UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI FAKULTET

DETEKCIJA ORIJENTACIJE U 3 DIMENZIJE SA MPU-9250 SENZOROM I STM32F051R8 MIKROKONTROLEROM

Diplomski rad

Mentor:
Prof. dr. Nenad Jovičić

Kandidat:
Danijel Čamdžić, 0205/2016

Beograd, Septembar 2020.

Sadržaj

1. UVOD

U ovom diplomskom radu objašnjava se način na koji je izvedena fuzija podataka sa senzora akcelerometra, žiroskopa i magnetometra u cilju detektovanja rotacije senzora u tri dimenzije kao i način programiranja mikrokontrolera za obavljanje potrebnih funkcija. Upotrebljeni mikrokontroler je STM32F051R8 dok je upotrebljena periferija MPU-9250 u kojoj se nalaze sva tri senzora (akcelerometar, žiroskop i magnetometar). Radno okruženje je *Keil*[1] dok je programski jezik u kom je mikrokontroler programiran C.

Algoritam fuzije senzorskih podataka uključuje filtriranje upotrebom Kalmanovog filtra nad podacima dobijenih iz senzora akcelerometra i žiroskopa u cilju dobijanja rezultata koji ne pate od problema sa kojima se suočavaju ova dva senzora. Tačnije, podaci sa senzora akcelerometra pate od velikog šuma dok podaci sa senzora žiroskopa pate od gubitka tačnosti usled integraljenja podataka jer se integrali u diskretnim vremenskim trenucima. Ova dva senzora zajedno sa kalmanovim filtriranjem podataka omogućavaju da se odrede dvije od tri ose rotacije tj. *roll* i *pitch* dok treću osu koju čini *yaw*[2] nije moguće odrediti jer je tu osu nemoguće dobiti iz senzora akcelerometra već samo iz senzora žiroskopa pa se podaci ne mogu filtrirati jer ih imamo iz samo jednog izvora. Treću osu *yaw* dobijamo iz senzora magnetometra. Podaci iz magnetometra pate od problema koji uključuju šum i problem samog senzora koji nam podatke šalje brzinom od 12 Hz, što je daleko sporije od podataka za druge dvije ose. Problem šuma rješava se niskofrekventnim filtrom u kombinaciji sa Kalmanovim filtrom - fuzijom sa podacima iz žiroskopa koji u prethodnom slučaju nisu našli svoj adekvatan par za Kalmanovo filtriranje. Metoda *kompenzacije nagiba* neophodna je da vrijednost *yaw* bude tačna i u slučaju kada senzor nije paralelan sa ravni Zemlje. Za tu vrstu kompenzacije koriste se matematičke formule koji zahtijevaju unaprijed poznate vrijednosti druge dvije ose *roll* i *pitch*.

Takođe važan dio diplomskog rada jeste i rad sa mikrokontrolerom koji je programiran *bare-metal* metodom koja predstavlja način programiranja mikrokontrolera sa niskim nivoom apstrakcije u kojoj se ne koriste ugrađene i dostupne biblioteke i funkcije[3] već se sve radi i programira na nivou registara i adresa unutar mikrokontrolera. Time se obezbjeđuje potpuna kontrola programera nad mikrokontrolerom i što manje bacanje vremenskih resursa na bespotrebne provjere i postavke koje ugrađene funkcije mogu imati.

Detekcija orijentacije predstavlja važan aspekt mnogih inženjerskih disciplina među kojima su i vojni, hirurški i navigacioni sistemi, virtuelna realnost, prepoznavanje pokreta i slično. Za razliku od drugih poznatih metoda koje uključuju eksterne senzore (akustične, radarske) za detekciju položaja tijela, a koje imaju velike limitacije u vidu interferencije, problem sijenki, interapcija svijetla i dr., upotreba MEMS (*Micro Electro-Mechanical System*) senzora ne uključuje takve limitacije jer se zasnivaju na detektovanju fizičkih atributa tijela koje se prati.

[1] Keil® MDK 5, www.keil.com
[2] Roll, pitch i yaw su uglovi valjanja, propinjanja i skretanja u nautičkim terminima
[3] Jedna od takvih dostupnih biblioteka je HAL biblioteka

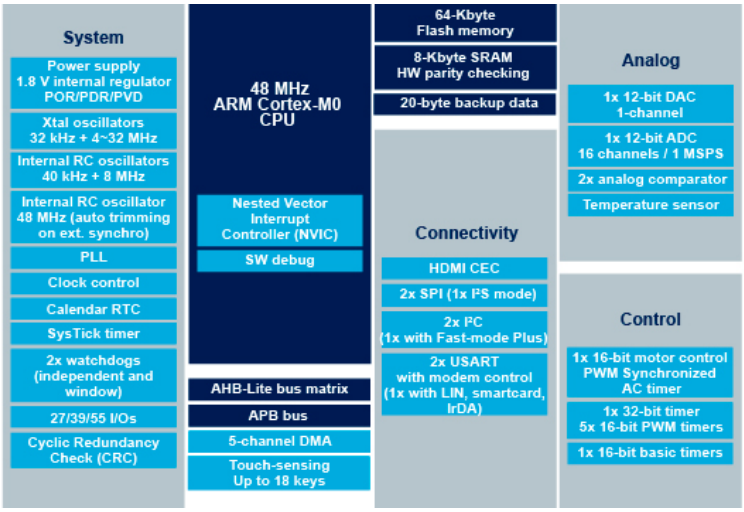
2. INICIJALIZACIJA MIKROKONTROLERA

Pod inicijalizacijom mikrokontrolera podrazumijeva se konfiguracija i startovanje periferija koje su neophodne da bi se uspostavila pravilna veza između senzora i mikrokontrolera (I2C), periferija koje su neophodne za pravilno funkcionisanje projekta (TIMx, USART), kao i periferija čije je konfigurisanje neophodno za željeno funkcionisanje mikrokontrolera koji koristimo (GPIO...).

Ploča koja je korištena u ovom projektu je NUCLEO-F030R8 ploča sa STM32F030R8 MCU (*Microcontroller Unit*) koja podržava arduino i ST morpho konekciju. STM32F030R8 MCU bazira se na ARM Cortex-M0 procesoru i u sebi sadrži 64 KB Flash memorije i CPU maksimalne brzine od 48 MHz. Na slici 2.0[4] prikazane su karakteristike STM32F030R8 mikrokontrolera sa svim periferijama koje on sadrži dok je na slici 2.1 prikazan izgled NUCLEO ploče. U ovom poglavlju bazirat ćemo se na konfiguraciji samo onih periferija koje su nam neophodne za pravilno funkcionisanje projekta. Pod tim se podrazumijeva konfiguracija registara za rad sa:

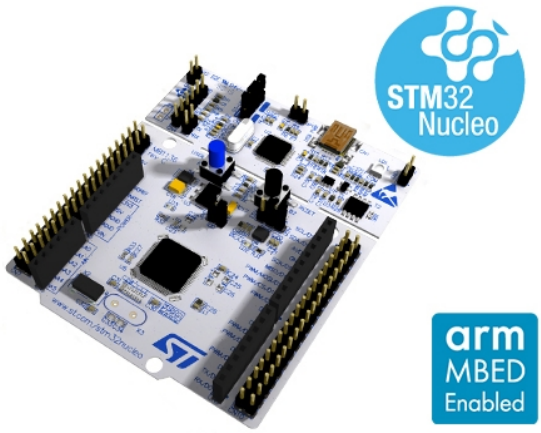
- GPIOx
- TIM3
- USART2
- I2C1

kao i konfiguracija takta mikrokontrolera za rad na 48 MHz.



Slika 2.0. Karakteristike STM32F030R8 mikrokontrolera

[4] Slike 2.0 i 2.1 preuzete su sa st.com



Slika 2.1. NUCLEO-F030R8 ploča

Naredna potpoglavlja bazirat će se na detaljnijem prikazu načina konfiguracije pojedinih periferija sa priloženim dijelovima koda za programiranje periferije kao i dijelovima datasheet-a u kojima je dat slikovit prikaz potrebnih registara.

2.1. Konfiguracija RCC registara (Reset and Clock Control)

Konfiguracija RCC registara uključuje odabiranje PLL-a (*Phase Locked Loop*) kao izvora za takt sistema a zatim podešavanje PLL-a da radi na 48 MHz. Na slici 2.1.0[2] prikazan je RCC_CR registar u kome se vrši kontrola rada (uključivanje i isključivanje PLL-a), kao i provjera spremnosti (PLLRDY bit). Prilikom uključivanja ili isključivanja PLL-a kao i promjene režima rada neophodno je sačekati PLLRDY bit kako bi smo znali da je PLL spreman za nastavak rada i da je pravilno konfigurisan.

Na slici 2.1.1[2] prikazan je RCC_CFGR registar u kom se bitima PLLMUL3...0 konfiguriše brzina rada. Upisivanjem bitskog 12 na to mjesto brzina PLL-a postavlja se na 48 MHz. SW bitima se kontroliše izvor takta i upisivanjem bitske 2 odabira se PLL kao izvor za takt. Programski kod koji konfiguriše registre dat je ispod i jednostavnim pozivanjem funkcije prije glavne petlje registri će biti konfigurisani na pravilan način.

```
/**
 * @brief Clock_init
 *
 * Function that intializes the clock frequency and the source of clock for the STM32F030R8 microncontroller.
 *
 */
void Clock_Init(void){
   /* Use PLL as Clock Source and Mul by 12 */
   RCC->CR &= ~(RCC_CR_PLLON);                  /**< Turn off PLL */
   while(RCC->CR & RCC_CR_PLLRDY);               /**< Wait until PLL is ready */
   RCC->CFGR |= RCC_CFGR_PLLMUL12;               /**< Multiply PLL value by 12. Must not exceed 48 Mhz (Max
frequency)! */
   RCC->CR |= RCC_CR_PLLON;                      /**< Turn on PLL */
```

```
   while(!(RCC->CR & RCC_CR_PLLRDY));          /**< Wait until PLL is ready in order to avoid messing stuff up
*/

   FLASH->ACR |= FLASH_ACR_LATENCY;          /**< Add ONE wait state for the flash acces time because 24
Mhz <= SYSCLOCK <= 48MHz */
   RCC->CFGR &= ~(RCC_CFGR_PPRE);            /**< Clear PCLK prescaler settings for safety */
   RCC->CFGR |= RCC_CFGR_PPRE_DIV16;         /**< HCLK divided by 16 */
   RCC->CFGR &= ~(RCC_CFGR_HPRE);            /**< Clear HCLK prescaler settings */
   while(!(RCC->CR & RCC_CR_PLLRDY));        /**< Wait until PLL is ready */
   RCC->CFGR &= ~(RCC_CFGR_SW);              /**< Clear System clcok switch settings */
   RCC->CFGR |= RCC_CFGR_SW_PLL;             /**< Select PLL as source */
   RCC->CFGR &= ~(RCC_CFGR_PPRE);           /**< Clear preslacer settings set before for safety */
}
```

6.4.1 Clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	PLL RDY	PLLON	Res.	Res.	Res.	Res.	CSS ON	HSE BYP	HSE RDY	HSE ON
						r	rw					rw	rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSI RDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Slika 2.1.0. RCC_CR registar

6.4.2 Clock configuration register (RCC_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PLL NODIV	MCOPRE[2:0]			MCO[3:0]				Res.	Res.	PLLMUL[3:0]				PLL XTPRE	PLL SRC[1]
rw	rw	rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PLL SRC[0]	ADC PRE	Res.	Res.	Res.	PPRE[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw				rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Slika 2.1.1. RCC_CFGR registar

2.2. Konfiguracija GPIO registara (General Purpose I/Os)

Konfiguracija GPIO registara podrazumijeva omogućavanje takta određenim registrima koji se koriste za slanje/primanje podataka ka/iz senzora kao i omogućavanje takta za pin koji je vezan za LED diodu koja se koristi za signalizaciju prilikom kalibracije magnetometra. Na slici 2.2.0[2] prikazan je RCC_AHBENR registar koji služi za kontrolu pristupa takta nad određenim portovima.

