SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4819

Identifikacija parametara kretanja vozila u stvarnom vremenu

Dominik Bartolović

Zagreb, lipanj 2017.

Sažetak

Potrebno je realizirati programski sustav za identifikaciju parametarskog modela vozila u kretanju. Programski sustav mora obnavljati model u stvarnom vremenu na temelju mjernih podataka o kretanju vozila te veličina s ostalih senzora i elemenata upravljačkog sustava. Model mora obuhvaćati osnovne fizikalne komponente sustava drugog reda te vanjske i unutarnje sile koje uzrokuju kretanje. Proučiti i umanjiti utjecaj šuma i mjernih nesigurnosti na identifikaciju. Fizikalni model kretanja preslikati u virtualni korisnički model, uz odabrane opcije i parametre preslikavanja. Koristiti ARM zasnovano STM32 F4 ugradbeno računalo s operacijskim sustavom za rad u stvarnom vremenu (RTOS) te komunikacijskim podsustavom. U programskom jeziku C napisati potrebnu programsku potporu. Po potrebi koristiti programsko okruženje MATLAB za provjeru i simulaciju algoritama.

Sadržaj

Tablica sadržaja

Popis tablica

Indeks tablica

Popis slika

Indeks slika

1. Uvod

Prvi automobil na benzin je proizveden 1886. godine. Bio je to dosta složen stroj. Vozač je morao kontrolirati prigušni ventil, tempirati paljenje i čak sam pokrenuti motor pomoću ručice. Tu su naravno bile i standardne kontrole koje se mogu vidjeti i u današnjim automobilima, gas, kočnica i kvačilo. Ako se pojača gas, automobil ima veću akceleraciju, a ako se pusti, prestaje ubrzavati.

Od prvog automobila pa sve do danas motori na unutarnje izgaranje nisu u potpunosti ukroćeni. Jako je teško upravljati njima precizno. Vozač nema puno više kontrole od običnog gasa i kočnice. Ljudi koji rade taj automobil mu mogu prilagođavati razne vozne osobine, ali često kada se jednom definiraju takve ostaju i može ih se mijenjati tek opsežnijim mehaničkim zahvatima. Automobili sa motorom na unutarnje izgaranje zahtijevaju profesionalce da bi se iz njih izvukao maksimum. A čak i profesionalac nekada zakaže. Dovoljan je samo trenutak nepažnje i gubitak koncentracije da bi se dogodila nezgoda. Zar ne bi bilo bolje da možemo ukrotiti automobil i bolje kontrolirati njegove vozne osobine?

Glavni problem je što se motori na unutrašnje izgaranje mogu kontrolirati uglavnom mehaničkim putem, a to je uglavnom jako sporo. Postoje načini da se, na primjer, odziv na pritisak papučice gasa ubrza, ali to košta, i svako poboljšanje povećava mehaničku složenost, što smanjuje pouzdanost motora i čini ga podložnijim kvarovima.

Najbolje bi bilo da motor bude što mehanički jednostavniji. Ako usporedimo električni motor sa benzinskim, električni je jako jednostavan. I ne samo da je mehanički jednostavan, već je bolji u svakom pogledu. Brži odziv, ravnomjerniji moment, gotovo nebrojene mogućnosti upravljanja, i manja težina za istu snagu. Električni signali su puno brži od mehaničkih i moment motora se može mijenjati gotovo trenutno. Da ne govorimo o pametnim načinima upravljanja kao što je momentno vektoriranje uz koje jedno vozilo može imati karakteristike njih 5.

Električna vozila polako ali sigurno dolaze na ceste i zamjenjuju vozila sa motorom na unutarnje izgaranje. Najveća prepreka napuštanju unutarnjeg

izgaranja i okretanju električnim vozilima nije motor, već baterije, odnosno skladištenje energije. Ako se taj problem riješi, neće postojati veće zapreke izbacivanju motora sa unutrašnjim izgaranjem.

Sa novim BLDC motorima može se postići upravljačka karakteristika po želji, jedino što je potrebno je odgovarajuće sklopovlje i programska potpora koji će generirati pogonske signale kako bi se pogonio motor. To je upravo ono čime se ovaj završni rad bavi. Problematikom pri generiranju upravljačkih signala.

2. Bicikl bez lanca

Pametnim upravljanjem električnim automobilom se mogu raditi svakakve zanimljive stvari. Mogu mu se prilagoditi karakteristike da bude pogodniji za "drift" na primjer, ali u ovom radu se neću baviti problematikom upravljanja električnim automobilom, već električnim biciklom.

Ali ne radi se o klasičnom električnom biciklu koji ima motor/generator na jednom kotaču i koji može poslužiti kao pomoć pri vožnji, a i dalje su tu pedale i prijenos kao na klasičnim, neelektričnim, biciklima. Ovdje se radi o biciklu kojemu su pedale i kotač u potpunosti mehanički odvojeni. Nema lanca. Ta činjenica pruža ogromnu fleksibilnost, ali i veću složenost za upravljanje.

Kada su pedale i kotač mehanički povezani, na pedalama se "osjeti" težina bicikla. Vozač točno zna koliko jako mora gaziti kako bi: usporavao, kretao se približno jednakom brzinom i ubrzavao. Sve je jako jednostavno, ali i vrlo nefleksibilno. Klasični električni bicikli imaju malo više fleksibilnosti, ali i dalje ne iskorištavaju u potpunosti sve prednosti koje današnja elektronika može ponuditi.

Kako je već prije spomenuto, mehanički sustavi imaju dosta nedostataka u odnosu na usporedive električne. Teško se njima upravlja i veća složenost donosi veću šansu za kvarove. Najbolje bi bilo riješiti se lanca.

Zato je potreban pravi električni bicikl. On ne treba lanac već ima generator na pedalama i motor na kotačima. Tako dobijemo potpunu kontrolu nad svim parametrima sustava i možemo pružiti najbolje vozačko iskustvo.

