonu je nepoznat iz razloga što je projekt označen kao „visoko povjerljiv“[1, 2].

Slika 2.1 SGR-A1[1]

Godine 2007. izraelska vojska razmještala je sustav automatskih obrambenih kupola duž ograde granice u Gazi smještenih na posebno postolje u razmacima od nekoliko stotina metara. Na automatske kupole montirane su strojnice .50 kalibra M2. Komunikacija između automatske kupole i kontrolnog centra ostvarena je putem svjetlovodne opreme. Ove automatske kupole služe kao vrsta robotskog snajpera koji pokriva zonu od 1.500 metara. Obrambena kupola se temelji na *Samson* tehnologiji za daljinsko upravljanje obrambenim kupolama[1, 2].

U prosincu 2010. južnokorejska tvrtka Dodam predstavila je *Super Aegis II* (slika 2.2), automatiziranu kupolu naoružanja na tornju koja koristi toplinsku sliku za detekciju vozila i ljudi do 3 km udaljenosti. Može funkcionirati tokom noći i bez obzira na vremenske uvjete. Sustav daje usmeno upozorenje prije početka pucanja. Iako je sposoban automatski pucati, kupola je konfigurirana na način da zahtijeva potvrdu od strane čovjeka. Ova kupola se koristi u raznim objektima u Ujedinjenim Arapskim Emiratima, Abu Dhabiju i Kataru te u korejskoj demilitariziranoj zoni [2, 3].



Slika 2.2 Super Aegis II [3]

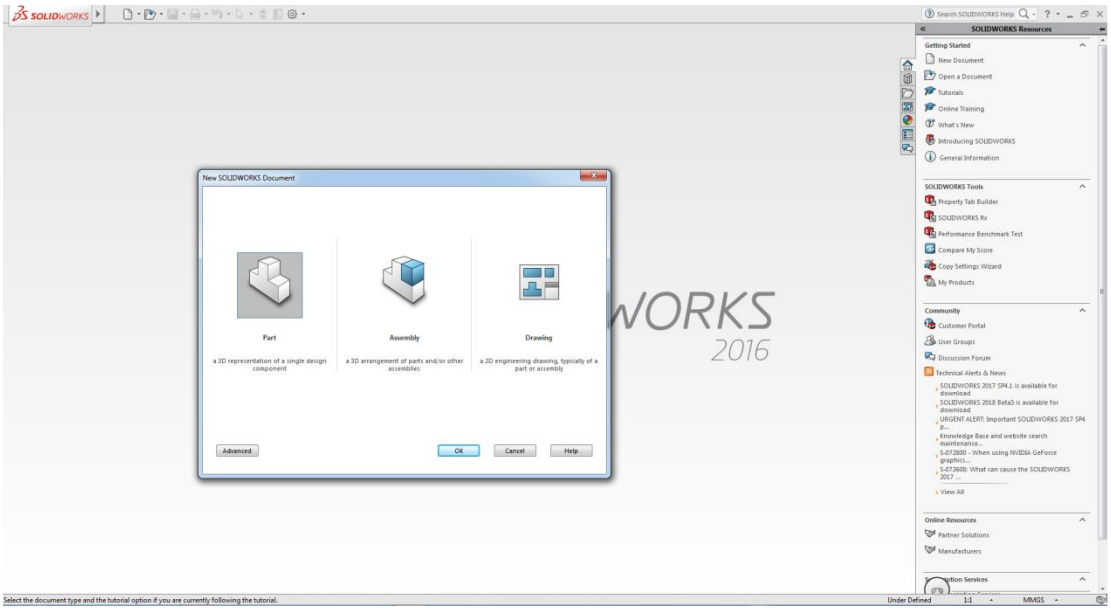
3 Modeliranje i izrada baze obrambenekupole

3.1 Kreiranje modela kupole

SolidWorks je programski alat za modeliranje 3D modela dijelova ili cijelih sklopova. Koristi parametarski pristup značajkama za stvaranje modela i sklopova. Ovaj programski alat napisan je na jezgri*Parasolid*. Parametri koji se koriste za modele odnose se na ograničenja čije vrijednosti određuju oblik ili geometriju modela ili sklopa. Parametri mogu biti numerički: duljine linija ili promjera kruga te geometrijski: tangentni, paralelni, koncentrični, horizontalni ili vertikalni. Numerički parametri se mogu povezati korištenjem međusobnih veza kako bi dizajnirani dio/sklop dobio namjenu i tako reagirao na promjene i ažuriranja (npr. rupa na limenci ili boci pića ostane uvijek na vrhu površine bez obzira na promjenu visine ili širine limene ili boce) .

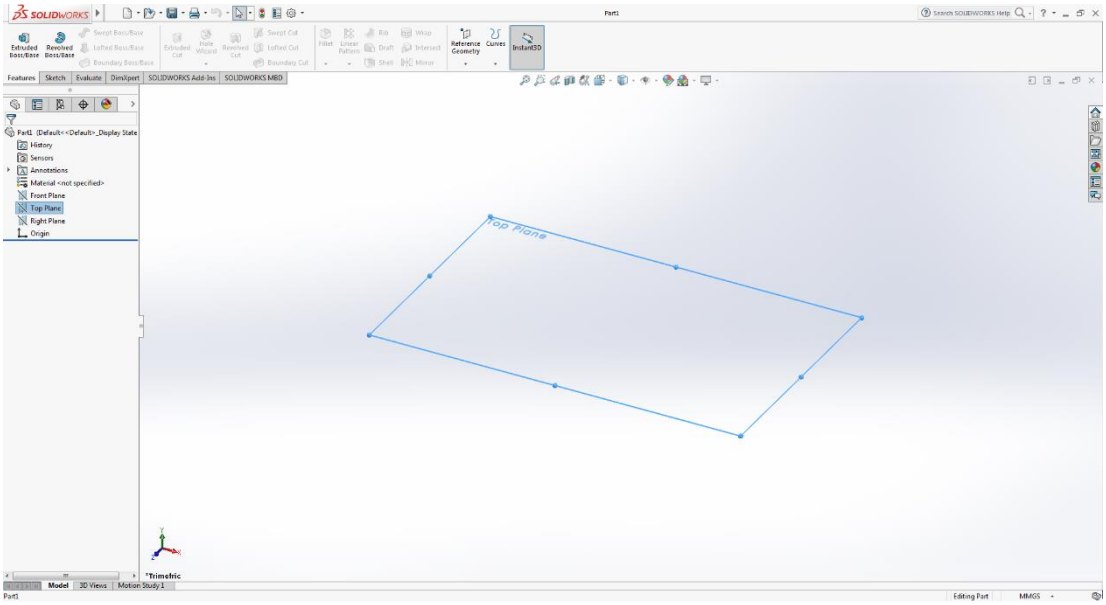
Kreiranje modela u programskom okruženju *SolidWorks* obično počinje s 2D skicom. Skica se sastoji od geometrije poput točaka, linija, lukova, itd. Dimenzije se dodaju skici kako bi odredili veličinu i položaj geometrije. Veze se koriste za definiranje atributa kao što su tangentnost, paralelnost i koncentričnost. Dimenzije u skici se mogu određivati samostalno ili prema vezama s drugim parametrima unutar ili izvan skice [4, 5].

Modeliranje samog sklopa počinje tako da se prvo kreira dio po dio. Dio se može kreirati tako da se odabere opcija *File*, zatim *New* (CTRL+N). Nakon toga se pojavi sučelje prikazano na slici 3.1 na kojem odabiremo opciju *Part* za 3D modeliranje pojedinog dijela zasebno ili *Drawing* za 2D modeliranje pojedinog dijela ili cijeloga sklopa.



Slika 3.1 Izrada dijelova u programskom okruženju *SolidWorks*

U ovom slučaju, odabrana je opcija *Part*. Nakon toga kreće samo modeliranje te se otvara sučelje u kojem je prikazan prostor za kreiranje modela (slika 3.2). Prostorom se krećemo po Kartezijevom trodimenzionalnom koordinatnom sustavu (x, y, z osi). Isto tako prostor je podijeljen u tri plana, a to su: gornji tlocrt, desni bokocrt i nacrt.

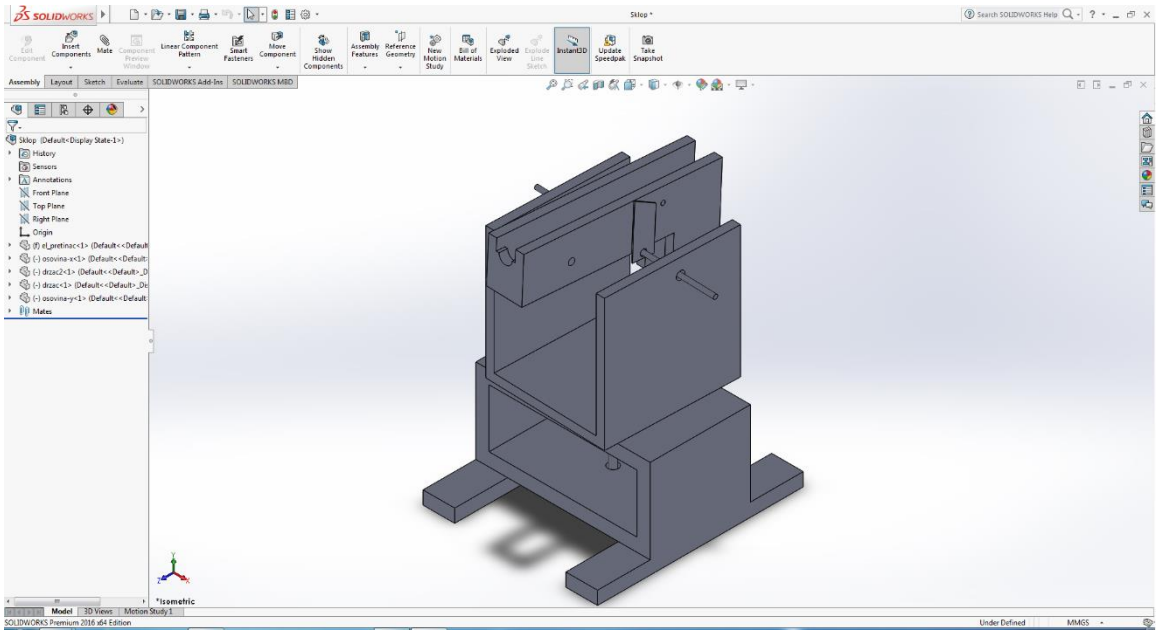


Slika 3.2 Prostor za modeliranje u programskom okruženju *SolidWorks*

Na alatnoj traci pod nazivom *Command Manager* nalaze se alati za modeliranje. Prilikom modeliranja, korištene su kartice *Features* i *Sketch*. Za kreiranje modela obrambene kupole korišteni su sljedeći alati na navedenim karticama:

- *Line* (L) – alat za crtanje pravocrtnih linija,
- *Circle* – alat za crtanje kružnice,
- *Extruded Boss/Base* – alat za izvlačenje modela iz 2D u 3D projekciju,
- *Extruded Cut* – alat za ekstrudirane rezove.

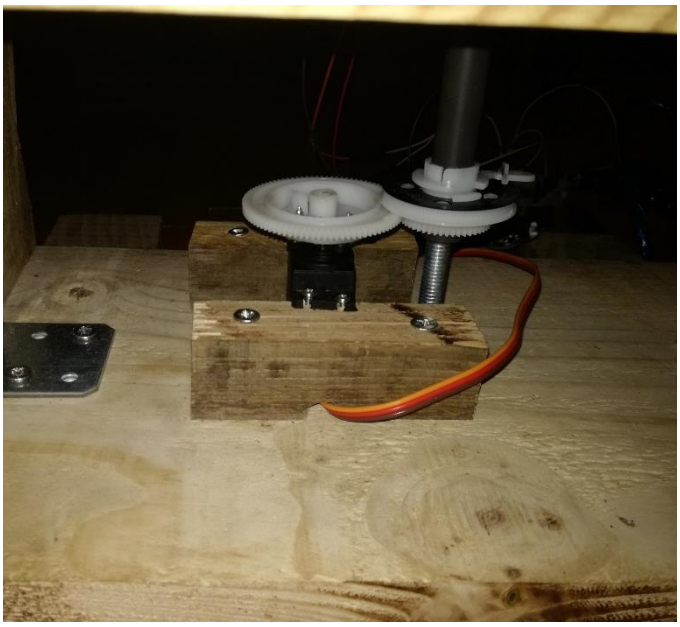
Na lijevom izborniku prikazan je popis korištenih alata. Svaki korak može se tim putem pratit, te naknadno modificirati ili ukloniti po potrebi. Na slici 3.3 prikazan je kompletan model obrambene kupole.