Upisivanjem logičke jedinice u bit IOPAEN takt je dostupan svim pinovima iz tog registra. Na slici 2.2.1[2] prikazan je registar GPIOx_MODER koji služi za određivanje režima rada određenog pina u okviru registra u kom se taj pin nalazi. Za kontrolu LED diode (pin 5) neophodno je da taj pin bude deklarisan kao izlazni pin u GPIOA_MODER registru. MODER5 biti moraju biti postavljeni na bitsku jedinicu za izlazni režim. Programski kod za ovu jednostavnu konfiguraciju dat je ispod.

```
/**
 * @brief GPIO_Init
 *
 * Function that initializes the GPIO pins.
 *
 */
void GPIO_Init(void) {
   /* Enable clock for GPIOs */
   RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
   RCC->AHBENR |= RCC_AHBENR_GPIOBEN;
   RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
   RCC->AHBENR |= RCC_AHBENR_GPIOFEN;

   GPIOA->MODER |= GPIO_MODER_MODER5_0;       /**< Output mode (01) for bit 5 */
}
```

6.4.6 AHB peripheral clock enable register (RCC_AHBENR)

Address offset: 0x14

Reset value: 0x0000 0014

Access: no wait state, word, half-word and byte access

Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	TSCEN	Res.	IOPFEN	IOPEEN	IOPDEN	IOPCEN	IOPBEN	IOPAEN	Res.
							rw		rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CRCEN	Res.	FLITFEN	Res.	SRAMEN	DMA2EN	DMAEN
									rw		rw		rw	rw	rw

Slika 2.2.0. RCC_AHBENR registar

8.4.1 GPIO port mode register (GPIOx_MODER) (x =A..F)

Address offset:0x00

Reset values:

- 0x2800 0000 for port A
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Slika 2.2.1. GPIOx_MODER registar

2.3. Konfiguracija TIM3 Tajmera (TIMER 3)

Periferija tajmera (TIM3) ima važnu ulogu pri uspostavljanju komunikacije između mikrokontrolera i senzora. Za pravilnu konfiguraciju senzora, neophodno je nakon određenih izmjena režima rada sačekati određeni vremenski interval da bi se rad sa senzorom mogao nastaviti tj. da bi se znalo da je senzor pravilno konfigurisan. Svi vremenski intervali koji se čekaju mjere se ovom periferijom. Takođe, njegova uloga značajna je i pri samoj obradi podataka pristiglih iz senzora jer TIM3 periferija mjeri vremenki interval *dt* koji je protekao od prethodnog pristiglog validnog podatka. Taj vremenski interval koristi se za integraciju podataka iz senzora žiroskopa da bi se ugaona brzina koju dobijamo iz senzora pretvorila u ugaonu poziciju koju pokušavamo da odredimo u ovom projektu. Diskretna priroda ovog načina integracije podataka dovodi do akumulacije grešaka pri računanju pa ugaona pozicija dobijena iz senzora žiroskopa veoma brzo, vremenom, gubi tačnost.

Na slici 2.3.0[2] i 2.3.1[2] prikazan je izgled dva najvažnija registra za konfiguraciju periferije TIM3, TIM3_PSC i TIM3_ARR registri respektivno. Pošto je TIM3 16-bitni tajmer, TIM3_ARR registar služi za postavljanje granične vrijednosti do koje će taj tajmer brojati nakon čega se vrijednost tajmera restartuje i započinje novo brojanje ukoliko je brojanje tajmera omogućeno. TIM3_PSC registar služi za postavljanje vrijednosti preskaliranja brzine brojanja. Ta vrijednost će inkrementirana za jedan, dijeliti sistemsku brzinu mikrokontrolera i konačan rezultat biće brzina brojanja periferije TIM3. Formula 2.3.0 prikazuje matematičku vezu između vrijednosti preskaliranja, sistemske brzine i brzine brojanja periferije TIM3. Bez dodatne konfiguracije, tajmer je postavljen u režim brojanja na gore.

$$F_{TIM3} = \frac{F_{sys}}{PSC+1} \tag{2.3.0}$$

18.4.11 TIM2 and TIM3 prescaler (TIM2_PSC and TIM3_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Slika 2.3.0. TIM3_PSC registar

18.4.12 TIM2 and TIM3 auto-reload register (TIM2_ARR and TIM3_ARR)

Address offset: 0x2C

Reset value: 0xFFFFFFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ARR[31:16] (TIM2 only)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Slika 2.3.1. TIM3_ARR registar

Vrijednost preskaliranja tajmera postavljena je na 24000 što u kombinaciji sa sistemskom brzinom od 48MHz čini brzinu brojanja od 2KHz. Vrijednost TIM3_ARR registra postavljena je na 2000 što u kombinaciji sa brzinom brojanja od 2KHz čini period brojanja od 1 sekunde. S obzirom da se svi događaji unutar projekta dešavaju sa znatno manjim periodom, ovaj način konfiguracije tajmera sasvim zadovoljava potrebe našeg projekta. Startovanje tajmera predstavlja upisivanje logičke jedinice kao vrijednost bita CEN unutar TIM3_CR1 registra dok se očitavanje vrijednosti do koje je tajmer izbrojao vrši iz TIM3_CNT registra. Programski kod kojim se periferija TIM3 konfiguriše na pomenute vrijednosti dat je ispod kao i dio koda koji prikazuje startovanje tajmera, čekanje da izbroji do vrijednosti koja predstavlja 500ms[5] i očitavanje te vrijednosti.

```
/**
 * @brief TIMER3_Init
 *
 * Function that initializes the TIMER3 peripheral with it's frequency and up count max.
 *
 */
void TIMER3_Init(void) {
   /* Configuring Timer (TIM3) */
   RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
   TIM3->SR = 0;                                  /**< Resetting the SR register of TIM3 */
   TIM3->PSC = (uint16_t)(24000-1);               /**< SYSCLK/(PSC+1) = TIM3_frequency */
   TIM3->ARR = (uint16_t)1999;                    /**< Counts to 1999 and then goes to 0. */
}


/* Starting the timer 3 peripheral and counting for 500 ms to change the mode of operation */
TIM3->CNT = 0x00;
TIM3->CR1 |= TIM_CR1_CEN;            /**< Enabling the timer 3 peripheral */
while(1){
  if(TIM3->CNT > 1000)
      break;
}
TIM3->CR1 &= ~TIM_CR1_CEN;         /**< Disabling the timer 3 peripheral */
TIM3->CNT = 0x00;
```

2.4. Konfiguracija USART2 Registara (Universal synchronous asychronous receiver trasmitter)

Konfiguracija USART2 registara nije neophodna za pravilno funkcionisanje mikrokontrolera ili senzora ali jeste za dio projekta koji se zasniva na vizuelizaciji detekcije orijentacije u vidu 3D kutije koja se prikazuje na ekranu računara i predstavlja trenutni položaj senzora. Taj vid komunikacije odvija se između mikrokontrolera i računara preko *Processing Software-a* u kom se pomenuta kutija iscrtava. Računar prima podatke o sve tri ose rotacije i osvježava trenutni prikaz slika na svakih 50Hz što predstavlja brzinu kojom dobijamo nove podatke.

[5] Ovu vrijednost koristimo kao vremenski period koji mora da protekne između dvije promjene režima rada senzora magnetometra

Konfiguracija USART2 registara podrazumijeva da se omogući sistemski takt pinovima kojima je dodijeljena ta funkcija. Dodijeljivanje funkcije tim pinovima vrši se u GPIOA_AFR i GPIOA_MODER registrima pri čemu se odabira da je funkcija pinova alternativna (pin 2 i pin 3). Neophodno je omogućiti prijem i slanje podataka upisivanjem logičke jedinice u bite RE i TE, respektivno, koji se nalaze u USART2_CR1 registru a prije toga onemogućiti korištenje te periferije upisivanjem logičke jedinice na mjesto bita UE (USART ENABLE). Pored toga, neophodno je upisati i odgovarajući baud rate[6] koji je odabran da bude 115200. Ovo se postiže upisivanjem vrijednosti 417 (brzina sistemskog takta podijeljena sa baud rate-om) u registar USART2_BRR nakon čega je potrebno ponovo omogućiti periferiju upisivanjem logičke jedinice na mjesto bita UE. Na slici 2.4.0[2] dat je prikaz USART2_CR1 registra. Programski kod kojim se konfiguriše USART dat je ispod. Način na koji se konfiguriše i inicira slanje podataka sa mikrokontrolera na računar biće detaljno prikazan u poglavlju koji se bavi vizuelizacijom rezultata.

```
/**
 * @brief USART2_Init
 *
 * Function that initializes UART for the STM32F030R8 microcontroller to be used to transfer data to the PC in order
 * to visualize orientation.
 *
 */
void USART2_Init(void) {
   /* Configuring UART (USART2) */
   RCC->APB1ENR |= RCC_APB1ENR_USART2EN;                /**< Enable clock for the USART2 */

   GPIOA->AFR[0] |= (GPIO_AFRL_AFSEL2 & 0x100) | (GPIO_AFRL_AFSEL3 & 0x1000);
   GPIOA->MODER |= GPIO_MODER_MODER2_1 | GPIO_MODER_MODER3_1;    /**< Alternate function for pins 2
and 3 */

   USART2->CR1 &= ~(USART_CR1_UE);                      /**< Disable UART */
   USART2->CR1 |= USART_CR1_TE | USART_CR1_RE;          /**< Enable transmit and receive */

   USART2->BRR = (uint32_t)417;                         /**< 48MHz/115200 = 416.66 -> 417 */

   USART2->CR1 |= USART_CR1_UE;                         /**< Enable UART */
}
```

27.8.1 Control register 1 (USART_CR1)

Address offset: 0x00

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	M1	EOBIE	RTOIE	DEAT[4:0]					DEDT[4:0]				
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	CMIE	MME	M0	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	UESM	UE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Slika 2.4.0. USART2_CR1 registar

[6] Brzina prenosa u komunikacionom kanalu. Označava maksimalan broj bita koji se mogu prenijeti u sekundi

2.5. Konfiguracija I2C1 Registara (Inter-Integrated Circuit)

Komunikacija između mikrokontrolera i senzora obavlja se preko I2C protokola. Početna stvar kod konfigurisanja I2C1 registara jeste konfiguracija registara pinova kojima će biti dodijeljena funkcija I2C pinova (SCL za takt i SDA za slanje podataka). Za ovu vrstu odabrani su pinovi 8 i 9 u B seriji registara. Konfiguracija ovih registara podrazumijeva selekciju alternativne funkcije za pinove 8 i 9, slično kao u slučaju za USART portove (GPIOB_AFR i GPIOB_MODER registri), odabir brzine promjene izlaza porta u registru GPIOB_OSPEEDR (u našem slučaju to je high-speed), kao i odabiranja tipa porta koji je u našem slučaju open-drain sa pull-up otpornikom što se konfiguriše u registrima GPIOB_OTYPER i GPIOB_PUPDR respektivno. Na slici 2.5.0[2] prikazan je izgled GPIOB_PUPDR registra gdje se upisivanjem bitske vrijednosti 01 na mjesto portova 8 i 9 odabira pull-up otpornik. Programski kod koji vrši ovakva podešavanja nad registrima I2C pinova dat je ispod.

```
RCC->AHBENR |= RCC_AHBENR_GPIOBEN;          /**< Enable clock for the GPIOB (Pins for I2C1) */

GPIOB->AFR[1] |= (GPIO_AFRH_AFSEL8 & 0x01) | (GPIO_AFRH_AFSEL9 & 0x10);
GPIOB->MODER |= GPIO_MODER_MODER8_1 | GPIO_MODER_MODER9_1;      /**< Alternate function for the
pins 8 and 9 */
GPIOB->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR8_1 | (GPIO_OSPEEDR_OSPEEDR8_1 >> 1)|
GPIO_OSPEEDER_OSPEEDR9_1 | (GPIO_OSPEEDR_OSPEEDR9_1 >> 1);  /**< High speed */
GPIOB->OTYPER |= GPIO_OTYPER_OT_8 | GPIO_OTYPER_OT_9;              /**< Open drain type */
GPIOB->PUPDR |= GPIO_PUPDR_PUPDR8_0 | GPIO_PUPDR_PUPDR9_0;      /**< Pull-up resistor */
```

8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..F)

Address offset: 0x0C

Reset values:

- 0x2400 0000 for port A
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Slika 2.5.0. GPIOB_PUPDR registar