Ali sada kada smo se riješili lanca pojavilo se dosta problema. Treba ostvariti električnu vezu između pedala i kotača, jer je mehanička veza nestala. Prijenos energije i povratna veza vozaču, koji dolaze prirodno sa mehaničkom vezom, sve je potrebno dodati kao programsko ili elektroničko riješenje. O protumomentu na pedali odlučuje programska podrška, isto kao i o momentu na motoru.

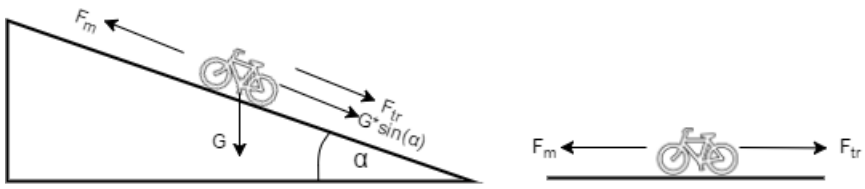
Upravo ova činjenica daje bezbroj mogućnosti upravljanja i zbog nje je ovakav električni bicikl zapravo bolji od klasičnog električnog. Na primjer, bez problema se može simulirati klasični bicikl uz bilo koji omjer prijenosa zupčanika. To je samo jedan jednostavniji slučaj, a moguće su i zanimljivije stvari. Na primjer dok se vozi po uzbrdici može se simulirati kao da se vozi po nizbrdici, ili obrnuto. Ili još bolje, zašto bicikl ne bi automatski prilagođavao težinu vožnje umoru vozača. Sve to i još puno više je samo stvar prilagođenja momenta na kotaču i protumomenta na pedali.

Nabrojani su neki slučajevi koji očito troše više energije nego vozač daje. To naravno nije moguće, ali nije spomenuto da ovaj bicikl također ima baterije. Kada se radi u modu koji treba više energije nego vozač daje, potrebni višak se izvlači iz baterija, a kada se radi u modu u kojem vozač proizvodi više nego se koristi za kretanje, višak ide u baterije. Tjelovježba bi bila primjer gdje se proizvodi više nego je potrebno.

3. Problematika

3.1. Problem nepoznate mase

Pogledajmo problematiku kod ovakvog pristupa električnom biciklu na primjeru. Uzmimo slučaj kada se bicikl vozi uzbrdo, a mi bi htjeli da se vozaču čini kao da ide po ravnome. Neka je uzbrdica konstantnog nagiba. Ako napravimo dijagram sila u ovoj situaciji on bi izgledao ovako:



Slika 1: Dijagrami sila na uzbrdici(lijevo) i na ravnome(desno)

Ovdje su prikazani dijagrami sila i za uzbrdicu(lijevo) i za ravninu(desno). Vidimo da je jedina razlika u sili *G*sin(α)*. Ta se sila pojavljuje na uzbrdici jer težina na uzbrdici nije okomita na ostale sile već je pod nekim kutem *α* u odnosu na njih. Zbog toga je moramo rastaviti na dvije komponente, jednu u smjeru kretanja, a drugu okomito na smjer kretanja. Komponenta težine koja je okomita na smjer kretanja potpomaže sili trenja, a komponenta koja je paralelna usporava ili ubrzava bicikl, u ovom primjeru usporava jer se radi o uzbrdici.

Da bi simulirali vozaču ravninu dok se vozi na uzbrdici moramo mu poništiti tu komponentu *G*sin(α)*. To možemo lako, samo trebamo motor jače pogoniti. Ali ovdje ipak postoji jedan problem. On izlazi na vidjelo ako raspišemo težinu *G*. Čemu je težina *G* jednaka? Težina *G* jednaka je:

$$G = mg \tag{1}$$

umnošku akceleracije sile teže koja je konstanta i mase. Upravo je masa ono što stvara problem u našem primjeru, jer je to masa bicikla plus masa vozača. Da je samo masa bicikla ne bi bilo problema jer nam je ona poznata, ali masa vozača nam je veliki upitnik. Ne možemo niti uzeti neku prosječnu masu ljudske populacije jer se masa dvoje ljudi može puno razlikovati.

Također nije baš niti najelegantnija opcija tražiti ljude da se izvažu svaki puta prije nego idu voziti bicikl. Kao najbolje riješenje se nameće da treba nekako odrediti ukupnu masu u vožnji. Problematikom određivanja te mase se bavi ovaj završni rad.

3.2. Složeno trenje

Za sada je bilo riječi samo o problemu ukupne mase bicikla, a na slici 1 su označene još dvije dodatne sile osim sile teže i njene *G*sin(α)* komponente. To su sile F_m i F_{tr}. F_m je sila motora i za nju ne bi trebalo dodatnih objašnjenja. Nju motorom možemo kontrolirati. F_{tr} s druge strane ne možemo kontrolirati, i ona je takva kakva je.

Na prvi pogled F_{tr} bi trebala biti konstantna jer kako se uči još u školi vrijedi:

$$F_{trP} = \mu_P G \tag{2}$$

gdje je μ koeficijent trenja koji je konstanta koja ovisi o podlogama koje se taru. G je težina tijela i ona ovisi također o konstanti g i masi koja je konstantna kada neka osoba sjedne na bicikl. Međutim u ovom završnom radu ćemo je ipak definirati malo drugačije. Stavit ćemo u nju trenje guma i podloge po kojoj se vozi, trenje na samom biciklu između raznih mehaničkih dijelova, jer je za njih koeficijent trenja drugačiji, i stavit ću jednu promjenjivu silu a to je sila otpora zraka. Uz sve ovo F_{tr} izgleda ovako:

$$F_{tr} = F_{oz} + F_{trP} + F_{trMEH} \tag{3}$$

F_{trP} i F_{trMEH} možemo spojiti u jednu silu F_{trP} pa dobijemo:

$$F_{tr} = F_{oz} + F_{trP} \tag{4}$$

To ćemo napraviti jer su mehanički gubici jako mali i nema ih potrebe promatrati odvojeno od glavnog uzročnika trenja, a to su dodir gume-podloga. Samo ćemo iza koeficijenta trenja izbaciti indeks radi korektnosti, jer se više neće raditi samo o trenju s podlogom.