Slika 3.3 Model automatske obrambene kupole

3.2 Izrada baze obrambene kupole

Baza obrambene kupole izrađena je od drveta i iverice. Od drva je izrađen odjeljak za elektroniku zbog veće stabilnosti obrambene kupole, dok je držač oružja izrađen od iverice radi smanjenja opterećenja na servo motore prilikom mirovanja i gibanja pomičnog dijela kupole. Držač oružja i pretinac za elektroniku povezani su pomoću osovine koja se koristi za gibanje po X osi i kugličnog aksijalnog ležaja. Za gibanje osovina po X i Y osi koristi se zupčasti prijenos (slike 3.4 i 3.5). Okidanje je odrađeno putem poluge između servo motora i okidala oružja.



Slika 3.4 Zupčasti prijenos - x os

Korištenjem zupčanika za gibanje po X i Y osi dobiva se veći okretni moment i preciznost kod kretanja same kupole. Servo motori su na taj način puno manje opterećeni, te mogu lakše i brže pratiti metu koja se giba većom brzinom. Isto tako, korištenjem zupčanika poboljšana je i preciznost kupole.

Slika 3.5 Zupčasti prijenos - y os

Za prijenos gibanja između same baze i nosača kupole korišten je dvoredni aksijalni kuglični ležaj prikazan na slici 3.6. Navedeni ležaj ima mogućnost prenošenja aksijalnih opterećenja u oba smjera, te podnosi dosta velike brzine vrtnje. Konstrukcija ležaja je rastavljiva zbog čega se svaki dio može pojedinačno ugraditi što olakšava i cjelokupnu ugradnju ležaja.

Slika 3.6 Dvoredni aksijalni kuglični ležaj

3.3 Servo motori

Servo motor je tip električnog motora koji ima mogućnost precizne kontrole kutnog položaja, brzine i ubrzanja. Sastoji se od motora na čiji je rotor spojen enkoder za povratnu informaciju o položaju. Servo motori za razliku od koračnih motora imaju konstantni okretni moment, a zbog sustava zatvorene petlje nema bojazni od gubitka koraka [6].

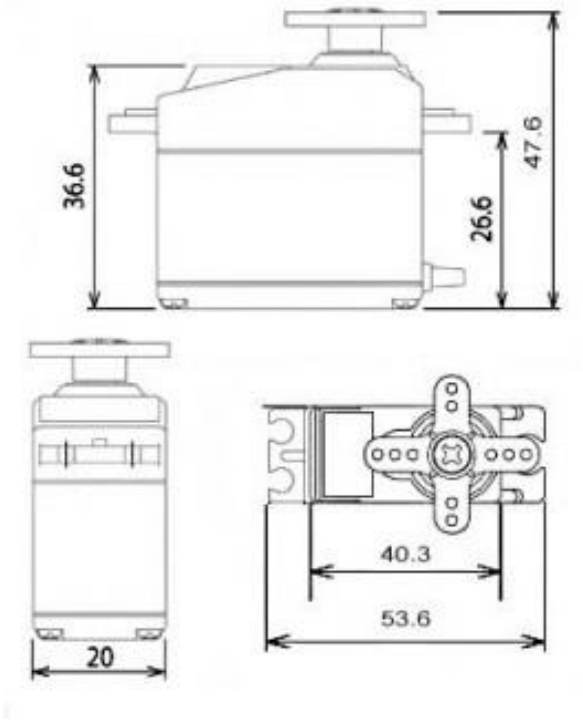
Vrsta motora nije ključan dio kod izrade servo motora te se zbog jednostavnosti i niske cijene najčešće koriste četkasti motori s permanentnim magnetom. Za velike industrijske servo motore koriste se AC indukcijski motori s promjenjivim frekvencijskim pogonom kako bi se omogućila kontrola nad njihovom brzinom [6].

Servo motori dijele se na:

1. istosmjerni servo motori (DC servo motor) i
2. izmjenični servo motori (AC servo motor).

Slika 3.7 TowerPro MG996R[7]

Za upravljanje automatskom obrambenom kupolom koriste se sveukupno 3 DC servo motora *MG996R*(slika 3.7). Dva servo motora koriste se za gibanje kupole po X i Y osi, dok je treći servo motor za okidanje oružja. Servo motori koriste tri žice, od čega su dvije za napajanje i treća za signal. Vrlo često crvena žica je spojena na pozitivan pol (+), a crna ili smeđa spojena na negativan pol (-). Žica za signal može biti žute, narančaste ili bijele boje te se koristi za komunikaciju između servo motora i razvojnog okruženja *Arduino*. Shema za MG996R prikazana je na slici 3.8 [6, 7, 8].



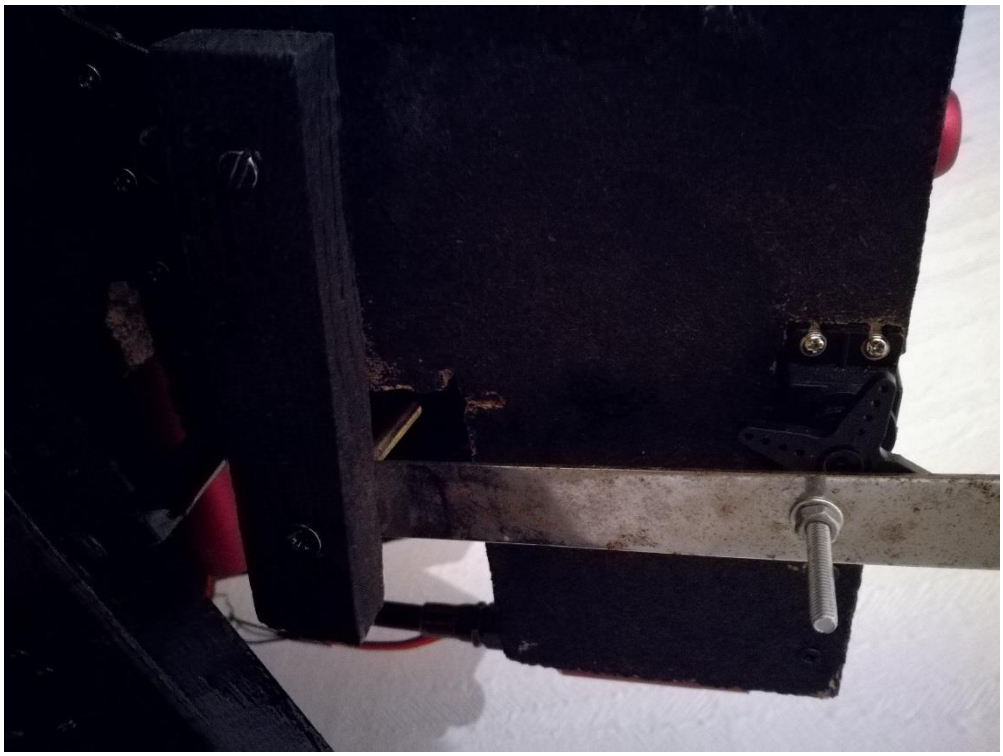
Slika 3.8 Konstrukcija shema - *MG966R* [7]

Specifikacije *MG996R* servo motora:

- Masa: 55g,
- Dimenzije: 40.7 x 19.7 x 42.9 mm,
- Okretni moment: 9.4 kg/cm (4.8V) , 11 kg/cm (6.0V),
- Radna brzina: 0.19 sec/ 60 stupnjeva (4.8V) , 0.15 sec/ 60 stupnjeva (6.0V),
- Radni napon: 4.8V ~ 6.6 V,
- Vrsta zupčanika: metalni.

3.4 Sustav za okidanje

Sustav za okidanjekoristi *MG996R* servo motor koji je programiran da ima funkciju automatske i poluautomatske paljbe. Sustav se može vidjeti na slici 3.9.



Slika 3.9 Mehanizam za okidanje

3.5 Popis korištenih dijelova za izradu obrambene kupole

U tablici3.1 nalazi se popis dijelova korištenih za konstrukciju obrambene kupole.

Tablica 3.1 Popis dijelova korištenih za izradu obrambene kupole

Naziv	**Svrha**
Servo motor x3	Gibanje po x i y osi, okidanje.
Drvo, iverica	Materijal za izradu baze kupole i nosača za oružje.
Breadboard	Lakše i organiziranije ožičenje.
USB kabel tipa B	Napajanje za razvojno okruženje *Arduino*, komunikacija između računala i razvojnog okruženja *Arduino.*
Web kamera	Praćenje mete.
Spremnik baterija	Napajanje servo motora pomoću 6x AAA baterija.
Arduino Uno R3	Upravljanje servo motorima, komunikacija s računalom.
Žice	Komunikacija između servo motora i Arduina, napajanje.
Vijci, zupčanici, i ostali materijali	Materijal potreban za konstrukciju baze i nosača.
Oružje	Marker za *paintball.*

4 Automatizacija obrambene kupole

4.1 Arduinorazvojno okruženje

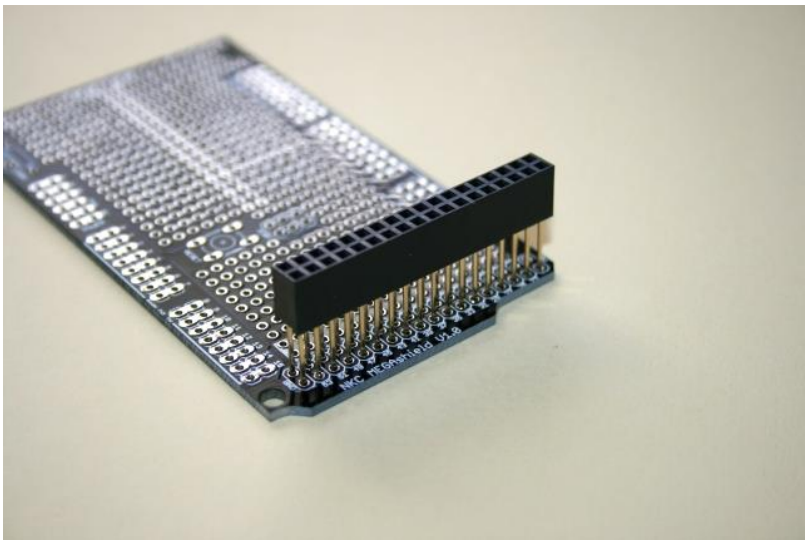
*Arduino*razvojno okruženje je alat pomoću kojega možemo napraviti interaktivne aplikacije, dizajniran da bi se pojednostavili zadaci početnicima, ali je fleksibilan i za stručnjake. *Arduino* platforma se temelji na jednostavnoj razvojnoj pločici s ulazno/izlaznim konektorima i besplatnom programskom podrškom s jednostavnim korisničkim sučeljem. Cjelokupno programiranje izvodi se u programskom jeziku C++, a izvodi se iz integriranog razvojnog okruženja koje postoji za *Linux, Windows* i *Mac* platforme [9, 10].

Vrste osnovnih *Arduino*razvojnih okruženja su:

- *Arduino Uno,*
- *Arduino Leonardo,*
- *Arduino 101,*
- *Arduino Esplora,*
- *Arduino Micro,*
- *Arduino Nano,*
- *Arduino Mini,*
- i drugi.

Postoje i naprednije izvedbe razvojnog okruženja *Arduino* s naprednim funkcijama i bržim performansama te se koriste za kompleksnije projekte [9, 10, 11].

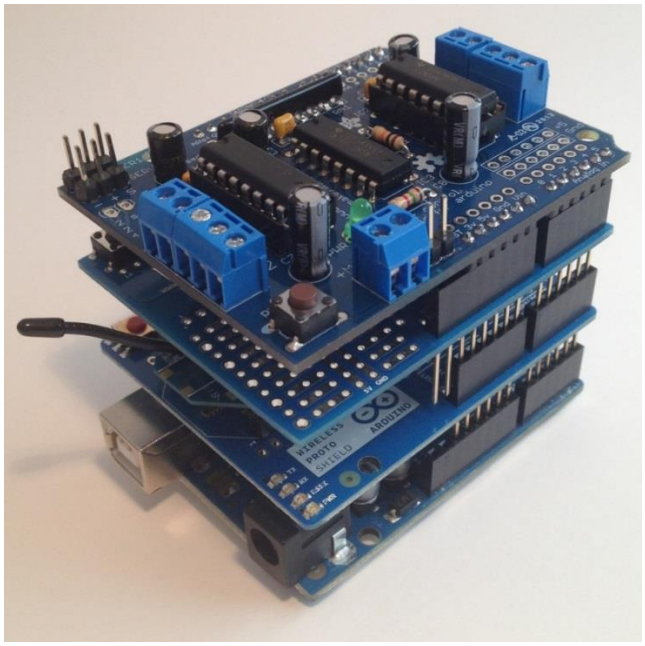
Većina *Arduino*razvojnih okruženja sastoji se od *Atmel* 8-bitnog AVR mikrokontrolera (*ATmega8, ATmega168, ATmega328, ATmega1280, ATmega2560*) s različitim količinama *flash* memorije, pinova i značajki. 32-bitni *Arduino Due,* temeljen na *Atmel SAM3X8E* uveden je 2012. godine. Arduino razvojna okruženja koriste jednodijelne ili dvostruke pinove ili ženske priključne pinove (slika 4.1)koji omogućuju priključke za programiranje i ugradnju u druge elektroničke krugove.