Jednostavniji dio vezan za konfiguraciju I2C1 protokola je konfiguracija samih I2C1 reigstara jer se sastoji samo od definisanja brzine protokola u I2C1_TIMINGR registru. U našem slučaju, brzina će biti podešena na 100KHz. Prilikom promjene brzine protokola, neophodno je prethodno onemogućiti rad periferije, upisivanjem logičke nule na mjestu bita PE (PERIPHERAL ENABLE) u I2C1_CR1 registru a zatim je ponovo omogućiti kada se završi sa željenom

konfiguracijom. Na slici 2.5.1[2] prikazan je izgled registra I2C1_CR1 dok je programski kod kojim se konfiguriše brzina dat ispod paragrafa.

```
RCC->APB1ENR |= RCC_APB1ENR_I2C1EN;                 /**< Enable clock for I2C1 */

I2C1->CR1 &= ~(I2C_CR1_PE);                         /**< Disable I2C1 peripheral */
I2C1->TIMINGR = 0x2000090E & 0xF0FFFFFFU;           /**< Adjust timing */
I2C1->CR1 |= I2C_CR1_PE;
```

26.7.1 Control register 1 (I2C_CR1)

Address offset: 0x00

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

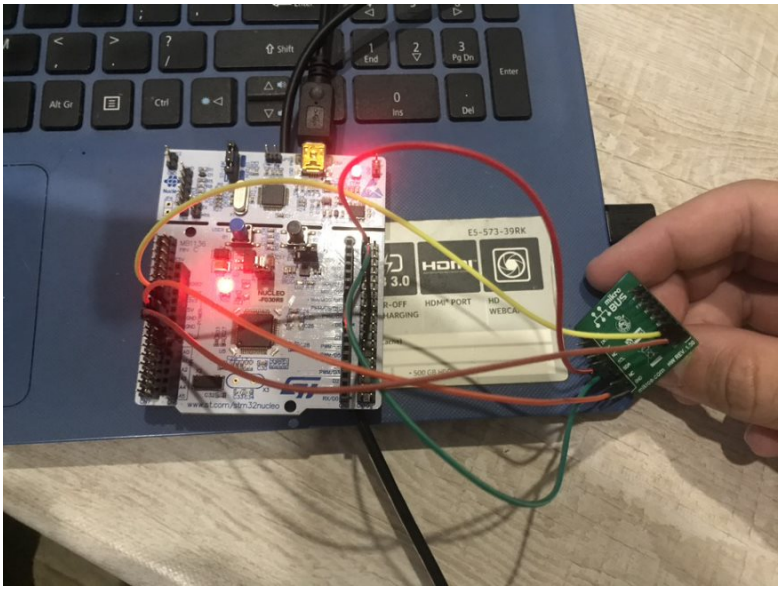
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PECEN	ALERT EN	SMBD EN	SMBH EN	GCEN	WUPE N	NOSTR ETCH	SBC
								rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXDMA EN	TXDMA EN	Res.	ANF OFF	DNF				ERRIE	TCIE	STOP IE	NACK IE	ADDR IE	RXIE	TXIE	PE
rw	rw		rw	rw				rw	rw	rw	rw	rw	rw	rw	rw

Slika 2.5.1. I2C1_CR1 registar

3. KOMUNIKACIJA SA SENZOROM

Komunikacija između mikrokontrolera i senzora obavlja se preko I2C protokola. Za obavljanje I2C funkcije izabrani su portovi 8 i 9 iz GPIOB grupe pinova. Na slici 3.1 prikazana je povezanost mikrokontrolera i senzora. Portovi na senzoru i mikrokontroleru označeni sa SDA i SCL predstavljaju osnov I2C protokola. SDA linija služi za obostrano slanje podataka dok SCL linija služi kao I2C takt koji radi brzinom od 100KHz. Druge linije veza prikazane na slici su za napajanje od 3.3V i uzemljenje (GND).



Slika 3.1. Povezanost mikrokontrolera i senzora

U datasheet-u MPU-9250 senzora[4] može se pronaći opis I2C protokola koji je potrebno ispoštovati da bi se uspostavila pravilna komunikacija sa senzorom. Na slici 3.2[4] mogu se vidjeti dijagrami upisa i čitanja iz senzora koji predstavljaju *I2C Master Write* i *I2C Master Read* protokol. I2C protokol upisa započinje start sekvencom nakon čega se šalje slave adresa senzora sa bitom W (0) koji označava da je u pitanju sekvenca upisa. Nakon toga, neophodno je poslati adresu registra u koji se želi upisati podatak pa tek nakon toga podatak ili seriju podataka ukoliko je u pitanju proces slanja više podataka. I2C protokol čitanja nešto je drugačiji od procesa upisa s tim što se proces započinje slanjem slave adrese senzora sa bitom W (0) koji predstavlja pokaznicu da je u pitanju proces upisa, kao i u prethodnom slučaju, sa adresom registra nakon toga ali potom se mora poslati ponovo start sekvenca (restart) nakon čega se šalje ponovna slave adresa senzora sa R bitom (1) koji označava da je u pitanju proces čitanja. Nakon toga se iz senzora prima podatak ili serija podataka ukoliko je u pitanju proces čitanja više podataka.

Single-Byte Write Sequence

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

Slika 3.2. I2C protokoli čitanja i upisa podataka

U naredna dva potpoglavlja biće prikazan način na koji je mikrokontroler programiran tako da protokoli budu ispoštovani i u slučaju sekvence upisa i sekvence čitanja.

3.1. Implementacija I2C Sekvence Upisa

Upis podataka putem I2C protokola implementiran je u vidu jedne *I2C_Write* funkcije u kojoj će se manipulisati sa I2C1 registrima a koja za parametre uzima slave adresu senzora, adresu registra i pokazivač na strukturu koja sadrži podatke koji će se upisivati u senzor. Sekvenca započinje provjeravanjem bita BUSY koji se nalazi unutar registra I2C1_ISR. Provjera je neophodna da bi se obezbijedilo da se prethodna I2C sekvenca izvršila do kraja. Prateći protokol opisan na slici 3.2[4] prvo je potrebno poslati start sekvencu a zatim slave adresu senzora. Konfiguracija registra I2C1_CR2 omogućava nam upravo to. U tom registru moguće je konfigurisati režim u kojem se zadaje start sekvenca, upisuje slave adresa senzora i zadaje broj bita za transfer. Adresni mod definisan bitima SADD postavljen je kao 7-bitni adresni mod[7] dok

[7] Ovakav mod čini da je slave adresa senzora u slučaju akcelerometar-žiroskop module 0x69 pomjerena za jedno mjesto u lijevo dok je slave adresa magnetometra 0x0C pomjerena za jedno mjesto u lijevo

je broj podataka za transfer određen bitima NBYTES postavljen na jedan 8-bitni podatak za transfer. RELOAD bit označava da će NBYTES biti ponovo upisano u registru ukoliko je setovan na logičku jedinicu dok u suprotnom znači da će u transferu slijediti STOP ili RESTART sekvenca. Na slici 3.1.0[2] prikazan je izgled I2C1_CR2 registra na kojima se vide odgovarajući biti i njihovi nazivi. Ovim se sekvenca upisa startuje i programski kod dat ispod prikazuje opisanu konfiguraciju.

```
/* Wait in case of busy */
while(I2C1->ISR & I2C_ISR_BUSY);

/* Sending Device Address */
I2C1->CR2 |= (Slave_Address & I2C_CR2_SADD);
I2C1->CR2 |= ((0x01 << 16U) & I2C_CR2_NBYTES);
I2C1->CR2 |= I2C_CR2_RELOAD;
I2C1->CR2 |= I2C_CR2_START;
```

26.7.2 Control register 2 (I2C_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	PEC BYTE	AUTO END	RE LOAD	NBYTES[7:0]							
					rs	rw	rw	rw							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NACK	STOP	START	HEAD 10R	ADD10	RD_ WRN	SADD[9:0]									
rs	rs	rs	rw	rw	rw	rw									

Slika 3.1.0. I2C1_CR2 registar

Nakon ove konfiguracije, start sekvenca i adresa senzora su poslati ka senzoru i sekvenca upisa je započeta. Neophodno je sačekati dok se adresa uređaja kompletno ne pošalje, tj. dok se transfer ne završi pa treba ispitivati bit TXIS koji se nalazi unutar registra I2C1_ISR sve dok on ne postane jednak logičkoj jedinici. U tom trenutku znamo da je sve iz registra I2C1_TXDR poslato i da se u njega može upisati naredni podatak. U njega potom upisujemo adresu registra u koji želimo da upišemo podatak a koji nam je proslijeđen kao parametar funkcije. Nakon upisa potrebno je ispitivati bit TCR koji se nalazi unutar I2C1_ISR registra a koji nam govori da su NBYTES poslati tj. da je slanje adrese registra završeno. Sledeće po protokolu jeste slanje podatka. Neophodno je ukloniti RELOAD bit i upisati logičku jedinicu u AUTOEND bit registra I2C1_CR2 da bi se nakon transfera poslao STOP bit tj. stop sekvenca koja označava kraj transfera. Nakon završetka transfera (što se provjera TXIS bitom) vrši se poslednja provjera STOPF bita unutar I2C1_ISR registra koja nam daje do znanja da je transfer konačno završen. Programski kod gore navedenog dijela konfiguracije dat je ispod.

```
/* Wait until TXIS is Set */
while(!(I2C1->ISR & I2C_ISR_TXIS));

/* Send Memory Address */
I2C1->TXDR = Register_Address;

/* Wait until TCR is Set */
while(!(I2C1->ISR & I2C_ISR_TCR));

I2C1->CR2 &= ~(I2C_CR2_AUTOEND | I2C_CR2_RELOAD);
I2C1->CR2 |= I2C_CR2_AUTOEND;

/* Wait until TXIS is Set */
while(!(I2C1->ISR & I2C_ISR_TXIS));
I2C1->TXDR = *pdata;

/* Wait until STOPF is Set and clear it */
while(!(I2C1->ISR & I2C_ISR_STOPF));
I2C1->ICR |= I2C_ICR_STOPCF;
```

3.2. Implementacija I2C Sekvence Čitanja

Sekvenca čitanja podataka započinje na isti način kao i sekvenca upisa podataka pa se nećemo bazirati na taj dio jer je već opisan. Ove dvije sekvence počinju da se razlikuju u trenutku kada se pošalje adresa registra iz kojeg želi da se čita podatak. Nakon ispitivanja TC bita (*Transfer Complete*) koji označava da je transfer adrese registra završen a koji se nalazi u I2C1_ISR registru ponovo se konfiguriše I2C1_CR2 registar tako da se ponovo aktivira START bit, aktivira AUTOEND bit i RD_WRN bit koji po protokolu označava da se radi o sekvenci čitanja. Nakon toga se prelazi u ispitivanje RXNE bita koji se također nalazi u I2C1_ISR registru a koji označava da je prijem podataka završen pa se podatak koji se sada nalazi u I2C1_RXDR registru može prebaciti u našu strukturu. Time se završava sekvenca čitanja. Programski kod koji rekonfiguriše I2C1_CR2 registar i vrši prijem podataka je dat ispod.

```
/* Sending Device Address and Restart */
I2C1->CR2 |= I2C_CR2_AUTOEND;
I2C1->CR2 |= I2C_CR2_RD_WRN;
I2C1->CR2 |= I2C_CR2_START;

/* Waiting RXNE to be Set */
while(!(I2C1->ISR & I2C_ISR_RXNE));

/* Reading data from the receive register */
*pdata = I2C1->RXDR;

/* Waiting for STOPF to detect STOP bit */
while(!(I2C1->ISR & I2C_ISR_STOPF));

/* Clear STOP bit */
I2C1->ICR |= I2C_ICR_STOPCF;
```

4. OBRADA PODATAKA SA SENZORA

U ovom poglavlju bavit ćemo se analizom i fuzijom podataka iz senzora akcelerometra, žiroskopa i magnetometra. U prvom potpoglavlju biće riječi o neophodnim procedurama koje se moraju vršiti nad senzorom prije procesa glavnog prikupljanja podataka ili procedurama koje se vrše u toku prikupljanja podataka ali nije zgodno pominjati ih u tim dijelovima da se izbjegne konfuzija i prenatrpanost poglavlja. Naredna potpoglavlja bavit će se svakim senzorom pojedinačno, prikazati njegove prednosti i mane, dok će fuzija podataka biti obrađena u potpoglavlju koji uvodi pojam Kalmanovog filtra tako što će pokazati kako dvije vrste srodnih veličina koje pate od neusaglašenih mana uspijevaju biti očišćene Kalmanovim filtriranjem.

