Digitalna vaga za mjerenje dinamičkog elisnog potiska

Palčok, Juraj

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

DIGITALNA VAGA ZA MJERENJE DINAMIČKOG ELISNOG POTISKA

Završni rad

Juraj Palčok

Osijek, 2017.

1.UVOD

U današnje vrijeme dronovi postaju sve popularniji i osobe koje se bave s tim zanimljivim hobijem imaju potrebu usavršiti performanse svojih naprednih igračaka. Iz tog razloga odabran je završni rad "Digitalna vaga za mjerenje dinamičkog elisnog potiska". Ovaj uređaj omogućavat će korisniku da se uvjeri u stvarne performanse motora koji su ujedno i najvažniji dio drona. Na tržištu već postoje ovakve vage koje se mogu kupiti za svega 50-60 $ [1] što je pristupačna cijena i za one malo plićeg džepa. Postoje i improvizirane verzije u kojima se motori pričvršćuju na kuhinjske vage te pomoću vage mjere potisak motora, ali pošto ti motori postižu brzine i do 10000 o/min što je opasno ako nešto krene kako ne treba. Većina ovakvih sustava za mjerenje omogućuju samo prikaz potiska bez ikakve ovisnosti o brzini vrtnje. Ideja je izraditi maketu koja će uz ovo što već rade postojeći sustavi imati mogućnost detaljnijeg prikaza karakteristika motora. Za ovaj rad je zamišljeno koristiti mikroupravljač ATMega 328P (Arduino nano) koji je relativno jeftin i jednostavan za korištenje. Mjerenje će se raditi preko softvera koji je izrađen u Microsoft Visual studiu. Seminarski rad će sadržavati opis postojećih sustava i uređaja te podrobniji opis same izrade makete te razvijanje sučelja za upravljanje..

1.1.Zadatak i ciljevi rada

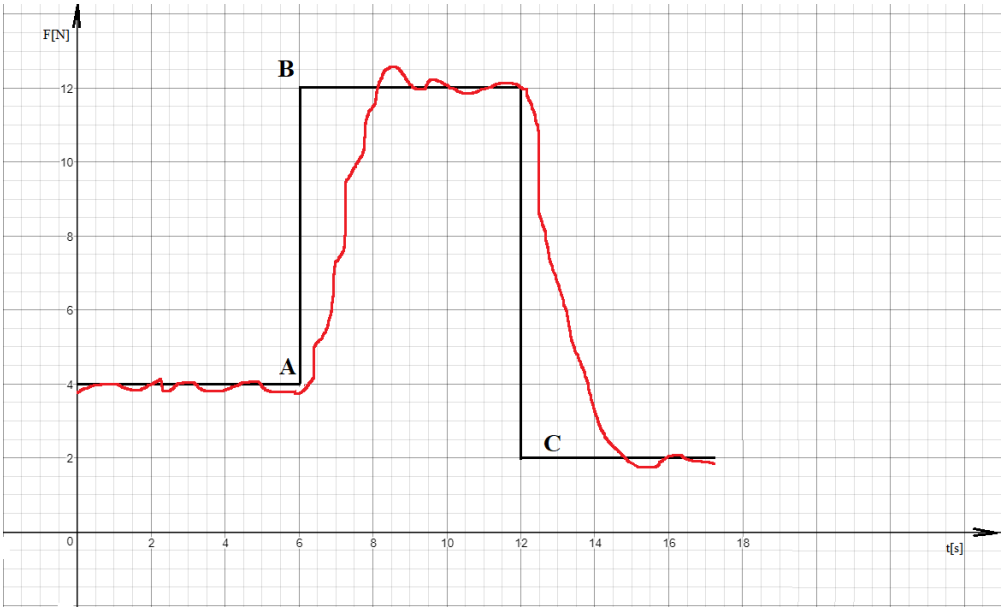
Zadatak rada je izraditi maketu digitalne vage za mjerenje dinamičkog elisnog potiska, izraditi softver za upravljanje te snimiti dinamičku ovisnost potiska u gramima i brzine vrtnje motora. Prikazati odnos u obliku grafa na računalu i usporediti izmjereno s očekivanim te na temelju toga odlučiti dali je vaga dovoljno precizna za upotrebu.

2.MJERENJE DINAMIČKOG ELISNOG POTISKA

Mjerenje dinamičkog elisnog potiska se može izvesti s vertikalnim ili horizontalnim rotorom motora, ali postupak mora biti isti jer se promatra vrijeme koje je potrebno da se neka promjena ostvari.

2.1.Dinamički elisni potisak

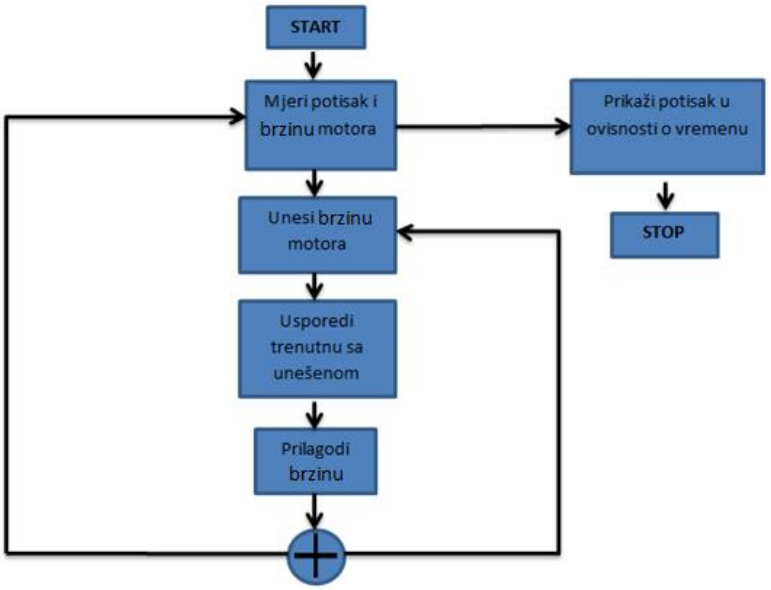
Same dinamičke karakteristike nam govore (izražavaju) sposobnost nekog uređaja da iz položaja A dođe u položaj B. U ovome slučaju to je neka početna snaga i krajnja snaga motora. Najprecizniji uvid u dinamičke karakteristike motora dobivamo ako karakteristika koja sadrži prijelaz iz točke A u točku B sadrži i povratak u točku A ili neku treću C (slika 2.1.). Na grafu se razlikuju dvije karakteristike. Karakteristika iscrtana crnom bojom predstavlja idealnu dok crvena predstavlja stvarno ponašanje motora. Ovakva karakteristika nam je beskorisna ako ne sadrži vrijeme koje je potrebno da bi se ti prijelazi odradili, tj. ona mora prikazati točno vrijeme trajanja prijelaznih stanja. Iz grafa se uočava da idealna karakteristika iz točke A u B prijelazi u trenutku tj. nema kašnjenja dok crvena ima kašnjenje od 2 sekunde. Kod prijelaza iz stanja B u C prijelaz je veći te samim time imamo i kašnjenje koje je veće od dvije sekunde. Takva karakteristika korisniku prikazuje podatke koji su važni pri samom odabiru elise jer sadrži potpuni prikaz rasta i pada potiska.



Slika 2.1. Dinamička karakteristika ovisnost sile [N] i vremena [s].

2.2.Sustav za mjerenje potiska, upravljanje i algoritam upravljanja

Sustav za mjerenje potiska trebao bi sadržavati vagu za mjerenje, algoritam za upravljanje i dio za prikaz mjerenja. Ta tri sustava se mogu realizirati na više načina ali njihov glavni zadatak je uvijek isti. Takav sustav najčešće je izveden s analognom vagom koja može direktno prikazivati izmjereno ili može izmjereno preko A/D pretvornika transformirati u digitalni signal. Ovisno o tipu vage postoje ograničenja pri prikazu izmjerenog podatka. U slučaju pretvorbe podatka u digitalni nastaju gubici i javlja se kašnjenje koje je gotovo zanemarivo ali ipak postaji te usporava i čini mjerenje nesigurnijim. Neovisno o upotrebljenoj vazi za mjerenje njen zadatak je izmjeriti vrijednost i tu vrijednost poslati na daljnju obradu. Upravljački sustav se izvodi preko kontrolera koji je programabilan i na sebi sadrži program za upravljanje. Postoje i jednostavnija rješenja kao što su npr. kontroleri u obliku džojstika koji imaju mogućnost upravljanja motorom, ali takvi sustavi ne sadrže nikakvu povratnu informaciju. Zaključeno je da upravljanje džojstikom nije dovoljno te da nam je potreban malo složeniji sustav upravljanja. U nastavku će biti objašnjen glavni postupak i način na koji bi ovakvi sustavi trebali raditi.

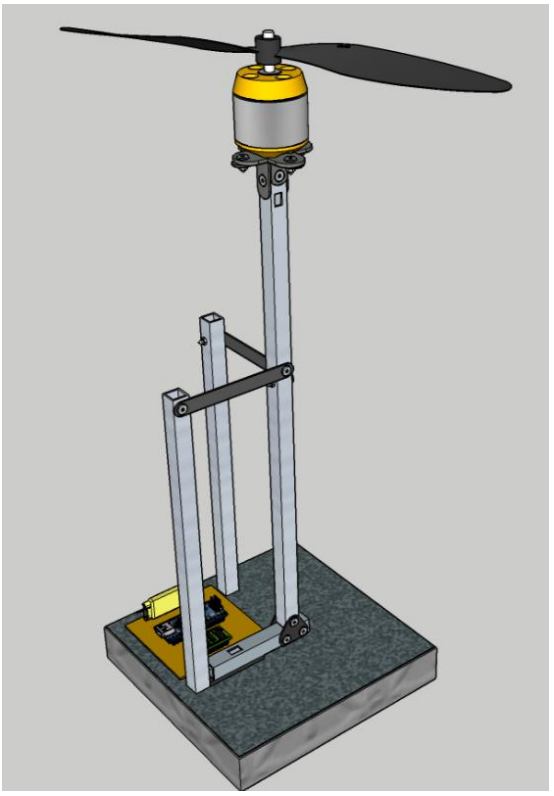


Slika 2.2 Sustav za mjerenje, upravljanje i prikaz

Na slici 2.2 prikazan je jednostavan sustav za mjerenje, upravljanje i prikaz izmjerenog. Od sustava se zahtjeva da korisnik stalno ima uvid u mjerenje stoga se uzorkovanje uzima na samome početku. Korisnik stalno može manipulirati mjerenjem i to tako da mijenja brzinu motor. Kada korisnik unese željenu brzinu sustav je usporedi s trenutnom te prema potrebi smanji ili poveća.

