UNIVERSITAS INDONESIA

RANCANG BANGUN SISTEM NAVIGASI INERSIA DENGAN KALMAN FILTER PADA MIKROKONTROLER AVR

SKRIPSI

NANDO KUSMANTO
04 05 03 0583

FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK ELEKTRO
DEPOK
JUNI 2009

UNIVERSITAS INDONESIA

RANCANG BANGUN SISTEM NAVIGASI INERSIA DENGAN KALMAN FILTER PADA MIKROKONTROLER AVR

SKRIPSI

Diajukan Untuk Melengkapi Sebagian Persyaratan Menjadi Sarjana Teknik

NANDO KUSMANTO
04 05 03 0583

FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK ELEKTRO
DEPOK
JUNI 2009

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,

dan semua sumber baik yang dikutip maupun dirujuk

telah saya nyatakan dengan benar.

Nama	**:**	**Nando Kusmanto**
NPM	**:**	**0405030583**
Tanda Tangan	**:**	
Tanggal	**:**	**17 Juni 2009**

LEMBAR PENGESAHAN

Skripsi ini diajukan oleh	:	
Nama	:	Nando Kusmanto
NPM	:	0405030583
Program Studi	:	Teknik Elektro
Judul Skripsi	:	Rancang Bangun Sistem Navigasi Inersia Dengan Kalman Flter Pada Mikrokontroler AVR

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Teknik pada Program Studi Teknik Elektro, Fakultas Teknik, Universitas Indonesia

DEWAN PENGUJI

Pembimbing	:	Dr. Abdul Muis S.T., M.Eng.	( )
Penguji	:	Ir. Wahidin Wahab M.Sc., Ph.D.	( )
Penguji	:	Dr. Ir. Ridwan Gunawan M.T.	( )
Ditetapkan di	:	Ruang Multimedia B Lt. 2 Departemen Teknik Elektro FTUI	
Tanggal	:	1 Juli 2009	

UCAPAN TERIMA KASIH

Puji syukur saya panjatkan kepada Tuhan Yang Maha Esa, karena atas berkat dan rahmat-Nya, saya dapat menyelesaikan skripsi ini. Penulisan skripsi ini dilakukan dalam rangka memenuhi salah satu syarat untuk mencapai gelar Sarjana Teknik Departemen Elektro pada Fakultas Teknik Universitas Indonesia. Saya menyadari bahwa, tanpa bantuan dan bimbingan dari berbagai pihak, dari masa perkuliahan sampai pada penyusunan skripsi ini, sangatlah sulit bagi saya untuk menyelesaikan penulisan skripsi ini. Oleh karena itu, saya mengucapkan terima kasih kepada:

(1) Dr. Abdul Muis, ST., M.Eng. selaku dosen pembimbing yang telah menyediakan waktu, tenaga, dan pikiran untuk mengarahkan saya dalam penyusunan skripsi ini;
(2) orang tua, keluarga saya, dan Cynthia Noviani yang telah banyak memberikan bantuan dukungan material dan moral; dan
(3) Nur Hidayat, Abe Dharmawan, D.Ari Wicaksono, dan sahabat-sahabat yang telah banyak membantu dalam menyelesaikan skripsi ini.

Akhir kata, saya berharap Tuhan Yang Maha Esa berkenan membalas segala kebaikan semua pihak yang telah membantu. Semoga skripsi ini membawa manfaat bagi pengembangan ilmu.

Depok, 17 Juni 2009

Penulis

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama	:	Nando Kusmanto
NPM	:	0405030583
Departemen	:	Elektro
Fakultas	:	Teknik
Jenis karya	:	Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty-Free Right*)** atas karya ilmiah saya yang berjudul :

Rancang Bangun Sistem Navigasi Inersia Dengan Kalman Filter Pada Mikrokontroler AVR

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.
Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 17 Juni 2009
Yang menyatakan

(Nando Kusmanto)

ABSTRAK

Nama : Nando Kusmanto
Program studi : Teknik Elektro
Judul : Rancang Bangun Sistem Navigasi Inersia Dengan Kalman Filter Pada Mikrokontroler AVR

Sistem navigasi merupakan komponen yang paling penting pada kendaraan di udara, air dan luar angkasa, termasuk juga pada roket dan misil yang dapat dikendalikan. Salah satu yang paling umum digunakan adalah sistem navigasi inersia. Skripsi ini membahas mengenai perancangan dan pembuatan sistem navigasi inersia untuk mendapatkan data posisi dan kemiringan, yaitu dengan sensor *rate-gyroscope*, *accelerometer*, dan mikrokontroler AVR ATMega16. Demikian juga pembahasan tentang sistem kalibrasi dan digital filter data dari sensor. Selain itu, karena *accelerometer* dipengaruhi percepatan gravitasi, maka dibutuhkan suatu koreksi gravitasi dimana membutuhkan data kemiringan yang sangat akurat. Dalam skripsi ini kalman filter digunakan untuk mendapatkan data kemiringan yang lebih akurat, dengan memanfaatkan dua masukan, dari *rate-gyroscope* dan *accelerometer*.

Kata kunci :
Sistem navigasi inersia, *gyroscope*, *accelerometer*, kalman filter

ABSTRACT

Name : Nando Kusmanto
Study Program: Electrical Engineering
Title : Inertial Navigation System Design with Kalman Filter by using AVR Microcontroller

Navigation system is the most important component in air-, space-, and watercraft, including guided missiles. One of the common navigation systems is inertial navigation system. This bachelor thesis discusses about designing and building inertial navigation system, to get information about position and tilt, using rate-gyroscope, accelerometer, and AVR ATmega16 microcontroller. Furthermore, this thesis also discusses about calibration system and digital filter of sensor's data. In addition, because accelerometer also measures gravity acceleration, to get the real position needs a gravity correction which needs very accurate information about tilt angle. In this study, kalman filter used to get more accurate tilt angle, using two inputs, from rate-gyroscope and accelerometer

Key Words:
Inertial navigation system, gyroscope, accelerometer, kalman filter

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

DAFTAR SINGKATAN

ADC	*Analog to Digital Converter*
IMU	*Inertial Measurement Unit*
INS	*Inertial Navigation System*
PC	*Personal Computer*
USART	*Universal Synchronous and Asynchronous serial Receiver and Transmitter*

DAFTAR ISTILAH

Accelerometer	alat pengukur percepatan linear
Bit	digit biner dalam sistem bilangan biner, dapat bernilai 0 atau 1
Byte	seri 8-bit
Rate-Gyroscope	alat pengukur kecepatan putar
Least Significant Bit (LSB)	bit dengan nilai pangkat terendah dalam satuan biner
Master	divais yang mengontrol pesan
Mikrokontroler	single purpose processing unit yang didesain untuk melakukan program pengendalian kecil, kadang real time
Most Significant Bit (MSB)	bit dengan nilai pangkat tertinggi dalam satuan biner
Pitch	perputaran pada sumbu y
Roll	perputaran pada sumbu x
SCL	Serial Clock pada I^2C *interface*
SDA	Serial Data pada I^2C *interface*
Slave	divais yang dikontrol oleh *master*
Yaw	perputaran pada sumbu z

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Untuk menjalankan suatu wahana ataupun robot otomatis, dibutuhkan suatu sistem navigasi. Sistem navigasi merupakan komponen yang paling penting pada mesin otomatis, baik robot, pesawat maupun kendaraan otomatis lainnya seperti kapal, mobil ataupun pada satelit, termasuk juga pada roket dan misil yang dapat dikendalikan. Salah satu sistem yang umum digunakan untuk navigasi adalah sistem navigasi inersia atau *inertial navigation system* (INS), yaitu dengan memanfaatkan *Inertial Measurement Unit (IMU)*. Sebuah IMU bekerja dengan mendeteksi gerakan (percepatan linear dan orientasi dari gerakan tersebut) dengan menggunakan kombinasi dari *accelerometer* dan *rate-gyroscope*. Data yang didapat dari sensor ini selanjutnya diolah komputer untuk mengetahui posisi kendaraan tersebut.