Slika 4.1*Arduino Shield* i ženski priključni pinovi [12]

Arduino okruženja mogu se povezati s dodatnim modulima nazvanim *Shield*. Višestruki i složeni*Shield*moduli mogu se pojedinačno adresirati putem *I2C* serijske sabirnice (ili neke druge). Većina ploča sadrži stabilizator napona (3.3 V i 5 V) i 16 MHz kristalni oscilator ili keramički rezonator. Neke izvedbe, kao što je *LilyPad*, pokreću se pri 8 MHz i odstupaju od regulatora napona zbog specifičnih ograničenja na oblik.Ploča razvojnog okruženja *Arduino*ima otkriven veći dio *I/O*pinova mikrokontrolera kako bi se mogli upotrijebiti i u drugim upravljačkim krugovima. *Diecimila, Duemilanove* i trenutni *Uno*osiguravaju 14 digitalnih *I/O*pinova, od kojih šest može proizvesti signale modulirane širine impulsa i šest s analognim ulazima, koji se također mogu koristiti kao šest digitalnih *I/O*pinova. Ovi pinovi nalaze se na vrhu ploče, te su izvedeni preko ženskog priključnog pina polumjera od 0,1 inča (2,54 mm).

Nekoliko zaštitnih modula *Shield* za dodatke koji se mogu spojiti na *Arduino* (slika 4.2), također su komercijalno dostupni. *Arduino Nano,* i *Arduino-Bare Bones* koji su kompatibilni i *Boarduino*ploče mogu pružiti muške priključne pinove na donjoj strani ploče koji se mogu priključiti na nelemljive upravljačkeploče.Postoje mnoge *Arduino*-kompatibilne i izvedene verzije ploča. Neke su funkcionalno ekvivalentne srazvojnim okruženjem *Arduino* i mogu se koristiti međusobno. Mnoge poboljšavaju osnovnu*Arduino* ploču dodavanjem izlaznih upravljačkih komponenti, te se često koristi za edukacijske svrhe na razini škole, kako bi se pojednostavila izrada vozila *Buggy* i malih robota. Drugi su električki ekvivalentni, ali im je promijenjen oblik, ponekad zadržavajući kompatibilnost sa shieldovima, a ponekad i ne. Neke inačice koriste različite procesore, te su različitih kompatibilnosti [12].



Slika 4.2 Korištenje višestrukih *ArduinoShield*modula [12]

4.2 ArduinoUno R3

*Arduino UNO*razvojno okruženje(slika 4.3) je bazirano na *ATMEGA328* mikrokontroleru trvtke *Atmel*koji radi na 16 MHz. Pločica sadrži 14 digitalnih ulaza/izlaza i 6 analognih ulaza. Povezivanje na računalo i programiranje se vrši pomoću *USB* kabela. *ArduinoUno*je moguće napajati zasebnim napajanjem od 7-12V ili putem računala (preko *USB* kabela). *Arduino UNO R3* se od svojih prethodnih ploča razlikuje u tome što ne koriste *FTDIUSB* serijski čip, već umjesto toga ima *Atmega16U2* (*Atmega8U2* verzije *R2*) programiran kao *USB* serijski konverter.

Mikrokontroler izvodi jedan program koji je zapisan u njegovu Flash memoriju, a u *EEPROM* memoriji se čuvaju podaci nakon isključenja uređaja. Funkcijama *pinMode*(), *digitalWrite*() i *digitalRead*() određuje se da li će digitalni pinovi biti ulazni ili izlazni. Pinovi imaju radni napon od 5V i mogu biti opterećeni maksimalnom strujomiznosa 40 mA. Pritezni otpornik je standardno isključen te je njegov otpor 20-50 kΩ. Određeni pinovi imaju i specijalne funkcije kao što su: *RX* i *TX* za primanje i prijenos *TTL* serijskih podataka, vanjski prekidi, *PWM*, *LED*, itd. Detaljniji opis digitalnih pinova može se vidjeti u tablici 4.1[9, 10, 11].

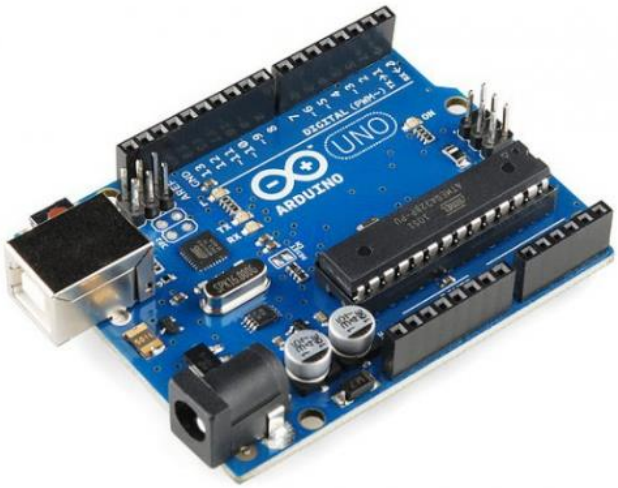
Tablica 4.1 Funkcije digitalnih pinova

PIN	FUNKCIJA PINA
0, 1	*RX* i *TX* pinovi za primanje i slanje *TTL* serijskih podataka.
2, 3	Vanjski prekidi – ovi pinovi mogu se konfigurirati da aktiviraju prekid na niskoj vrijednosti, rastućem ili padajućem bridu ili kod promijene vrijednosti pomoću funkcije *attachInterrupt ()*.
3, 5, 6, 9, 10, 11	Mogu se koristiti za 8-bitni *PWM* izlaz koji si omogućuje funkcijom *analogWrite()*.
10 (SS), 11(MOSI), 12(MISO), 13(SCK)	Podržavaju *SPI* komunikaciju koristeći *SPI*biblioteka.
13	Ugrađeni *LED* koji se aktivira kada je digitalni pin 13 *HIGH* vrijednosti, a isključen kada je na *LOW* vrijednosti.

Specifikacije za *ArduinoUno R3*razvojno okruženje:

- 8 bitni *CMOS* mikrokontroler,
- Mikroprocesor: *Atmega328*,
- Frekvencija procesora: 16 MHz,
- Radni napon 5 V,
- Ulazni napon (preporučen) : 7-12 V,
- Ulazni napon (ograničen) : 6-12 V,
- Digitalni *I/0* pinovi: 14 (6 pružaju *PWM* izlaz),
- Analogni ulazni pinovi: 6,

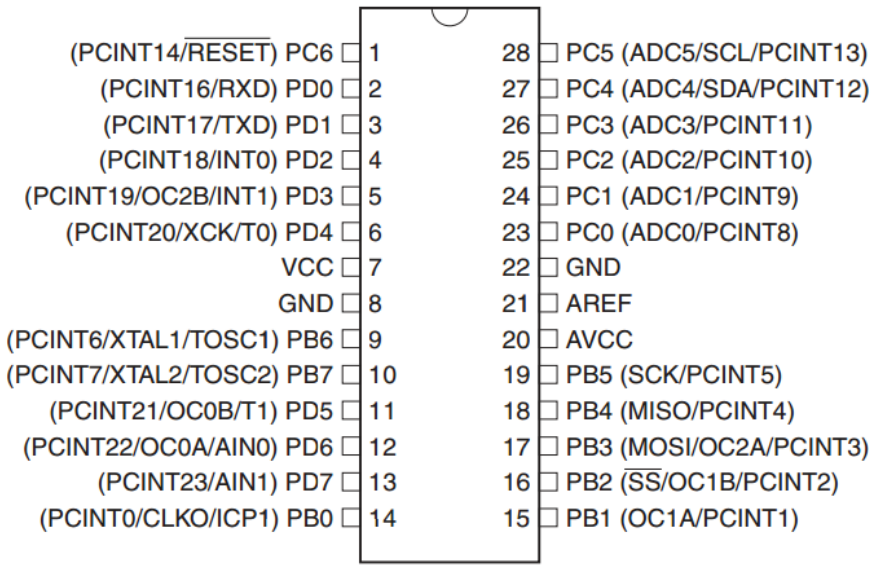
- *Flash* memorija: 32 kB *Flash* programska memorija sa programom *Bootloader* 2 kB *SRAM* za podatke 1 kB *EEPROM*,
- Povezivanje sa računalom preko komunikacijske veze *USB*.



Slika 4.3*Arduino Uno R3*[11]

4.3 Atmel ATmega328P

Atmel AVR® kombinira bogati skup instrukcija s 32 radna registra opće namjene. Svih 32 registra izravno su povezani s ALU (aritmetičko logičkom jedinicom), dopuštajući da se dva neovisna registra koriste u jednoj instrukciji izvršenoj u jednom radnom ciklusu. Rezultirajuća arhitektura je kodno učinkovitija jer postiže do deset puta bržu propusnost od konvencionalnih *CISC* mikrokontrolera.*Atmega328/P* nudi sljedeće značajke: 32Kb programibilneintegrirane flash memorije s mogućnošću „*Read-while-Write*“, 1Kb *EEPROM*, 2K *SRAM*, 23 *I/O* linije opće namjene, 32 radna registra opće namjene, *RTC*, 1 serijski programirani *USART*, 1Bajtni orijentirani 2-žični serijski međusklop (*I2C*), 6-kanalni 10-bitni *ADC*, 3 fleksibilna tajmera/brojača s usporednim načinima rada i *PWM* programibilni*Watchdog Timer* s unutarnjim oscilatorom, *SPI* serijski port i šest načina rada koji štede energiju [13]. Shema pinova za *Atmega328P* prikazana je na slici 4.4.

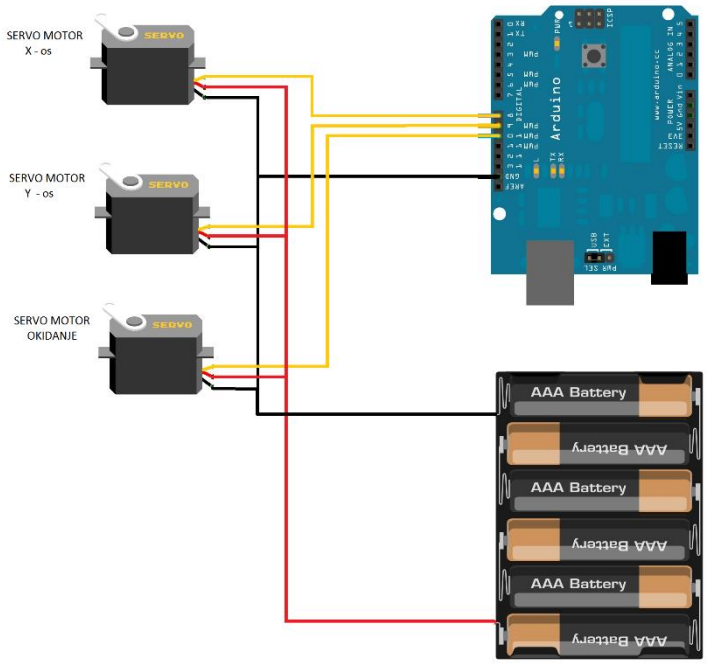


Slika 4.4 Shema Atmega328P mikrokontrolera[13]

Način rada u praznom hodu zaustavlja *CPU*, a istovremeno omogućava nastavak funkcioniranja sustava *SRAM*, tajmera/brojača, *SPI* porta i prekida. *Power-down* način rada sprema sadržaj registra, ali zamrzava oscilator, onemogućuje sve druge funkcije čipa do sljedećeg prekida ili resetiranja hardvera. U načinu rada *Power-saving*, asinkroni mjerač vremena nastavlja s radom, omogućujući korisniku održavanje baze tajmera dok ostatak uređaja nije u funkciji. *ADC Noise reduction* način rada zaustavlja *CPU* i sve *I/O* module, osim asinkronog mjerača vremena i analogno digitalnog konvertera, kako bi se smanjio šum uklapanja tijekom *ADC* pretvorbi. U *Standby* načinu rada, kristal/rezonator oscilator radi dok ostatak uređaja nije u funkciji. To omogućuje vrlo brz početak rada uz nisku potrošnju energije. U modu *Extended Standby* i glavni oscilator i asinkroni mjerač vremena nastavljaju raditi [13].