4.1. Prerada senzorskih podataka i rad sa senzorom

Ovo potpoglavlje ima za cilj da objasni tri stvari koje su bitne prije samog procesa prikupljanja podataka. Te tri stvari čine:

- Startovanje i promjena aktivnog senzora
- Regulacija podataka sa senzora magnetometra
- Kalibracija magnetometra

4.1.1. Startovanje i promjena aktivnog senzora

Startovanje senzora podrazumijeva da se po uključivanju na napajanje u registar kontrole napajanja upiše heksadecimalna vrijednost 0x00 da bi senzor izašao iz režima spavanja i ušao u aktivan režim čime bi podaci sa akcelerometra i ostalih senzora postali validni i spremni za dohvatanje. Upisivanje vrijednosti 0x00 u PWR_MGMT_1 registar obavlja se sekvencom upisom podatka prateći I2C protokol kao što je to opisano u prethodnim poglavljima. U mapi registara MPU-9250 senzora može se vidjeti izgled PWR_MGMT_1 registra i funkcija svih bita unutar njega. Na slici 4.1.1.0[4] prikazan je opis RESET i SLEEP bita ovog registra. Upisivanjem logičke jedinice na mjesto bita SLEEP senzor prelazi u režim spavanja. Programski kod koji vrši izlazak iz režima spavanja i ulaz u aktivan režim prikazan je ispod. On se izvršava prije glavne petlje u kojoj se prikupljaju podaci i bez njega podaci koji se očitavaju nisu validni. Unutar koda nalazi se

i provjera očitane vrijednosti WHO_AM_I registra koji sadrži fiksnu vrijednost da bi se potvrdila korektna veza sa senzorom od strane mikrokontrolera.

```
/**
 * @brief WakeUpSensor
 *
 * Function checks the connection to the MPU-9250 sensor by reading the WHO_AM_I register and then writes to
 * power management register to turn off sleep mode.
 *
 */
void WakeUpSensor(void) {
   /* Reading the WHO_AM_I register of the Accel-Gyro Slave */
   I2C_Read(AccelGyroAddress, WHO_AM_I, data);
   if(data[0] != 0x68) {
      strcpy((char*)buf, "Invalid WHO_AM_I Register value!\r\n");
      SendToUART(buf);
   }

   data[0] = 0x00;
   /* Writing to the PWR_MGMT_1 register to turn off the sleep mode */
   I2C_Write(AccelGyroAddress, PWR_MGMT_1, data);
}
```

Name: PWR_MGMT_1

Serial IF: R/W

Reset value: (Depends on PU_SLEEP_MODE bit, see below)

BIT	NAME	FUNCTION
[7]	H_RESET	1 – Reset the internal registers and restores the default settings. Write a 1 to set the reset, the bit will auto clear.
[6]	SLEEP	When set, the chip is set to sleep mode (After OTP loads, the PU_SLEEP_MODE bit will be written here)

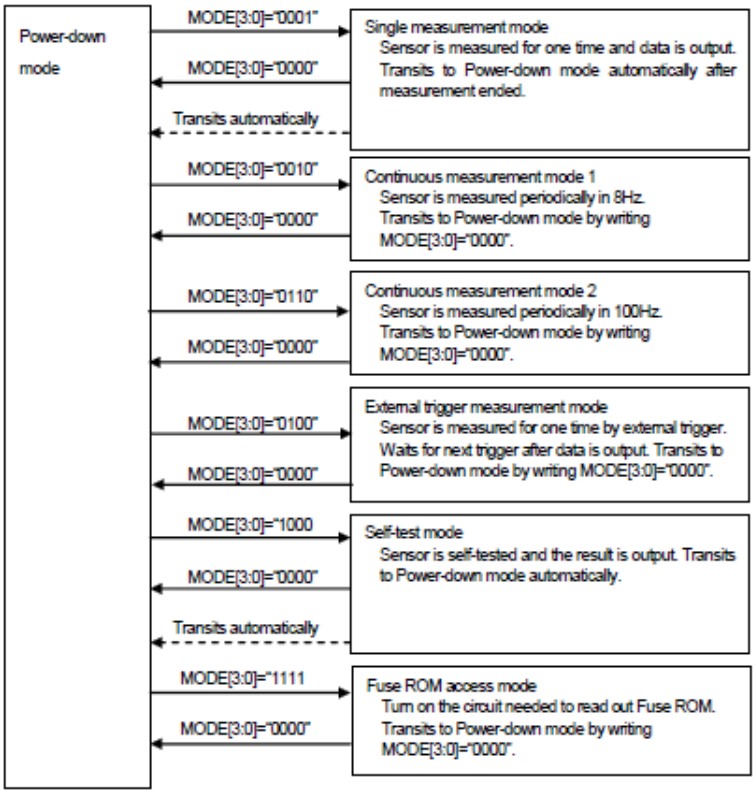
Slika 4.1.1.0. PWR_MGMT_1 registar

Pod promjenom aktivnog senzora podrazumijeva se modifikacija vrijednosti upisane u registru INT_BYPASS_CONFIG_AD koji se nalazi u senzoru. Naime, u MPU-9250 senzoru nije moguće pristupiti modulu akcelerometra, žiroskopa i magnetometra u takoreći isto vrijeme tj. bez modifikacije aktivnog modula u jednom trenutku. Modul akcelerometra i žiroskopa nalaze se zajedno i njihovi registri se mogu pristupati bez promjene režima rada. Međutim, pristup modulu magnetometra je moguć jedino ako se taj modul postavi za aktivan upisivanjem vrijednosti 0x02 u INT_BYPASS_CONFIG_AD registar. Upisivanjem vrijednosti 0x00 modul akcelerometra i žiroskopa postaje trenutni aktivni modul. Time je naređeno da se prije čitanja podataka iz ovih senzora mora naglasiti koji senzor je trenutni aktivan za čitanje. Ova promjena modula implementirana je preko funkcije koja za parametar uzima adresu koja se upisuje u INT_BYPASS_CONFIG_AD registar i vrši upis te vrijednosti. Programski kod koji obavlja ovu funkciju dat je ispod.

```
/**
 * @brief SwitchSlaveDevice
 *
 * Function switches between the AccelGyro sensor and Magnetometer sensor based on the command it receives.
 *
 */
void SwitchSlaveDevice(uint8_t command) {
   data[0] = command;
   /* Writing to the INT_BYPASS_CONFIG_AD register */
   I2C_Write(AccelGyroAddress, INT_BYPASS_CONFIG_AD, data);
}
```

4.1.2. Regulacija podataka sa senzora magnetometra

Regulacija podataka sa senzora magnetometra predstavlja proces u kom se sa aktivnim modulom magnetometra mijenja režim rada magnetometra pri kom je moguće pristupiti FUSE-ROM registrima u kojima se nalaze vrijednosti kojima se regulišu podaci koji su sa magnetometra očitani. Takvih registara ima tri – jedan za svaku osu koja se detektuje.Promjena režima rada vrši se upisivanjem određene vrijednosti u CNTL1_AD registar magnetometra. Na slici 4.1.2.0[7] prikazani su neki režimi rada i način promjene iz jednog režima u drugi.



Slika 4.1.2.0. Režimi rada i promjena režima rada

Sa slike je očigledno da se pri svakoj promjeni režima rada mora ući u međurežim koji je označen sa POWER-DOWN mod. Takođe, u datasheet-u senzora govori se da se pri svakoj promjeni režima rada mora sačekati minimalno 100ms.[8]

$$M_{adj} = M\left(\frac{(ASA-128)}{256} + 1\right) \qquad (4.1.2.0)$$

Regulacija vrijednosti očitanih sa magnetometra vrši se pomoću formule 4.1.2.0 u kojoj vrijednost ASA označava regulacionu vrijednost za određenu osu pročitanu iz FUSE-ROM registra. Funkcija koja vrši očitavanje regulacionih vrijednosti koristi periferiju TIM3 da odbrojava 500ms do promjene sledećeg režima rada. Programski kod koji vrši to očitavanje dat je ispod.

```
data[0] = 0x00;
/* Writing to the CNTL1_AD register (POWER-DOWN MODE)*/
I2C_Write(MagnetometerAddress, CNTL1_AD, data);

/* Starting the timer 3 peripheral and counting for 500 ms to change the mode of operation */
TIM3->CNT = 0x00;
TIM3->CR1 |= TIM_CR1_CEN;            /**< Enabling the timer 3 peripheral */
while(1){
   if(TIM3->CNT > 1000)
      break;
}
TIM3->CR1 &= ~TIM_CR1_CEN;        /**< Disabling the timer 3 peripheral */
TIM3->CNT = 0x00;

data[0] = 0x0F;
/* Writing to the CNTL1_AD register (FUSE-ROM-MODE)*/
I2C_Write(MagnetometerAddress, CNTL1_AD, data);

TIM3->CNT = 0x00;
TIM3->CR1 |= TIM_CR1_CEN;
while(1){
   if(TIM3->CNT > 1000)
      break;
}
TIM3->CR1 &= ~TIM_CR1_CEN;
TIM3->CNT = 0x00;

/* Reading the adjustment values of the magnetometer */
for(uint8_t i = 0; i < 3; i++)
{
   I2C_Read(MagnetometerAddress, (ASAX_REG + i), (data + i));
}

ASAX = data[0];
ASAY = data[1];
ASAZ = data[2];
```

Nakon što se regulacione vrijednosti dohvate iz FUSE-ROM registara, ulazi se u *Continuous-Mode 2* režim koji podrazumijeva da se podaci iz magnetometra dobijaju brzinom od

[8] Da bismo bili sigurni da je sve ispoštovano do kraja, u ovom projektu čekaćemo 500ms

12 Hz, što je daleko od idealnog ali je jedini režim koji je omogućen.[9] On se aktivira upisivanjem vrijednosti 0x02 u CNTL1_AD registar što ujedno znači da će podaci biti 14-bitni.[10] Na slici 4.1.2.1[5] može se vidjeti registar CNTL1_AD koji je bitan za promjenu režima rada magnetometra. Bit 5 ovog registra diktira da li su podaci 14-bitni ili 16-bitni.

5.8 CNTL1: Control 1

Addr	Register name	D7	D6	D5	D4	D3	D2	D1	D0
Read-only register									
0AH	CNTL1	0	0	0	BIT	MODE3	MODE2	MODE1	MODE0
Reset		0	0	0	0	0	0	0	0

Slika 4.1.2.1. CNTL1 registar

4.1.3. Kalibracija magnetometra

Kalibracija magnetometra predstavlja tehniku kojom se uravnotežava magnetno polje detektovano od strane magnetometra. Implementirana je u okviru funkcije koja se pokreće samo jednom da bi se otkrile težine koje su neophodne da se sve tri magnetne ose uravnoteže nakon svakog očitavanja. Naime, funkcija vrši uzastopna očitavanja magnetometra dok korisnik vrti senzor u obliku broja 8, time mijenjajući položaj svih osa. Time se određuje maksimum i minimum očitavanja za svaku osu i nalazi takozvana sredina kruga koja bi u idealnom slučaju trebala da bude u centru tj. da se maksimalna i minimalna vrijednost razlikuju samo u znaku. Ukoliko to nije slučaj, nalazi se težina koja kompenzuje taj pomjeraj kruga pa ga vraća na koordinatni početak. Ovo je termin poznat kao *hard-iron compensation*. Dio programskog koda funkcije koja vrši kalibraciju magnetometra dat je ispod.

```
for(i = 0; i < calibration_count; i++) {
ReadMagnetometer();

   X_OUTPUT = ((float)(X_AXIS))/4.0;
   Y_OUTPUT = ((float)(Y_AXIS))/4.0;
   Z_OUTPUT = ((float)(Z_AXIS))/4.0;

   /* Finding the maximum and minimum of each axis */
if(X_OUTPUT > MAGX_MAX)
      MAGX_MAX = X_OUTPUT;
   if(X_OUTPUT < MAGX_MIN)
      MAGX_MIN = X_OUTPUT;
   if(Y_OUTPUT > MAGY_MAX)
      MAGY_MAX = Y_OUTPUT;
   if(Y_OUTPUT < MAGY_MIN)
      MAGY_MIN = Y_OUTPUT;
```

[9] MPU-9250 ne podržava rad magnetometra na brzini od 100Hz prema datasheet-u
[10] MPU-9250 takođe ne podržava 16-bitni režim podataka iz magnetometra već samo 14-bitni režim

```
    if(Z_OUTPUT > MAGZ_MAX)
        MAGZ_MAX = Z_OUTPUT;
    if(Z_OUTPUT < MAGZ_MIN)
        MAGZ_MIN = Z_OUTPUT;
}

/* Store the magnetic offsets from the hard iron distortion in order to subtract from the actual readings */
MagneticOffset[0] = ((float)MAGX_MAX + (float)MAGX_MIN)/2;
MagneticOffset[1] = ((float)MAGY_MAX + (float)MAGY_MIN)/2;
MagneticOffset[2] = ((float)MAGZ_MAX + (float)MAGZ_MIN)/2;
```

4.2. Akcelerometar

Ovo potpoglavlje, kao i naredna potpoglavlja bavit će se prikupljanjem i obradom podataka iz senzora, i njihovim pretvaranjem iz jedne vrste veličina u drugu. Modul akcelerometra sadrži brojne karakteristike koje je moguće postaviti upisom u određene registre. Neke karakteristike nećemo dirati, jer su dovoljne takve kakve jesu dok ćemo neke po potrebi mijenjati.