Untuk mendapatkan data dari IMU yang lebih akurat, dibutuhkan suatu sistem kalibrasi. Selain itu, karena *accelerometer* dipengaruhi percepatan gravitasi, maka dibutuhkan suatu koreksi gravitasi yang membutuhkan data kemiringan yang akurat. Untuk itu, skripsi ini menggunakan kalman filter untuk mendapatkan data kemiringan yang lebih akurat, dengan memanfaatkan dua input yaitu dari *rate-gyroscope* dan *accelerometer*.

Skripsi ini membahas mengenai perancangan dan pembuatan suatu sistem navigasi, sehingga dapat diketahui posisi dan kemiringan dari kendaraan tersebut. Selain itu, untuk menghemat biaya dalam membuat sebuah robot ataupun wahana otomatis, maka dibutuhkan suatu algoritma sistem kalibrasi dan kalman filter yang sederhana, sehingga dapat diterapkan pada *low-cost* mikrokontroler. Pada skripsi ini, sistem navigasi inersia akan diterapkan pada *low-cost* mikrokontroler yang umum digunakan, yaitu tipe AVR.

1.2 Tujuan Penulisan

Tujuan utama dari penulisan skripsi ini adalah untuk merancang dan membuat sistem navigasi dengan biaya rendah, mempunyai presisi yang tinggi,

dan memberikan informasi secara cepat, dengan menggunakan mikrokontroler dan PC sebagai pengukur dan penyimpan data.

1.3 Pembatasan Masalah

Pembahasan dalam skripsi ini difokuskan dalam pengujian kalibrasi dan kalman filter untuk mendapatkan data kemiringan *roll* dan *pitch*, serta pengujian *accelerometer* untuk mendapatkan data posisi dengan menggunakan pemograman C pada mikrokontroler AVR ATmega16.

1.4 Metode Penelitian

Metode penelitian yang digunakan dalam penyusunan skripsi ini meliputi:

1. Pendekatan tinjauan pustaka, yaitu dengan melakukan studi literatur dari buku-buku pustaka, atau *manual book* serta *reference book* dari suatu perangkat yang digunakan.
2. Pendekatan diskusi dengan pembimbing skripsi ataupun teman-teman yang berkaitan dengan topik bahasan skripsi.
3. Perancangan perangkat keras dan perangkat lunak.
4. Pengujicobaan dan pengambilan data.

1.5 Sistematika Penulisan

Untuk mempermudah penulisan dan agar pembahasan yang disajikan lebih sistematis, maka laporan ini dibagi ke dalam lima bab. Isi masing – masing bab diuraikan secara singkat dibawah ini :

BAB 1 PENDAHULUAN

Bab ini menjelaskan tentang latar belakang masalah, tujuan, batasan masalah, metode penulisan, dan sistematika penulisan seminar.

BAB 2 LANDASAN TEORI

Pada bab ini menjelaskan tentang dasar teori dari sistem navigasi inersia, mikrokontroler AVR ATmega16 dan kalman filter.

BAB 3 PERANCANGAN SISTEM NAVIGASI INERSIA

Bab ini menjelaskan segala sesuatu yang digunakan dalam

merancang dan membangun sistem navigasi inersia, dari sisi perangkat keras dan perangkat lunak.

BAB 4 PENGUJIAN DAN ANALISA

Bab ini berisikan mengenai pengujian dari sistem navigasi inersia yang telah dibangun, meliputi pengujian digital filter, kalman filter, dan keakuratan data yang didapatkan beserta analisanya.

BAB 5 KESIMPULAN

Bab terakhir ini berisikan pernyataan singkat dan tepat yang dijabarkan dari hasil studi literatur atau landasan teori dari penyusunan seminar.

BAB 2
LANDASAN TEORI

2.1 Prinsip Dasar Sistem Navigasi Inersia

Sistem navigasi inersia merupakan salah satu sistem navigasi yang paling sering digunakan pada wahana di udara, air dan luar angkasa, termasuk juga pada roket dan misil yang dapat dikendalikan. Keuntungan dari sistem navigasi inersia adalah biaya pembuatannya yang cukup rendah, dan sistem yang sederhana, sehingga sistem ini sangat praktis untuk digunakan. Selain itu, kelebihan utama dari sistem navigasi inersia adalah sistem tidak dipengaruhi oleh keadaan lingkungan. Pada sistem navigasi GPS, sistem akan mengalami masalah ketika GPS tidak mendapatkan data karena *noise* yang sangat besar akibat terhalang benda (di dalam ruang), keadaan atmosfer, hujan, dan lain-lain. Hal ini tidak terjadi pada sistem navigasi inersia.

Sistem navigasi inersia bekerja dengan memanfaatkan IMU. *Inertial Measurement Unit* atau IMU merupakan sebuah sensor yang digunakan untuk mengukur percepatan linear (x, y dan z) dan kecepatan putar (*roll*, *pitch* dan *yaw*). Prinsip dasar dari sistem navigasi inersia adalah dengan mengintegralkan percepatan linear dan kecepatan putar yang didapatkan dari IMU, sehingga didapatkan data posisi dan kemiringan. Posisi dan kemiringan yang didapatkan dari sistem navigasi inersia ini relatif terhadap kondisi awal (posisi dan kemiringan awal). Data posisi akan didapatkan dengan mengintegrasikan sebanyak dua kali dari data percepatan, dapat dilihat persamaan (2.3), sedangkan data sudut kemiringan didapatkan dengan mengintegrasikan kecepatan putar sekali, yang dapat dilihat pada persamaan (2.4).

$$v = \int a \; dt \tag{2.1}$$

$$s = \int v \; dt \tag{2.2}$$

dengan a adalah percepatan linear, v adalah kecepatan linear, dan s adalah jarak atau posisi. Dari persamaan (2.1) dan (2.2) maka didapatkan:

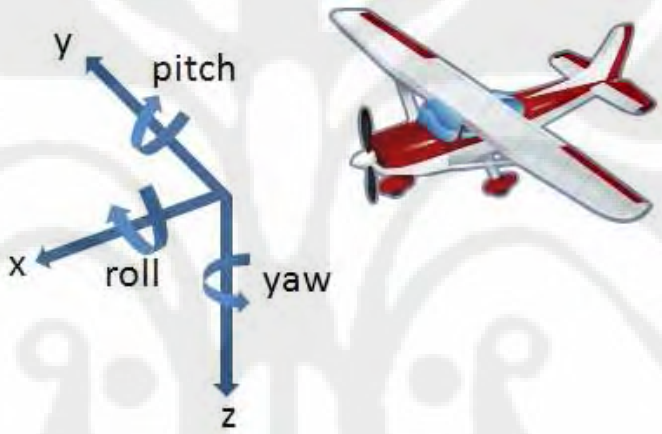
$$s = \iint a \; dt \; dt \tag{2.3}$$

Sedangkan

$$\theta = \int \omega \; dt \tag{2.4}$$

dengan ω adalah kecepatan putar dan Θ adalah sudut.