4.1 Ožičenje obrambene kupole i Arduino razvojnog okruženja

Napajanje servo motora na obrambenoj kupoli i razvojnog okruženja *Arduino* je odvojeno. Za napajanje razvojnog okruženja *Arduino* koristi se *USB* kabel povezan na prijenosno računalo. Servo motori su napajani pomoću AAA baterija. Signalna žica servo motora za gibanje po X osi spojena je na digitalni *I/O* pin 8. Na digitalni *I/O* pin 9 spojena je signalna žica servo motora za gibanje po Y osi, dok je signalna žica servo motora za okidanje spojena na digitalni *I/O* pin 10. Negativan pol (-) servo motora se osim na napajanje spajaju i na *GND* pin na razvojnom okruženju *Arduino*. Shema ožičenja prikazana je na slici 4.5.



Slika 4.5 Ožičenje obrambene kupole

4.2 Web kamera

Web kamera (slika 4.6) je video kamera koja u stvarnom vremenu prikazuje njezinu sliku. Kada računalo „snimi“ video, video *stream* može biti spremljen, pregledan ili poslan na druge mreže putem sustava poput interneta ili poslan e-poštom kao privitak. Kada se šalje na udaljenu lokaciju, videostream se može spremiti, pregledavati ili tamo poslati. Za razliku od *IP* kamere (koja se povezuje putemmrežne tehnologije *Ethernet* ili *Wi-Fi* mreže), web kamera se obično povezuje *USB* kabelom, sličnim kabelom ili je ugrađena u računalni hardver kao kod prijenosnih računala.Izraz "web kamera" također se može koristiti u izvornom smislu video kamere spojene na web, kontinuirano i na neodređeno vrijeme, a ne za određenu sesiju, obično opskrbljujući pogled za svakoga tko posjeti njegovu web stranicu preko interneta. Primjer takvog načina korištenja web kamere je korištenje web kamera za prikaz cestovnog prometa na mreži. Web kamere obično uključuju leću, senzor slike, elektroničku podršku i mogu sadržavati i mikrofon za zvuk. Dostupne su različite leće, najčešće u web-kamerama potrošačke klase koristi se plastična leća kojoj se može pomoću vijka podešavati fokus kamere. Također su dostupne leće s fiksnim fokusom, koje nemaju opcija za prilagodbu. Kako je dubina slike fotoaparata veća kod malih formata slike, a veća za leće s velikim f-brojem (mali otvor), sustavi koji se koriste u web kamerama imaju dovoljno veliku dubinsku sliku da koriste objektiv s fiksnim fokusom što u ne utječe na oštrinu slike [14].

Slika 4.6*Gigatech* web kamera[15]

Senzor slike može biti *CMOS* ili *CCD*, pri čemu je prijašnji*CMOS* dominantan za jeftine kamere, ali *CCD* kamere ne moraju nužno nadmašiti *CMOS* fotoaparate kada se uspoređuju niske cijene. Većina web kamera korisnicima može pružiti *VGA* razlučivost videozapisa i brzinu od 30 sličica u sekundi. Mnogo novijih uređaja može proizvesti video u rezolucijama od više megapiksela, a nekoliko njih može doseći i visoke brzine prijenosa, kao što je *PlayStation Eye* koji može proizvesti 320 × 240 piksela videozapis sa 120 sličica u sekundi.Elektronička podrška očitava sliku senzora i odašilje ga na glavno računalo. Fotoaparat *Sonix SN9C101* za prijenos slike koristi *USB*. Tipično, svaki okvir se prenosi nekomprimiran u *RGB* ili *YUV* datoteci ili komprimiran kao *JPEG* datoteka. Neki fotoaparati, kao što su kamere na mobilnim telefonima, koriste *CMOS* senzor s podrškom za elektroniku "*On die*", tj. senzor i pripadajuća elektronika se nalaze na jednom silicijskom čipu za uštedu prostora i troškova proizvodnje. Većina web kamera ima ugrađene mikrofone kako bi video poziv i videokonferencija bila prikladnija.Specifikacija klase *USB* video uređaja (*UVC*) omogućuje interkonektivnost web kamera s računalima bez potrebe za upravljačkim programom uređaja. *Microsoft Windows XP SP2*, *Linux* i *Mac OS X* (od listopada 2005.) implementirali su *UVC* podršku i ne zahtijevaju dodatne upravljačke programe uređaja, iako se često instaliraju zbog dodatnih opcija koje omogućuje upravljački program [14].

4.3 Komunikacija između Arduino Uno R3 i računala

*Arduino Uno R3*razvojno okruženje ima brojne mogućnosti za komuniciranje s računalom, *Arduino/Genuino* ili drugim verzijama razvojnih okruženja. *ATmega328* ima *UART TTL* (5V) serijsku komunikaciju koja je postavljena na pinove 0 (*RX*) i 1 (*TX*). *ATmega16u2* na razvojnom okruženju kanalizira ovu serijsku komunikaciju preko *USB*veze i pojavljuje se kao virtualna veza

softvera na računalu. *Firmware 16U2* koristi standardni *USB COM* upravljački program i nije potreban nikakav vanjski upravljački program. U *Arduino IDE* programskom okruženju je uključen i serijski monitor koji omogućuje primanje/slanje jednostavnih tekstualnih podataka na ploču. Prilikom primanja/slanja podataka na ploču *RX* i *TX LED* diode će na ploči bljeskati kada se prijenos/primanje podataka vrši putem *USB* veze. Korištenjem *SoftwareSerial* biblioteke omogućuje se serijska komunikacija na bilo kojem od digitalnih pinova razvojnog okruženja *Arduino Uno*. *ATmega328* također podržava i *I2C* (*TWI*) i *SPI* komunikaciju. Prilikom prijenosa koda na *Arduino Uno*, nije više potreban fizički reset putem tipkala već je ploča dizajnirana na način koji omogućuje softverski reset [16].

4.4 Programiranje ArduinoUno R3

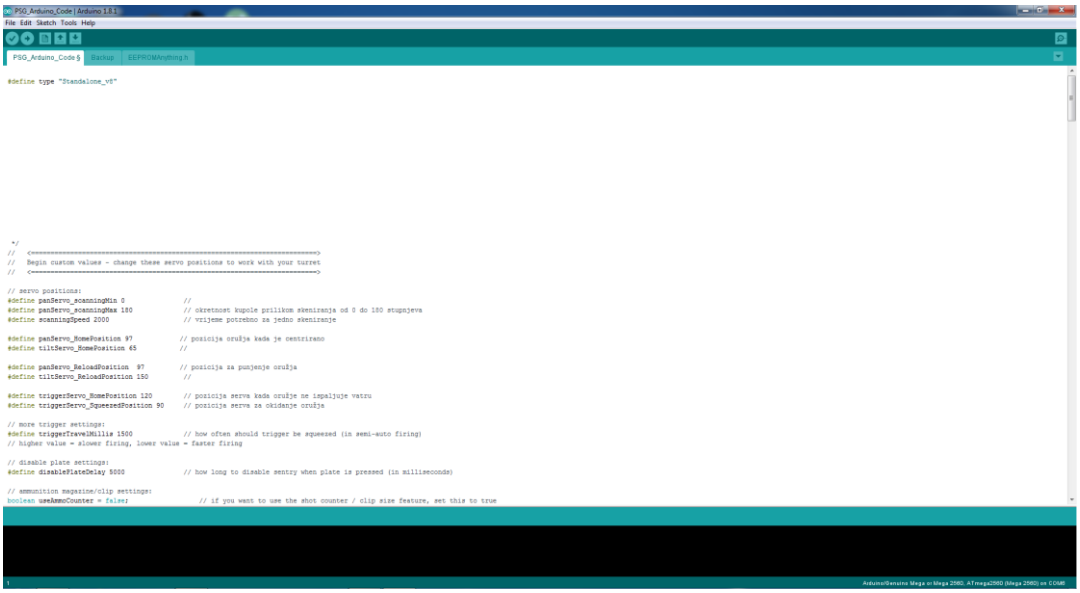
Za programiranje razvojnog okruženja*Ardino* potrebni su *USB* kabel (slika 4.7) koji spaja računalo,*Arduino* i softver *Arduino IDE*. *Arduino IDE*je besplatni program za programiranje samog razvojnog okruženja *Arduino*koji se može preuzeti besplatno putem interneta. *Arduino Uno* koristi *ATmega382* koji dolazi s unaprijed programiranim programom *Bootloader*. *Bootloader* omogućuje prenošenje koda bez korištenja vanjskog hardverskog programatora. Komunikacija između računala i mikrokontrolera je putem *STK500* protokola. Moguće je i zaobići sami *Bootloader* i programirati mikrokontroler kroz *ICSP(In-Circuit Serial Programming*) koristeći *Arduino ISP* ili slično [17].

Slika 4.7 USB kabel tipa B[17]

Arduino IDE je razvijen u programskom jeziku *Java* te je potekao iz programskih razvojnih okruženja za programske jezike *Processing* i *Wiring*. Ima implementiran uređivač teksta sa značajkama kao što su rezanje i lijepljenje teksta, pretraživanje i zamjena teksta, automatsko uvlačenje, označavanje sintakse te vrlo jednostavan prijenos programa na *Arduino* ploču sa samo

jednim klikom miša. Osim uređivača teksta, *Arduino IDE* sadrži područje za poruke, tekstnu konzolu koja nam javlja greške i status *Arduino* ploče, te alatnu traku s gumbima za navedene funkcije.

Program napisan u *Arduino IDE* (slika 4.8) naziva se skica, a skice se spremaju na računalo u obliku .ino datoteke.*Arduino IDE* podržava *C* i *C++* programske jezike pomoću posebnih pravila strukturiranja koda programa [18, 19].



Slika 4.8*Arduino IDE*

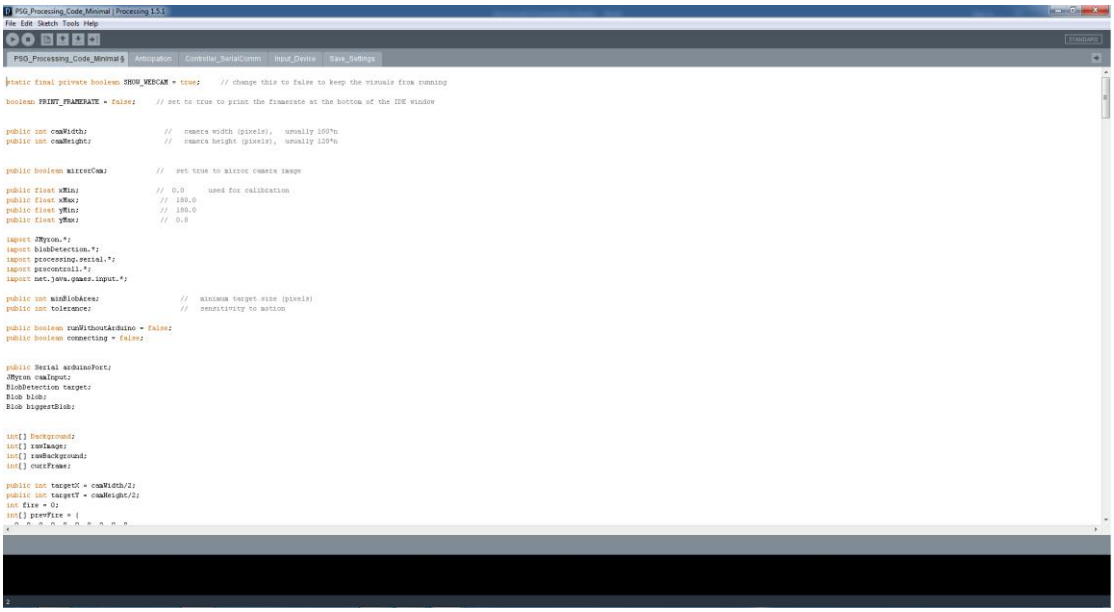
4.4.1 Konfiguracija početnih pozicija

Biblioteka korištena za obrambenu kupolu je *servo.h* koja u sebi sadrži naredbe koje su potrebne za upravljanje servo motorima. Biblioteka se poziva naredbom: *#include servo.h*. U samom početku programskog koda za obrambene kupole potrebno je odabrati da li se koristi samostalni*Arduino, Arduino* sa modulom *Shield* ili unaprijed izrađena pločica specifično dizajnirana za upravljanje obrambenom kupolom. Za obrambenu kupolu korištenu u završnom radu korišten je samostalni*Arduino*, stoga se vrsta kontrolera postavlja naredbom*#define type „Arduino_bare“*. Nakon što se odredi vrsta kontrolera, slijedi postavljanje početnih pozicija servo motora. Početnu poziciju servo motora za gibanje po x-osi postavlja se naredbom*#define panServo_HomePosition*, pozicija servo motora za gibanje po y-osi naredbom *#define tiltServo_HomePosition* dok se za servo motor za okidanje koristi naredba*#define triggerServo_HomePosition*. Prilikom odabira pozicije unose se vrijednosti od 0°-180°, te

vrijednosti predstavljaju stupnjeve otklona motora u od početne pozicije koja je na 90° u realnom svijetu. Omjer vrijednosti unutar programa i stvarnosti je 1:1.Osim početnih pozicija, potrebno je postaviti parametre i za skeniranje kao što su minimalni kut*#define panServo_scanningMin*, maksimalni kut skeniranja *#define panServo_scanningMax*, te brzina skeniranja *#define scanningSpeed*. Brzinu okidanja prilikom poluautomatskog režima rada postavljamo pomoću naredbe *#define triggerTravelMillis*, pri čemu unesena vrijednost predstavlja milisekunde što znači da što veću vrijednost unesemo to će brzina okidanja biti sporija.Odabir režima okidanja određen je naredbom *public boolean firingMode*, pri čemu je za vrijednost *true* uključen poluautomatski režim, a za vrijednost *false* automatski režim okidanja.