Modul akcelerometra sadrži MEMS akelerometar sa tri ose sa programibilnom skalom detekcije. Kao što se može vidjeti na slici 4.2.0[4], na kojoj je prikazana tabela specifikacija akcelerometra, u dijelu *Full-Scale Range* selekcijom određene vrijednosti *AFS_SEL* može se odabrati maksimalna skala detekcije. U našem slučaju, s obzirom da se radi o detekciji pokreta, default-na vrijednost od 2G je sasvim dovoljna. U tom slučaju osjetljivost detekcije je 16384 LSB/G. S obzirom da se za svaku osu akcelerometra u senzoru nalaze dva 8-bitna registra, što čini ukupni 16-bitni podatak, dobijeni podatak se dijeli sa osjetljivošću da bi se dobila pravilna vrijednost koja se može kasnije koristiti. Svi podaci akcelerometra dati su u komplementu dvojke. Za modul akcelerometra, nije potrebno ništa mijenjati niti podešavati.

3.2 Accelerometer Specifications

Typical Operating Circuit of section 4.2, VDD = 2.5V, VDDIO = 2.5V, TA=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
Full-Scale Range	AFS_SEL=0		±2		g
	AFS_SEL=1		±4		g
	AFS_SEL=2		±8		g
	AFS_SEL=3		±16		g
ADC Word Length	Output in two's complement format		16		bits
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/g
	AFS_SEL=1		8,192		LSB/g
	AFS_SEL=2		4,096		LSB/g
	AFS_SEL=3		2,048		LSB/g
Initial Tolerance	Component-Level		±3		%
Sensitivity Change vs. Temperature	-40°C to +85°C AFS_SEL=0 Component-level		±0.026		%/°C
Nonlinearity	Best Fit Straight Line		±0.5		%
Cross-Axis Sensitivity			±2		%
Zero-G Initial Calibration Tolerance	Component-level, X,Y		±60		mg
	Component-level, Z		±80		mg
Zero-G Level Change vs. Temperature	-40°C to +85°C		±1.5		mg/°C
Noise Power Spectral Density	Low noise mode		300		µg/√Hz
Total RMS Noise	DLPFCFG=2 (94Hz)			8	mg-rms
Low Pass Filter Response	Programmable Range	5		260	Hz
Intelligence Function Increment			4		mg/LSB
Accelerometer Startup Time	From Sleep mode		20		ms
	From Cold Start, 1ms V_{DD} ramp		30		ms
Output Data Rate	Low power (duty-cycled)	0.24		500	Hz
	Duty-cycled, over temp		±15		%
	Low noise (active)	4		4000	Hz

Slika 4.2.0. Specifikacije akcelerometra

Prikupljanje podataka sa senzora akcelerometra vrši se na sledeći način. Na slici 4.2.1[5] prikazan je dio registarske mape MPU-9250 senzora na kojoj se vide registri koji sadrže podatke sa osa. Vrši se očitavanje ovih registara I2C sekvencom čitanja gdje su registarske adrese sekvencijalno poređane od 0x3B do 0x40. Pročitani podaci se čuvaju u strukturi *data*. Te podatke koji su 8-bitni potrebno je naknadno spojiti u jedan 16-bitni podatak koji predstavlja kompletnu informaciju o jednoj osi akcelerometra. Nakon toga, neophodno je podatke podijeliti sa odgovarajućom osjetljivošću da bi se dobio pravilan podatak u skali koju smo odabrali. Sve što je ostalo nakon toga jeste da podatke o linearnom ubrzanju sve tri ose pretvorimo u ugaonu poziciju matematičkim transformacijama koje su date formulama 4.2.0 i 4.2.1. Na slici 4.2.2[4] prikazana je orijentacija osa senzora akcelerometra i žiroskopa dok je na slici 4.2.3[11] prikazana fizička predstava rotacionih osa *roll, pitch* i *yaw*. Programski kod koji obavlja funkciju prikupljanja i transformacije podataka iz akcelerometra priložen je ispod.

```
ReadAccelerometer();

/* Storing the data inside the variables */
X_AXIS = (data[0] << 8) | data[1];
Y_AXIS = (data[2] << 8) | data[3];
Z_AXIS = (data[4] << 8) | data[5];

/* Adjusting for the noise and transforming the data to represent true values */
X_OUTPUT = X_AXIS/16384.0;
Y_OUTPUT = Y_AXIS/16384.0;
Z_OUTPUT = Z_AXIS/16384.0;

/* Calculating the roll and pitch values from the accelerometer data */
float AccelAngleX = atan(Y_OUTPUT / sqrt(pow(X_OUTPUT, 2) + pow(Z_OUTPUT, 2)))*180/PI; /**< Roll */
float AccelAngleY = atan(-1*X_OUTPUT / sqrt(pow(Y_OUTPUT, 2) + pow(Z_OUTPUT, 2)))*180/PI; /**< Pitch
*/
```

$$ROLL = \operatorname{atan}\left(\frac{Y}{\sqrt{X^2+Z^2}}\right) \tag{4.2.0}$$

$$PITCH = \operatorname{atan}\left(\frac{-1*X}{\sqrt{Y^2+Z^2}}\right) \tag{4.2.1}$$

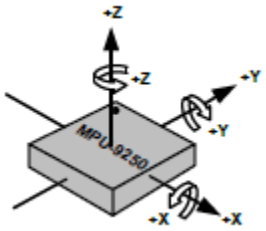
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]

Slika 4.2.1. Dio registarske mape MPU-9250

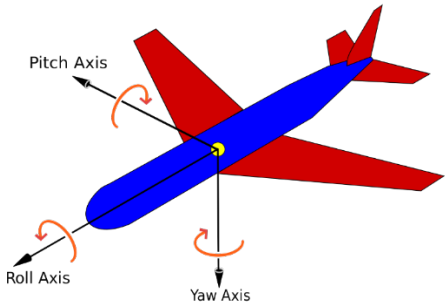
Na slikama 4.2.4 i 4.2.5 prikazan je rezultat određivanja *roll* i *pitch* vrijednosti. *Yaw* vrijednost nije moguće odrediti preko akcelerometra zbog gravitacionog djelovanja na senzor. Iz dijagrama se da vidjeti velika mana akcelerometarskog mjerenja. Izlaz je jako šumovit. Šum je moguće ukloniti određenim NF filtriranjima ali onda odziv izlaza postaje spor što nam nije

[11] Slika preuzeta sa wikipedije

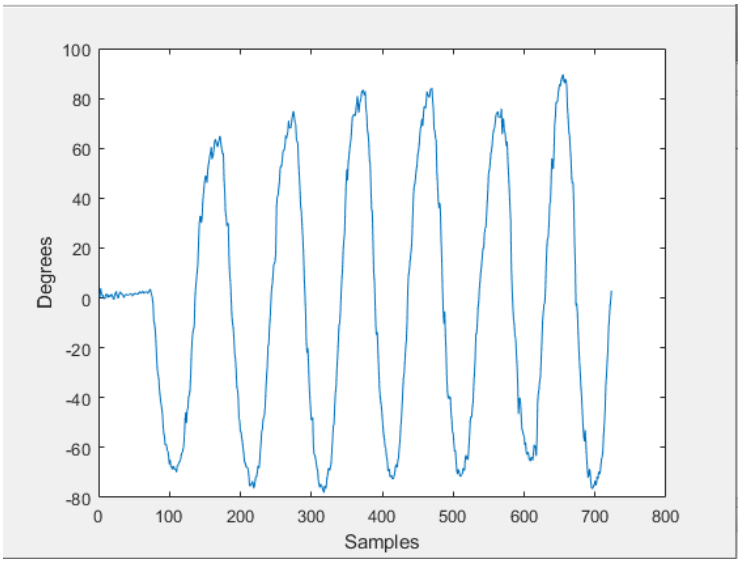
pogodno. Ove vrijednosti kasnije ćemo kombinovati sa žiroskopskim rezultatima da dobijemo izlaz koji nije šumovit.



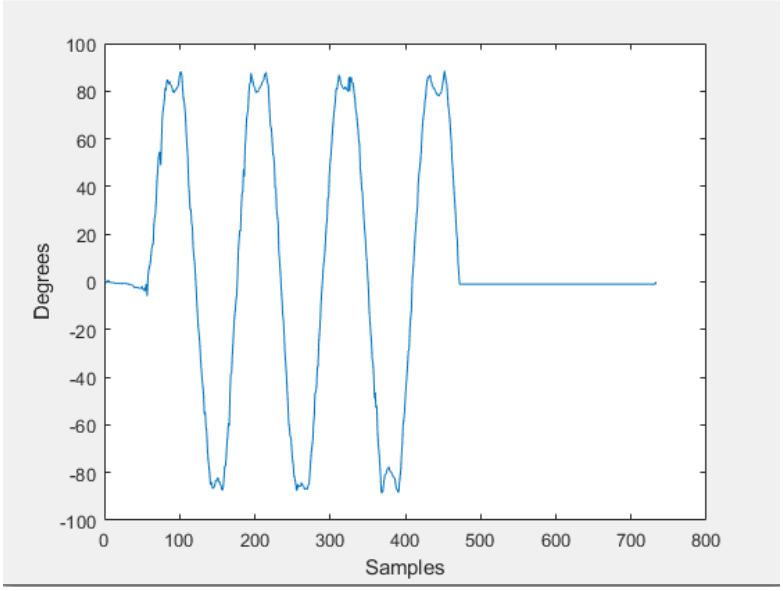
Slika 4.2.2. Orijentacija osa akcelermetra i žiroskopa



Slika 4.2.3. Roll, pitch, yaw ose



Slika 4.2.4. Roll određen akcelerometrom



Slika 4.2.5. Pitch određen akcelerometrom

4.3. Žiroskop

Očitavanje vrijednosti žiroskopa obavlja se na sličan način kao i očitavanje vrijednosti akcelerometra. I2C sekvenca čitanja nad registrima u kojima se nalaze žiroskopski podaci u kombinaciji sa preuređivanjem vrijednosti i skaliranjem daju nam ugaonu brzinu iz sve tri ose žiroskopa. Korištenjem periferije TIM3 koja je se pokreće da odbrojava milisekunde nakon svakog očitavanja podataka iz žiroskopa daje nam proteklo vrijeme koje nam je neophodno za integraciju. Integracija nam potom daje od ugaone brzine osa ugaonu poziciju čija vrijednost odgovara vrijednosti naših rotacionih osa. Senzor žiroskopa nam može dati sve tri rotacione ose za razliku od senzora akcelerometra ali svaka od njih ima problem gubitka tačnosti nakon određenog vremena.

Na slici 4.3.0[4] prikazane su karakteristike senzora žiroskopa koje se mogu naći u datasheet-u MPU-9250 ploče[4]. Kao i u slučaju akcelerometra, nisu potrebna dodatna podešavanja jer je naš projekat detekcija pokreta te su default-ne opcije sasvim dovoljne. Ono što je za nas bitno jeste dio *Sensitivity Scale Factor* koji nam govori na koji način vrijednosti koje dobijamo iz registara da interpretiramo. Vrijednost 131 nam govori da sve vrijednosti moramo podijeliti sa 131 da bi podaci imali smisla. Registri u kojima se nalaze podaci sa žiroskopa počinju na adresi 0x67 i završavaju na adresi 0x72.