Otpor zraka, koji nije konstanta kako je već navedeno, ovisi o brzini kojom se vozi. Za veće brzine je funkcija ovisnosti otpora zraka o brzini kvadratna, ali za male brzine, kao što je brzina vožnje bicikla, ta funkcija se može aproksimirati pravcem pa glasi:

$$F_{oz} = \vartheta v \tag{5}$$

gdje je ϑ konstanta otpora zraka, a v brzina kretanja bicikla.

Sve to ćemo upakirati pod jednu oznaku (F_{tr}) i nazvati sila trenja. Formula za tako definiranu silu trenja je:

$$F_{tr} = \vartheta v + \mu G \tag{6}$$

Zašto nam je uopće bitna sila trenja postavlja se pitanje. Sila trenja zapravo predstavlja gubitke sustava. Ona je ono što nas vuče nazad kad bi htjeli ići naprijed. Ako se vratimo kratko na primjer sa penjanjem na uzbrdicu gdje bi htjeli da nam se čini kao da se vozimo po ravnom onda nam sila trenja zapravo smeta. Smeta nam jer se zbraja sa komponentom *G*sin(α)* sile teže i mi vidimo samo ukupnu silu. Zato je moramo nekako isfiltrirati. Kasnije će biti dodatno pojašnjeno, za sada je samo bitno da se zna što ona točno je i što predstavlja u sustavu.

Zadnji dodatak koji treba napraviti je uzeti u obzir promjenu sile teže u ovisnosti o nagibu podloge. To ćemo napraviti tako da dodamo *cos(α)* trenju podloge. Uz taj mali dodatak ukupno trenje je:

$$F_{tr} = \vartheta v + \mu G \cos(\alpha) \tag{7}$$

3.3. Sila masi daje ubrzanje

Postoji još jedna sila koje se nismo dotaknuli, koja čak nije niti očito naznačena na slici 1. Ona je ipak tamo a dobije se vektorskim zbrajanjem svih ovih sila o kojima je do sada bilo govora. Radi se o ukupnoj sili F. To je sila koja bicikl tjera naprijed. Njena formula glasi:

$$F = ma \tag{8}$$

Iz ove formule proizlazi i naslov ove podcjeline. m je ukupna masa bicikla, a *a* je ubrzanje koje nastaje kao posljedica djelovanja ukupne sile *F* na bicikl ukupne mase *m*. Budući da je sila *F* veličina na koju možemo utjecati, a ubrzanje je veličina koja se mijenja kao posljedica promjene sile, logičnije je prethodnu formulu zapisati na način:

$$a = \frac{F}{m} \tag{9}$$

, ali kod zapisivanja sustava ćemo ipak koristiti prethodnu formulu, jer će se raditi o jednadžbi sila. Također nema potrebe koristiti vektorski zapis jer su sve sile na istom pravcu.

3.4. Sažetak

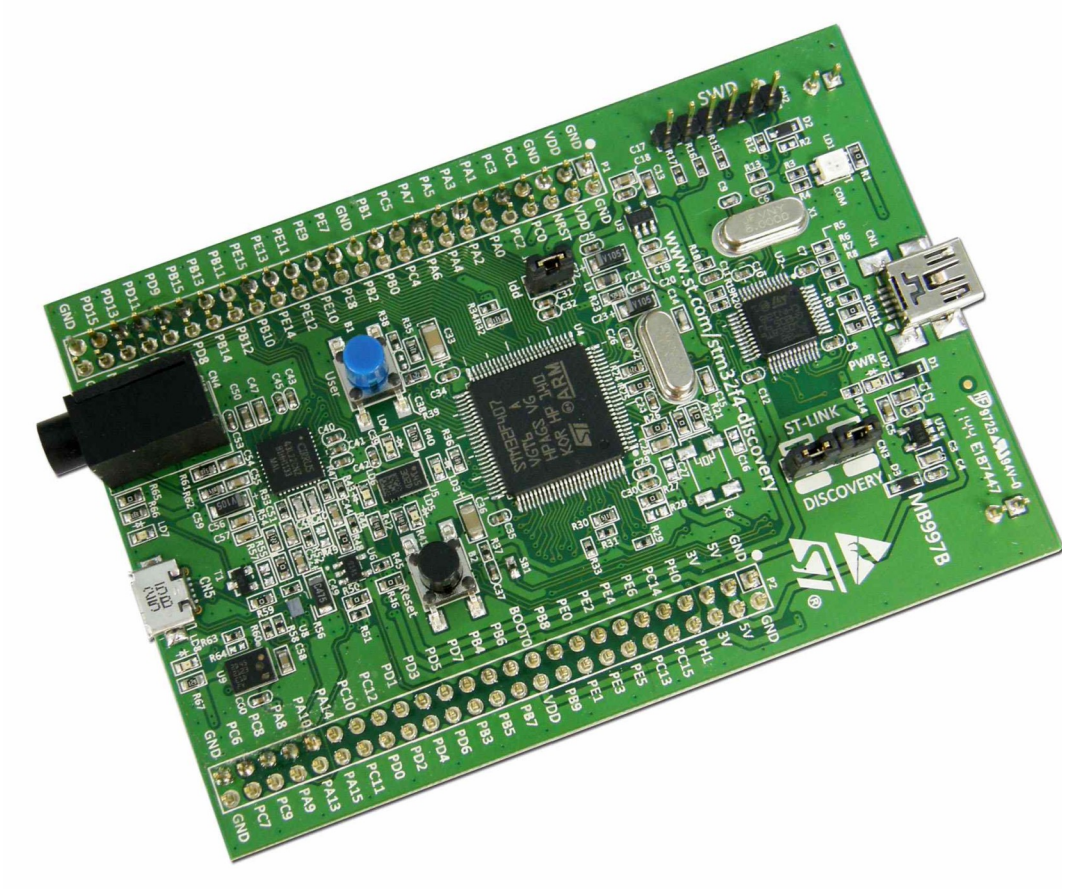
Puno je toga napisano u 3. poglavlju, pa samo da ponovimo kratko što je krajnji cilj ovog završnog rada i kako se sve napisano uklapa jedno s drugim. Potrebno je određivati parametre sustava u stvarnom vremenu kako bi znali kojom snagom treba goniti motor i koju silu stvarati na pedalama. Parametri sustava su masa, nagib, koeficijent otpora zraka i koeficijent trenja.