3.SUSTAV ZA MJERENJE DINAMIČKOG ELISNOG POTISKA

Nakon dugog proučavanja odlučeno je da će sustav za mjerenje dinamičkog elisnog potiska sadržavati postolje na kojem će motor biti pričvršćen s vertikalnim rotorom te će u podnožju sadržavati vagu i hardver (slika 3.1). Razlog postavljanja motora u vertikalan položaj prvenstveno je zato što su motori tako postavljenu na dronovima. Drugi razlog je taj što u tome slučaju možemo postaviti elisu velikog radijusa krakova. Sustav za upravljanje će biti izrađen u računalnom sučelju Arduino IDE i učitan na mikrokontroler te će s kabelskom vezom biti spojen s računalom preko kojeg će korisnik moći upravljati sustavom. Program za upravljanje će biti izrađen u Microsoft visual studio zato što se oba programska sučelja temelje na C programskom jeziku i može se ostvariti međusobna komunikacija. Ovo poglavlje sadržavat će detaljan opis postupka izrade same makete te elemenata koji su korišteni.



Slika 3.1 3D izgled zamišljene makete.

3.1 Izrada hardvera i programska podrška

Zamišljenu maketu potrebno je realizirati te izraditi hardver i napisati program za upravljanje. Odlučeno je izraditi tiskanu pločicu te na nju polemiti elemente hardvera, a program će biti izrađen u dva dijela. Prvi dio programa će sadržavati algoritam upravljanja dok će drugi

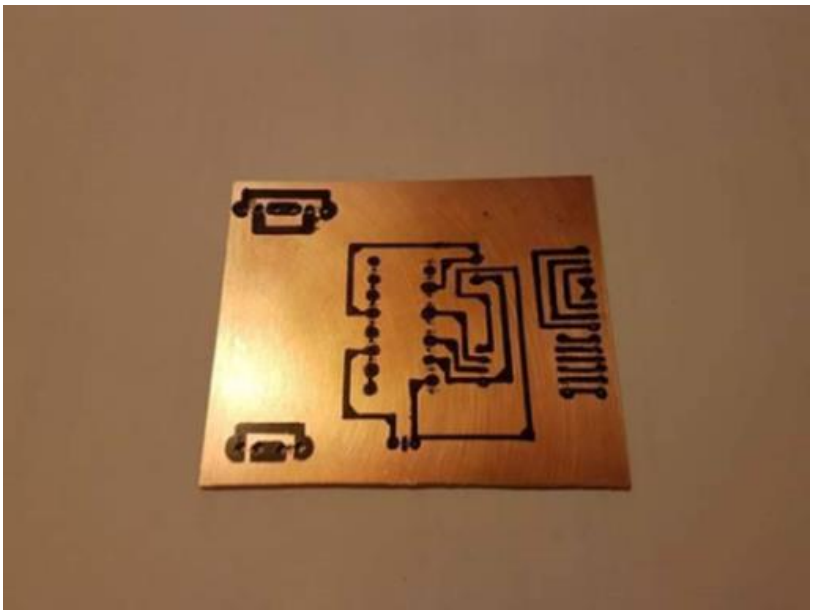
dio služiti za upravljanje i prikaz mjerenja. U nastavku će biti detaljnije navedeno o načinu izrade hardvera i razvoja programske podrške.

3.1.1. Izrada hardvera

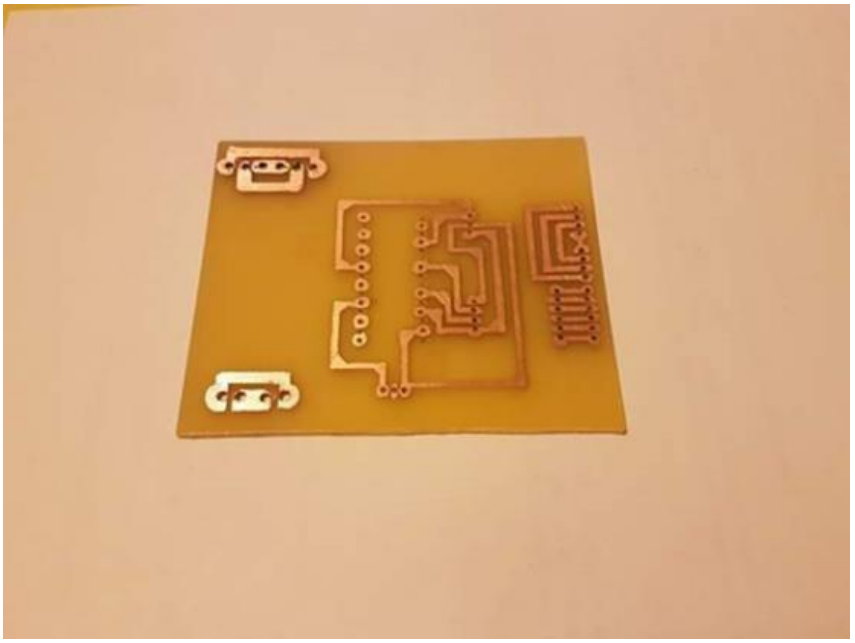
Međusobna povezanost elemenata hardvera izvedena je izradom tiskane pločice koja je izrađena na računalnom programu EAGLE [2]. U eaglu je prvo izrađena električka shema (prilog 5.) s potrebnim elementima te je ona prerađena u virtualni izgled (prilog 4.) same pločice. Nakon što je virtualna pločica izrađena potrebno ju je prenijeti na tiskanu pločicu tako da vodljivi dijelovi ostanu zatamnjeni a ostatak proziran. Postoje razni načini označavanja vodljivih dijelova na pločicu npr. osvjetljavanje, sa peglom, crtanje alkoholnim markerom i dr.. Označenu (slika 3.2.) pločicu potrebno je izjetkati.

Jetkanje [3] je process nakon kojeg na pločici ostanu samo vodljivi dijelovi koji su bili zatamnjeni. Jetkanje se najčešće na konvencionalan način vrši pomoću nagrizajuće otopine koja sadržava vodu, vodikov peroksid odnosno hydrogen (H_2O_2) i solnu kiselinu i to u omjeru 0.5 : 0.15 : 0.2. Jako važno je da se otopina miješa u plastičnoj posudi. Pločicu je potrebno uroniti u otopinu i sačekati (cca 15-tak min.) sve dok svi nepotrebni vodljivi dijelovi nestanu. Izjetkanu pločicu (slika 3.3) potrebno je izbušiti na za to predviđenim mjestima upotrebom bušilice i svrdla promjera 1mm.

Nakon što je pločica završena potrebno ju je očistiti i postaviti elemente na za to predviđeno mjesto. Postavljene elemente potrebno je polemiti kako bi se ostvario električni spoj između pojedinih elemenata hardvera. Lemljene je postupak u kojem pomoću lemne legure (60% kositar 39%, olovo 1%, bakar) i lemilice koja topi tu leguru i zaljeva kontakte te ostvaruje električki spoj.



Slika 3.2. Otiskana pločica alkoholnim markerom.



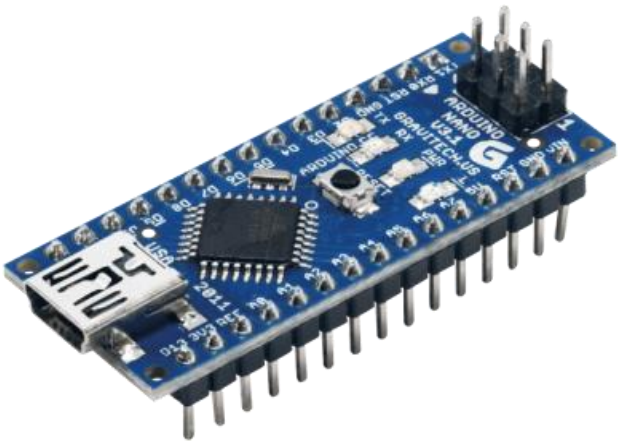
Slika 3.3. Izjetkana pločica.

Elementi koje hardver (slika 3.9) sdrži su:

- arduino nano
- ćelija za mjerenje sile (load cell)
- hx711
- elektronički kontroler brzine -30A

3.1.2. Arduino Nano

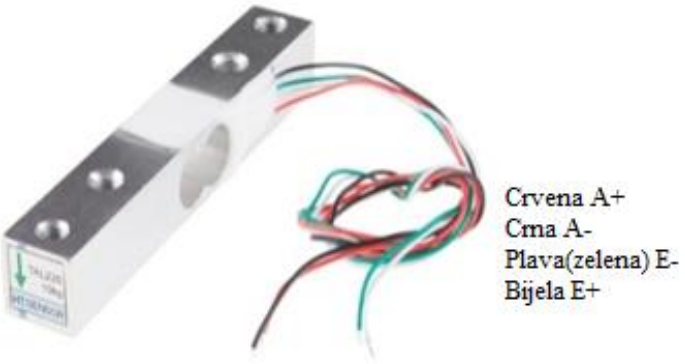
Arduino Nano (slika 3.4.) jest otvoreni softver ili u engleskom originalu Open source čiji je izvorni kod dostupan svim korisnicima koji ga mogu mijenjati, prepravljati i poboljšavati njegov sadržaj. To znači da uz 'open source' programe dolazi i čitav izvorni kod u nekom programskom jeziku, pa se može i mijenjati sam program. Koristi se u raznim granama industrije od elektronike pa sve do medicine. Ima trideset priključnica, jedan USB ulaz, jedan ICSP ulaz te tipku za resetiranje. Od 30 priključnica koliko ih Arduino sadrži za primanje/slanje, dvanaest njih su digitalni, a osam njih služi za analogne signale. Arduino također sadrži dva serijska ulaza (TX/RX) i ostalih se pet koristi za napajanja. Arduino nano pokreče 8-bitni mikrokontroler ATMega328p (više o samom mikrokontroleru u prilogu 1.).



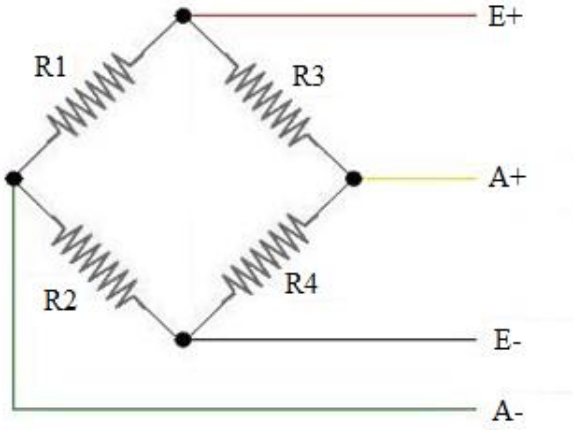
Slika 3.4. Arduino nano.