4.5 Programski jezik Processing

Processing je *open source* računalni programski jezik i integrirano razvojno okruženje (slika 4.9) izgrađeno za elektroničku umjetnost, novu medijsku umjetnost i zajednice vizualnog dizajna s ciljem podučavanja osnova računalnog programiranja u vizualnom kontekstu te da služi kao temelj elektroničkim knjigama za skiciranje. Projekt je započeo 2001. godine od strane osnivača Casey Reas i Benjamin Fry. Oboje su bivši članovi grupe za estetiku i računanje u ustavnovi *MIT Media Lab*. U 2012. su započeli zakladu za *Processing* zajedno s novim članom pod imenom Daniel Shiffman koji se pridružio kao treći voditelj projekta. Jedan od ciljeva programskog okruženja *Processing* je omogućiti, ljudima koji nisu programeri, računalno programiranje s vizualnom povratnom informacijom. *Processing* se temelji na programskom jeziku *Java*, ali koristi pojednostavljenu sintaksu i korisničko sučelje grafičkog prikaza [20, 21].



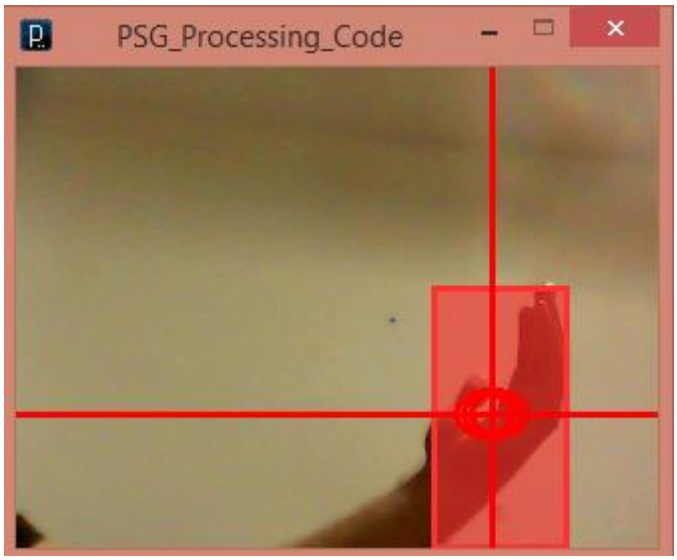
Slika 4.9*Processing IDE*

4.6 Prepoznavanje mete

Prepoznavanje mete odrađeno je pomoću programskog koda u programskog okruženju *Processing*, računala i web kamere. Isto kao u *Arduino IDE*, potrebna je bibliotekaili više njih kako bi se mogle pozivati već unaprijed isprogramirane funkcije te ih povezati u cjelokupan program. Korištene sučetiri biblioteke, a to su: *BlobDetection*, *GUI Components*, *JMyron* i *Procontroll*. Kako bi bibliotekabila aktivna u programu da se mogu iz njega pozivati funkcije, potrebno ga je uključiti pomoću naredbe *import* (naziv biblioteke).

4.6.1 BlobDetection

BlobDetection je jedan od glavnih biblioteka za prepoznavanje mete i funkcionira na način da pronalazi mrlje na slici koje imaju svjetlinu višu ili nižu od zadane vrijednosti. Kada pronađe mrlju ili grupu mrlja čija svjetlina odstupa od zadane vrijednostiizračunava i detektira poziciju mete kako je prikazano na slici 4.10. Ova biblioteka ne prati mrlje već ih pronalazi za svaki *frame* koji kamera pošalje.Veličinu mrlje koju prati kamera određuje se naredbom: *public int minBlobArea*, pri čemu je vrijednost koja se unosi izražena u pikselima.



Slika 4.10 Prepoznavanje mete

Nakon što se odredi vrijednost varijable *minBlobArea*, potrebno je kreirati varijablu s kojom će se ona uspoređivati. U ovom slučaju naziv nove varijable je *biggestBlobArea*, te joj je u početku dodijeljena vrijednost 0. U novu varijablu spremaju se vrijednosti mrlja koje šalje kamera, te ih se uspoređuje s vrijednosti zadanoj u varijabli *minBlobArea*. Na slici 4.11 može se vidjeti realizacija usporedbe dviju varijabli. Kako bi mrlja bila ispravno prikazana, potrebno ju je prikazati u 2D obliku po x i y osi, te mora imati širinu i visinu. Stoga je potrebno kreirati nove varijable koje će predstavljati širinu i visinu mrlje (*blobWidth* i *blobHeight*). Nove varijable množe se sa visinom i širinom kamere, te ih se nakon toga uspoređuje s varijablom *biggestBlob*. Rezultat umnoška navedenih varijabli daje nam poziciju mrlje koja predstavlja detekciju pokreta, te se tako daljnje instrukcije šalju s računala na *Arduino* koji upravlja servo motorima.

```
int biggestBlobArea = 0;
target.computeBlobs(screenPixels);
for (int i = 0; i < target.getBlobNb()-1; i++) {
  blob = target.getBlob(i);
  int blobWidth = int(blob.w*camWidth);
  int blobHeight = int(blob.h*camHeight);
  if (blobWidth*blobHeight >= biggestBlobArea) {
    biggestBlob = target.getBlob(i);
    biggestBlobArea = int(biggestBlob.w*camWidth)*int(biggestBlob.h*camHeight);
  }
}
possibleX = 0;
possibleY = 0;

if (biggestBlobArea >= minBlobArea) {
  possibleX = int(biggestBlob.x * camWidth);
  possibleY = int(biggestBlob.y * camHeight);
}
```

Slika 4.11 Usporedba varijabli za detekciju[21]

4.6.2 GUI Components

GUI Components koristi se za kreiranje grafičkog sučelja koji služi za kalibriranje, ručno upravljanje te biranje postavki koje se mogu koristiti na obrambenoj kupoli. Tako se za grafičko sučelje mogu kreirati klizači, gumbi, prekidači, tekstualna polja, kontrolni okviri, itd. Kako bi dio grafičkog sučelja mogao biti kreiran, potrebno je odrediti parametre funkcije. Parametri se odnose na naziv komponente koja se kreira (klizač, gumb, itd.), x i y koordinate gornjeg lijevog kuta, širina i visina željene komponente, te tekst koji će se prikazati u komponenti.U nastavku slijede dijelovi koda korišteni za kreiranje navedenih dijelova grafičkog sučelja:

label_xMin = new GLabel(*this*, "*xMin*: 000", 35, 362, 150, 20);
label_xMax = new GLabel(*this*, "*xMax*: 180", 145, 362, 150, 20);
label_yMin = new GLabel(*this*, "*yMin*: 000", 35, 392, 150, 20);
label_yMax = new GLabel(*this*, "*yMax*: 180", 145, 392, 150, 20);

Kako bi se gore navedene grafičke komponente dodale na glavno sučelje, potrebno je svaku dodati pomoću naredbe: *panel_main.add*(label_“naziv_komponente“).

4.6.3 JMyron

*JMyron*Služi za prijenos slike u stvarnom vremenu s web kamere na računalo. Prvo je potrebno odrediti visinu i širinu slike kamere prikazanu u pikselima pomoću naredbi*public int camWidth* i *public int camHeigth*.Kamera sa svakom sličicom učitava novu pozadinu te na

temelju toga detektira promijene na slici koje vidi kao metu. Linija koda *camInput.adaptivity*koja je prikazana na slici 4.12sprječava detekciju minimalnih promjena koje mogu uzrokovati sijena, pad lišća, promjena svijetlosti sunca, na koje bi inače program reagirao s otvaranjem vatre na validnu metu i trošio municiju.

```
camInput = new JMyron();
camInput.start(camWidth, camHeight);
camInput.findGlobs(0);
camInput.adaptivity(1.01);
camInput.update();
currFrame = camInput.image();
rawImage = camInput.image();
Background = camInput.image();
rawBackground = camInput.image();
screenPixels = camInput.image();
```

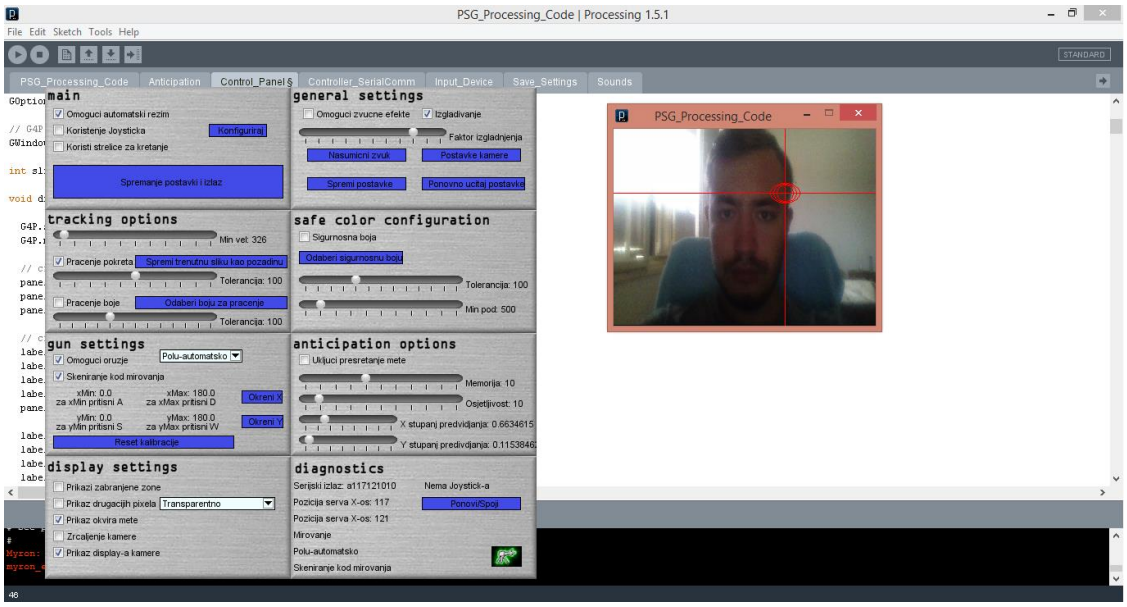
Slika 4.12 Processing kod –*Jmyron*[22]

4.6.4 Procontroll

Procontroll je biblioteka koja omogućuje da se kupolom upravlja pomoću kontrolnih uređaja kao što je *joystick*, *gampad* ili neki drugi kontrolniuređaj.

4.7 Software za upravljanje obrambenom kupolom

Sami softver za upravljanje i kalibraciju automatske kupole (slika 4.13) sastoji se od *Arduino IDE* dijela programskog koda i *Processing IDE* dijela programskog koda. Kod napisan u *Arduino IDE* koristi za komunikaciju i upravljanje servo motorima na obrambenoj kupoli, dok kod napisan u *Processing IDE* služi za grafičko sučelje i komunikaciju između web kamere, računala i *Arduino Uno R3*razvojnog okruženja.