Ako bi bicikl uvijek radio u modu u kojem je prijenos momenta konstantan, određivanje ovih parametara ne bi bilo potrebno, ali ako želimo neke zanimljivije modove rada onda je potrebno. Jedan od tih modova bi bio onaj opisan u primjeru, kada se bicikl penje a vozač želi da mu se čini kao da vozi po ravnome. Parametri sustava nam omogućuju da poništimo samo silu koja je tu zbog nagiba. Naravno moguće bi bilo poništiti i samo silu trenja podloge, ili silu otpora zraka, ili sve 3 u bilo kojem iznosu, ali ih se zato mora znati, a da bi znali kolike one jesu moramo znati parametre sustava.

4. Sklopovlje

4.1. STM32F4

Električni bicikli neizbježno imaju računalo. Tako i bicikl na kojem mi radimo ima računalo. U našem slučaju je to ugradbeno računalo STM32F407. Ono se nalazi na razvojnoj pločici koja izgleda ovako:



Slika 2: STM32F407 (u sredini) na razvojnoj pločici

Ovo računalo ima raznih mogućnosti za povezivanje s drugom elektronikom. Dosta ulaza/izlaza se koristi za nešto, ali u okviru ovog završnog rada ćemo se fokusirati na SPI2 koji se nalazi na izvodima PB12(SS), PB13(SCK), PB14(MISO) i PB15(MOSI) ove razvojne pločice. Na ovo komunikacijsko sučelje ćemo spojiti modul za bežičnu komunikaciju RFM69.

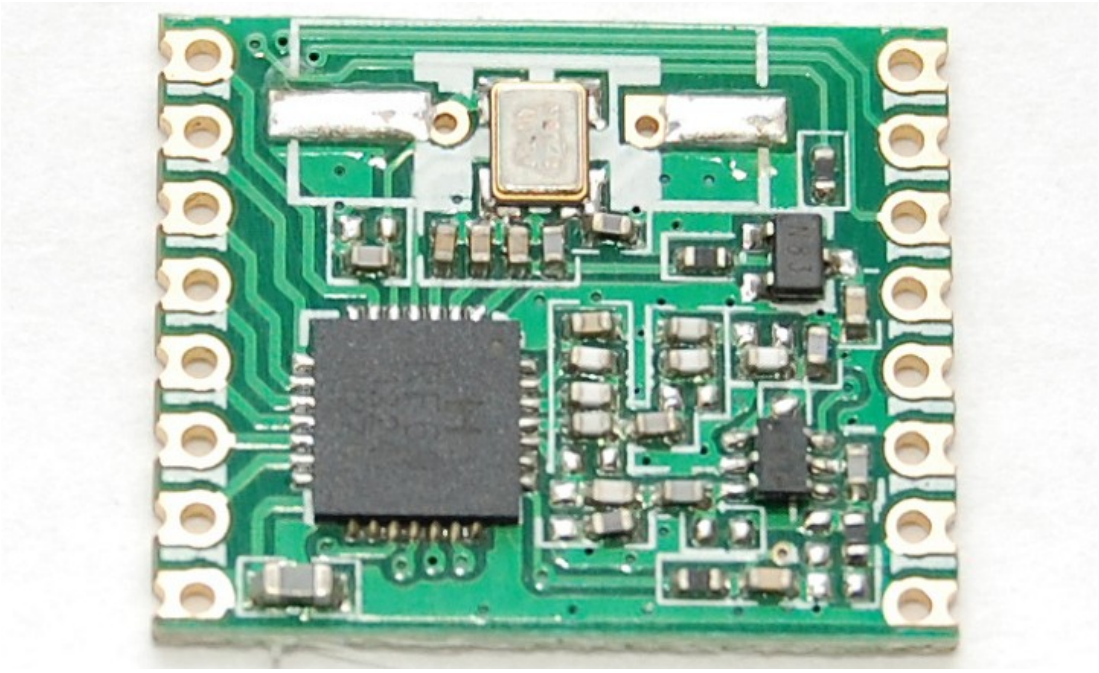
Na ugradbenom računalu će biti pokrenut ChibiOS, što je mali operacijski sustav koji se izvršava u stvarnom vremenu. To znači da je puno lakše upravljati vanjskim uređajima u odnosu na pravi operacijski sustav, ali

da su i dalje dostupne neke prednosti koje pruža operacijski sustav, kao na primjer izvršavanje programa u dretvama i apstrakcijski sloj nad vanjskim jedinicama.

Nažalost ne postoji gotova biblioteka funkcija napisana za STM32F407 koja bi omogućila upravljanje bežičnim modulom koji koristimo. Srećom postoji za Arduino, pa je ta biblioteka prilagođena za ugradbeno računalo koje mi koristimo, ali više o tome kasnije.

4.2. RFM69

Kao što je spomenuto, za bežičnu komunikaciju ćemo koristiti bežični modul RFM69. On koristi frekvenciju od 433MHz za prijenos informacija i u ovome slučaju je potrebna brzina prijenosa od barem 4 kbps. I više će biti potrebno kada se uzme u obzir da postoje određene neefikasnosti u prijenosu. Potrebno je poslati podatke za zaštitu poruke od greške i za šifriranje poruke. Modul izgleda ovako:



Slika 3: RFM69 bežični komunikacijski modul

Modulu se može dodati antena kako bi se povećao domet i prijenos učinio pouzdanijim. Nakon što se modul ispravno postavi, i programski i

sklopovski, komunikacija se odvija nesmetano. Ako bi veza slučajno pukla moduli je ponovno uspostave bez intervencije.