3.1.3. Ćelija za mjerenje sile

Ćelija za mjerenje sile ili na engleskom Load cell (slika 3.5.) je pretvarač koji se koristi za stvaranje električnog signala čija je veličina izravno proporcionalna sili koja se mjeri. Ćelija za mjerenje sile upotrebu nalazi u izradi uređaja za mjerenje sile (npr. vaga, dinamometar i dr.). Ćelija za mjerenje sile zadrži mehaničku konstrukciju od četiri dijela u sklopu Weatston-ovog mosta (slika 3.6.), a sila koja se mjeri deformira jedan dio. Deformacija materijala mijenja električki otpor, što je mjera sile koja djeluje na nju. Ćelija za mjerenje sile obično dolazi u konfiguraciji od četiri žice. Žice označene s E+ i E- označavaju pobudu, dok žice s oznakama A+ i A- označavaju signale. U idealnom slučaju razlika napona između A+ i A- bez opterećenja iznosi nula, te raste proporcionalno mehaničkom opterećenju ćelije za mjerenje sile.



Slika 3.5. Ćelija za mjerenje sile.



Slika 3.6. Električka shema weanstonovog mosta.

3.1.4.HX711 A/D pretvarač

HX711 (slika 3.7.) je 24-bitni analogno digitalni pretvarač ADC koji male ulazne analogne napone pretvara u digitalni oblik s kojime onda može komunicirati s mikrokontrolerom. Za komunikaciju s mikrokontrolerom potrebne su dvije IO linije (DT, SCK). Mikrokontroler ne bi mogao registrirati tako male napone stoga HX711[7] ima ugrađeno pojačalo s pojačanjem od 32, 64 ili 128. Postoje i dva kanala (izlaza), navedena pojačanja vrijede za kanal A, dok je za B moguće fiksno pojačanje od 32. Ovakav integriran krug kreiran je specifično za ćelije za mjerenje sile te se najčešće i koristi u kombinaciji s njima. Radi na naponu 2.6-5,5 v i dolazi u SOP16L tipu enkapsulacije čipa.

Slika 3.7. HX711 A/D pretvarač.

3.1.5. Elektronički kontroler brzine 30A

Elektronički kontroler brzine ili ESC (slika 3.8.) je elektronički sklop s namjerom da mijenja brzinu elektromotora, njegov smjer ili eventualno djeluje kao dinamička kočnica. Elektronički kontroleri brzine se često koriste na električno pogonjenim radio kontroliranim modelima, a varijacija koja se najčešće upotrebljava za motore bez četkica u osnovi osigurava elektronski generiran trofazni izvor električne energije niskog napona za motor. ESC može biti samostalna jedinica koja se priključuje na kanal za upravljanje gasom prijamnika ili ugrađen u sam prijemnik.

Slika 3.8. Električki kontroler brzine (ESC).

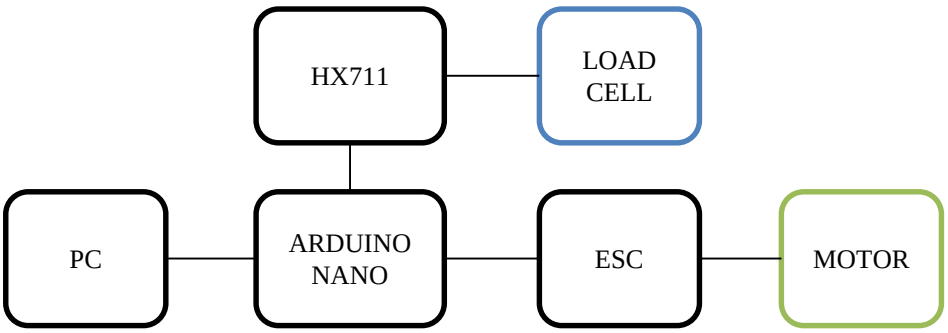
Slika 3.9. Hardver.

3.1.6. Programska podrška

Softver se sastoji od dva dijela. Prvi dio je sam upravljački program Arduina koji je napisan u Arduino IDE [4] programskom okruženju dok je drugi dio aplikacija izrađena u visual studio 2012 [5] koja omogućuje prikaz mjerenja i upravljanje vagom.

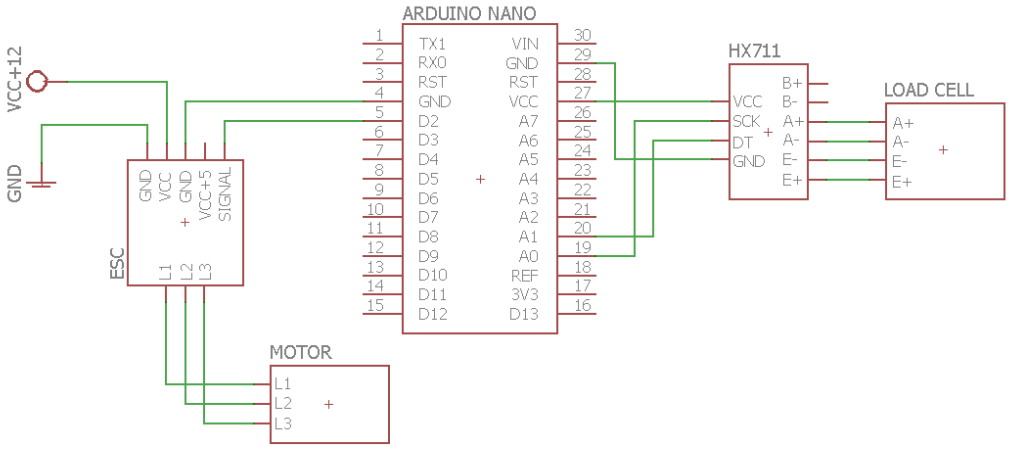
3.2. Električka shema sustava

Na slici 3.10. je prikazana električka blok shema digitalne vage. Radi lakšeg snalaženja izražena je u tri boje. Plava boja označava uređaj za mjerenje, zelena uređaj s kojim se upravlja dok crna predstavlja sustav za upravljanje vagom.



Slika 3.10. Blok dijagram električke sheme digitalne vage.

Blok “PC” predstavlja računalo na kojem se nalazi softverski program koji prikazuje rezultate mjerenja i pruža lako upravljanje vagom. Veza računala s blokom “ARDUINO NANO” ostvarena je usb kablom B mini tipa čija je maksimalna signalna brzina 480 Mbit/s. Brzina od 480 Mbit/s zadovoljava potrebe mjerenja stoga je veza ostvarena tako, a ne neki drugi. Prije samog rada Arduino kod (prilog 3.) se s računala šalje na mikrokontroler Arduino nano te je s time korakom vaga spremna za upotrebu. “ESC” (Elektronički kontroler brzine) predstavlja uređaj s vlastitim napajanje koji omogućuje spajanje motora na digitalni ulaz Arduina. Ovaj uređaj služi kao električni regulator brzine a radi tako da primi signal sa Arduina te prema jačini tog signala pojačava/smanjuje brzinu vrtnje motora, opširnije o samom uređaju u nastavku. Trofazni motor se “oprezno” spaja na ESC, oprezno jer zamjenom faza motor se vrti u suprotnom smjeru. Ćelija z amjerenje sile radi na principu Winstonovog mosta te izmjerenu vrijednost u obliku analognog napona šalje na analogni-digitalno pretvornik HX711 koji dobivenu digitalnu vrijednost šalje na digitalni ulaz Arduina nano.

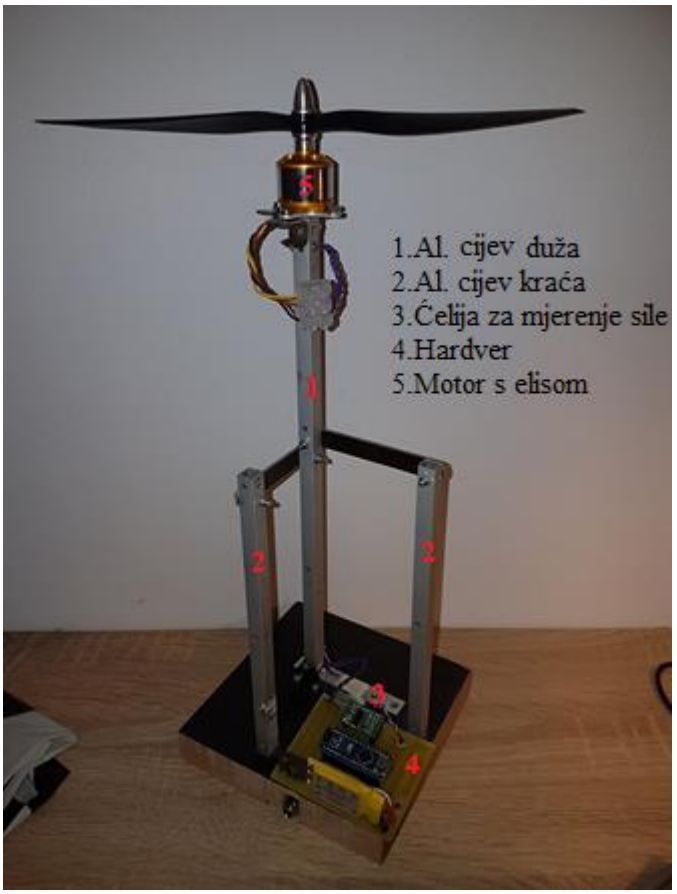


Slika 3.11. Električka shema digitalne vage.

Na slici 3.11. prikazana je detaljna električka shema digitalne vage sa svim elementima. Ova shema je izrađena u računalnom program Eagle. Arduino nano u shemi nije spojen na napajanje zato što se napaja preko usb kabela preko kojeg je komunikacija s računalom ostvarena.

3.3. Mehanički ustroj

Za samo podnožje makete korišteno je drvo dimenzija 150x150x25 mm. Na podnožju se nalazi ćelija za mjerenje sile, hardver te aluminijska konstrukcija. Aluminijska konstrukcija izrađena je od aluminijske cijevi (dvije) četvrtastog prijesjeka 200x10x10 mm te jedne duže dimenzija 300x10x10 mm koja je iskorištena kao čvrsta veza motora i vage. Fizički izgled i način na koji je maketa složena možete pogledati na slici 3.12..