Biasanya IMU merupakan rangkaian yang terdiri dari 3 buah *accelerometers* dan 3 buah *rate-gyroscope*. Percepatan linear akan diukur dari tiga *accelerometer* yang dipasang secara orthogonal satu sama lain dan kecepatan putar diukur dengan menggunakan tiga *rate-gyroscope* yang juga dipasang dengan pola orthogonal. *Accelerometer* digunakan untuk mengukur percepatan inersia yang dikenal dengan sebutan *G-force* (1 G = 9.80665 m/s^2) atau percepatan gravitasi. Sedangkan *rate-gyroscope* mengukur kecepatan putar relatif terhadap sumbu koordinat yang digunakan. Untuk lebih jelasnya pada gambar 2.1



Gambar 2.1 Pengukuran percepatan linear dan orientasi oleh sensor IMU
Sumber: http://zone.ni.com/devzone/cda/tut/p/id/8163

Kekurangan utama dari sistem navigasi inersia adalah sistem mengalami akumulasi kesalahan. Kesalahan kecil pada pengukuran percepatan dan kecepatan putar akan menyebabkan kesalahan yang semakin berkembang pada komponen kecepatan dan posisi akibat proses integrasi.

Saat ini sistem navigasi inersia digunakan untuk semua kendaraan otomatis. Hampir semua kendaraan air komersial ataupun militer dilengkapi dengan sistem ini. Kebanyakan pesawat terbang juga menggunakannya.

2.2 Mikrokontroler AVR ATmega 16

AVR merupakan seri mikrokontroler CMOS 8-bit buatan Atmel, berbasis arsitektur RISC (*Reduced Instruction Set Computer*) yang ditingkatkan. Hampir semua instruksi dieksekusi dalam satu siklus *clock*. AVR mempunyai 32 *register*

general-purpose, *timer/counter* fleksibel dengan mode *compare*, *interrupt* internal dan eksternal, serial UART, *programmable Watchdog Timer*, *two-wire* (I^2C), ADC, PWM internal dan mode *power saving*. AVR juga mempunyai *In-System Programmable Flash on-* dalam sistem menggunakan hubungan serial SPI. ATmega16 adalah mikrokontroler CMOS 8-bit daya-rendah berbasis arsitektur RISC yang ditingkatkan. ATmega16 mempunyai *throughput* mendekati 1 MIPS per MHz untuk mengoptimasi komsumsi daya terhadap kecepatan proses.

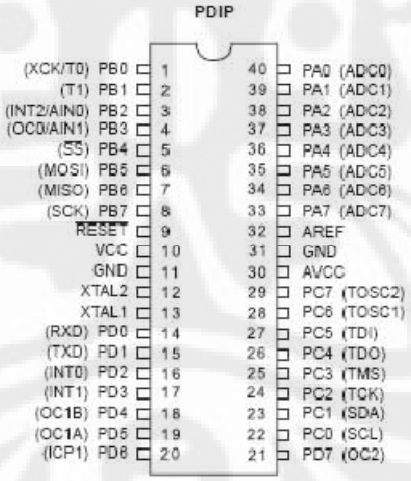
Beberapa keistimewaan dari AVR ATmega16 antara lain:

1. *Advanced RISC Architecture*
    - *130 Powerful Instructions – Most Single Clock Cycle Execution*
    - *32 x 8 General Purpose Fully Static Operation*
    - *Up to 16 MIPS Throughput at 16 MHz*
    - *On-chip 2-cycle Multiplier*
2. *Nonvolatile Program and Data Memories*
    - *8K Bytes of In-System Self-Programmable Flash*
    - *Optional Boot Code Section with Independent Lock Bits*
    - *512 Bytes EEPROM*
    - *512 Bytes Internal SRAM*
    - *Programming Lock for Software Security*
3. *Peripheral Features*
    - *Two 8-bit Timer/Counters with Separate Prescalers and Compare Mode*
    - *Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes*
    - *One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode*
    - *Real Time Counter with Separate Oscillator*
    - *Four PWM Channels*
    - *8-channel, 10-bit ADC*
    - *Byte-oriented Two-wire Serial Interface*
    - *Programmable Serial USART*
4. *Special Microcontroller Features*

- *Power-on Reset and Programmable Brown-out Detection*
- *Internal Calibrated RC Oscillator*
- *External and Internal Interrupt Sources*
- *Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby*

5. *I/O and Package*
   - *32 Programmable I/O Lines*
   - *40-pin PDIP, 44-lead TQFP, 44-lead PLCC, and 44-pad MLF*
6. *Operating Voltages*
   - *2.7 - 5.5V for ATmega16L*
   - *4.5 - 5.5V for Atmega16*

Pin-pin pada ATmega16 dengan kemasan 40-pin DIP (*dual inline package*) ditunjukkan oleh gambar 2.2. Kemasan pin tersebut terdiri dari 4 Port yaitu Port A, Port B, Port C,Port D yang masing masing Port terdiri dari 8 buah pin. Selain itu juga terdapat RESET, VCC, GND 2 buah, VCC, AVCC, XTAL1, XTAL2 dan AREF.



Gambar 2.2 Pin-pin ATMega 16 kemasan 40 pin

Sumber: Datasheet ATmega16

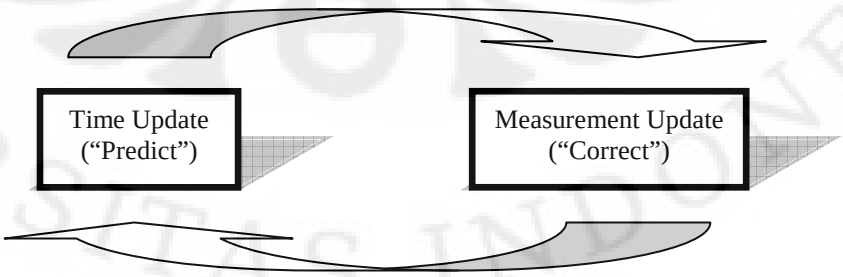
2.3 Kalman Filter

Kalman filter merupakan sebuah *recursive* filter yang efisien, yang mengestimasi *state* pada *linear dynamic system* dari rentetan pengukuran *noise*. Disebut *recursive* sebab untuk menghitung state estimasi saat ini, hanya

membutuhkan data state estimasi satu waktu sebelumnya dan data pengukuran saat ini. Teknik filter ini dinamakan berdasarkan penemunya, Rudolf E. Kalman. Kalman filter sangat berguna terutama dalam navigasi dan lingkungan dengan *Gaussian noise.*

Pada teori kendali, kalman filter merupakan sebuah algoritma atau kumpulan persamaan matematika yang menghasilkan sebuah perhitungan yang efisien untuk mengestimasi state dari proses, dengan tujuan meminimalkan *noise* atau variansi terhadap referensi lain. Filter ini sangat bagus dalam beberapa aspek: mendukung estimasi state sebelumnya, saat ini dan berikutnya. Bahkan hal ini tetap dapat dilakukan meskipun model sistem yang sebenarnya tidak diketahui. Kalman filter disebut juga sebuah estimator stokastik yang optimal.