5. Opis rješenja

5.1. Prijelaz na kružno gibanje

Sve do sada smo koristili veličine za opis linearnog gibanja, međutim da bi ušli u svijet kotača morat ćemo koristiti neke poveznice na kružno gibanje. To srećom nije komplicirano pa se može sažeti u tablicu.

Linearno gibanje	Kružno gibanje	Poveznica
F [N]	*M [Nm]*	*M = Fr*
v [ms^{-1}]	*ω [rads^{-1}]*	*ω = v*r^{-1}*
a [ms^{-2}]	*ω_α [rads^{-2}]*	*ω_α = a*r^{-1}*

Tablica 1: Analogija između linearnog i kružnog gibanja

Ovdje je *r* radijus kotača. Navedena je samo ukupna sila *F* ali ista formula vrijedi i za sve ostale sile s kojima ćemo raditi u sustavu.

5.2. Određivanje sustava

Sada kada je sve spremno treba postaviti sustav. Sustav će biti jednadžba sila koje djeluju na bicikl u vožnji. Imamo ukupnu silu, silu motora, silu trenja i silu težu kada bicikl nije na ravnom. Jednadžba izgleda ovako:

$$F = F_m - F_{tr} - F_G \tag{10}$$

Kada se ubace pripadajući izrazi za sile dobije se:

$$ma = F_m - \vartheta v - \mu G\cos(\alpha) - G\sin(\alpha) \tag{11}$$

Ovo je sustav drugog reda jer je akceleracija druga derivacija puta koji bicikl prelazi.

Odavde pa nadalje ćemo uzeti da je *μGcos(α)* nula jer je koeficijent trenja *μ* jako mali(oko 0.004 za gumu i asfalt). Također, uzet ćemo da je komponenta *Gsin(α)* konstantna jer se može pretpostaviti da se neće mijenjati za vrijeme mjerenja pomaka kotača. Zašto se pomak kotača mora mjeriti? Mora se mjeriti kako bi se iz njega izračunala brzina kretanja bicikla i njegova akceleracija.

Neka je *φ(n)* diskretna funkcija kuta o vremenu. Razmak između svake vrijednosti kuta je fiksno vrijeme *T*. To vrijeme se još naziva period uzorkovanja i on nam je poznat jer ga mi određujemo. Iz kuta se lako da izračunati prijeđeni put:

$$x(n)=\varphi(n)r \tag{12}$$

Sada možemo zapisati brzinu i akceleraciju kao diferenciju prijeđenog puta:

$$v(n)=x'(n)\approx\frac{x(n)-x(n-1)}{T} \tag{13}$$

$$a(n)=x''(n)\approx\frac{x'(n)-x'(n-1)}{T} \tag{14}$$

Ako se u jednadžbu za akceleraciju (14) umjesto *x′* ubaci izraz (13) dobije se:

$$a(n)=x''(n)\approx\frac{\frac{x(n)-x(n-1)}{T}-\frac{x(n-1)-x(n-1-1)}{T}}{T} \tag{15}$$

I kada se još malo sredi dobije se:

$$a(n)=x''(n)\approx\frac{x(n)-2x(n-1)+x(n-2)}{T^2} \tag{16}$$

Kao što je rečeno, pretpostavit ćemo da je kut nagiba otprilike konstantan za vrijeme mjerenja, pa ćemo ga kraće zapisati sa F_C:

$$F_C=G\sin(\alpha) \tag{17}$$

I kada ovo ubacimo u jednadžbu sustava dobije se:

$$ma=F_m-\vartheta v-F_C \tag{18}$$

Što je jednadžba sa 3 nepoznanice.

Prebacimo otpor zraka i konstantnu silu na lijevu stranu, i ubacimo pripadajuće izraze za akceleraciju (16) i brzinu (13):

$$\frac{m}{T^2}[x(n)-2x(n-1)+x(n-2)]+\frac{\vartheta}{T}[x(n)-x(n-1)]-F_C=F_m(n) \tag{19}$$

Također dodajmo argument *n* ispred sile motora, jer se i ona mijenja sa vremenom.

Sada je potrebno dodatno srediti ovu jednadžbu na način da se grupiraju svi članovi sa jednakim kašnjenjem:

$$[\frac{m}{T^2}+\frac{\vartheta}{T}]x(n)-[\frac{2m}{T^2}+\frac{\vartheta}{T}]x(n-1)+\frac{m}{T^2}x(n-2)+F_C=F_m(n) \quad (20)$$

I još da ovaj zapis učinimo kompaktnijim koeficijente uz pozicije *x* zamjenimo svaki sa jednom oznakom:

$$a_0 x(n)-a_1 x(n-1)+a_2 x(n-2)+F_C=F_m(n) \quad (21)$$

Sada imamo kompaktnu jednadžbu sustava bicikla u vožnji.

U ovoj jednadžbi poznati su nam *x(n)* (i *x(n-1), x(n-2)* ... itd.), što je pozicija kotača i $F_m(n)$ što je moment motora. Koeficijenti uz ove varijable su nepoznate konstante. Da bi ih odredili morali bi imati barem 4 ovakve jednadžbe ali sa različitim vrijednostima varijabli. To se može jednostavno postići, samo je potrebno vršiti mjerenja u nekim periodičkim vremenskim razmacima. Argument *(n)* predstavlja trenutno mjerenje, a argument *(n-1)* predstavlja prethodno mjerenje i tako dalje. Dakle trebalo bi napraviti mjerenja do *(n-5)* kako bi se mogle napisati 4 jednadžbe (21), riješio sustav i izvukle nepoznate konstante.

4 jednadžbe su dovoljne, ali 4 jednadžbe znače (samo) 6 mjerenja. Želimo uključiti više od 6 mjerenja u izračun rješenja jer tako smanjujemo pripadajuću pogrešku. Točan broj mjerenja se određuje eksperimentalno, metodom pokušaja i pogreške.