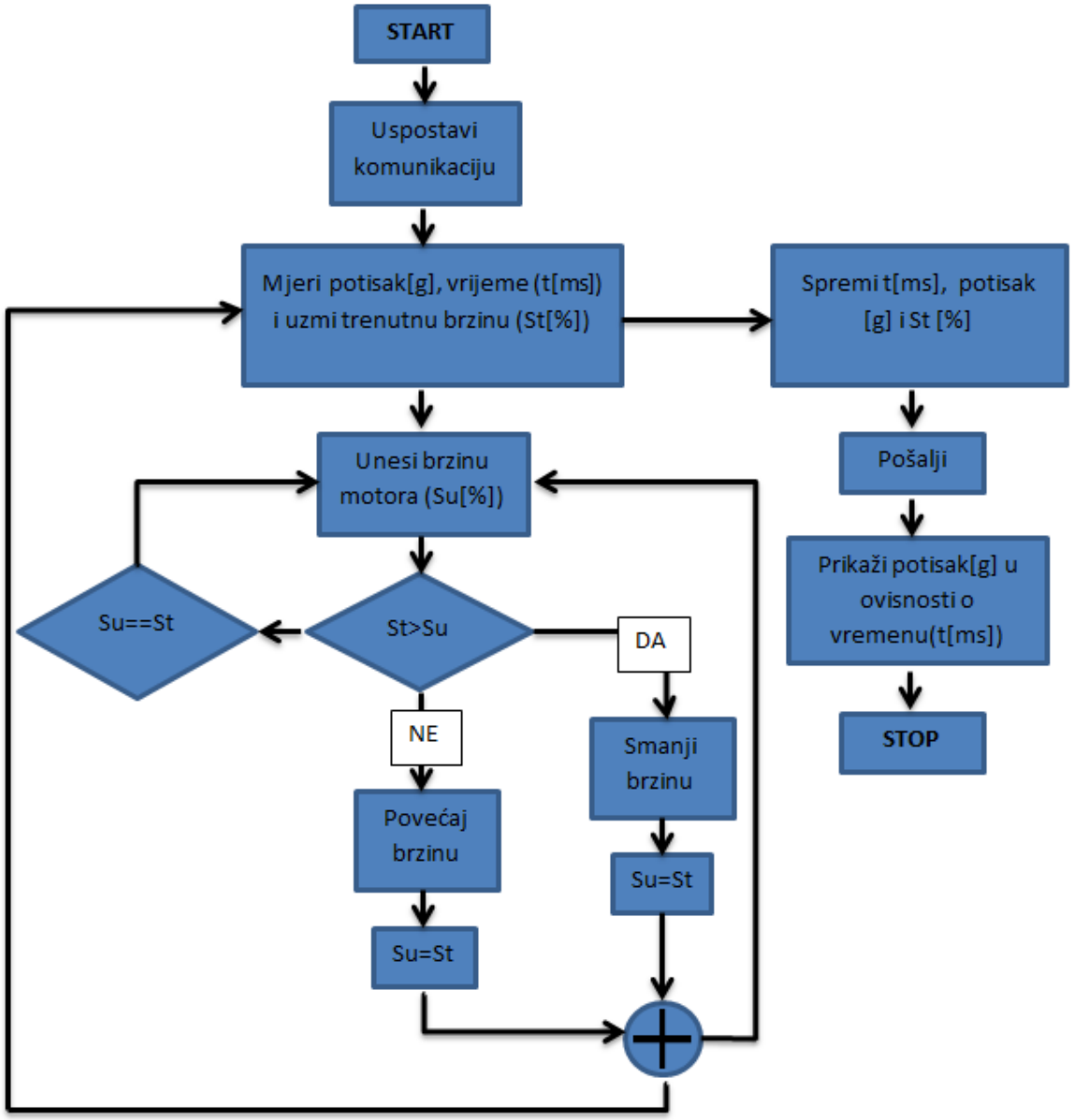


Slika 3.12. Maketa digitalne vage.

3.4. Algoritam upravljanja

Algoritam upravljanja izrađen je u Arduino IDE programskom sučelju (slika 3.13.) koje se temelji na C programskom jeziku. Ćelija za mjerenje sile pri komunikaciji sa Arduinom koristi HX711 pretvarač koji se u Arduino program mora umetnuti u obliku hx711 knjižnice (library, vidi prilog 2 i 2.1) u suprotnome program neće raditi. Unutar knjižnice pohranjen je unaprijed zadan način rada pojedinog elementa. Knjižnice služe da bi nama olakšale i ubrzale

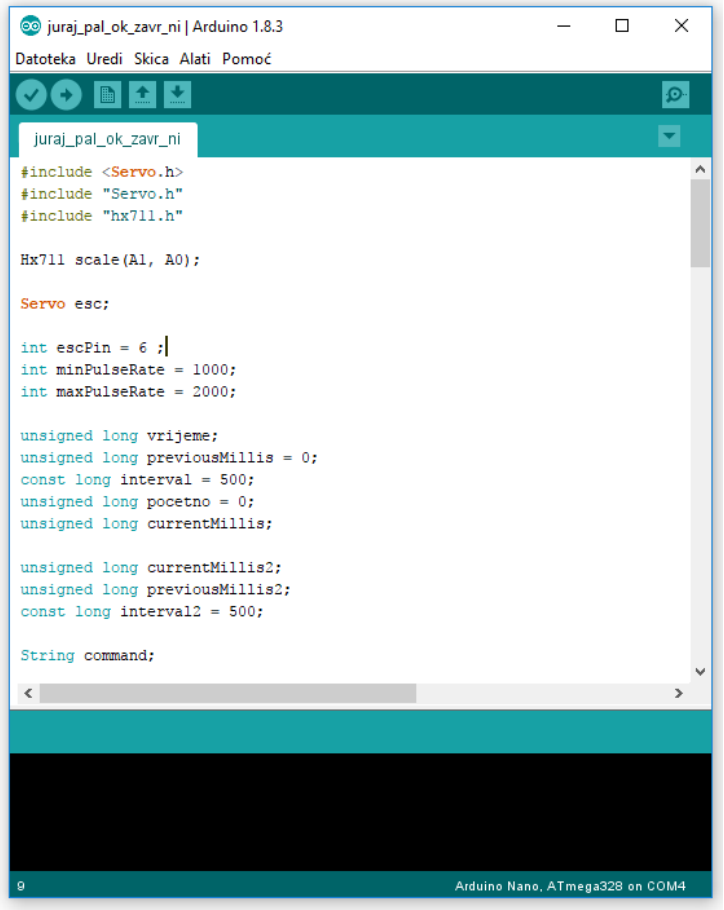
pisanje koda. Knjižnice se uključuju s ključnom riječju npr: Servo myservo;. Servo je ključna riječ iz knjižnice u kojoj je već spremljen način rada. Na slici 3.14. prikazan je detaljan opis algoritma upravljanja.



Slika 3.14. Algoritam upravljanja.

Na samome početku potrebno je uspostaviti komunikaciju upravljačkog programa i uređaja za mjerenje. Kada se komunikacija uspostavi sustav počinje spremati vrijednosti u datoteku koja se zatim šalje. Poslani podaci se razvrstavaju i ispisuju na graf. Ispisivanje podataka na graf ovisi o vremenu uzorkovanja, a ne o intervalima u kojima vaga uzima mjerenje. Korisnik u svakome trenutku mjerenja može mijenjati brzinu motora i to na način da unese brzinu koja se usporedi s trenutnom zatim ovisno o ishodu usporedbe sustav smanjuje/povećava

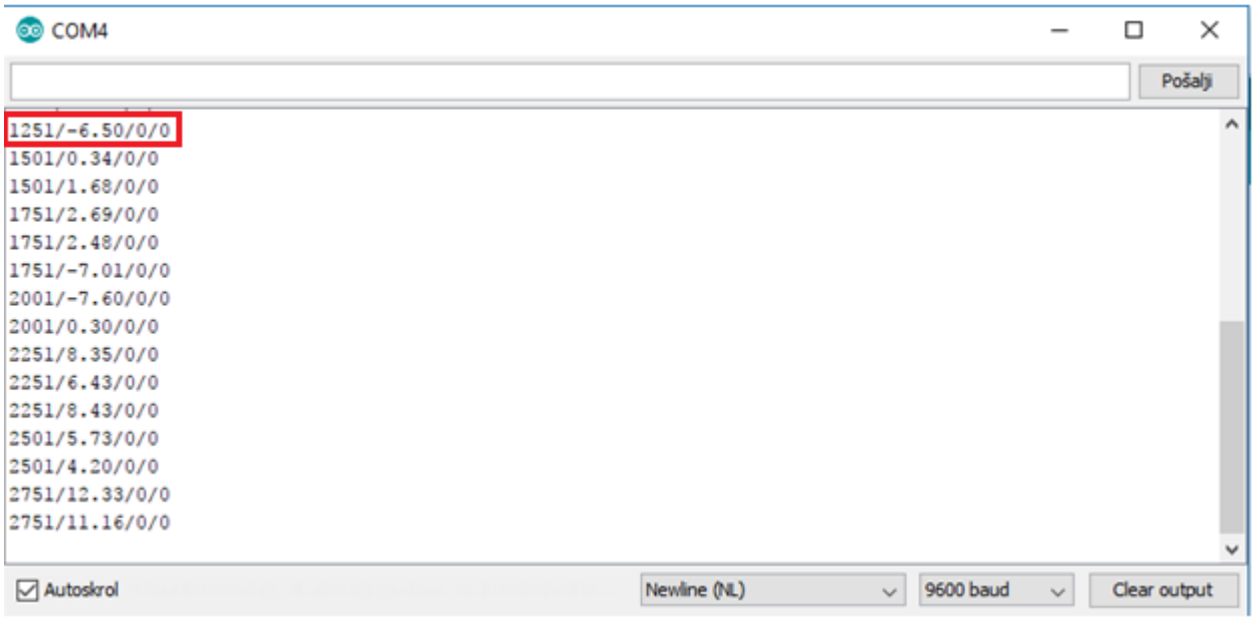
brzinu motora. Nakon završetka mjerenja korisniku kao rezultati pokusa ostaju graf i datoteka u kojoj su spremljeni svi trenuci koji su uzorkovani i ispisani na graf.



Slika 3.13. Programsko sučelje Arduino IDE.

3.5. Komunikacija i HM sučelje

Komunikacija se ostvarena pomoću zbroja stringova koje Arduino šalje preko usb kabela. String sadrži četiri podatka: vrijeme koje počinje u trenutku početka komunikacije između računala i Arduina, potiska koji mjeri ćelija za mjerenje sile, brzine vrtnje koju mi postavimo te trenutne brzine motora (slika3.15.). Konkretno iz primjera (slika 3.15.) prvi podatak (označen crvenom bojom) koji je poslan sadrži da je od trenutka pokretanja komunikacije prošlo 1251 ms, ćelija za mjerenje sile mjeri -6,5g, a trenutna i postavljena brzina motora iznose 0. Podaci koje Arduino pošalje spremaju se u tekstovnu datoteku kako bi svi trenutci uzorkovanja bili zabilježeni.