Slika 4.13 Upravljački program

4.8 Kalibracija obrambene kupole

Kalibracija obrambene kupole je postupak izjednačavanja vidnog polja kamere u stvarnom svijetu s pokretima kupole u svrhu što preciznijeg praćenja pokreta meta koje su u pokretu ili miruju. Odrađuje se u samom programu za upravljanje prikazanom na slici 4.14. Isto tako kalibraciju je potrebno odraditi svaki puta kada se promijeni lokacija/scena koju kamera skenira.

Slika 4.14 Sekcija programa za kalibraciju kupole

Za kalibraciju su korištene sljedeće naredbe:

- *public float xMin,*
- *public float xMax,*
- *public float yMin,*
- *public float yMax.*

Vrijednosti koje se unose za navedene naredbe su stupnjevi otklona motora od 0° do 180°. Naredbom *public int tolerance* se određuje tolerancija na kretanje mete, te što se veći broj postavi to je veća tolerancija.

5 ZAKLJUČAK

Prilikom izrade ovog rada korištena su znanja koju obuhvaćaju veći dio područja koja zapravo i predstavljaju mehatroniku. Za izradu konstrukcije potrebna su znanja iz strojarstva, te dio u kojem se odrađuje modeliranje i automatizacija same kupole obuhvaća elektroniku i informatiku. Novčana sredstva korištena za izradu rada su minimalna iz razloga što su iskorišteni materijali i komponente koje su izvađene iz neispravnih printera i skenera. Cilj mehatronike je automatizacija, ne samo pojedinačnih komponenti, već i cijelih sustava. Automatizacijom se smanjuju troškovi rada čovjeka te izbjegava ljudska pogreška.

6 LITERATURA

[1] Wikipedia®. Samsun SGR-A1 [Online]. 2017. Dostupno na:https://en.wikipedia.org/wiki/Samsung_SGR-A1. (10.08.2017.)

[2] Wikipedia®. Sentry gun [Online]. 2017. Dostupno na: https://en.wikipedia.org/wiki/Sentry_gun. (10.08.2017.)

[3] Prigg M. Who goes there? Samsung unveils robot sentry that can kill from two miles away. Daily Mail [Elektronički časopis]. 2014. Dostupno na: http://www.dailymail.co.uk/sciencetech/article-2756847/Who-goes-Samsung-reveals-robot-sentry-set-eye-North-Korea.html. (12.08.2017.)

[4] SolidWorks Corporation. SolidWorks Premium. 2017. Dostupno na: http://www.solidworks.com/sw/products/3d-cad/solidworks-premium.htm. (12.08.2017.)

[5] Wikipedia®. SolidWorks [Online]. 2017. Dostupno na: https://bs.wikipedia.org/wiki/SolidWorks. (12.08.2017.)

[6] Wikipedia®. Servomotor [Online]. 2017. Dostupno na: https://en.wikipedia.org/wiki/Servomotor. (17.08.2017.)

[7] Electronico Scaldas. MG996R_Tower-Pro [Online]. 2015. Dostupno na: http://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf. (17.08.2017.)

[8] Automatika.rs. RC servo motori [Online]. 2016. Dostupno na: https://www.automatika.rs/baza-znanja/mehatronika/rc-servo-motori.html. (17.08.2017.)

[9] Wikipedia®. Arduino [Online]. 2017. Dostupno na: https://en.wikipedia.org/wiki/Arduino. (17.08.2017.)

[10] Arduino. Arduino Uno Rev3 [Online]. 2017. Dostupno na: https://store.arduino.cc/arduino-uno-rev3. (19.08.2017.)

[11] Arduino. Software [Online]. 2017. Dostupno na: https://www.arduino.cc/en/Main/Software. (19.08.2017.)

[12] NKC Electronics. Header pack for MEGAshield for Arduino MEGA, MEGA 2560, MEGA 2560 R3 amd Due [Online]. 2014. Dostupno na: http://www.nkcelectronics.com/header-pack-for-megashield-for-arduino-mega-stackable.html. (20.08.2017.)

[13] Atmel Corporation. ATMega328/P Datasheet-Complete [Online]. 2016. Dostupno na: http://www.atmel.com/images/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. (20.08.2017.)

[14] Wikipedia®. Webcam [Online]. 2017. Dostupno na: https://en.wikipedia.org/wiki/Webcam. ()

[15] Dodatna oprema. Web kamera Gigatech W-054, 1.3/20Mpix Pink [Online]. 2013. Dostupno na: http://dodatnaoprema.com/web-kamera-gigatech-w-054-13-20mpix-pink/web-kamere-periferija-cene/32478. (20.08.2017.)

[16] Arduino. Updating the Atmega8u2 and 16u2 on an Uno or Mega2560 using DFU [Online]. 2017. Dostupno na: https://www.arduino.cc/en/Hacking/DFUProgramming8U2. (23.08.2017.)

[17] Fisher T. USB Type B [Elektronički časopis]. 2017. Dostupno na: https://www.lifewire.com/usb-type-b-connector-2626033. (23.08.2017.)

[18] Arduino. Environment [Online]. 2015. Dostupno na: https://www.arduino.cc/en/Guide/Environment. (23.08.2017.)

[19] Arduino. Software [Online]. 2017. Dostupno na: https://www.arduino.cc/en/Main/Software. (23.08.2017.)

[20] Wikipedia®. Processing (Programming language) [Online]. 2017. Dostupno na: https://en.wikipedia.org/wiki/Processing_(programming_language). (24.08.2017.)

[21] GitHub. Project-Sentry-Gun [Online]. 2017. Dostupno na: https://github.com/sentryGun53/Project-Sentry-Gun. (24.08.2017.)

[22] Processing Foundation. Environment (IDE) [Online]. 2001. Dostupno na: https://processing.org/reference/environment/. (24.08.2017.)

7 OZNAKE I KRATICE

AC – Alternating Current

ADC – Analog to digital Converter

CCD – Charge Coupled Device

CISC – Complex Instruction Set Computer

CMOS – Complementary Metal Oxide Semiconductor

CPU – Central Processing Unit

DC – Direct Current

EEPROM – Elecrtricaly Erasable Programmable Read-Only Memory

GND - Ground

I/O – Input / Output – ulaz / izlaz

I2C – Inter-Integrated Circuit

IDE – integrated development environment

LED – Light-Emitting Diode

PWM – Pulse Width Modulation

RGB – RED GREEN BLUE

RTC – Real Time Clock

RX - Recieve

SPI – Serial Peripheral Interface

SRAM – Static Random Access Memory

TTL – Transistor-Transistor Logic

TX - Transmit

UART – Universal asynchronus reiever/transmitter

USB – Universal Serial Bus

UVC – Ultraviolet C

VGA – Video Graphics Array

8 SAŽETAK

Naziv: Upravljanje obrambenom kupolom pomoću *Arduino* razvojnog okruženja

U radu su opisani dijelovi korišteni za izradu automatske obrambene kupole kojom upravlja razvojno okruženje*Arduino Uno R3*, te prijenosno računalo. Web kamera se koristi kao senzor pokreta, te sliku obrađuje prijenosno računalo koje šalje daljnje upute do razvojnog okruženja *Arduino* putem *USB* konekcije. Upravljački kodovi su pisani u *Arduino IDE* i *Processing* jeziku. 3D modeliranje same kupole odrađeno je u programskom okruženju *SolidWorks*.

Ključne riječi: mikrokontroler, Arduino, obrambena kupola, automatizacija.

9 ABSTRACT

Title:Gun turret Control by Means of Arduino

The paper describes the parts used to create an sentry gun controlled by the Arduino Uno R3 development environment and the laptop. The webcam is used as a motion sensor, and the image is processed by a laptop that sends further instructions to Arduino via USB connection. The codes are written in Arduino IDE and Processing languages. 3D modeling of the sentry gun itself was done in SolidWorks.