Kada imamo više od 6 mjerenja, odnosno 4 jednadžbe, onda imamo preodređen sustav, jer imamo samo 4 nepoznanice, a 5 ili više jednadžbi. Takav sustav nema rješenje koje zadovoljava sve jednadžbe. Umjesto toga je moguće pronaći rješenje koje u nekoj mjeri zadovoljava sve jednadžbe. Za pronalazak takvog rješenja se koristi metoda najmanje kvadratne podrške.

5.3. Metoda najmanje kvadratne pogreške

Ideja iza metode najmanje kvadratne pogreške je da se u preodređenom sustavu jednadžbi nađe rješenje koje u nekoj mjeri zadovoljava sve jednadžbe u sustavu. Takvih rješenja ima beskonačno, ali ova metoda pronalazi ono koje najbolje zadovoljava jednadžbe sustava.

Da bi koristili ovu metodu potrebno je imati preodređeni sustav jednadžbi u matričnom obliku, pa ćemo napisati sustav (21) za više mjerenja u matričnom obliku:

$$\begin{bmatrix} x(n) & x(n-1) & x(n-2) & 1 \\ x(n-1) & x(n-2) & x(n-3) & 1 \\ x(n-2) & x(n-3) & x(n-4) & 1 \\ x(n-3) & x(n-4) & x(n-5) & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x(n-(N-1)) & x(n-N) & x(n-(N+1)) & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ F_C \end{bmatrix} = \begin{bmatrix} F_M(n) \\ F_M(n-1) \\ F_M(n-2) \\ F_M(n-3) \\ \vdots \\ F_M(n-(N-1)) \end{bmatrix} \quad (22)$$

Lijevu matricu, matricu pozicija, označimo sa Ψ, srednju matricu, matricu nepoznatih koeficijenata, označimo sa Θ, a desnu matricu, matricu sila motora, označimo sa Y. Matrica Ψ je dimenzije *Nx4*, a matrica Y je dimenzije *Nx1*. Ovu jednadžbu sada možemo napisati u kompaktnijem obliku:

$$\Psi\Theta = \mathrm{Y} \quad (23)$$

U ovoj jednadžbi još fali vektor stupac pogreške ε, jer metoda najmanjih kvadrata ne daje rješenje koje će lijevu stranu ove jednadžbe učiniti jednakom desnoj. Dat će rješenje koje dovodi lijevu stranu desnoj najbliže moguće. Zbog toga je potrebno dodati vektor pogreške kako bi jednadžba bila ispravna.

$$\Psi\Theta = \mathrm{Y} + \varepsilon \quad (24)$$

Budući da Ψ nije kvadratna matrica ne postoji njen inverz. Zbog toga je na prvi pogled jednadžba (24) nerješiva. Ako ne postoji njen inverz ne može se prebaciti na drugu stranu i "podijeliti" sa Y. Rješenje ovog problema daje metoda najmanje kvadratne pogreške, koja kaže da vrijedi sljedeće:

$$\Psi^T(\Psi\Theta - \mathrm{Y}) = 0 \quad (25)$$

0 na desnoj strani jednadžbe predstavlja nul-matricu. Ovdje vektor stupac pogreške ε više nije potreban i ova jednadžba vrijedi bez ikakvih dodatnih prilagodbi. Daljnjim sređivanjem se dobije:

$$\Psi^T\Psi\Theta = \Psi^T\mathrm{Y} \quad (26)$$

Sada je $\Psi^T\Psi$ kvadratna matrica i cijela jednadžba se može podijeliti s njom kako bi se dobio konačni izraz za nepoznate koeficijente:

$$\Theta = (\Psi^T \Psi)^{-1} \Psi^T \mathrm{Y} \tag{27}$$

Ovako određeni koeficijenti su najbolja procjena rješenja. Ako se ubace u jednadžbu (24) iz nje se može izračunati pogreška aproksimacije. Ta pogreška može poslužiti kako bi odredili kvalitetu rješenja i odlučili hoćemo li rješenje odbaciti ili zadržati.

5.4. Priprema za računanje koeficijenata

Kako bi izbjegli računanje inverza matrice vratimo se na jednadžbu (24). U njoj se i dalje jasno vidi da je ona zapravo sustav jednadžbi zapisan u matričnom obliku. Ovaj puta je to određeni sustav jednadžbi, jer se sastoji od 4 jednadžbe sa 4 nepoznanice. Da bi se takav sustav riješio može se napisati njegova izmijenjena matrica, u kojoj se zapravo spoje poznate matrice na lijevoj i desnoj strani na sljedeći način:

$$[\Psi^T \Psi : \Psi^T \mathrm{Y}] \tag{28}$$

Nakon toga je potrebno dovesti ovu matricu u reducirani oblik iz kojega se onda u najdesnijem stupcu iščita rješenje sustava. To rješenje je rješenje koje ima najmanju kvadratnu pogrešku kako je objašnjeno u prošloj podcjelini.

5.5. Određivanje nagiba (kut α)

Uz poznatu silu F_C i masu sustava kut se može lako odrediti. Budući da vrijedi:

$$F_C = G \sin(\alpha) \tag{29}$$

Pa je kut jednak:

$$\alpha = \arcsin\left(\frac{F_C}{G}\right) \tag{30}$$

Pozitivna vrijednost predstavlja uzbrdicu, a negativna nizbrdicu.

6. Sila na pedali

6.1. Opis problema

Budući da su pedale direktno spojene na dva motor-generatora, može se podešavati protumoment na njima. Taj protumoment će ovisiti o željenim karakteristikama prijenosa, ali ako vozač ne pritišće pedalu ne želimo da pedala ide unatrag. Zato moramo znati koliki moment djeluje na pedalu kako bismo mogli reagirati sa jednakim ili manjim protumomentom. Mjerenje momenta na pedali je izvedeno pomoću senzora pritiska. Taj senzor zatim bežično šalje podatke do glavnog računala(STM32F4) na obradu.