Slika 3.15. Izgled paketa koje arduino šalje.

Kreiranje aplikacije koja omogućava mjerenje dinamičkih karakteristika izvršeno je u Microsoft Visual Studio 2012. Najprije je kreiran dizajn, postavljene su tipke i određeni tekst blokovi koji se kasnije programiraju programskim jezikom C# te spajaju u cjelinu.

Prvo treba inicijalizirati port koji dohvaća podatke koje Arduino šalje, a inicijalizacija se može obaviti na dva načina (slika 3.16. i slika 3.17.). Oba odrade istu funkciju ali prvi je prikazan radi lakšeg objašnjenja pojedinog dijela.

```
private void InitializePort(){    // inicijalizacija porta

      myport = new SerialPort();   // definiranje serijskog porta
      myport.BaudRate = 9600;  // definiranje brzine prijenosa podataka
      myport.Parity = Parity.None;    // definiranje kontrolnih bitova, netreba,
      myport.DataBits = 8; // veličina dolaznog podatka definira se na 8 bitova
      myport.StopBits = StopBits.One; // definiranje broja bitova za prekid (1)

myPort.DataReceived += new SerialDataReceivedEventHandler(ReceivedSerialHandler);
      // definiranje dohvačenih podataka
      myport.Open(); // otvaranje porta
}
```

Slika 3.16. Inicijalizacija usb ulaza za serijsku komunikaciju.

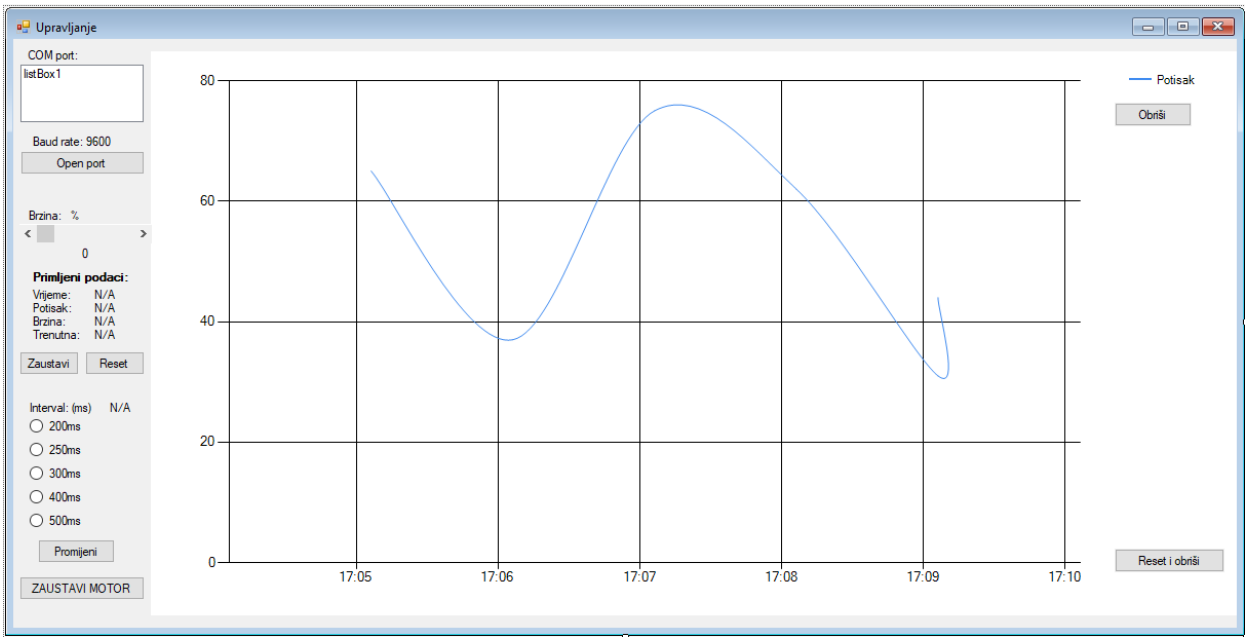
```
myPort = new SerialPort(comPort, baudRate, Parity.None, 8, StopBits.One);
```

Slika 3.17. Inicijalizacija usb ulaza za serijsku komunikaciju.

Izrađen je serijski port “myport” i dodijeljena mu je brzina prijenosa podataka serijske komunikacije. Za odabir porta kreirana je labela koja sama pronalazi zauzeti ulaz na računalu. Pronalazak ulaza omogućava komunikaciju računala sa Arduino nano razvojnom platformom. USB komunikacijski ulaz ne koristi kontrolni biti pa je taj biti definiran kao nepostojeći. Veličina dolaznog podatka je 8 bitna, dok se jedan od bitova koristi za raskid komunikacije. Ako prilikom inicijalizacije porta dođe do pogreške pojavljuje se prozor u kojem je navedena odsutnost ulaza prema kojem se pokušava ostvariti komunikacija.

3.5.1. HM sučelje

Na početku kreiranja vizualnog izgleda aplikacije postojao je samo prozor sa zadanim izgledom. Razvijanje dizajna odvijalo se dodavanjem teksta, tekstualnih prozora, prozora za prikaz grafa i tipki (slika 3.18.). Funkcija tekstualnih prozora definirana je unutar programskog koda ovisno o tome dali služi za upis porta ili ispis teksta. Definiran je događaj na temelju kojeg se vrši ispisivanje podataka iz tekstualnog prozora u na graf uz uvjet da se to ponavlja u odabranim intervalima. Tipka „Open port“ definira početak inicijalizacije porta koja je ranije objašnjena. Dok tipka „Zaustavi“ prekida serijsku komunikaciju.



Slika 3.18. Izgled krajnje aplikacije iz Visuala Studio 2012.

4.TESTIRANJE I REZULTATI

Potrebno je testirati izrađenu maketu te prokomentirati rezultate. Za potrebe ovog seminarskog rada bit će potrebno odraditi testiranje s jednom elisom i jednim motorom.

4.1. Metode testiranja

Elisa će biti testirana tako da će se motor prvo zaletjeti na 30 % brzine zatim držati 8.5s nakon toga će se zaletjeti na 80% brzine te tu brzinu držati 8.5 sekundi i na kraju ugasiti motor. Promatranje će biti usmjerenu na vrijeme koje je proteklo u promjenama brzine vrtnje motora.

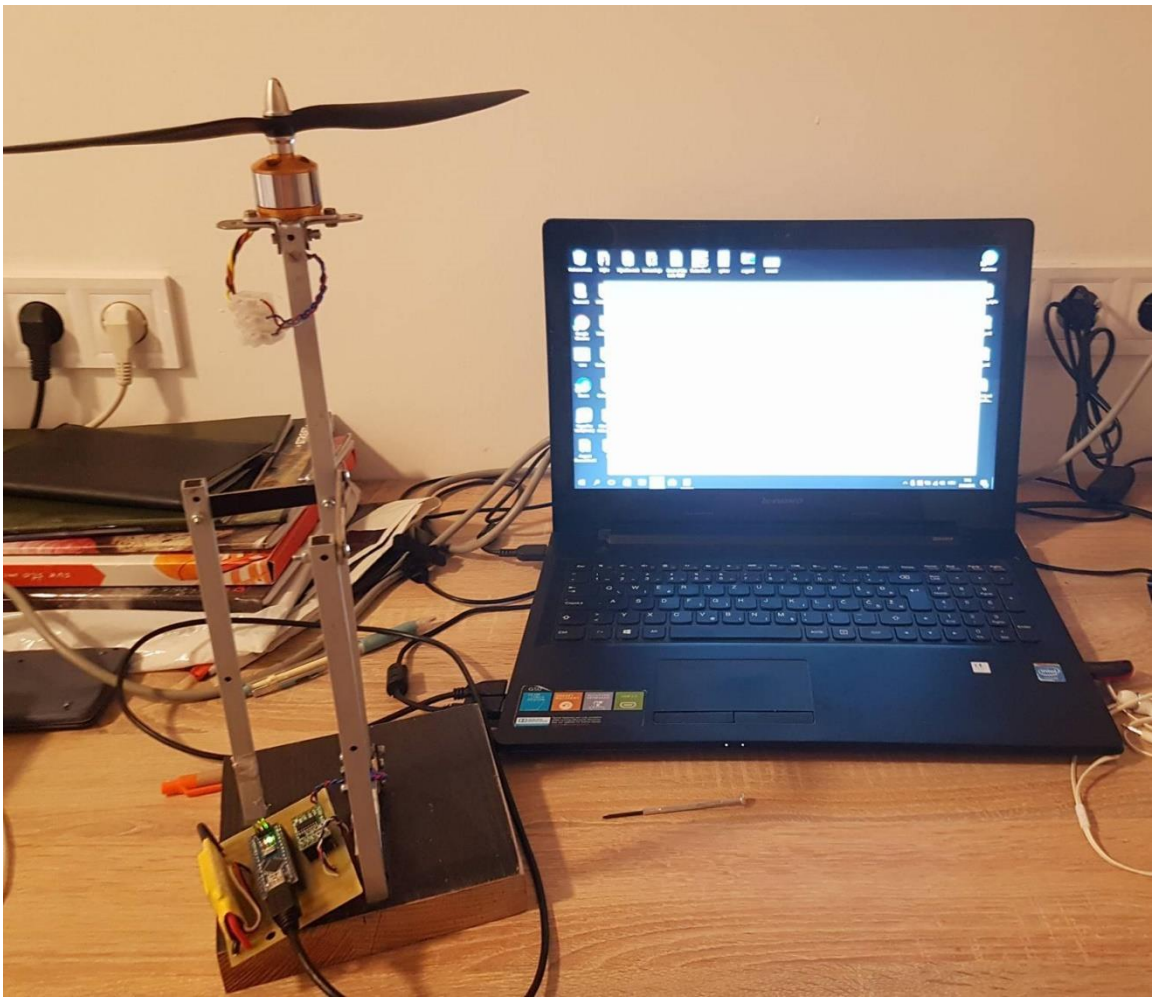
Elisa sadrži dva kraka radijusa 122mm (slika 4.1.).

Slika 4.1. Elisa.

Nakon što je elisa montirana na motor potrebno je programski kod učitati na Arduino nano te zatim pokrenuti aplikaciju. Potrebno je odabrati ulaz na računalu preko kojeg je Arduino spojen te pokrenuti komunikaciju. Kada je komunikacija ostvarena unosimo željene varijable u aplikaciju te pokrećemo testiranje, ako slučajno dođe do problema prilikom pokretanja aplikacija sadržava sigurnosno dugme koje automatski gasi sve započeto.