Pada sistem navigasi, kalman filter yang digunakan adalah kalman filter diskrit. Kalman filter akan mengestimasi proses dengan menggunakan bentuk pengendali *feedback* : filter mengestimasi state proses pada beberapa waktu dan kemudian mendapatkan umpanbalik (*feedback*) dalam bentuk pengukuran (*noise*). Oleh karena itu, persamaan kalman filter dibagi menjadi dua kelompok: persamaan *time update* dan persamaan *measurement update. Time update* dapat disebut juga sebagai proses *predict*, yaitu menggunakan estimasi state dari satu waktu sebelumnya untuk mendapatkan sebuah estimasi state pada saat ini. Sedangkan *measurement update* disebut juga sebagai proses *correct*, yaitu informasi pengukuran pada saat ini digunakan untuk memperbaiki prediksi, dengan harapan akan didapatakan state estimasi yang lebih akurat. Sehingga dalam aplikasinya, algoritma kalman filter akan menggunakan proses berulang dari *predict* dan *correct*. Seperti yang dapat dilihat pada gambar 2.3.



Gambar 2.3 Perputaran algoritma kalman filter

Sumber: An Introduction to Kalman Filter hal. 5, telah diolah kembali

Berikut ini adalah persamaan *predict* dan *correct* dalam kalman filter:

***Time Update* (*Predict*)**

Predicted state $\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1}$ (2.5)

Predicted error covariance $P_k^- = P_{k-1}A^T + Q$ (2.6)

***Measurement Update* (*Correct*)**

Innovation $inno = z_k - H\hat{x}_k^-$ (2.7)

Innovation merupakan perbedaan antara matrix *observation* (z_k) dengan nilai *predicted* sensor $H\hat{x}_k^-$. Matrix *observation* (z_k) diukur dengan sebuah sensor. Kemudian tugas pertama dalam *measurement update* adalah menghitung kalman gain dengan rumus (2.8). Perlu diketahui bahwa persamaan ini hanya berlaku untuk satu observer, jika lebih dari satu observer maka persamaan lain harus digunakan.

Optimal Kalman gain $K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$ (2.8)

Kemudian akan didapatkan estimasi optimal dari state yang baru dengan menggunakan *innovation*.

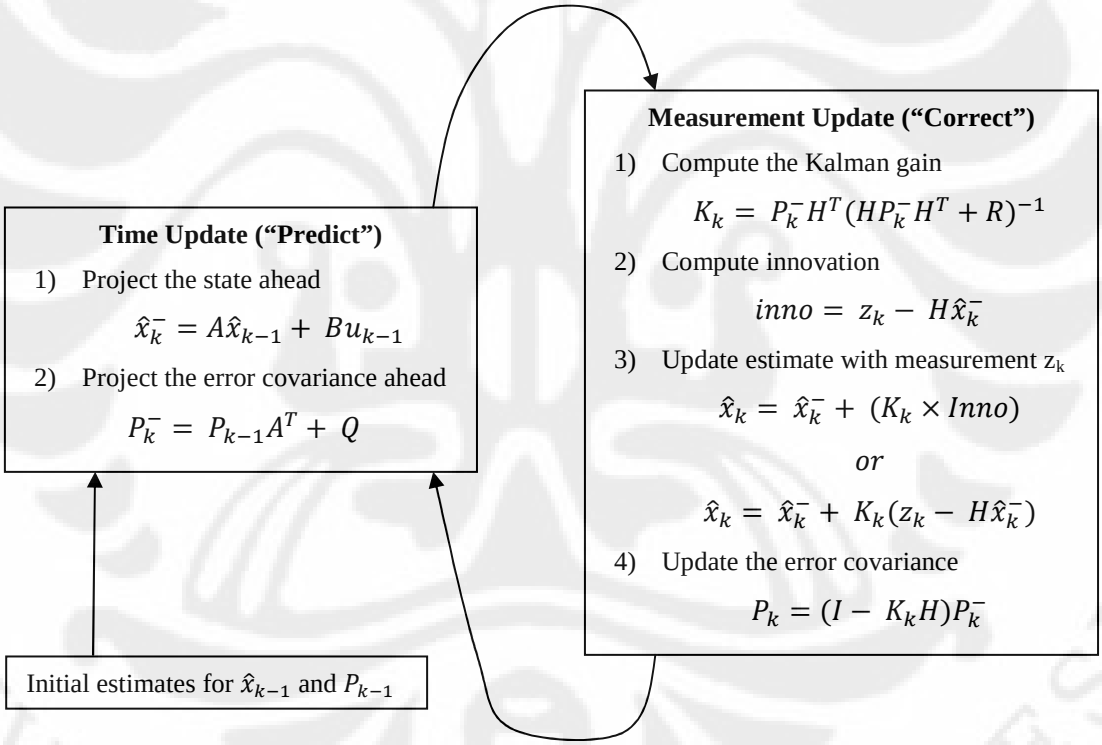
Updated state estimate $\hat{x}_k = \hat{x}_k^- + (K_k \times Inno)$ (2.9)

Updated error covariance $P_k = (I - K_k H)P_k^-$ (2.10)

Pada persamaan di atas terdapat variabel Q yang merupakan *process noise covariance* dan R yang merupakan *measurement noise covariance*. Dalam implementasi sebenarnya dari filter ini, R dan Q biasanya diukur terlebih dahulu untuk operasi filter. Penentuan parameter ini sangat tergantung dari model sistem yang digunakan. Perlu digarisbawahi bahwa R dan Q sebaiknya adalah nilai yang konstan, sebab pada kondisi demikian *estimation error covariance* P_k dan kalman gain akan stabil lebih cepat dan akan tetap konstan.

Selain itu ketika memulai proses kalman filter diperlukan *initial condition* dari P_{k-1} dan $\hat{x}_{k-1}$ atau bisa disebut juga P_0 dan $\hat{x}_0$. Penentuan *initial condition* ini juga berdasarkan sistem yang digunakan. $\hat{x}_0$ biasanya didapatkan dengan memperkirakan state sistem pada keadaan awal, sedangkan P_0 sebaiknya bernilai tidak sama dengan nol, sebab apabila $P_0 = 0$ akan menyebabkan filter menginisialisasi dan selalu percaya bahwa $\hat{x}_k$=$\hat{x}_0$ (Welch and Bishop, 2006). Selain itu, Welch dan Bishop (2006) juga menambahkan bahwa penentuan P_0 tidak begitu penting sebab filter akan menyesuaikan dengan sendirinya.