Podaci se primaju pomoću RFM69 modula sa obje pedale 80 puta u sekundi. Glavno računalo ih zatim obrađuje i primjenjuje odgovarajući protumoment na pedale. Ovdje ću samo ukratko opisati način na koji se podaci primaju i način njihove obrade.

6.2. Algoritam

Da bi upravljali RFM69 modulom koristimo biblioteku funkcija od LowPowerLab-a. Nažalost nije postojala gotova biblioteka za ugradbeno računalo STM32F4, pa je je bilo potrebno prilagoditi. To je uključivalo pretvaranje C++ klase u C strukturu, a klasinih metoda u funkcije. Također je bila potrebna prilagodba cijelog načina funkcioniranja biblioteke, jer je originalno pisana za Arduino i dosta se stvari razlikuje između Arduina i STM32F4.

Biblioteka funkcionira tako da se nakon inicijalizacije modula stalno provjerava jesu li došli neki novi podaci. Ako jesu funkcija koja služi za provjeru vraća identifikacijski broj modula koji je poslao podatak. Ako nije stiglo ništa vraća se nula.

Primljeni podatak se nalazi u 3 odvojena bajta i potrebno ga je spojiti u jedno. Također, budući da je primljeni podatak sirova vrijednost sa senzora, potrebno ga je dodatno prilagoditi. To podrazumjeva nuliranje i množenje kalibracijskom konstantom.

Da bi podatak spojili u jedan koristit ćemo ILI po bitovima (C operator |):

```
data = (data_piece[2] << 16) |
       (data_piece[1] << 8) |
       data_piece[0];
```

U pravom kodu je malo drukčije izvedeno ali ideja je ista.

Sirova vrijednost koju senzor daje je proporcionalna sa masom. Ovisnost mase o izlazu senzora je pravac koji ne prolazi kroz ishodište.

$$(masa\,na\,pedali)=(cal\,const)*((sirova\,vrijednost\,sa\,senzora)-(nulta\,masa)) \quad (31)$$

Taj pravac ne prolazi kroz ishodište zato što će uvijek neka masa biti prisutna na senzoru. U ovome slučaju to je masa podloge za vaganje, ali čak i kada podloge ne bi bilo i sam senzor ima neku težinu koja bi učinila da funkcija ovisnosti mase o izlazu senzora ne prolazi kroz ishodište. Taj problem je prisutan i na drugim vagama, ne samo na našoj. On se riješi tako da se početna masa svega na vazi(u našem slučaju pedali sa senzorom mase) oduzme od svake veće izmjerene mase. Tako se postigne da pravac ovisnosti mase o izlazu senzora prolazi kroz ishodište. Nagib toga pravca je kalibracijska konstanta. Ona se odredi tako da se uteg poznate mase postavi na vagu (pedalu) i pogleda se koju vrijednost senzor tada pokazuje. Od te se vrijednosti oduzme nulta masa i zatim se poznata masa utega podijeli s tim brojem. Tako se dobije kalibracijska konstanta. Ona ima jedinstvenu ali stalnu vrijednost za svaki senzor. Kada se od vrijednosti sa senzora oduzme nulta masa i to sve pomnoži s kalibracijskom konstantom dobije se trenutna masa na pedali. Sada je podatak sa senzora spreman za daljnju uporabu.

7. Zaključak

Puno fleksibilnosti se može dobiti izbacivanjem lanca i oslanjanjem na elektroniku da veže pedale i kotač. Upravljanje takvim sustavom može biti složeno ali zanimljivo. Šum je potrebno što više minimizirati kako ne bi pogubno utjecao na mjerenja. Pogotovo kada se radi o mjerenju kritičnih veličina za sustav, kao što su na primjer akceleracija i brzina i masa na pedali u ovom primjeru. Sve skupa je ispalo jako dobro i može se nazvati uspjehom.

8. Literatura

1. "Car", s Interneta, https://en.wikipedia.org/wiki/Car
2. Khan, Salman: "Least squares approximation", s Interneta, https://www.khanacademy.org/math/linear-algebra/alternate-bases/orthogonal-projections/v/linear-algebra-least-squares-approximation, 2009
3. Khan, Salman: "A projection onto a subspace is a linear transformation", s Interneta, https://www.khanacademy.org/math/linear-algebra/alternate-bases/orthogonal-projections/v/lin-alg-a-projection-onto-a-subspace-is-a-linear-transforma, 2009

Dodatak A: Programska podrška

Ovdje će biti navedeni izvorni programski kodovi nekih funkcija koje su bitne za funkcioniranje sustava za primanje podataka o pritisku na pedalu bežično.

Krenimo sa strukturom koja drži podatke o bežičnom komunikacijskom modulu:

```
typedef struct{
    volatile uint8_t DATA[RF69_MAX_DATA_LEN];
    volatile uint8_t DATALEN;
    volatile uint8_t SENDERID;
    volatile uint8_t TARGETID;
    volatile uint8_t PAYLOADLEN;
    volatile uint8_t ACK_REQUESTED;
    volatile uint8_t ACK_RECEIVED;
    volatile int16_t RSSI;
    volatile uint8_t _mode;
    uint8_t _slaveSelectPin;
    uint8_t _address;
    bool_t _promiscuousMode;
    uint8_t _powerLevel;
    bool_t _isRFM69HW;

}RFM69;
```

Kada funkcije barataju sa RFM69 modulom onda koriste ovu strukturu za spremanje podataka. DATA je jako bitna varijabla u koju se spremaju dolazni podatci, DATALEN govori o duljini primljenih podataka. DATALEN će uvijek biti 3 jer se sa pedale uvijek šalju 3 bajta. Još jedna bitna varijabla je SENDERID koja govori s koje pedale je stigao zadnji primljeni podatak.