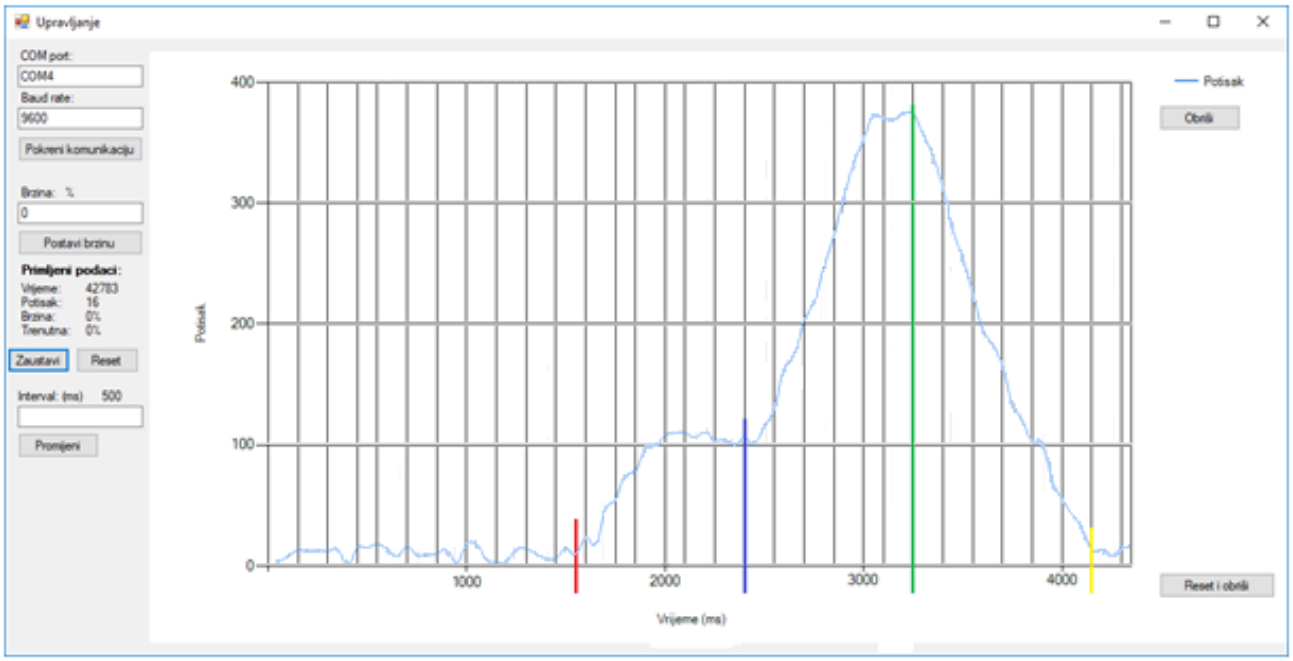
4.2. Rezultati testiranja

Prije samog početka testiranja potrebno je maketu pričvrstiti za podlogu kako bi se pokus mogao odraditi na siguran način. Nakon što su sve pripreme odrađene i algoritam za upravljanje se pošalje na mikroupravljač Arduino nano potrebno je samo pokrenuti aplikaciju i mjerenje može početi (slika 4.2.).



Slika 4.2. Priprema za mjerenje.

Na slici 4.3. su prikazani rezultati mjerenja koji će biti prokomentirani kasnije.



Slika 4.3. Rezultati testiranja.

Iz pokusa se vidi vrijednost potiska u gramima. Do trenutka označenog crvenom bojom motor nije radio, a ova očitavanja vage su uslijedila zbog njene pre velike osjetljivosti i ne savršenosti same izrade makete. U trenutku označenom crvenom bojom motor se počine zalijetati na svojih 30% brzine. Vremenski razmak između plave i crvene boje iznosi 8.5s, a promjena brzine motora od 0% do 30% odvila se za 4,5s. Sa 30% brzine odabrana elisa ostvaruje 100g potiska. U trenutku označenom plavom crtom motor se počinje zalijetati na 80% brzine i ta promjena traje 6,5s. S 80% brzine elisa ostvaruje 375g potiska. Ako bismo izmjerene vrijednosti malo proračunali (375g-100g) dobili bismo da ova elisa može za 6.5 s razviti potisak od 275g. Ponovno imamo vremenski razmak između zelene i plave boje iznosi 8.5s. U trenutku označenom zelenom bojom motor se postavlja na 0% brzine te zbog tromosti elise i rotora motora ta promjena traje 6 sekundi.

5.ZAKLJUČAK

Cilj završnog rada bio je izraditi maketu digitalne vage za mjerenje dinamičkog elisnog potiska. Pažnju treba obratiti na to da postolje vage bude dobro pričvršćeno za podlogu jer bi u protivnom veća elisa mogla srušiti vagu. Kao glavni uređaj za upravljanje korišten je Arduino nano. Komunikacija s računalom uspostavljena je kabelskom vezom te omogućuje korisniku stalno upravljanje mjerenjem. Usporede li se mjerenjem dobiveni rezultati s teorijskim vrijednostima vaga radi na ispravan način uz pogrešku od 5g. Ova vaga može se koristiti za mjerenje elisnog potiska raznih vrsta trofaznih motora pokretanih električkim kontrolerom brzine, kao i raznih tipova elisa.

LITERATURA

[1] Turnigy Thrust Stand
https://hobbyking.com/en_us/turnigy-thrust-stand.html?___store=en_us pristup: 3.6.2017

[2] Autodesk eagle
https://www.autodesk.com/products/eagle/free-download pristup: 5.6.2017

[3] Tiskana pločica
https://hr.wikipedia.org/wiki/Tiskana_plo%C4%8Dica pristup: 5.6.2017

[4] Arduino playground
https://playground.arduino.cc/ pristup.3.6.2017

[5] Visual Studio 2013
https://msdn.microsoft.com/en-us/library/dd831853(v=vs.120).aspx pristup.4.8.2017

[6] Dino Car, diplomski
https://bib.irb.hr/datoteka/776579.diplomski_Zavrna_verzija_DinoCarFESB.pdf, 16.06.2017.

[7] HX711 datasheet
https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf, 23.06.2017

SAŽETAK

Izrađena je maketa digitalne vage za mjerenje dinamičkog elisnog potiska za istosmjerne motore bez četkica (neg. Brushless DC electric motor). Potisak se mijeri pomoću ćelije za mjerenje sile (eng. Load cell). Izrađen je hardver koji se sastoji od dva podsustava, energetskog na kojem se nalazi elektronički kontroler brzine (eng. ESC) koji služi za upravljanje motorom (max struja 30A), te upravljačkog na kojem se nalaze elektroničke komponente HX711 A/D pretvarač te Arduino nano (ATMega 328P). Na mikroupravljaču Arduino nano učitan je algoritam za upravljanje koji je napisan u računalnom sučelju Arduino IDE. Program za upravljanje mikrokontrolerom izrađen je u Microsoftovom Visual studio 2012. Prije samog mjerenja opisani su svi elementi i postupak izrade makete. Nakon mjerenja rezultati su uspoređeni s očekivanim vrijednostima te je zaključeno da izrađena vaga odstupa vrlo malo, najviše zbog velike osjetljivosti sustava.

Ključne riječi: Istosmjerni motor bez četkica, elektronički kontroler brzine, dinamički elisni potisak, HX711 A/D pretvarač

ABSTRACT

Digital Scale for dynamic airscrew measurement

A digital model of a scale for dynamic airscrew boost measurement for brushless DC electric motors was designed. The boost is measured via a load cell. The hardware that was constructed consists of two subsystems, an energy system with an ESC which is used for controlling the motor (Max current: 30A) and a control system containing an HX711 A/D converter and the Arduino Nano ATMega 328P. A control algorithm, written in Arduino IDE, is loaded into the Arduino microcontroller. The control program for the microcontroller was designed using Microsoft Visual Studio 2012. Before initiating measurement, all elements and the design process behind the model were detailed. After the measurements the measured results were compared with the expected values and it was concluded that the scale's result debate very little, mostly due to different system sensitivity

Keywords: Brushless DC electric motor, electronic speed controller, dynamic airscrew boost measurements, HX711 A/D converter

ŽIVOTOPIS

Juraj Palčok rođen je 05.07.1995. godine u Osijeku. U Marijancima završava „Osnovnu školu Matije Gubca“, nakon čega upisuje elektrotehničku školu u Valpovu. Po završetku srednjoškolskog obrazovanja upisuje preddiplomski studij elektrotehnike na Elektrotehničkom fakultetu Osijek. Na 2. godini studija se opredjeljuje za smjer komunikacije i informatika. Te je sada student 3. godine fakulteta elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

U Osijeku, lipanj 2017. Juraj Palčok

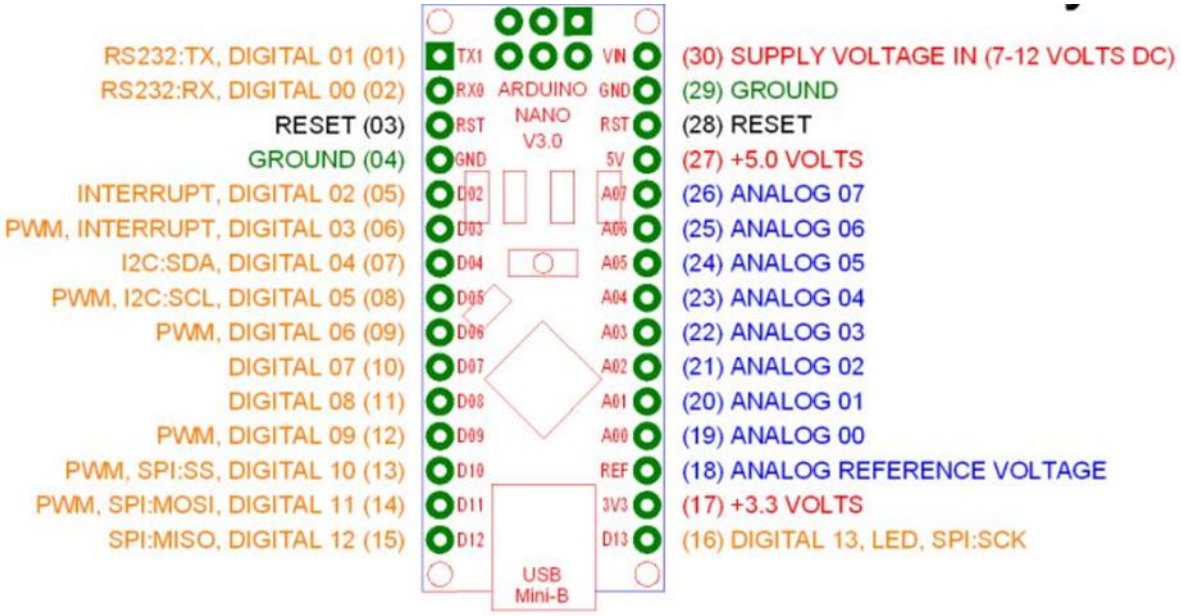
_____

(Vlastoručni potpis)