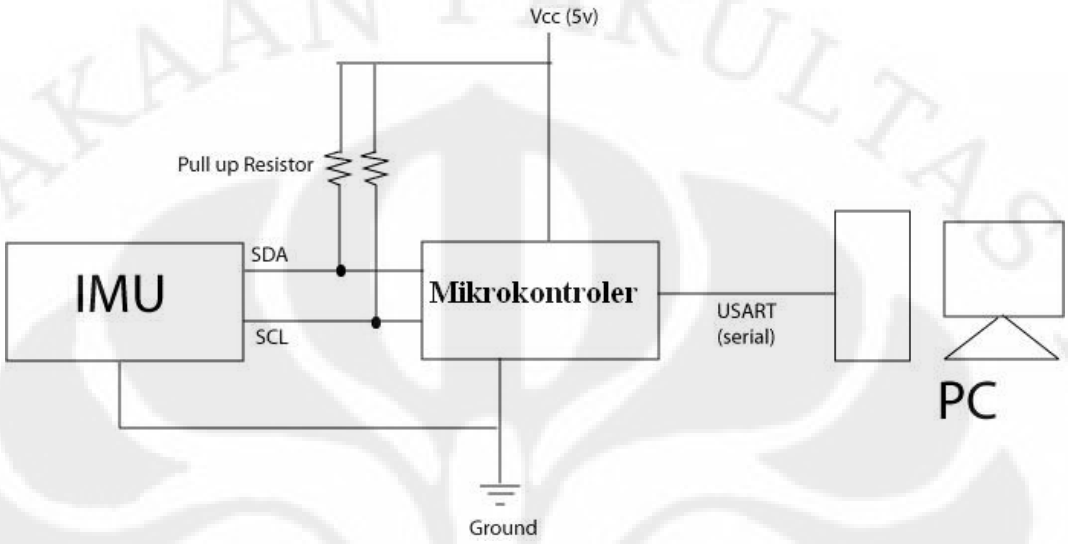
Secara umum, proses kalman filter terdiri dari proses *predict* dan *correct* yang dilakukan berulang terus-menerus. Gambar (2.4) menunjukkan gambaran lengkap dari operasi kalman filter.



Gambar 2.4 Gambaran lengkap proses kalman filter

Sumber : An Introduction to Kalman Filter hal. 6, telah diolah kembali

BAB 3
PERANCANGAN SISTEM NAVIGASI INERSIA



Gambar 3.1 Rangkaian sistem navigasi inersia

Gambar 3.1 menunjukkan rangkaian sistem navigasi inersia yang digunakan dalam skripsi ini. Komunikasi antar IMU dengan mikrokontroler menggunakan I^2C *interface*. Mikrokontroler berfungsi dalam pengambilan data dari sensor dan mengolahnya sehingga didapatkan data mengenai posisi dan kemiringan. Data akhir mengenai posisi dan kemiringan akan ditampilkan ke PC. Pengiriman data dari mikrokontroler ke PC dilakukan secara serial (USART). PC yang digunakan sebagai penampil data, menggunakan Microsoft Windows, dan memanfaatkan *hyper terminal*. Selain itu untuk mempermudah dalam mengamati pengaruh kalman filter dan parameternya terhadap sudut kemiringan, juga digunakan perangkat lunak *phyton*.

Perancangan sistem navigasi inersia dibagi ke dalam dua hal yaitu perangkat keras dan perangkat lunak. Perancangan perangkat keras terbagi menjadi perancangan setiap perangkat keras yang digunakan dan setiap perangkat keras komunikasi yang digunakan untuk menghubungkan perangkat keras tersebut. Sedangkan perancangan perangkat lunak adalah semua algoritma pemrograman yang digunakan pada mikrokontroler AVR Atmega16, meliputi

algoritma pengambilan data, integrasi, kalibrasi, digital filter, kalman filter, dan koreksi gravitasi.

3.1 Perancangan Perangkat Keras

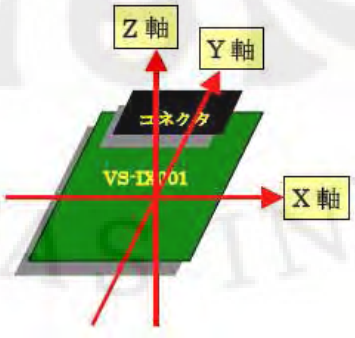
Rangkaian perangkat keras yang digunakan dalam membangun sistem navigasi inersia dapat dilihat dari gambar 3.1. Rangkaian perangkat keras ini disusun dengan menggunakan perangkat keras sebagai berikut:

- Modul VS-IX001 sebagai IMU
- Mikrokontroler AVR ATmega16 dan sistem minimumnya
- Perangkat keras rangkaian komunikasi I^2C *interface*
- Perangkat keras komunikasi serial dengan PC

3.1.1 VS-IX001

VS-IX001 merupakan sensor IMU yang terdiri dari 3 *accelerometer* (untuk mengukur percepatan linear sumbu x, y dan z) dan 2 *rate-gyroscope* (untuk mengukur kecepatan putar terhadap sumbu x dan y atau kecepatan putar *roll* dan *pitch*).