3.1 Gyroscope Specifications

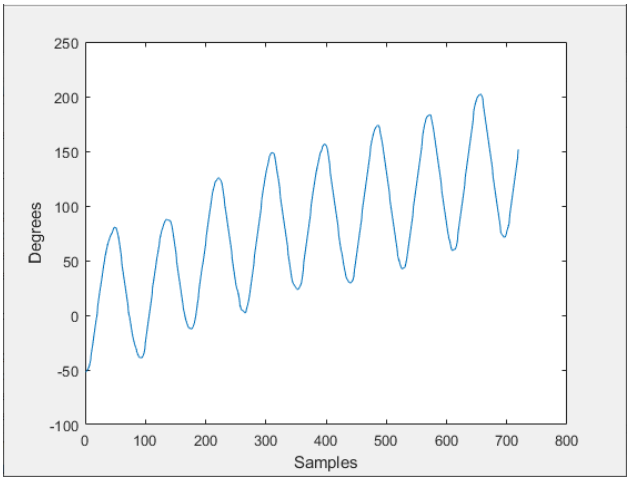
Typical Operating Circuit of section 4.2, VDD = 2.5V, VDDIO = 2.5V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
Full-Scale Range	FS_SEL=0		±250		°/s
	FS_SEL=1		±500		°/s
	FS_SEL=2		±1000		°/s
	FS_SEL=3		±2000		°/s
Gyroscope ADC Word Length			16		bits
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)
	FS_SEL=1		65.5		LSB/(°/s)
	FS_SEL=2		32.8		LSB/(°/s)
	FS_SEL=3		16.4		LSB/(°/s)
Sensitivity Scale Factor Tolerance	25°C		±3		%
Sensitivity Scale Factor Variation Over Temperature	-40°C to +85°C		±4		%
Nonlinearity	Best fit straight line; 25°C		±0.1		%
Cross-Axis Sensitivity			±2		%
Initial ZRO Tolerance	25°C		±5		°/s
ZRO Variation Over Temperature	-40°C to +85°C		±30		°/s
Total RMS Noise	DLPFCFG=2 (92 Hz)		0.1		°/s-rms
Rate Noise Spectral Density			0.01		°/s/√Hz
Gyroscope Mechanical Frequencies		25	27	29	KHz
Low Pass Filter Response	Programmable Range	5		250	Hz
Gyroscope Startup Time	From Sleep mode		35		ms
Output Data Rate	Programmable, Normal mode	4		8000	Hz

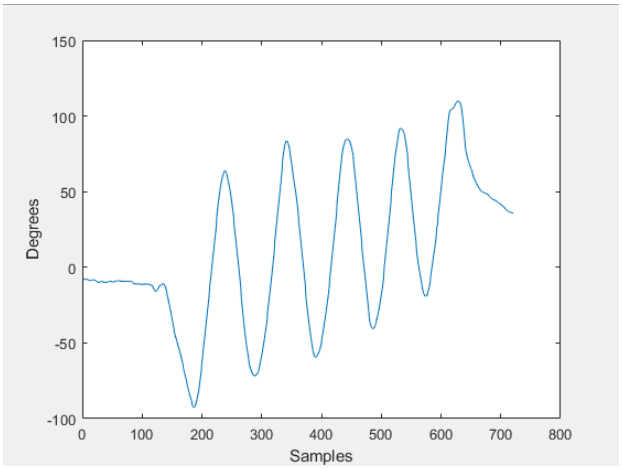
Slika 4.3.0. Specifikacije žiroskopa

Programski kod koji vrši očitavanje vrijednosti i integraciju podataka dat je ispod paragrafa. Na slikama 4.3.1, 4.3.2 i 4.3.3 prikazan je rezultat integracije sve tri ose sa kojih se vidi da se integracijom gubi tačnost iako su sami rezultati bez šuma. Integracijom se male greške proračuna akumuliraju pa se nakon određenog vremena mjerenja javljaju velika odstupanja.

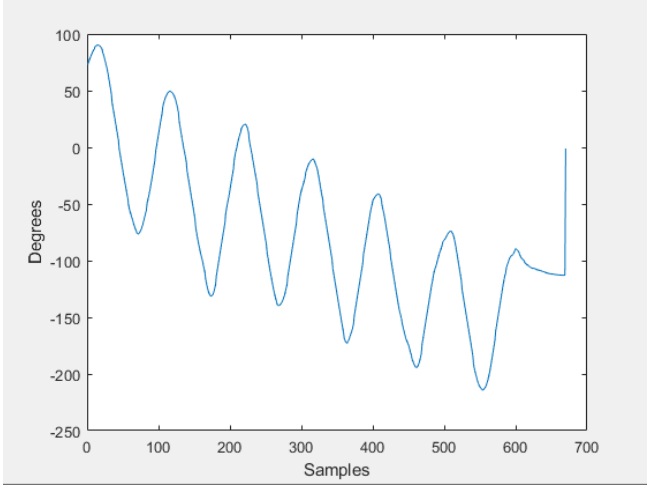
```
/* Getting the time it took for the data aquisition from the counter in order to integrate the gyroscope data */
float current_time = (float)(TIM3->CNT)*TurnToMilisecond;
float dt = (current_time) / 1000.0;
TIM3->CNT = 0;                                  /**< Reseting the counter value */

ReadGyroscope();

X_AXIS = (data[0] << 8) | data[1];
Y_AXIS = (data[2] << 8) | data[3];
Z_AXIS = (data[4] << 8) | data[5];

/* Adjusting for the noise and transforming the data to represent true values */
float GyroRateX = X_AXIS/131.0 - GyroXError;
float GyroRateY = Y_AXIS/131.0 - GyroYError;
float GyroRateZ = Z_AXIS/131.0 - GyroZError;

/* Integrate the data from the gyroscope in order to turn the angular velocity to angular position */
GyroAngleX += GyroRateX*dt; /**< Roll */
GyroAngleY += GyroRateY*dt; /**< Pitch */
GyroAngleZ += GyroRateZ*dt; /**< Yaw */
```



Slika 4.3.1. Roll određen žiroskopom



Slika 4.3.2. Pitch određen žiroskopom



Slika 4.3.3. Yaw određen žiroskopom

4.4. Magnetometar

Na slici 4.4.0[4] prikazane su karakteristike magnetometra koje se mogu pronaći u datasheet-u uređaja[4]. Bitna informacija za nas iz ovih karakteristika jeste ta da su podaci koji se čitaju iz registara 14-bitni tj. da su dva bita iz 16-bitnog registra jedne ose nepotrebna. Takođe, bitno je i uočiti *Sensitivity Scale Factor* koji je 0.6 a koji predstavlja vrijednost kojom podatke iz magnetometra treba pomnožiti da bi se dobile smislene vrijednosti. Pored ovih korekcija očitanih vrijednosti potrebno je pomenuti i neke što su pomenute u ranijim poglavljima. Te korekcije sastoje se od *Axis Adjustment* vrijednosti[12], kao i magnetskog ofseta izračunatog kalibracijom magnetometra. Ofset se oduzima od izračunatih vrijednosti dok se podaci množe korekcijama koje su pomenute.

Očitavanje vrijednosti magnetometra izvodi se na nešto drugačiji način od očitavanja vrijednosti akcelerometra i žiroskopa. Na slici 4.4.1[5] prikazan je izgled serije registara magnetometra koje je neophodno pročitati da bi se sekvenca čitanja podataka smatrala uspješnom. Ti registri čine ST1 registar, registre podataka HXL-HZH, i registar ST2. Naime, u ST1 registru nalazi se bit DRDY koji označava da su podaci spremni. U našem slučaju, podaci dolaze brzinom od 12 Hz. Registri HXL-HZH se čitaju standardno dok se registar ST2 mora pročitati na kraju sekvence čitanja da bi se sekvenca uspješno završila. Takva procedura opisana je u datasheet-u senzora. Kod ovakvog načina očitavanja postoji problem. Naime, čekanje na DRDY bit usporava sistem toliko da sve tri rotacione ose postaju jako spore. Podaci iz akcelerometra i žiroskopa stižu mnogo brže nego što to rade podaci iz magnetometra pa čekanje na te podatke usporava cijeli sistem. Sekvenca čitanja je modifikovana tako da se ti podaci ne čekaju već se čita istom brzinom kao i čitanje ostalih podataka pa će se novonastali problemi naknadno riješiti. Programski kod koji vrši navedena očitavanja i korekcije dat je ispod. Na slici 4.4.2[4] prikazana je orijentacija osa magnetometra gdje se vidi da je orijentacija z-ose suprotna od orijentacije z-ose akcelerometra i magnetometra pa se to kompenzuje mijenjanjem znaka *MAGZ* vrijednosti u toku računanja.

```
ReadMagnetometer();

/* Reading the 14-bit data from the magnetometer slave */
X_AXIS = (data[1] << 8) | data[0];
Y_AXIS = (data[3] << 8) | data[2];
Z_AXIS = (data[5] << 8) | data[4];

X_AXIS = (X_AXIS << 2);
Y_AXIS = (Y_AXIS << 2);
Z_AXIS = (Z_AXIS << 2);

X_OUTPUT = ((float)(X_AXIS))/4.0;
Y_OUTPUT = ((float)(Y_AXIS))/4.0;
Z_OUTPUT = ((float)(Z_AXIS))/4.0;

/* Transform the data to represent the true readings by calibrating and subtracting offset */
MAGX = X_OUTPUT * MagneticResolution * calibrationX - MagneticOffset[0];
MAGY = Y_OUTPUT * MagneticResolution * calibrationY - MagneticOffset[1];
MAGZ = -1*(Z_OUTPUT * MagneticResolution * calibrationZ - MagneticOffset[2]);
```

[12] Ovo su regulacione vrijednosti koje smo dohvatili iz Fuse-Rom registara

3.3 Magnetometer Specifications

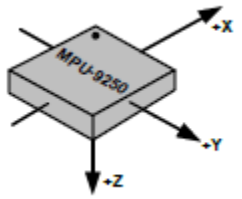
Typical Operating Circuit of section 4.2, VDD = 2.5V, VDDIO = 2.5V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
MAGNETOMETER SENSITIVITY					
Full-Scale Range			±4800		µT
ADC Word Length			14		bits
Sensitivity Scale Factor			0.6		µT / LSB
ZERO-FIELD OUTPUT					
Initial Calibration Tolerance			±500		LSB

Slika 4.4.0. Specifikacije magnetometra

Name	Address	READ/ WRITE	Description	Bit width	Explanation
WIA	00H	READ	Device ID	8	
INFO	01H	READ	Information	8	
ST1	02H	READ	Status 1	8	Data status
HXL	03H	READ	Measurement data	8	X-axis data
HXH	04H			8	
HYL	05H			8	Y-axis data
HYH	06H			8	
HZL	07H			8	Z-axis data
HZH	08H			8	
ST2	09H	READ	Status 2	8	Data status
CNTL	0AH	READ/ WRITE	Control	8	
RSV	0BH	READ/ WRITE	Reserved	8	DO NOT ACCESS
ASTC	0CH	READ/ WRITE	Self-test	8	
TS1	0DH	READ/ WRITE	Test 1	8	DO NOT ACCESS
TS2	0EH	READ/ WRITE	Test 2	8	DO NOT ACCESS
I2CDIS	0FH	READ/ WRITE	I^2C disable	8	
ASAX	10H	READ	X-axis sensitivity adjustment value	8	Fuse ROM
ASAY	11H	READ	Y-axis sensitivity adjustment value	8	Fuse ROM
ASAZ	12H	READ	Z-axis sensitivity adjustment value	8	Fuse ROM