PRILOZI

Prilog 1: ATMega328p specifikacije

Mikrokontroler	ATmega328
Operativni napon	5V
Ulazni napon (preporučeno)	7 – 12 V
Ulazni napon (ograničenja)	6 – 20 V
Digitalni ulazno/izlazni pinovi	14
PWM kanali	6
Analogni ulazni pinovi	8
DC struja po ulaz/izlaz pinu	40 mA
Flash memorija	32 KB (ATmega 328) od čega 2 KB koristi za pokretanje sustava (bootloader)
SRAM	2 KB
EEPROM	1 KB
Brzina sata	16 MHz
Veličina	1.85cm x 4.3cm



Prilog 2: Hx711 library

```
Hx711::Hx711(uint8_t pin_dout, uint8_t pin_slk) :
		_pin_dout(pin_dout), _pin_slk(pin_slk)
{
	pinMode(_pin_slk, OUTPUT);
	pinMode(_pin_dout, INPUT);

	digitalWrite(_pin_slk, HIGH);
	delayMicroseconds(100);
	digitalWrite(_pin_slk, LOW);

	averageValue();
	this->setOffset(averageValue());
	this->setScale();
}

Hx711::~Hx711()
{

}

long Hx711::averageValue(byte times)
{
	long sum = 0;
	for (byte i = 0; i < times; i++)
	{
		sum += getValue();
	}

	return sum / times;
}

long Hx711::getValue()
{
	byte data[3];

	while (digitalRead(_pin_dout))
		;

	for (byte j = 3; j--;)
	{
		for (char i = 8; i--;)
```

```
		{
			digitalWrite(_pin_slk, HIGH);
			bitWrite(data[j], i, digitalRead(_pin_dout));
			digitalWrite(_pin_slk, LOW);
		}
	}

	digitalWrite(_pin_slk, HIGH);
	digitalWrite(_pin_slk, LOW);

	data[2] ^= 0x80;

	return ((uint32_t) data[2] << 16) | ((uint32_t) data[1] << 8)
			| (uint32_t) data[0];
}

void Hx711::setOffset(long offset)
{
	_offset = offset;
}

void Hx711::setScale(float scale)
{
	_scale = scale;
}

float Hx711::getGram()
{
	long val = (averageValue() - _offset);
	return (float) val / _scale;
}
```

Prilog 2.1: Hx711 library

```
#ifndef HX711_H_
#define HX711_H_

#include "Arduino.h"

class Hx711
{
public:
	Hx711(uint8_t pin_din, uint8_t pin_slk);
	virtual ~Hx711();
	long getValue();
	long averageValue(byte times = 32);
	void setOffset(long offset);
	void setScale(float scale = 742.f);
	float getGram();

private:
	const uint8_t _pin_dout;
	const uint8_t _pin_slk;
	long _offset;
	float _scale;
};

#endif /* HX711_H_ */
```

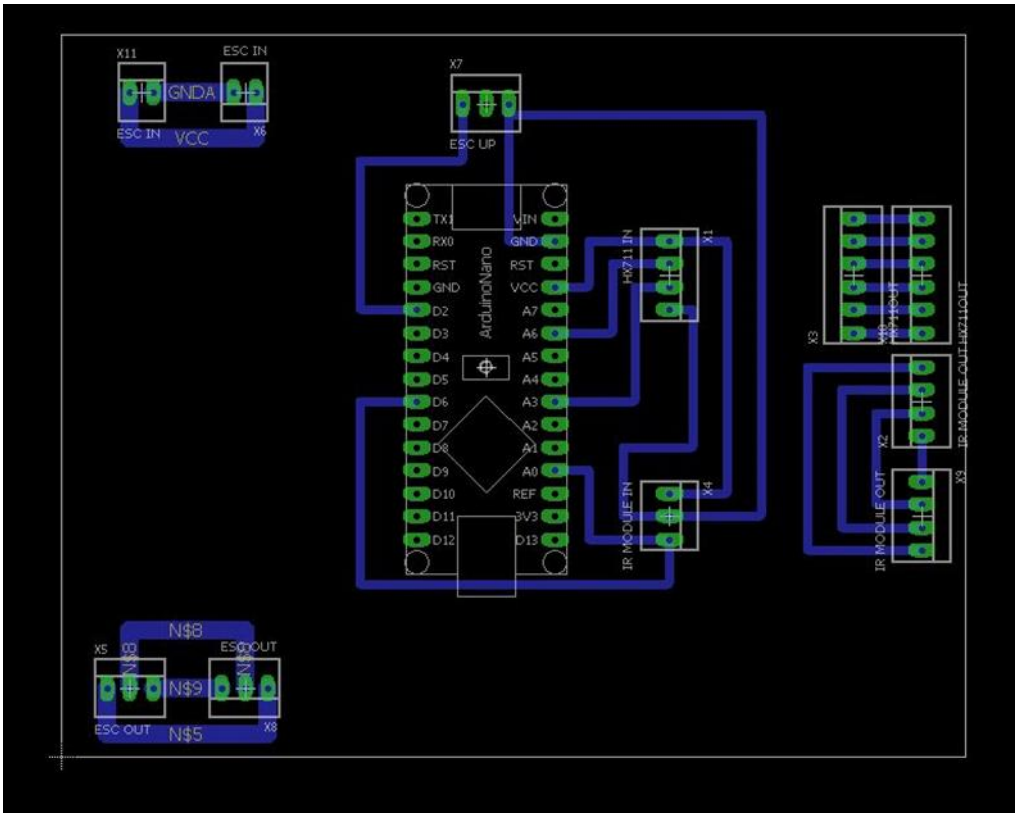
Prilog 3: Kod programa za upravljanje arduinom

```
#include <Servo.h>
#include "Servo.h"
#include "hx711.h"

Hx711 scale(A1, A0);

Servo esc;

int escPin = 6 ;
int minPulseRate = 1000;
int maxPulseRate = 2000;
int throttleChangeDelay = 100;
volatile int counter = 0;

unsigned long vrijeme;
unsigned long previousMillis = 0;
const long interval = 250;
String command;
unsigned long pocetno = 0;
unsigned long currentMillis;
unsigned long currentMillis2;
unsigned long previousMillis2;
const long interval2 = 500;
int gas=29;

void setup() {

  Serial.begin(9600);
  Serial.setTimeout(500);
  scale.setOffset(8623702);
  scale.setScale(169);
  // Attach the the servo to the correct pin and set the pulse range
  esc.attach(escPin, minPulseRate, maxPulseRate);
  // Write a minimum value (most ESCs require this correct startup)
  esc.write(0);

}

void loop() {
  currentMillis2 = millis();
  if (currentMillis2 - previousMillis2 >= interval2) {
    previousMillis2 = currentMillis2;
    changeThrottle(gas);
  }

  currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
```

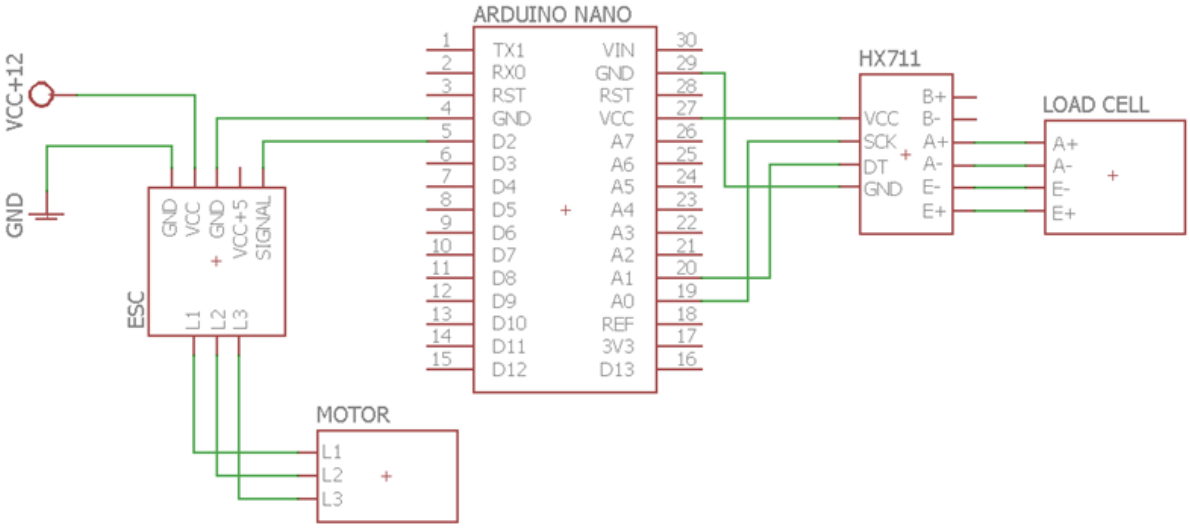
```
    vrijeme = currentMillis - pocetno;
  }

  // Wait for some input
  if (Serial.available() > 0)
  {
    char c = Serial.read();
    if (c == '\n')
    {
      int throttle = parseCommand(command);
      gas = throttle;

      //String zajedno = String (vrijeme) + "/" + String(scale.getGram()) + "/" + String(throttle);
      //Serial.println(zajedno);
      // Change throttle to the new value
      changeThrottle(throttle);
      command = "";
    }
    else
    {
      command += c;
    }
  }
}

void changeThrottle(int throttle) {

  // Read the current throttle value
  int currentThrottle = readThrottle();
  int potisak = scale.getGram();
  String zajedno = String (vrijeme) + "/" + String(potisak) + "/" + String(throttle) + "/" +
String(currentThrottle);
  //delay(10);
  Serial.println(zajedno);

  // Are we going up or down?
  int step = 1;
  if ( throttle < currentThrottle )
    step = -1;

  // Slowly move to the new throttle value
  if ( currentThrottle != throttle ) {
    esc.write(currentThrottle + step);
    // currentThrottle = readThrottle();
    // Serial.println(zajedno);
  }

}

int readThrottle() {
```

```
  int throttle = esc.read();
  /*
    Serial.print("Trenutni gas: ");
    Serial.println(throttle);
    Serial.print(counter * 30);
    Serial.println("rpm.");
    Serial.print(" ");
    Serial.println(scale.getGram());
  */
  return throttle;
}

int parseCommand(String com)
{
  if (com == "res")
  {
    resetiranje();
  }
  else {
    int value = com.toInt();
    if ( value < 29 ) {
      return 29;
    }
    else if ( value > 150 ) {
      return 150;
    }
    else
      return value;
  }
}

void resetiranje()
{
  pocetno = currentMillis;
}
```