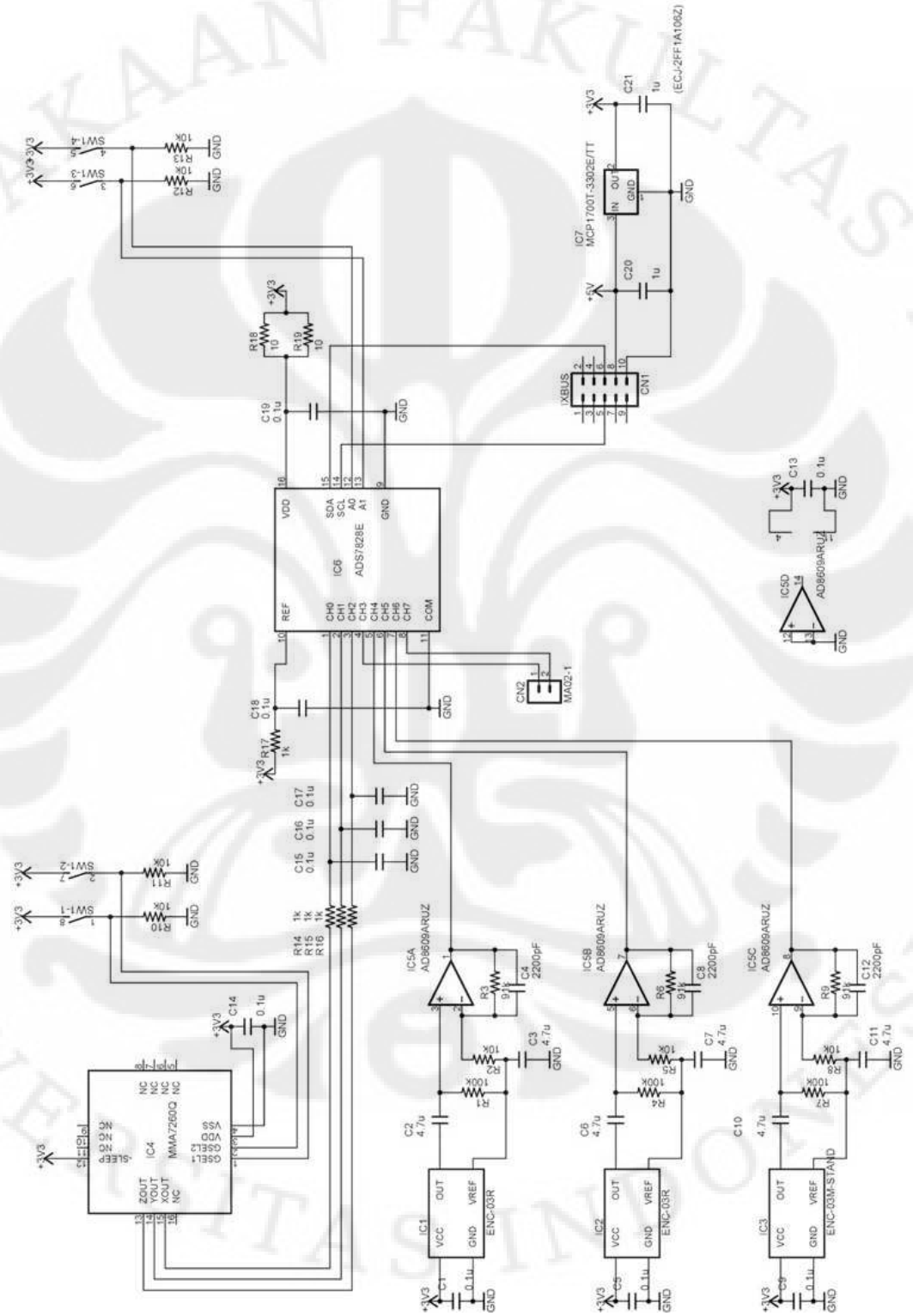
Gambar 3.2 VS-IX001



Gambar 3.3 Orientasi modul VS-IX001

Sumber : Datasheet VS-IX001

Cara kerja alat ini dapat diketahui dengan mempelajari bentuk rangkaiannya. Berikut ini adalah rangkaian skematik pada modul IMU VS-IX001:



Gambar 3.4 Rangkaian skematik VS-IX001

Sumber : Datasheet VS-IX001

Dari gambar 3.4 terdapat beberapa komponen penting pada modul IMU ini, yaitu:

- IC MMA7260Q yang merupakan *accelerometer*
- 2 buah ENC-03R yang merupakan *rate-gyroscope*
- ADS 7828E yang merupakan *analog to digital converter*
- Switch (SW 1-1, SW 1-2, SW 2-1, dan SW 2-2)
- MCP1700T yang merupakan *voltage regulator*
- 2x5 pin konektor

Secara garis besar cara kerja IMU ini adalah: data dari *accelerometer* dan *rate-gyroscope* yang merupakan data analog akan dikonversi dalam bentuk digital oleh *analog to digital converter* yang kemudian keluarannya dapat diambil datanya dengan I^2C *interface*. Modul ini akan bertindak sebagai divais *slave* dalam I^2C *interface* atau dengan kata lain yang menerima perintah dari divais *master*.

3.1.1.1 MMA7260Q

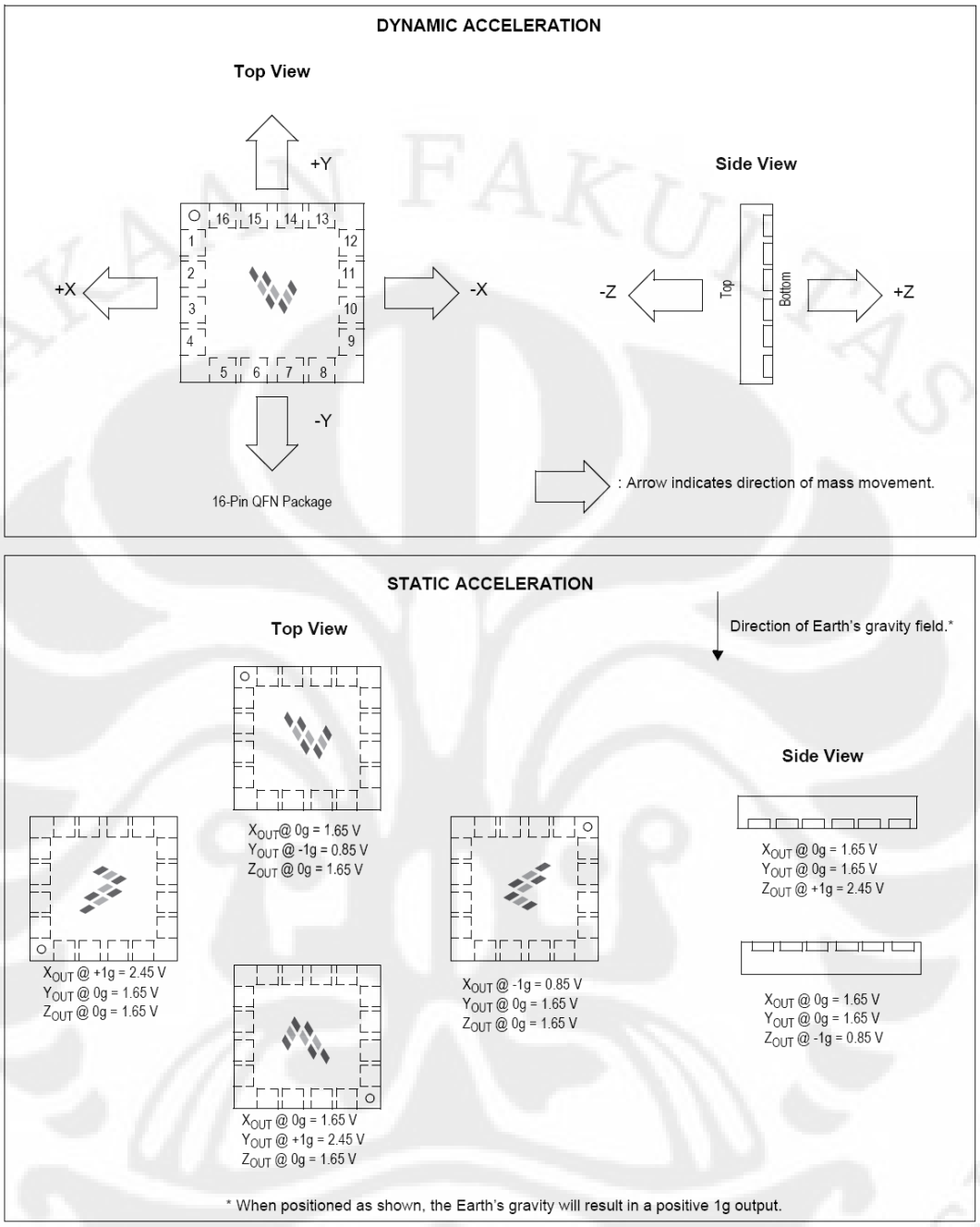
MMA7260Q merupakan *accelerometer* dengan sensitifitas yang dapat diatur untuk mendapatkan data percepatan sumbu x, y dan z. Keluaran dari IC ini berupa data analog yaitu tegangan. Sensitifitas dari IC ini dapat diatur dari switch 1-1 dan switch 1-2.