Keywords: *microcontroller, Arduino, gun turret, automation*

10 PRILOZI

10.1 Arduino kod

```
#define type "Standalone_v8"

#define panServo_scanningMin 0
#define panServo_scanningMax 180
#define scanningSpeed 2000

#define panServo_HomePosition 97
#define tiltServo_HomePosition 65

#define panServo_ReloadPosition  97
#define tiltServo_ReloadPosition 150
#define triggerServo_HomePosition 120
#define triggerServo_SqueezedPosition 90
#define triggerTravelMillis 1500

#define disablePlateDelay 5000
boolean useAmmoCounter =false;
int clipSize = 5;


int panServoPin;
int tiltServoPin;
int triggerServoPin;
int firingIndicatorLEDPin;
int USBIndicatorLEDPin;
int modeIndicatorLEDPin;
int reloadSwitchPin;
int disablePlatePin;
int electricTriggerPin;
boolean invertInputs;

typedefstruct config_t
{
  int controlMode;
  int safety;
  int firingMode;
  int scanWhenIdle;
  int trackingMotion;
  int trackingColor;
  int leadTarget;
  int safeColor;
  int showRestrictedZones;
  int showDifferentPixels;
  int showTargetBox;
  int showCameraView;
  int mirrorCam;
  int soundEffects;


  int camWidth;
  int camHeight;
  int nbDot;
  int antSens;
  int minBlobArea;
```

```
  int tolerance;
  int effect;
  int trackColorTolerance;
  int trackColorRed;
  int trackColorGreen;
  int trackColorBlue;
  int safeColorMinSize;
  int safeColorTolerance;
  int safeColorRed;
  int safeColorGreen;
  int safeColorBlue;
  int idleTime;


  double propX;
  double propY;
  double xRatio;
  double yRatio;
  double xMin;
  double xMax;
  double yMin;
  double yMax;

}
configuration;
configuration configuration1;

#include<Servo.h>

Servo pan;
Servo tilt;
Servo trigger;
int xPosition;
int yPosition;
int fire = 0;

int fireTimer = 0;
int fireSelector = 1;

int idleCounter = 0;
int watchdog = 0;
int watchdogTimeout = 2000;
boolean idle =true;
boolean scanning =false;
boolean scanDirection =true;
boolean disabled =false;
unsignedlongint disableEndTime;
int scanXPosition = panServo_scanningMin;

int shotCounter = 0;
boolean clipEmpty =false;


byte indicator;
byte x100byte;
byte x010byte;
byte x001byte;
byte y100byte;
byte y010byte;
byte y001byte;
```

```
byte fireByte;
byte fireSelectorByte;
byte scanningByte;

voidsetup(){
  assignPins();

  pan.attach(panServoPin);
  pan.write(panServo_HomePosition);
  tilt.attach(tiltServoPin);
  tilt.write(tiltServo_HomePosition);
  pinMode(electricTriggerPin,OUTPUT);
  digitalWrite(electricTriggerPin,LOW);
  trigger.attach(triggerServoPin);
  trigger.write(triggerServo_HomePosition);

  pinMode(USBIndicatorLEDPin,OUTPUT);
  pinMode(modeIndicatorLEDPin,OUTPUT);
  pinMode(firingIndicatorLEDPin,OUTPUT);
  pinMode(reloadSwitchPin,INPUT);
  pinMode(disablePlatePin,INPUT);
  if(invertInputs) {
    digitalWrite(reloadSwitchPin,HIGH);
    digitalWrite(disablePlatePin,HIGH);
  }
  Serial.begin(4800);
}

voidloop() {
  if (Serial.available() >= 10) {
    watchdog = 0;
    indicator =Serial.read();
    if(indicator =='a') {
      idle =false;
      idleCounter = 0;
      digitalWrite(USBIndicatorLEDPin,HIGH
      x100byte =Serial.read();
      x010byte =Serial.read();
      x001byte =Serial.read();
      y100byte =Serial.read();
      y010byte =Serial.read();
      y001byte =Serial.read();
      fireByte =Serial.read();
      fireSelectorByte =Serial.read();
      fireSelector =int(fireSelectorByte) - 48;
      scanningByte =Serial.read();
      if((int(scanningByte) - 48) == 1) {
        scanning =true;
      }
      else{
        scanning =false;
      }
    }
    elseif(indicator =='z'){
      idle =true;
    }
    elseif(indicator =='b'){

      backup();
```

```
    }
    elseif(indicator =='r'){

      restore();

    }
  }
  else{
    watchdog++;
    if(watchdog > watchdogTimeout) {
      idle =true;
    }
  }
  if(idle) {
    Serial.write('T');
    idleCounter++;
    if(idleCounter > 1000) {
      sequenceLEDs(1, 100);
      delay(10);
     idleCounter = 0;
    }
    else{
      digitalWrite(USBIndicatorLEDPin,LOW);
    }
    xPosition = panServo_HomePosition;
    yPosition = tiltServo_HomePosition;
    fire = 0;
  }
  else{
    xPosition = (100*(int(x100byte)-48)) + (10*(int(x010byte)-48)) +
(int(x001byte)-48);
    yPosition = (100*(int(y100byte)-48)) + (10*(int(y010byte)-48)) +
(int(y001byte)-48);
    fire =int(fireByte) - 48;
  }

  if(scanning) {
    digitalWrite(modeIndicatorLEDPin,HIGH);
    if(scanDirection) {
      scanXPosition += 1;
      if(scanXPosition > panServo_scanningMax) {
        scanDirection =false;
        scanXPosition = panServo_scanningMax;
      }
    }
    else{
      scanXPosition -= 1;
      if(scanXPosition < panServo_scanningMin) {
        scanDirection =true;
        scanXPosition = panServo_scanningMin;
      }
    }
    xPosition = scanXPosition;
    yPosition = tiltServo_HomePosition;
    fire = 0;
    delay(scanningSpeed/abs(panServo_scanningMax-panServo_scanningMin));
  }
  else{
    digitalWrite(modeIndicatorLEDPin,LOW);
  }
```

```
  if((digitalRead(disablePlatePin) ==HIGH&&!invertInputs) ||
(digitalRead(disablePlatePin) ==LOW&& invertInputs)) {
    disabled =true;
    disableEndTime =millis() + disablePlateDelay;
  }
  if(millis() > disableEndTime) {
    disabled =false;
  }

  if((digitalRead(reloadSwitchPin) ==HIGH&&!invertInputs)
(digitalRead(reloadSwitchPin) ==LOW&& invertInputs)) {
    xPosition = panServo_ReloadPosition;
    yPosition = tiltServo_ReloadPosition;
    fire = 0;
    digitalWrite(modeIndicatorLEDPin,HIGH);
    delay(100);
    digitalWrite(modeIndicatorLEDPin,LOW);
    delay(100);
  }

  if(disabled) {
    xPosition = panServo_ReloadPosition;
    yPosition = tiltServo_ReloadPosition;
    fire = 0;
    digitalWrite(modeIndicatorLEDPin,HIGH);
    delay(50);
    digitalWrite(modeIndicatorLEDPin,LOW);
    delay(50);
  }

  pan.write(xPosition);
  tilt.write(yPosition);

  if(useAmmoCounter && shotCounter >= clipSize) {
    clipEmpty =true;
  }
  else{
    clipEmpty =false;
  }


  if(fire == 1 &&!clipEmpty) {
    Fire(fireSelector);
  }
  else{
    ceaseFire(fireSelector);
  }
}

void Fire(int selector) {
  if(selector == 1) {
    fireTimer++;
    if(fireTimer >=0 && fireTimer <= triggerTravelMillis) {
      digitalWrite(electricTriggerPin,HIGH);
      trigger.write(triggerServo_SqueezedPosition);
      digitalWrite(firingIndicatorLEDPin,HIGH);
    }
    if(fireTimer > triggerTravelMillis && fireTimer <
1.5*triggerTravelMillis) {
```

```
      digitalWrite(electricTriggerPin,LOW);
      trigger.write(triggerServo_HomePosition);
      digitalWrite(firingIndicatorLEDPin,LOW);
    }
    if(fireTimer >= 1.5*triggerTravelMillis) {
      fireTimer = 0;
      if(useAmmoCounter) {
        shotCounter++;
      }
    }
  }
  if(selector == 3) {
    digitalWrite(electricTriggerPin,HIGH);
    trigger.write(triggerServo_SqueezedPosition);
    digitalWrite(firingIndicatorLEDPin,HIGH);
  }
}

void ceaseFire(int selector) {
  if(selector == 1) {
    fireTimer = 0;
    digitalWrite(electricTriggerPin,LOW);
    trigger.write(triggerServo_HomePosition);
    digitalWrite(firingIndicatorLEDPin,LOW);
  }
  if(selector == 3) {
    digitalWrite(electricTriggerPin,LOW);
    trigger.write(triggerServo_HomePosition);
    digitalWrite(firingIndicatorLEDPin,LOW);
  }
}

void sequenceLEDs(int repeats,int delayTime) {
  int startDelay;
  for(int i = 0; i < repeats; i++) {

    digitalWrite(USBIndicatorLEDPin,LOW);
    digitalWrite(modeIndicatorLEDPin,LOW);

    startDelay =millis();
    while(millis()-startDelay < delayTime) {
      digitalWrite(firingIndicatorLEDPin,HIGH);
    }
    digitalWrite(firingIndicatorLEDPin,LOW);

    startDelay =millis();
    while(millis()-startDelay < delayTime) {
      digitalWrite(USBIndicatorLEDPin,HIGH);
    }
    digitalWrite(USBIndicatorLEDPin,LOW);

    startDelay =millis();
    while(millis()-startDelay < delayTime) {
      digitalWrite(modeIndicatorLEDPin,HIGH);
    }
    digitalWrite(modeIndicatorLEDPin,LOW);

    startDelay =millis();
    while(millis()-startDelay < delayTime) {
```

```
    }
  }
}

void assignPins() {
  if(type =="Arduino_bare"|| type =="Arduino_Bare") {
    panServoPin = 8;
    tiltServoPin = 9;
    triggerServoPin = 10
    firingIndicatorLEDPin = 12;
    USBIndicatorLEDPin = 11;
    modeIndicatorLEDPin = 13;
    reloadSwitchPin = 3;
    disablePlatePin = 2;
    electricTriggerPin = 7;
    invertInputs =true;
  }
}
```