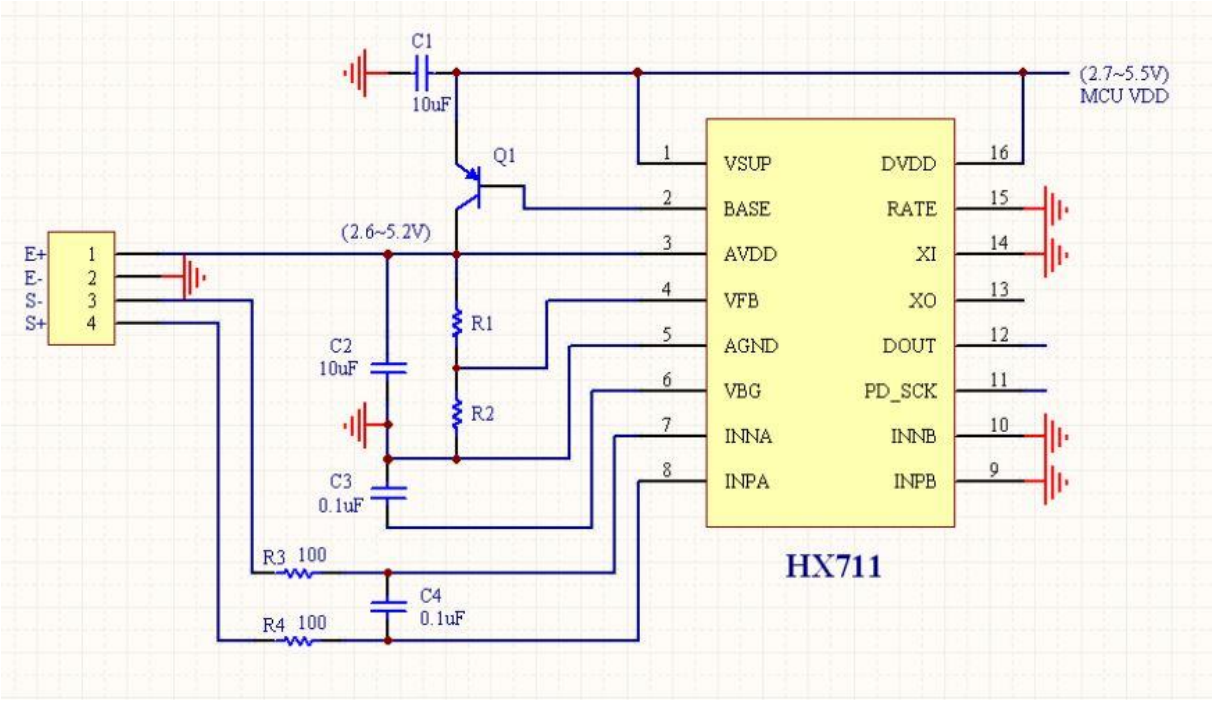
Prilog 4:Izgled virtualne pločice u program eagle



Prilog 6: Električka shema tiskane pločice



Prilog 5: Električna shema HX711 A/D pretvarača



Prilog 7. Program kod iz Visual Studia 2012

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.Windows.Forms.DataVisualization.Charting;

namespace eugen_diplo
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         foreach (string s in SerialPort.GetPortNames())
         {
            listBox1.Items.Add(s);
         }


         chart1.ChartAreas[0].AxisX.MinorGrid.Interval = 1;
         chart1.ChartAreas[0].AxisX.MinorGrid.Enabled = true;
         chart1.ChartAreas[0].AxisX.Title = "Vrijeme (ms)";
         chart1.ChartAreas[0].AxisY.Title = "Potisak (g)";
         chart1.ChartAreas[0].AxisX.MajorGrid.LineColor = Color.Gainsboro;
         chart1.ChartAreas[0].AxisY.MajorGrid.LineColor = Color.Gainsboro;
         chart1.ChartAreas[0].AxisX.MinorGrid.LineColor = Color.Gainsboro;
         chart1.ChartAreas[0].AxisY.MinorGrid.LineColor = Color.Gainsboro;
         button3.Enabled = false;
         if (timer1 != null)
         {
            timer1.Enabled = false;
         }
         radioButton5.Checked = true;


      }
      string brzina = "res";
      SerialPort myPort;
      string comPort = "COM4";
      int baudRate=9600
      int interval = 500;
      int postotak=0;
```

```
string[] data = { "", "", "", "" };
string[] data1 = { "", "", "", "" };

bool opened = false;
private void button1_Click(object sender, EventArgs e
{

   if (opened == false) {
      comPort = listBox1.GetItemText(listBox1.SelectedItem);
   label12.Text = interval.ToString();
   InitTimer();
   button1.Text = "Port opened";
   button1.Enabled = false;
   button3.Enabled = true;
   opened = true;

}
   else
   {
      timer1.Stop();
      myPort.WriteLine("ug");
      myPort.Close();
      button1.Text = "Open port";
   }

}

private Timer timer1
public void InitTimer
{
   timer1 = new Timer();
   timer1.Tick += new EventHandler(timer1_Tick);

   timer1.Interval = interval;
   timer1.Start();
}

private void timer1_Tick(object sender, EventArgs e)
{
   myPort = new SerialPort(comPort, baudRate, Parity.None, 8, StopBits.One);

   myPort.Open();
   myPort.WriteLine(brzina);
   if (brzina == "res
   {
      postotak = klizac1.Value
      double umnozak = (postotak * 1.21) + 29
      brzina = Math.Round(umnozak).ToString
      label9.Text = postotak.ToString() + "%";
   }
```

```
        myPort.Close();

        myPort.DataReceived += new
SerialDataReceivedEventHandler(ReceivedSerialHandler
        myPort.Open();
    }
    private void ReceivedSerialHandler(object sender, SerialDataReceivedEventArgs e)
    {
        SerialPort sp = (SerialPort)sender

        this.Invoke((MethodInvoker)delegate
        {
            string recvData = sp.ReadLine();

            System.IO.StreamWriter file1 = new
System.IO.StreamWriter("C:\\save\\recvData.txt");
            file1.WriteLine(recvData);
            file1.Close();

            data = recvData.Split('/');
            int duljina = data.Length;

            if (duljina < 4)
            {
                data = data1;
            }
            else
            {
                data1 = data;
                label7.Text = data[0
                label8.Text = data[1
                double broj1 = (Convert.ToInt32(data[2]) - 29) / 1.21
                double broj2 = (Convert.ToInt32(data[3]) - 29) / 1.21;
                string labela1=Math.Round(broj1).ToString() + "%";;
                string labela2=Math.Round(broj2).ToString() + "%";;
                label9.Text = labela1;
                label14.Text = labela2;

                System.IO.StreamWriter file2 = new
System.IO.StreamWriter("C:\\save\\data.txt");
                file2.WriteLine(data[0]+";"+data[1]+";"+data[2]+";"+data[3]);
                file2.Close();
                System.IO.StreamWriter file3 = new
System.IO.StreamWriter("C:\\save\\data2.txt");
                file3.WriteLine(data[0]+";"+data[1]+";"+labela1+";"+labela2);
                file3.Close();
            }
            chart1.Series["Potisak"].Points.AddXY(data[0], data[1]);
        }
        );
```

```
    myPort.Close();
}
bool zaustavljen = false;
private void button3_Click(object sender, EventArgs e)
{
    if (zaustavljen == true)
    {
        timer1.Start
        button3.Text = "Zaustavi graf";
        zaustavljen = false;
    }
    else
    {
        timer1.Stop();
        button3.Text = "Pokreni graf";
        zaustavljen = true;
    }

}

private void button4_Click(object sender, EventArgs e)
{
    brzina = "res

}


private void textBox1_KeyDown_1(object sender, KeyEventArgs e
{
    if (e.KeyCode == Keys.Enter
    {
        button1_Click(this, new EventArgs());
    }
}

private void textBox2_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        button1_Click(this, new EventArgs());
    }
}

private void button5_Click(object sender, EventArgs e)
{
    chart1.Series[0].Points.Clear();
}

private void button6_Click(object sender, EventArgs e)
{
```

```
    InitTimer();
}

private void textBox4_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        button6_Click(this, new EventArgs());
    }
}

private void button7_Click(object sender, EventArgs e)
{
    chart1.Series[0].Points.Clear();
    brzina = "res";
}

private void radioButton5_CheckedChanged(object sender, EventArgs e)
{
    if (opened == true)
    {
        timer1.Stop();
        interval = 500;
        label12.Text = interval.ToString();
        InitTimer();
    }
}

private void radioButton4_CheckedChanged(object sender, EventArgs e)
{
    if (opened == true)
    {
        timer1.Stop();
        interval = 400;
        label12.Text = interval.ToString();
        InitTimer();
    }
}

private void radioButton3_CheckedChanged(object sender, EventArgs e)
{
   if (opened == true)
    {
        timer1.Stop();
        interval = 300;
        label12.Text = interval.ToString();
        InitTimer();
    }
}
```

```
private void radioButton2_CheckedChanged(object sender, EventArgs e)
{
   if (opened == true)
   {
      timer1.Stop();
      interval = 250;
      label12.Text = interval.ToString();
      InitTimer();
   }

}

private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
   if (opened == true)
   {
      timer1.Stop();
      interval = 200;
      label12.Text = interval.ToString();
      InitTimer();
   }
}

private void button8_Click(object sender, EventArgs e)
{
   brzina = "ug";
   klizac1.Value = 0;
   postotak = klizac1.Value;
   double umnozak = (postotak * 1.21) + 29
   label9.Text = "STOPPED";
   label3.Text = "Brzina: STOPPED";

}

private void klizac1_Scroll(object sender, ScrollEventArgs e)
{
   label15.Text = (klizac1.Value).ToString();
}

private void klizac1_MouseLeave(object sender, EventArgs e)
{
   postotak = klizac1.Value;
   label3.Text = "Brzina: " + postotak.ToString() + "%";
   double umnozak = (postotak * 1.21) + 29;
   brzina = Math.Round(umnozak).ToString();
   label9.Text = postotak.ToString() + "%";
}}}
```