Tabel 3.1 Pengaturan sensitifitas *accelerometer* (MMA7260Q)

SW 1-1	SW 1-2	g-range	Sensitifitas
OFF	OFF	1,5g	800mV/g
OFF	ON	2g	600mV/g
ON	OFF	4g	300mV/g
ON	ON	6g	200mV/g

Sumber : Datasheet VS-IX001, telah diolah kembali

Sensitifitas menunjukkan besarnya penambahan tegangan keluaran tiap penambahan percepatan sebesar 1g dan pengurangan tegangan keluaran tiap penambahan percepatan sebesar 1g ke arah sebaliknya, dan ketika kondisi stabil (tanpa percepatan) keluaran tegangan adalah 1,65 Volt. IC ini memiliki *internal sampling frequency* sebesar 11 KHz.



Gambar 3.5 Tegangan keluaran *accelerometer* untuk berbagai kondisi

Sumber : Datasheet MMA7260Q

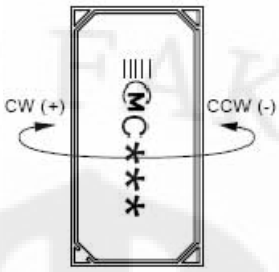
3.1.1.2 ENC-03R

ENC-03R merupakan *rate-gyroscope* buatan Murata, yang digunakan untuk mengukur kecepatan putar. Berikut ini adalah karakteristik dari ENC-03R:

Respon	: 50 Hz
Kecepatan putar maksimum	: +/- 300 deg/sec

Output (ketika kecepatan putar 0 deg/sec) : 1,35 Vdc

Sensitifitas : 0,67 mv/deg/sec

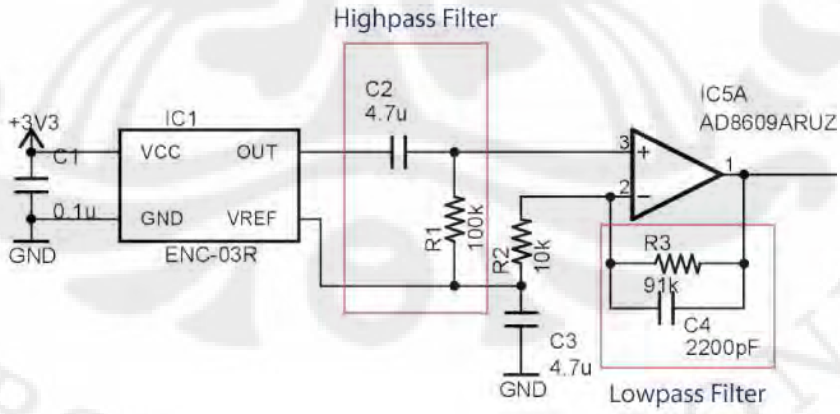


Gambar 3.6 Arah perputaran *rate-gyroscope*

Sumber: Datasheet Murata ENC-03R, telah diolah kembali

Jadi, setiap perputaran dengan kecepatan 1 deg/sec searah jarum jam akan membuat tegangan keluaran bertambah 0,67 mv dan sebaliknya perputaran dengan kecepatan 1 deg/sec berlawanan jarum jam akan membuat tegangan keluaran berkurang 0,67 mV.

Keluaran ENC-03R terhubung dengan rangkaian *low-pass filter* dan *high-pass filter*, seperti terlihat pada gambar 3.7. Adanya *low-pass filter* berfungsi dalam mengurangi *noise*, sedangkan *high-pass filter* berfungsi dalam mengurangi efek *temperature drift* (perubahan tegangan keluaran akibat suhu). Pada modul VS-IX001, frekuensi *cut-off* dari *low-pass filter* yang digunakan adalah ±795 Hz dan untuk *high-pass filter* adalah ±0.3 Hz.



Gambar 3.7 Rangkaian filter pada ENC-03R

3.1.1.3 ADS 7828E

ADS 7828E memiliki resolusi 12 bit, 8 *channel input sampling Analog to Digital Converter* dengan I^2C^{tm} *interface*. ADS 7828E mengubah data analog dari

accelerometer dan *rate-gyroscope* menjadi data digital. Dengan kuantisasi tegangan dalam 12 bit, keluarannya dapat dihitung dengan rumus :

$$ADC = \frac{Vin \times 2^{12}}{Vref} \quad (3.1)$$

Keterangan: ADC adalah keluaran data digital
Vin adalah tegangan input
Vref adalah tegangan referensi.

Tegangan referensi dari ADS7828E bisa didapat dari referensi internal sebesar 2,5 Volt atau menggunakan referensi eksternal. Pada modul VS-IX001 ini digunakan referensi eksternal sebesar 3,3 Volt.

Keluaran dari ADS 7828E menggunakan I^2C *interface* atau disebut juga *two wire*. Dimana prosesnya dengan memilih channel input yang ingin dikonversi, kemudian keluaran akan menggunakan dua pin (SDA dan SCL) untuk mendapatkan data digital hasil pengkonversian. Mengenai I^2C *interface* dan cara pengambilan data dari IC ini dengan I^2C *interface* dijelaskan pada subbab tersendiri.

ADS7828E berhubungan dengan switch 2-1 dan switch 2-2. Pengaturan switch ini akan berpengaruh pada pengalamatan *register* yang digunakan oleh ADS7828E ini untuk menyimpan data pada proses I^2C *interface*.

Tabel 3.2 Pengaturan pengalamatan yang digunakan untuk I^2C *interface* pada ADS7828E

SW 2-1	SW 2-2	Address
OFF	OFF	0x90
OFF	ON	0x92
ON	OFF	0x94
ON	ON	0x96

Sumber : Datasheet VS-IX001, telah diolah kembali

3.1.1.4 2x5 Pin Konektor

Keluaran dan masukkan dari modul VS-IX001 menggunakan 2x5 pin konektor. Fungsi dari masing-masing pin, adalah :

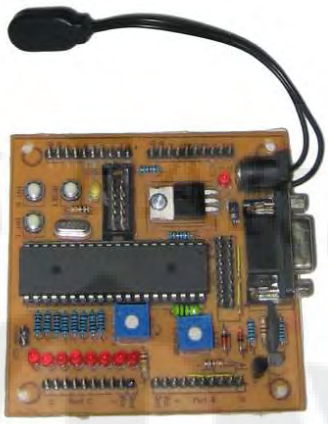
- Pin 5 sebagai SCL
- Pin 6 sebagai SDA
- Pin 8 sebagai Vdc (5 volt)
- Pin 10 sebagai *Ground*
- dan Pin lainnya tidak digunakan

3.1.2 Mikrokontroler AVR ATmega16

Spesifikasi dan kemampuan dari mikrokontroler AVR ATmega16 ini sudah dibahas pada subbab sebelumnya (subbab 2.2). Pemilihan Mikrokontroler AVR ATmega16 ini disebabkan karena harga mikrokontroler ini yang cukup murah, mudah untuk didapatkan di pasaran, handal, dan mempunyai kemampuan untuk berkomunikasi dengan I^2C *interface* dan dengan kabel serial (USART).