Postoji 31 funkcija za baratanje RFM69 modulom.

```
bool_t RFM69Initialize(RFM69 *rfm69, uint8_t freqBand,
uint8_t ID, uint8_t networkID);

void RFM69SetAddress(RFM69 *rfm69, uint8_t addr);

void RFM69SetNetwork(uint8_t networkID);

bool_t RFM69CanSend(RFM69 *rfm69);

void RFM69Send(RFM69 *rfm69, uint8_t toAddress, const
void* buffer, uint8_t bufferSize, bool_t requestACK);

bool_t RFM69SendWithRetry(RFM69 *rfm69, uint8_t
toAddress, const void* buffer, uint8_t bufferSize,
uint8_t retries, uint8_t retryWaitTime); // 40ms
roundtrip req for 61byte packets

bool_t RFM69ReceiveDone(RFM69 *rfm69);

bool_t RFM69ACKReceived(RFM69 *rfm69, uint8_t
fromNodeID);

bool_t RFM69ACKRequested(RFM69 *rfm69);

void RFM69SendACK(RFM69 *rfm69, const void* buffer,
uint8_t bufferSize);

uint32_t RFM69GetFrequency(void);

void RFM69SetFrequency(RFM69 *rfm69, uint32_t freqHz);

void RFM69Encrypt(RFM69 *rfm69, const char* key);

int16_t RFM69ReadRSSI(bool_t forceTrigger);

void RFM69Promiscuous(RFM69 *rfm69, bool_t onOff);

void RFM69SetHighPower(RFM69 *rfm69, bool_t onOFF); //
has to be called after initialize() for RFM69HW

void RFM69SetPowerLevel(RFM69 *rfm69, uint8_t level); //
reduce/increase transmit power level

void RFM69Sleep(RFM69 *rfm69);

uint8_t RFM69ReadTemperature(RFM69 *rfm69, uint8_t
calFactor); // get CMOS temperature (8bit)

void RFM69RcCalibration(void);

uint8_t RFM69ReadReg(uint8_t addr);

void RFM69WriteReg(uint8_t addr, uint8_t val);

void RFM69ReadAllRegs(void);

void RFM69SendFrame(RFM69 *rfm69, uint8_t toAddress,
const void* buffer, uint8_t size, bool_t requestACK,
bool_t sendACK);

uint8_t RFM69PollforData(RFM69*, uint32_t*);
```

```
void RFM69InterruptHandler(RFM69 *rfm69);
void RFM69ReceiveBegin(RFM69 *rfm69);
void RFM69SetMode(RFM69 *rfm69, uint8_t mode);
void RFM69SetHighPowerRegs(bool_t onOff);
void RFM69Select(void);
void RFM69Unselect(void);
```

RFM69InterruptHandler() je jedna od bitnijih.

```
void RFM69InterruptHandler(RFM69 *rfm69) {
     uint8_t i;
     if (rfm69->_mode == RF69_MODE_RX &&
     (RFM69ReadReg(REG_IRQFLAGS2) &
     RF_IRQFLAGS2_PAYLOADREADY))
     {
          RFM69SetMode(rfm69, RF69_MODE_STANDBY);
          RFM69Select();
          SPITransfer(spi, REG_FIFO & 0x7F);
          rfm69->PAYLOADLEN = SPITransfer(spi, 0);
          rfm69->PAYLOADLEN = rfm69->PAYLOADLEN > 66 ?
          66 : rfm69->PAYLOADLEN; // precaution
          rfm69->TARGETID = SPITransfer(spi, 0);
          if(!(rfm69->_promiscuousMode || rfm69-
          >TARGETID == rfm69->_address || rfm69-
          >TARGETID == RF69_BROADCAST_ADDR)
          || rfm69->PAYLOADLEN < 3)
          {
               rfm69->PAYLOADLEN = 0;
               RFM69Unselect();
               RFM69ReceiveBegin(rfm69);
               return;
          }
          rfm69->DATALEN = rfm69->PAYLOADLEN - 3;
          rfm69->SENDERID = SPITransfer(spi, 0);
          uint8_t CTLbyte = SPITransfer(spi, 0);
```

```
            rfm69->ACK_RECEIVED = CTLbyte &
            RFM69_CTL_SENDACK;
            rfm69->ACK_REQUESTED = CTLbyte &
            RFM69_CTL_REQACK;
            for (i = 0; i < rfm69->DATALEN; i++)
            {
                  rfm69->DATA[i] = SPITransfer(spi, 0);
            }
            if (rfm69->DATALEN < RF69_MAX_DATA_LEN) rfm69-
            >DATA[rfm69->DATALEN] = 0;
            RFM69Unselect();
            RFM69SetMode(rfm69, RF69_MODE_RX);
      }
      rfm69->RSSI = RFM69ReadRSSI(FALSE);
}
```

Ona služi za provjeravanje jesu li stigli kakvi novi podatci. Iako se zove "Interrupt Handler" što bi u prijevodu značilo rukovatelj prekidima, ova funkcija se ne poziva pomoću prekida. To ime je ostatak, dok biblioteka nije bila prilagođena za STM32. Sada je ovu funkciju potrebno pozvati negdje u kodu kako bi provjerila ima li novih podataka. Upravo se to radi u funkciji RFM69PollforData().

```
uint8_t RFM69PollforData(RFM69 *rfm69, uint32_t *data)
{
      uint8_t i;
      (*data) = 0;
      RFM69InterruptHandler(rfm69);
      if(RFM69ReceiveDone(rfm69) == FALSE)
            return 0;
      for (i = 0; i < rfm69->DATALEN; i++)
            (*data) |= (unsigned long)(rfm69->DATA[i]) <<
                        i*8;
      return rfm69->SENDERID;
}
```

Ova funkcija nije postojala u originalnoj biblioteci već je dopisana kako bi se olakšalo korištenje biblioteke. Ona poziva funkciju RFM69InterruptHandler() koja provjeri ima li novih podataka. Ako ima ti se podatci zapišu na adresu koja je predana kao argument funkciji i vrati se identifikacijski broj pošiljatelja (pedale), a ako nema vraća se 0.