Slika 4.4.1. Tabela registara magnetometra



Slika 4.4.2. Orijentacija osa magnetometra

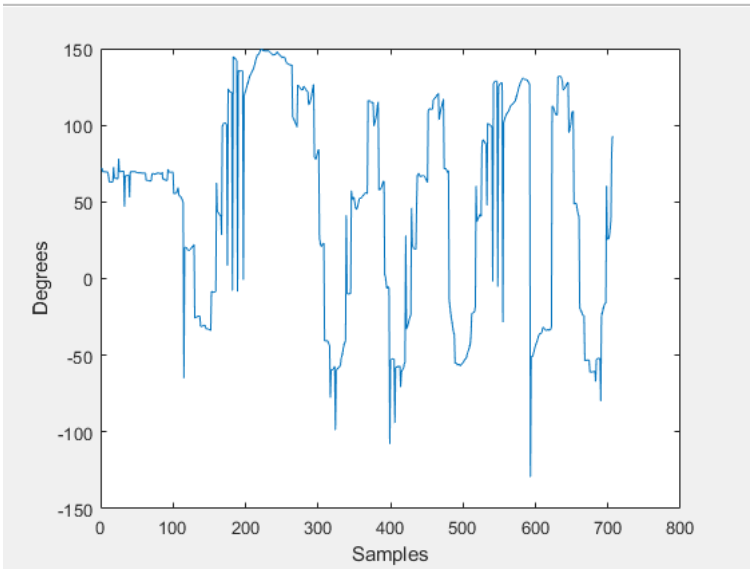
Važno je napomenuti da je ova vrijednost *yaw* ose validna samo ukoliko se senzor nalazi u ravni Zemlje tj. kada su vrijednosti ostale dvije rotacione ose jednake nuli. Da bi se kompenzovalo nakrivljenje senzora vrši se tehnika *kompenzacije nagiba* magnetometra gdje se matematičkim transformacijama sa uključenim vrijednostima ostalih osa nalazi pravilna vrijednost *yaw* ose. Te matematičke relacije date su formulama 4.4.0, 4.4.1 i 4.4.2. Na slici 4.4.3 prikazan je rezultat kompenzacije nakrivljenja senzora na *yaw* vrijednost ose. Programski kod koji vrši pomenutu kompenzaciju dat je ispod.

```
    /* Tilt compensation */
    /* Obtaining the yaw value from the magnetometer data based on the pitch and roll values and the read
magnetometer data */
    float Hy = MAGY*cos(pitch/(180/PI)) + MAGZ*sin(pitch/(180/PI));
    float Hx = MAGY*sin(roll/(180/PI))*sin(pitch/(180/PI))+MAGX*cos(roll/(180/PI)) -
MAGZ*sin(roll/(180/PI))*cos(pitch/(180/PI));

    /* Temporary yaw value which is really noise, obtained from the equation for the tilt compensation */
    float yaw_temp = atan2(Hy, Hx) * 180/PI;
```

$$H_y = Y * \cos(pitch) + Z * \sin(pitch) \tag{4.4.0}$$

$$H_x = Y * \sin(roll) * \sin(pitch) + X * \cos(roll) - Z * \sin(roll) * \cos(pitch) \tag{4.4.1}$$

$$YAW = \text{atan}(Hy, Hx) \tag{4.4.1}$$



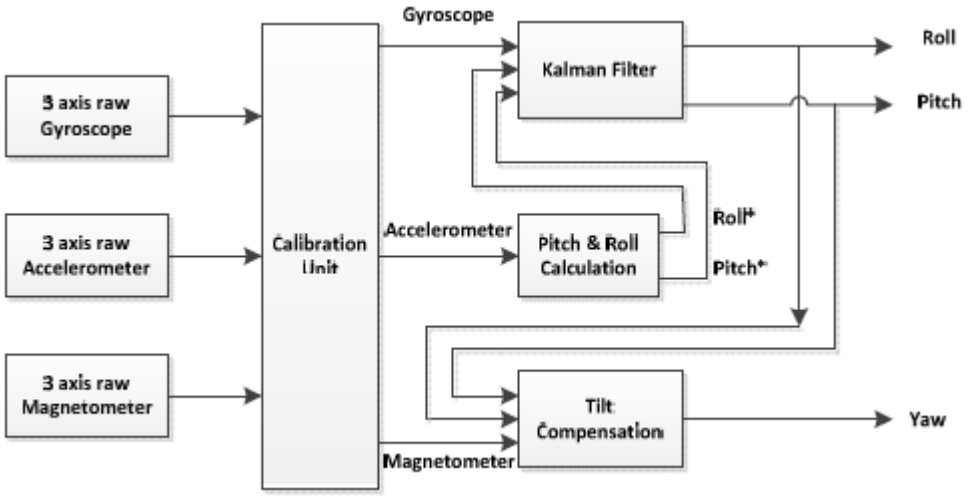
Slika 4.4.3. Yaw određen magnetometrom

4.5. Kalmanovo filtriranje

Sve što smo do sada uradili bilo je izvlačenje i pretvaranje senzorskih podataka u nešto što ima fizičkog smisla i može se fino vizuelizovati. Nažalost, podaci koje smo dobili pate od raznih problema i svaki od senzora ima svoj jedinstven problem. Akcelerometar ima problem šumovitog izlaza koji bi pri vizuelizaciji djelovao kao da osoba koja drži senzor neprestano pomjera taj senzor

u sve pravce, žiroskop ima problem gubljenja tačnosti usled integracije pa bi pri vizuelizaciji izgledalo kao da se nakon nekog vremena senzor sam pomjera dok magnetometar ima problem pre-diskretnih podataka što uvodi velike skokove između prikaza vrijednosti podataka. Sve ove probleme riješit ćemo uvođenjem Kalmanovih filtara.

Na slici 4.5.0[9] prikazana je struktura sistema koji je vrlo sličan našem sistemu. Razlikuje se u tome što će nama za određivanje *yaw* ose biti također potreban Kalmanov filtar koji ćemo kombinovati sa vrijednošću *yaw* ose koju dobijamo iz žiroskopa. Osim toga, za tu osu biće nam potreban i jak NF filtar koji će ublažiti velike skokove između dva sukcesivna podatka.



Slika 4.5.0. Struktura sistema

4.5.1. Kalmanov filtar

Kalmanovo filtriranje predstavlja rekurzivan algoritam koji je idealan za filtriranje šumovitih signala. Kalmanov filtar koristi fizičke atribute sistema koji se filtrira i na osnovu njih pravi predikcije sledećeg stanja sistema tako što estimira stanje sistema u trenutku *t* poznavajući stanje sistema u trenutku *t-1*. Kalmanov filtar ne zna tačnu poziciju ni brzinu sistema u trenutku *t* ali zna da su neke pozicije i brzine vjerovatnije od drugih jer smatra da je njihova distribucija gausova kao što je to prikazano na slici 4.5.1.0[13]. Sistem treba da bude opisan kroz prostore stanja kao što je to prikazano u jednakosti 4.5.1.0 gdje *p* predstavlja poziciju sistema koja je u našem slučaju ugaona pozicija tj. vrijednosti rotacionih osa koje tražimo dok *v* predstavlja brzinu sistema koja je u našem slučaju ugaona brzina koju dobijamo iz senzora žiroskopa za sve tri rotacione ose.

$$x_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix} \tag{4.5.1.0}$$

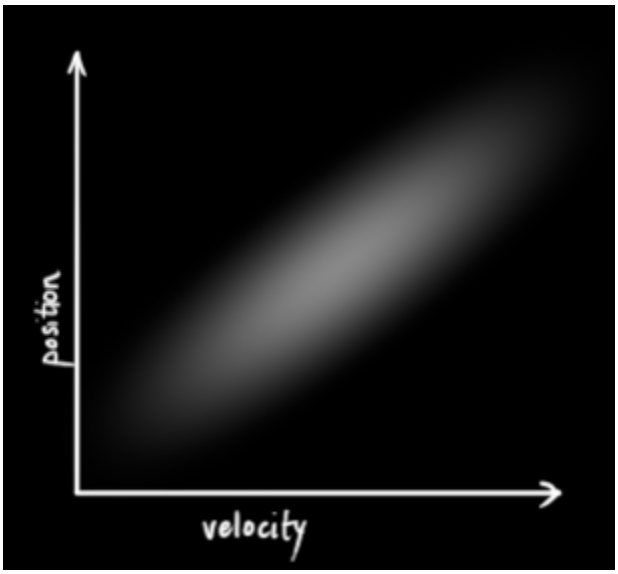
[13] Slika je preuzeta sa bzarg.com

Algoritam kalmanovog filtriranja započinje postavljanjem inicijalnih vrijednosti parametara filtra na nulu.

$$1) \qquad x_0 = 0; \quad P_0 = 0;$$

gdje je x matrica prostora stanja sistema a P predikcija kovarijanse greške (*Error covariance prediciton*). Inicijalizacija filtra programskim kodom riješena je inicijalizacijom strukture Kalmanovog filtra koja postoji za svaku osu, čineći ukupno tri strukture filtra. Inicijalizacija jedne od struktrura prikazana je ispod.

```
/* Kalman filter structure for the X-axis */
KalmanX.Q[0] = 0.001;
KalmanX.Q[1] = 0.003;
KalmanX.R = 0.003;
KalmanX.Xk[0] = 0.0;
KalmanX.Xk[1] = 0.0;
KalmanX.NoDriftGyroRate = 0.0;
KalmanX.P[0][0] = 0.0;
KalmanX.P[0][1] = 0.0;
KalmanX.P[1][0] = 0.0;
KalmanX.P[1][1] = 0.0;
```



Slika 4.5.1.0. Vjerovatnoća vrijednosti pozicije i brzine.

Algoritam se nastavlja sledećim korakom koji podrazumijeva predikciju stanja sistema. Vrijednost u je ugaona brzina koju pronalazimo u senzoru žiroskopa a predstavlja trenutnu brzinu sistema. Unutar funkcije filtriranja Kalmanovim filtrom u programskom kodu, sve strukture se primaju kao pokazivači. Dio koda vezan za ovaj korak prikazan je ispod. Mala modifikacija unutar koda uzima razliku ugaone brzine u stanju sistema i ugaone brzine iz žiroskopa da proslijedi ugaonu brzinu sistema koja neće imate integracione greške kao izlaz iz filtra.

$$2) \qquad x_k^- = A * x_{k-1} + \Delta t * u; \quad A = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix}$$

```
/* State prediction. This step uses the previous value to predict the next value */
Kalman->Xk[0] = Kalman->Xk[0] + dt * GyroRate - dt * Kalman->Xk[1];

/* Eliminating the drift from the gyroscope velocity readings */
Kalman->NoDriftGyroRate = GyroRate - Kalman->Xk[1];
```

Predikcija kovarijanse greške je naredni korak algoritma. Koristi se kovarijansa greške iz prethodnog koraka da bi se predvidjela kovarijansa greške u ovom koraku. Programski kod za ovaj korak priložen je ispod. Q je matrica kovarijanse i njena vrijednost je experimentalno određena prije implentacije filtra.

$$3) \qquad P_k^- = A * P_{k-1} * A^T + Q$$

```
/* Error covariance prediction. This step uses the prevoious error covariance to determine the current */
Kalman->P[0][0] = Kalman->P[0][0] - dt * Kalman->P[1][0] -dt * (Kalman->P[0][1] -dt * Kalman->P[1][1])+
Kalman->Q[0];
Kalman->P[0][1] = Kalman->P[0][1] - dt * Kalman->P[1][1];
Kalman->P[1][0] = Kalman->P[1][0] - dt * Kalman->P[1][1];
Kalman->P[1][1] = Kalman->P[1][1] + Kalman->Q[1];
```

Četvrti korak predstavlja određivanje Kalmanovih težina. H i R matrice se izračunavaju van filtra gdje R matrica predstavlja matricu kovarijanse koja je experimentalno određena a H matrica ima fiksan oblik. Kalmanove težine koriste se u estimaciji narednog stanja sistema i ažuriraju se pri svakoj promjeni stanja sistema. Programski kod ovog koraka takođe je priložen ispod.

$$4) \qquad K_k = P_k^- * H^T * \frac{1}{H * P_k^- * H^T + R}; \quad H = [1 \quad 0]$$

```
/* Kalman gain computation. It is later used as weight to determine the estimation of the position */
float K[2];
K[0] = Kalman->P[0][0] / ((float)(Kalman->P[0][0] + Kalman->R));
K[1] = Kalman->P[1][0] / ((float)(Kalman->P[0][0] + Kalman->R));
```

Sledeći korak je estimacija stanja sistema. U ovom koraku algoritam kompenzuje grešku između izmjerenog stanja i predviđenog stanja. Ovo je izlaz filtra u opštem slučaju. z je izmjereno stanje sistema u trenutku t.

$$5) \qquad x_k = x_k^- + K_k * H^T * (z_k - H * x_k^-)$$

```
/* Estimate computation. In this step, the algorithm determines the difference between the actual reading and
the estimated readings */
float difference = AccelAngle - Kalman->Xk[0];
Kalman->Xk[0] += K[0] * difference;
Kalman->Xk[1] += K[1] * difference;
```

Poslednji korak čini ažuriranje greške kovarijanse. Veća greška pokazuje manju tačnost estimacije stanja sistema.