10.2 Processing kod

```
publicint camWidth = 320;
publicint camHeight = 240;


boolean PRINT_FRAMERATE = false;

int[] diffPixelsColor = {
  255, 255, 0
};
publicint effect = 0;

publicboolean mirrorCam = false;

publicfloat xMin = 0.0;
publicfloat xMax = 180.0;
publicfloat yMin = 0.0;
publicfloat yMax = 180.0;

import JMyron.*;
import blobDetection.*;
import processing.serial.*;
import ddf.minim.*;
import java.awt.Frame;
import processing.opengl.*;
import procontroll.*;
import net.java.games.input.*;

publicint minBlobArea = 30;
publicint tolerance = 100;

publicboolean runWithoutArduino = false;
publicboolean connecting = false;


public Serial arduinoPort;
JMyron camInput;
BlobDetection target;
Blob blob;
Blob biggestBlob;


int[] Background;
int[] rawImage;
int[] rawBackground;
int[] currFrame;
int[] screenPixels;
publicint targetX = camWidth/2;
publicint targetY = camHeight/2;
int fire = 0;
int[] prevFire = {
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

float xRatio;
float yRatio;

int possibleX = camWidth/2;
```

```
int possibleY = camHeight/2;

int displayX = camWidth/2;
int displayY = camHeight/2;

int oldX = camWidth/2;
int oldY = camHeight/2;
int xdiff;
int ydiff;
publicfloat smoothingFactor = 0.8;
publicboolean activeSmoothing = true;

String strTargetx;
String strTargety;
String fireSelector;
String scanSelector;


publicboolean showDifferentPixels = false;
publicboolean showTargetBox = true;
publicboolean showCameraView = true;
publicboolean firingMode = true;
publicboolean safety = true;
publicboolean controlMode = false;
publicboolean soundEffects =  false;
publicboolean scanWhenIdle = true;
publicboolean trackingMotion = true;

int idleTime = 10000;
int idleBeginTime = 0;
boolean scan = false;

publicString serPortUsed;

int[][] fireRestrictedZones = newint[30][4];
int restrictedZone = 1;
boolean showRestrictedZones = false;

boolean selectingColor = false;
boolean trackingColor = false;
int trackColorTolerance = 100;
int trackColorRed = 255;
int trackColorGreen = 255;
int trackColorBlue = 255;

boolean selectingSafeColor = false;
boolean safeColor = false;
int safeColorMinSize = 500;
int safeColorTolerance = 100;
int safeColorRed = 0;
int safeColorGreen = 255;
int safeColorBlue = 0;

boolean useArrowKeys = false;

publicboolean useInputDevice = false;
publicboolean inputDeviceIsSetup = false;

public ControllIO controlIO;
public ControllDevice inputDevice;
```

```
public ControllButton[] buttons = new ControllButton[30];
public ControllSlider[] sliders = new ControllSlider[10];

public ControllButton[] fire_buttons = new ControllButton[0];
public ControllButton[] preciseAim_buttons = new ControllButton[0];
public ControllButton[] centerGun_buttons = new ControllButton[0];
public ControllButton[] autoOn_buttons = new ControllButton[0];
public ControllButton[] autoOff_buttons = new ControllButton[0];
public ControllButton[] inputToggle_buttons = new ControllButton[0];
public ControllButton[] randomSound_buttons = new ControllButton[0];

public ControllSlider[] pan_sliders = new ControllSlider[0];
public ControllSlider[] tilt_sliders = new ControllSlider[0];
public ControllSlider[] panInvert_sliders = new ControllSlider[0];
public ControllSlider[] tiltInvert_sliders = new ControllSlider[0];


publicfloat xPosition = camWidth/2;
publicfloat yPosition = camHeight/2;


String[] inStringSplit;
int controlMode_i, safety_i, firingMode_i, scanWhenIdle_i, trackingMotion_i,
trackingColor_i, leadTarget_i, safeColor_i,
showRestrictedZones_i, showDifferentPixels_i, showTargetBox_i, showCameraView
_i, mirrorCam_i, soundEffects_i;


voidsetup() {

  loadSettings();

  size(camWidth, camHeight);
  minim = new Minim(this);
  loadSounds();
  playSound(18);
  camInput = new JMyron();
  camInput.start(camWidth, camHeight);
  camInput.findGlobs(0);
  camInput.adaptivity(1.01);
  camInput.update();
  currFrame = camInput.image();
  rawImage = camInput.image();
  Background = camInput.image();
  rawBackground = camInput.image();
  screenPixels = camInput.image();
  target = new BlobDetection(camWidth, camHeight);
  target.setThreshold(0.9);
  target.setPosDiscrimination(true);

  retryArduinoConnect();

  xRatio = (camWidth / (xMax - xMin));
  yRatio = (camHeight/ (yMax - yMin));
  drawControlPanel();
}
```

```
voiddraw() {
  if (PRINT_FRAMERATE) {
    println(frameRate);
  }

  if (controlMode) {
    autonomousMode();
  }
  elseif (!controlMode) {
    manualMode();
  }

  if (fire == 1) {
    idleBeginTime = millis();
    scan = false;
  }
  else {
    if (millis() > idleBeginTime + idleTime && controlMode && scanWhenIdle) {
      scan = true;
    }
    else {
      scan = false;
    }
  }

  if (!safety) {
    fire = 0;
  }

  strTargetx = "000" + str(targetX);
  strTargetx = strTargetx.substring(strTargetx.length()-3);
  strTargety = "000" + str(targetY);
  strTargety = strTargety.substring(strTargety.length()-3);
  fireSelector = str(0);
  if (firingMode) {
    fireSelector = str(1);
  }
  else {
    fireSelector = str(3);
  }
  if (scan) {
    scanSelector = str(1);
  }
  else {
    scanSelector = str(0);
  }
  if (!runWithoutArduino && !connecting) {
    arduinoPort.write('a' + strTargetx + strTargety + str(fire) +
fireSelector + scanSelector);
  }

  if ((keyPressed&&key == 't') || showRestrictedZones) {
    for (int col = 0; col <= restrictedZone; col++) {
      noStroke();
      fill(0, 255, 0, 100);
      rect(fireRestrictedZones[col][0], fireRestrictedZones[col][2],
fireRestrictedZones[col][1]-fireRestrictedZones[col][0],
fireRestrictedZones[col][3]-fireRestrictedZones[col][2]);
    }
  }
```

```
  if (selectingColor) {
    stroke(190, 0, 190);
    strokeWeight(2);
    fill(red(currFrame[(mouseY*width)+mouseX]),
green(currFrame[(mouseY*width)+mouseX]),
blue(currFrame[(mouseY*width)+mouseX]));
    rect(mouseX+2, mouseY+2, 30, 30);
  }

  if (selectingSafeColor) {
    stroke(0, 255, 0);
    strokeWeight(2);
    fill(red(currFrame[(mouseY*width)+mouseX]),
green(currFrame[(mouseY*width)+mouseX]),
blue(currFrame[(mouseY*width)+mouseX]));
    rect(mouseX+2, mouseY+2, 30, 30);
  }

  soundTimer++;
  if (soundTimer == soundInterval) {
    randomIdleSound();
    soundTimer = 0;
  }

  for (int i = 9; i >= 1; i--) {
    prevFire[i] = prevFire[i-1];
  }
  prevFire[0] = fire;
  int sumNewFire = prevFire[0] + prevFire[1] + prevFire[2] + prevFire[3] +
prevFire[4];
  int sumPrevFire = prevFire[5] + prevFire[6] + prevFire[7] + prevFire[8] +
prevFire[9];

  if (sumNewFire == 0 && sumPrevFire == 5) {
    int s = int(random(0, 6));
    if (s == 0)
      playSound(1);
    if (s == 1)
      playSound(5);
    if (s == 2)
      playSound(9);
    if (s == 3)
      playSound(12);
    if (s == 4)
      playSound(13);
    if (s == 5)
      playSound(20);
  }

  if (fire == 1)
    strokeWeight(3);
  if (fire == 0)
    strokeWeight(1);
  stroke(255, 0, 0);
  noFill();
  line(displayX, 0, displayX, camHeight);
  line(0, displayY, camWidth, displayY);
  ellipse(displayX, displayY, 20, 20);
  ellipse(displayX, displayY, 28, 22);
  ellipse(displayX, displayY, 36, 24);
```

```
  updateControlPanels();
  prevTargetX = targetX;
  prevTargetY = targetY;
}

void autonomousMode() {
  if(inputDeviceIsSetup) {
    checkInputDevice();
  }

  if (selectingColor || selectingSafeColor) {
    cursor(1);
  }
  else {
    cursor(0);
  }
  camInput.update();
  rawBackground = camInput.retinaImage();
  rawImage = camInput.image();
  if (mirrorCam) {
    for (int i = 0; i < camWidth*camHeight; i++) {
      int y = floor(i/camWidth);
      int x = i - (y*camWidth);
      x = camWidth-x;
      currFrame[i] = rawImage[(y*camWidth) + x-1];
      Background[i] = rawBackground[(y*camWidth) + x-1];
    }
  }
  else {
    currFrame = rawImage;
    Background = rawBackground;
  }

  loadPixels();
  int safeColorPixelsCounter = 0;

  for (int i = 0; i < camWidth*camHeight; i++) {
    if (showCameraView) {
      pixels[i] = currFrame[i];
    }
    else {
      pixels[i] = color(0, 0, 0);
    }

    boolean motion = (((abs(red(currFrame[i])-red(Background[i])) +
abs(green(currFrame[i])-green(Background[i])) + abs(blue(currFrame[i])-
blue(Background[i]))) > (200-tolerance)) && trackingMotion);
    boolean isTrackedColor = (((abs(red(currFrame[i])-trackColorRed) +
abs(green(currFrame[i])-trackColorGreen) + abs(blue(currFrame[i])-
trackColorBlue)) < trackColorTolerance) && trackingColor);

    boolean isSafeColor = (((abs(red(currFrame[i])-safeColorRed) +
abs(green(currFrame[i])-safeColorGreen) + abs(blue(currFrame[i])-
safeColorBlue)) < safeColorTolerance) && safeColor);

    if (motion || isTrackedColor) {
      screenPixels[i] = color(255, 255, 255);
      if (showDifferentPixels) {
        if (effect == 0) {
```

```
          pixels[i] = color(diffPixelsColor[0], diffPixelsColor[1],
diffPixelsColor[2]);
        }
        elseif (effect == 1) {
          pixels[i] = color((diffPixelsColor[0] + red(currFrame[i]))/2,
(diffPixelsColor[1] + green(currFrame[i]))/2, (diffPixelsColor[2] +
blue(currFrame[i]))/2);
        }
        elseif (effect == 2) {
          pixels[i] = color(255-red(currFrame[i]), 255-green(currFrame[i]),
255-blue(currFrame[i]));
        }
        elseif (effect == 3) {
          pixels[i] = color((diffPixelsColor[0] + (255-red(currFrame[i])))/2,
(diffPixelsColor[1] + (255-green(currFrame[i])))/2, (diffPixelsColor[2] +
(255-blue(currFrame[i])))/2);
        }
      }
    }
    else {
      screenPixels[i] = color(0, 0, 0);
    }

    if (isSafeColor) {
      safeColorPixelsCounter++;
      pixels[i] = color(0, 255, 0);
      screenPixels[i] = color(0, 0, 0);
    }
  }



  updatePixels();

  int biggestBlobArea = 0;
  target.computeBlobs(screenPixels);
  for (int i = 0; i < target.getBlobNb()-1; i++) {
    blob = target.getBlob(i);
    int blobWidth = int(blob.w*camWidth);
    int blobHeight = int(blob.h*camHeight);
    if (blobWidth*blobHeight >= biggestBlobArea) {
      biggestBlob = target.getBlob(i);
      biggestBlobArea = int(biggestBlob.w*camWidth)*int(biggestBlob.h*camHeig
ht);
    }
  }
  possibleX = 0;
  possibleY = 0;

  if (biggestBlobArea >= minBlobArea) {
    possibleX = int(biggestBlob.x * camWidth);
    possibleY = int(biggestBlob.y * camHeight);
  }


  if ((biggestBlobArea >= minBlobArea)) {
    fire = 1;
    if (showTargetBox) {
      stroke(255, 50, 50);
      strokeWeight(3);
```

```
      fill(255, 50, 50, 150);
      rect(int(biggestBlob.xMin*camWidth), int(biggestBlob.yMin*camHeight),
int((biggestBlob.xMax-biggestBlob.xMin)*camWidth), int((biggestBlob.yMax-
biggestBlob.yMin)*camHeight));
    }

    anticipation();

    if (activeSmoothing) {
      xdiff = possibleX - oldX;
      ydiff = possibleY - oldY;
      possibleX = int(oldX + xdiff*(1.0-smoothingFactor));
      possibleY = int(oldY + ydiff*(1.0-smoothingFactor));
    }

    displayX = possibleX;
    displayY = possibleY;
    if (displayX < 0)
      displayX = 0;
    if (displayX > camWidth)
      displayX = camWidth;
    if (displayY < 0)
      displayY = 0;
    if (displayY > camHeight)
      displayY = 0;
    targetX = int((possibleX/xRatio)+xMin);
    targetY = int(((camHeight-possibleY)/yRatio)+yMin);
    oldX = possibleX;
    oldY = possibleY;
  }
  else {
    fire = 0;
  }

  boolean clearOfZones = true;
  for (int col = 0; col <= restrictedZone; col++) {
    if (possibleX > fireRestrictedZones[col][0] && possibleX <
fireRestrictedZones[col][1] && possibleY > fireRestrictedZones[col][2] &&
possibleY < fireRestrictedZones[col][3]) {
      clearOfZones = false;
      fire = 0;
    }
  }


  if (safeColorPixelsCounter > safeColorMinSize && safeColor) {
    noStroke();
    fill(0, 255, 0, 150);
    rect(0, 0, width, height);
    fire = 0;
    targetX = int((xMin+xMax)/2.0);
    targetY = int(yMin);
    displayX = camWidth/2;
    displayY = camHeight;
  }
}

void manualMode() {
  camInput.update();
  rawBackground = camInput.retinaImage();
```

```
rawImage = camInput.image();
if (mirrorCam) {
  for (int i = 0; i < camWidth*camHeight; i++) {
    int y = floor(i/camWidth);
    int x = i - (y*camWidth);
    x = camWidth-x;
    currFrame[i] = rawImage[(y*camWidth) + x-1];
    Background[i] = rawBackground[(y*camWidth) + x-1];
  }
}
else {
  currFrame = rawImage;
  Background = rawBackground;
}

loadPixels();
for (int i = 0; i < camWidth*camHeight; i++) {
  pixels[i] = currFrame[i];
}
updatePixels();

if(inputDeviceIsSetup) {
  checkInputDevice();
}
if(useInputDevice) {
  updateInputDevice();
  if(useArrowKeys) {
    if(keyPressed) {
      if (keyCode == 37) {
        xPosition -= 1;
      }
      if (keyCode == 38) {
        yPosition -= 1;
      }
      if (keyCode == 39) {
        xPosition += 1;
      }
      if (keyCode == 40) {
        yPosition += 1;
      }
      fire = 0;

    }
  }
}else{
  if(useArrowKeys) {
    if(keyPressed) {
      if (keyCode == 37) {
        displayX -= 1;
      }
      if (keyCode == 38) {
       displayY -= 1;
      }
      if (keyCode == 39) {
        displayX += 1;
      }
      if (keyCode == 40) {
        displayY += 1;
      }
      fire = 0;
```

```
      }
    }else{
      displayX = mouseX;
      displayY = mouseY;
      if (mousePressed) {
        fire = 1;
      }
      else {
        fire = 0;
      }
    }
    targetX = constrain(int((displayX/xRatio)+xMin), 0, 180);

    targetY = constrain(int(((camHeight-displayY)/yRatio)+yMin), 0, 180);

  }
}


voidmousePressed() {
  if (keyPressed&&key == 'r') {
    print("constraints:" + mouseX + ", " + mouseY);
    fireRestrictedZones[restrictedZone][0] = mouseX;
    fireRestrictedZones[restrictedZone][2] = mouseY;
  }
  if (selectingColor) {
    trackColorRed = int(red(currFrame[(mouseY*width)+mouseX]));
    trackColorBlue = int(blue(currFrame[(mouseY*width)+mouseX]));
    trackColorGreen = int(green(currFrame[(mouseY*width)+mouseX]));
    selectingColor = false;
  }

  if (selectingSafeColor) {
    safeColorRed = int(red(currFrame[(mouseY*width)+mouseX]));
    safeColorBlue = int(blue(currFrame[(mouseY*width)+mouseX]));
    safeColorGreen = int(green(currFrame[(mouseY*width)+mouseX]));
    selectingSafeColor = false;
  }
}

voidmouseReleased() {
  if (keyPressed&&key == 'r') {
    println(" ... " + mouseX + ", " + mouseY);
    fireRestrictedZones[restrictedZone][1] = mouseX;
    fireRestrictedZones[restrictedZone][3] = mouseY;
    if
(fireRestrictedZones[restrictedZone][1]>fireRestrictedZones[restrictedZone][0
] &&
fireRestrictedZones[restrictedZone][1]>fireRestrictedZones[restrictedZone][2]
) {
      restrictedZone++;
    }
  }
}

voidkeyReleased() {
  if ( key == 'p') {
    randomIdleSound();
  }
```

```
  if (key == ' ') {
    controlMode = !controlMode;
  }

  if (key == 'b') {
    camInput.adapt();
    playSound(15);
  }
  if (key == 'a') {
    xMin = float(targetX);
    xRatio = (camWidth / (xMax - xMin));
  }
  if (key == 'd') {
    xMax = float(targetX);
    xRatio = (camWidth / (xMax - xMin));
  }
  if (key == 's') {
    yMin = float(targetY);
    yRatio = (camHeight/ (yMax - yMin));
  }
  if (key == 'w') {
    yMax = float(targetY);
    yRatio = (camHeight/ (yMax - yMin));
  }
  if(key == CODED&&keyCode == 16) {
    useArrowKeys = !useArrowKeys;
  }

}



publicvoid viewCameraSettings() {
  camInput.settings();
  playSound(21);
}

publicvoid openWebsite() {
  link("http://psg.rudolphlabs.com/");
  playSound(15);
}

publicvoid setBackground() {
  camInput.adapt();
  playSound(11);
}

publicvoid playRandomSound() {
  randomIdleSound();
}

publicvoid selectColor() {
  selectingColor = true;
}

publicvoid selectSafeColor() {
  selectingSafeColor = true;
}

publicvoid radioEffect(int ID) {
```

```
  effect = ID + 1;
}


publicvoid stop() {
  if(soundEffects) {
    s1.rewind();
    s1.play();
    delay(2500);
    s1.close();
    s2.close();
    s3.close();
    s4.close();
    s5.close();
    s7.close();
    s6.close();
    s8.close();
    s9.close();
    s10.close();
    s11.close();
    s12.close();
    s13.close();
    s14.close();
    s15.close();
    s16.close();
    s17.close();
    s18.close();
    s19.close();
    s20.close();
    s21.close();
    minim.stop();
  }
  if (!runWithoutArduino) {
    arduinoPort.write("z0000000");
    delay(500);
    arduinoPort.stop();
  }
  camInput.stop();
  super.stop();
}
```

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 25. 09. 2014.	ANTONIO NIKOLIĆ	

Prema Odluci Visoke tehničke škole u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Visoke tehničke škole u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

ANTONIO NIKOLIĆ

ime i prezime studenta ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada

U Bjelovaru, 25.09.2017.

potpis studenta ice