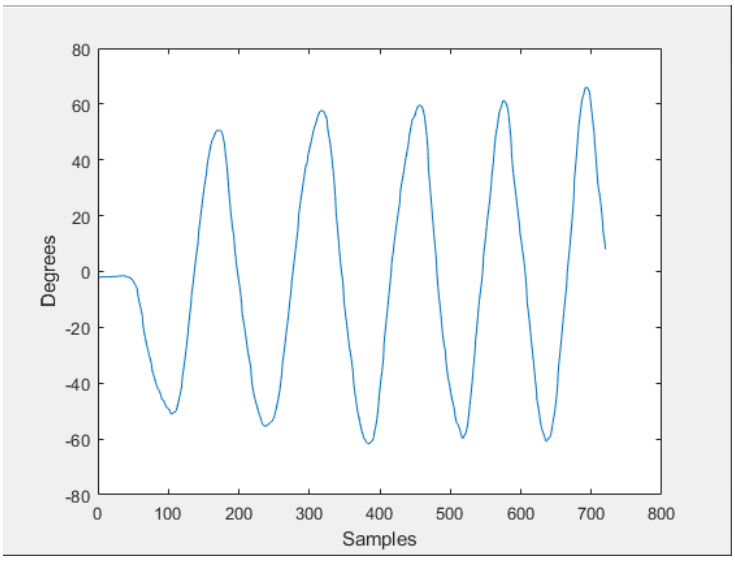
6) $$P_k = P_k^- - K_k * H * P_k^-$$

```
/* Error covariance computation. Larger Pk signifies a bigger error in estimation */
float P0 = Kalman->P[0][0];
float P1 = Kalman->P[0][1];

Kalman->P[0][0] -= K[0] * P0;
Kalman->P[0][1] -= K[0] * P1;
Kalman->P[1][0] -= K[1] * P0;
Kalman->P[1][1] -= K[1] * P1;
```

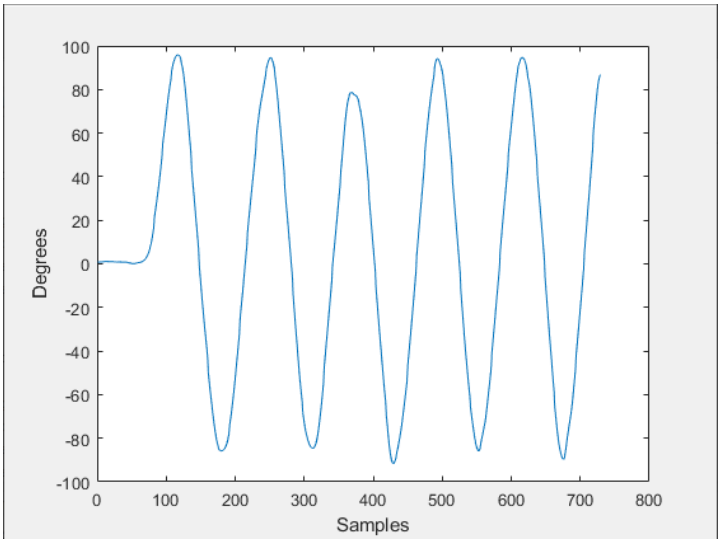
4.5.2. Roll i Pitch

Sada konačno imamo sve potrebne informacije i alate za određivanje rotacionih osa *roll* i *pitch*. U funkciju koja vrši Kalmanovo filtriranje za određivanje ovih rotacionih osa na mjestu ugaone pozicije šaljemo rotacione ose određene senzorom akcelerometra koje imaju problem šuma. Na mjesto ugaone brzine šaljemo rotacione ose iz žiroskopa koje imaju problem integracione akumulacije grešaka i kao rezultat iz ažurirane strukture Kalmanovog filtra za tu osu uzimamo ugaonu brzinu koja neće akumulirati greške u integraciji jer se u svakom koraku modifikuje unutar Kalmanovog filtra. Na slikama 4.5.2.0 i 4.5.2.1 prikazan je rezultat pomjeranja senzora i funkcija promjene ovih rotacionih osa gdje se vidi da su svi problemi eliminisani. Programski kod ovog dijela dat je ispod.

```
KalmanFilterXYZ(&KalmanX, GyroRateX, AccelAngleX, dt);
KalmanFilterXYZ(&KalmanY, GyroRateY, AccelAngleY, dt);

/* Determining roll and pitch from the gyroscope data with removed drift (Kalman filtering) */
roll += KalmanX.NoDriftGyroRate*dt;
pitch += KalmanY.NoDriftGyroRate*dt;
```



Slika 4.5.2.0. Roll nakon Kalmanovog filtriranja



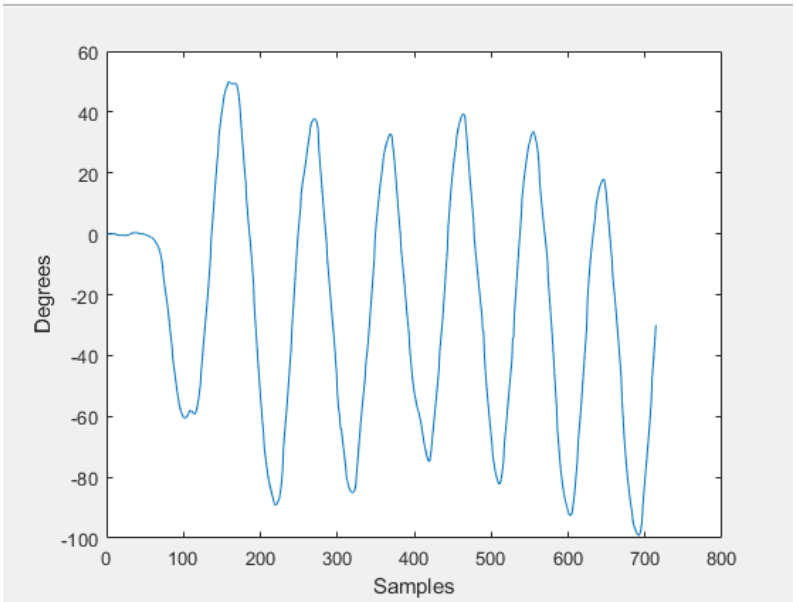
Slika 4.5.2.1. Pitch nakon Kalmanovog filtriranja

4.5.3. Yaw

Prije nego se odradi slična procedura i za *yaw* osu, neophodno je filtrirati podatke koje nam pristižu nakon kompenzacije tilta. To se postiže jednim niskofrekventnim filtrom koji čini da su skokovi unutar podataka mnogo manji ali je i odziv mnogo sporiji. Nakon ovog se izvodi Kalmanovo filtriranje ove ose sa srodnim podatkom iz žiroskopa za ovu osu. Na slici 4.5.3.0

prikazan je konačan rezultat okretanja senzora i funkcija promjene *yaw* ose. Programski kod priložen je ispod.

```
/* Temporary yaw value which is really noise, obtained from the equation for the tilt compensation */
float yaw_temp = atan2(Hy, Hx) * 180/PI;

/* Using NF filter to reduce the noise from the yaw value */
yaw_tempf = 0.5*yaw_tempf + 0.5*yaw_temp;

/* Using Kalman filter to reduce more noise and eliminate drift from the Z-axis gyroscope reading */
KalmanFilterXYZ(&KalmanZ, GyroRateZ, yaw_tempf, dt);

yaw += KalmanZ.NoDriftGyroRate*dt;
```



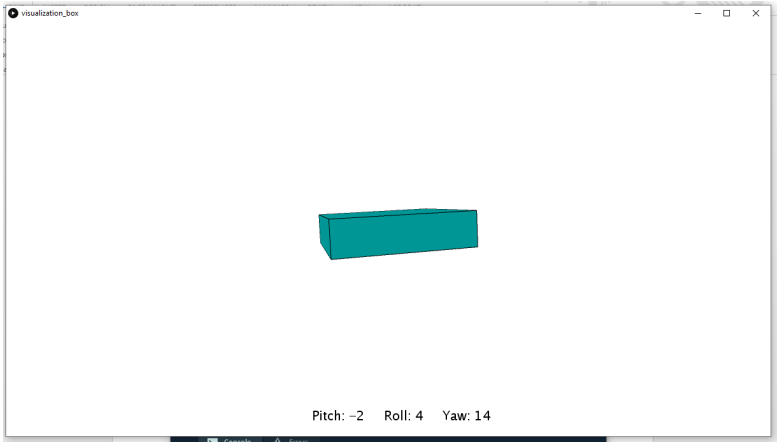
Slika 4.5.3.0. Yaw nakon Kalmanovog filtriranja

5. VIZUELIZACIJA DETEKCIJE ROTACIJE

Poslednji dio projekta, radi bolje preglednosti rezultata, bavio se vizuelizacijom rezultata obrade u vidu 3D kutije (box) koja se iscrtavala na ekranu računara preko programa *Processing*[14]. Mikrokontroler je komunicirao sa računarom preko USART-a i brzinom od 50 Hz slao nove podatke sve tri rotacione ose koje su se u *Processing* programu čuvale i koristile za kontrolu kutije koja se okretala onako kako je korisnik okretao senzor. Slanje podataka sa mikrokontrolera vršilo se preko USART2_TDR registra iz kog su se podaci slali. Slanje se završavalo onda kada bit TC iz registra USART2_ISR postane jednak logičkoj jedinici. Primjer programskog koda za slanje podataka preko USART-a dat je ispod.

```
/**
 * @brief SendToUART
 *
 * Function that sends the string data over UART.
 *
 */
void SendToUART(uint8_t *pbuf) {
	uint32_t size = strlen((char*) pbuf);
	for (uint32_t i=0; i<size; i++){
	   USART2->TDR = pbuf[i];                          /**< Send through the Transmit Data Register */
	   while(!(USART2->ISR & USART_ISR_TC));          /**< Check transmission complete flag in order to
continue */
	}
}
```

Na slici 5.0 prikazan je izgled prozora na kom je iscrtana kutija koja predstavlja senzor koji prati okretanje realnog senzora u prostoru dok su na donjem dijelu prozora prikazane vrijednosti rotacionih osa.



Slika 5.0. Vizuelizaciona kutija

[14] Processing software, www.procesing.org

6. ZAKLJUČAK

Algoritam fuzije podataka iz MEMS senzora projektovan uz pomoć Kalmanovog filtra i kompenzacije nagiba senzora pokazao se kao efikasan za detekciju rotacije u tri dimenzije. Algoritam je uspješno eliminisao sve probleme koje MEMS senzori pojedinačno izražavaju i konačan rezultat algoritma ne pokazuje znake postojanja nijednog od njih. Problem *yaw* vrijednosti rotacione ose riješen je dodavanjem niskofrekventnog filtra prije Kalmanovog filtriranja i time se znatno smanjio uticaj nesavršenosti senzora koji je obezbjeđivao podatke koji su bili daleko sporije dostupni od ostalih senzora. Nabavljanje kvalitetnijeg magnetometra od onog u MPU-9250 eliminisao bi potrebu za niskofrekventnim filtrom što bi sistem učinilo nešto bržim ali daleko efikasnijim i elegantnijim. Ovim je pokazano da se MEMS senzori mogu koristiti za određivanje rotacione pozicije sistema u slučajevima kada drugi vid detekcije nije pogodan zbog svojih ograničenja ili čak u slučajevima kada su oba raspoloživa ali je ovaj možda jeftiniji ili manje podložan mogućim smetnjama. Interesantan nastavak ovog problema bio bi određivanje pozicije objekta u tri dimenzije pomoću MEMS senzora kao i njegova upotreba u složenijim projektima koji zahtijevaju praćenje pozicije objekta u prostoru.

LITERATURA

[1] STM32F051x8 Datasheet, STMicroelectronics

[2] STM32F051x8 User Manual, STMicroelectronics

[3] STM32F051x8 Errata Sheet, STMicroelectronics

[4] MPU-9250 Datahseet, InvenSense

[5] MPU-9250 Register Map, InvenSense

[6] MPU-9250 User Manual, InvenSense

[7] AK8963 Datasheet, Asahi Kasei Microsystems

[8] Embedded Kalman Filter For Inertial Measurement Unit (IMU) on the Atmega8535 – Hany Ferdinando, Handry Khoswanto and Djoko Purwanto

[9] Implementing a Sensor Fusion Algorithm for 3D Orientation Detection with Inertial/Magnetic Sensors - Fatemeh Abyarjoo, Jonathan Cofino, Armando Barreto and Francisco Raul Ortega