Selain itu pada skripsi ini digunakan sistem minimum untuk AVR ATmega16 buatan Prasimax. Sistem minimum ini digunakan agar lebih praktis dalam menyuplai daya, memprogram mikrokontroler dan merangkai jalur komunikasi dari dan ke mikrokontroler. Pada sistem minimum mikrokontroler ini sudah terdapat header untuk kabel serial dan *voltage regulator*. Selain itu, sistem minimum ini juga menyediakan rangkaian *clock* eksternal (kristal) sebesar 8 MHz, switch untuk reset, dan interrupt, yang akan sangat berguna dalam penggunaannya.

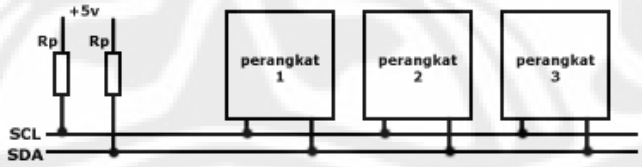
Dalam sistem navigasi inersia di skripsi ini, semua sumber daya perangkat yang digunakan, berasal dari sistem minimum AVR yang terhubung dengan adaptor (AC-DC converter) 9v. Oleh karena adanya *voltage regulator* pada sistem minimum ini, maka tegangan masukan ke mikrokontroler dan tegangan keluarannya adalah 5v.



Gambar 3.8 Sistem minimum AVR ATmega16

3.1.3 Rangkaian Komunikasi I^2C *Interface*

Rangkaian komunikasi I^2C *interface* digunakan untuk jalur komunikasi antara IMU dengan mikrokontroler. Komunikasi I^2C *interface* hanya membutuhkan dua jalur kabel, oleh karena itu kadang juga disebut dengan *two wire*. Dua jalur tersebut adalah SDA (*Serial Data*) dan SCL (*Serial Clock*)**.** SCL merupakan jalur yang digunakan untuk mensinkronisasi transfer data pada jalur I^2C, sedangkan SDA merupakan jalur untuk data. Jalur dari SCL dan SDA ini terhubung dengan *pull-up resistor* yang besar resistansinya tidak menjadi masalah (1K, 1.8K, 4.7K, 10K, 47K atau nilai diantara range tersebut).



Gambar 3.9 Rangkain I^2C

Dengan adanya *pull-up* disini, jalur SCL dan SDA menjadi *open drain*, yang maksudnya adalah perangkat hanya perlu memberikan output 0 (*LOW*) untuk membuat jalur menjadi *LOW*, dan dengan membiarkannya *pull-up resistor* sudah membuatnya *HIGH*.

Umumnya dalam I^2C terdapat satu perangkat yang berperan menjadi *master* (meskipun dimungkinkan beberapa perangkat, dalam jalur I^2C yang sama, menjadi *maste*r) dan satu atau beberapa perangkat *slave*. Tugas perangkat *slave*

hanya merespon apa yang diminta *master*. *Slave* dapat memberi data ke *master* dan menerima data dari *master* setelah *server* melakukan inisialisasi. *Master* dapat terhubung dengan banyak *slave*, dan cara membedakan *slave* tersebut adalah dengan menggunakan pengalamatan, dimana setiap perangkat *slave* itu mempunyai alamat yang unik. Dalam perancangan sistem navigasi inersia ini, ATmega16 akan berperan sebagai *master*, sedangkan modul IMU VS-IX001 sebagai *slave*.

Pada skripsi ini jalur sda adalah dengan menghubungkan pin SDA pada IMU VS-IX001, yaitu pin 6 dengan pin SDA pada mikrokontroler ATmega16, yaitu pin C0. Sedangkan jalur SCL adalah dengan menghubungkan pin 5 pada modul IMU dengan pin C1 pada mikrokontroler ATmega16. Kedua jalur ini terhubung dengan *pull-up* resistor ke Vcc seperti pada gambar 3.1.

Selain itu untuk memberikan daya kepada modul IMU VS-IX001 dibutuhkan tegangan sebesar 5v. Pada skripsi ini digunakan tegangan yang berasal dari sistem minimum AVR ATmega16 yang juga sebesar 5v. Sehingga pin Vcc dan *ground* pada sistem minimum AVR ATmega16 dihubungkan dengan pin Vcc (pin 8) dan pin ground (pin 10) pada IMU.

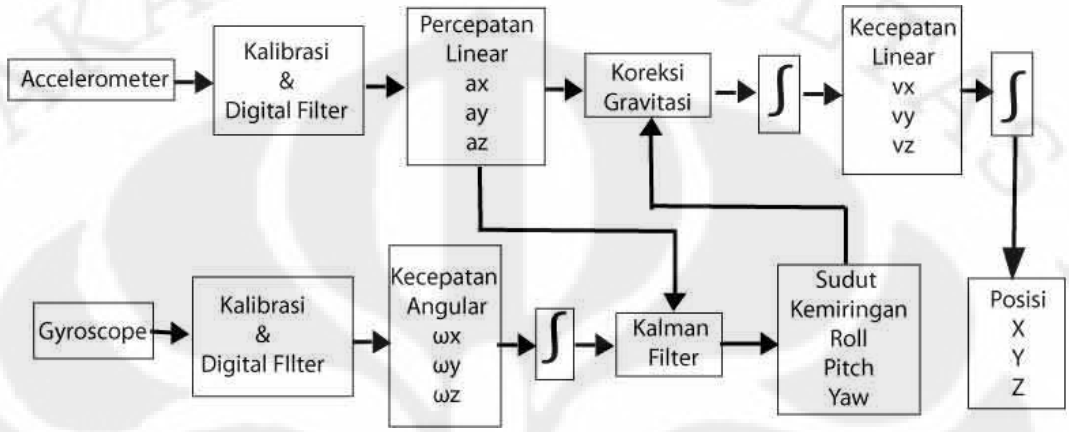
3.1.4 Komunikasi Serial dengan PC

Komunikasi serial dengan PC membutuhkan konektor *male to female* RS-232C DB-9. RS-232C DB-9 memiliki sembilan buah pin yang digunakan untuk komunikasi serial. Pemasangannya cukup dengan menghubungkan bagian ujung kabel *male* ke slot serial yang telah tersedia pada sistem minimum AVR dan bagian *female* ke slot serial pada PC.

Gambar 3.10 Konektor *male to female* RS-232 DB-9

3.2 Perancangan Perangkat Lunak

Arsitektur perancangan dan pembuatan algoritma dari sistem navigasi inersia untuk mendapatkan data posisi dan kemiringan adalah seperti pada gambar 3.11.



Gambar 3.11 Arsitektur algoritma sistem navigasi inersia

Pertama-tama pengambilan data dari *accelerometer* dan *rate-gyroscope* menggunakan algoritma I^2C interface, data yang didapat ini akan diolah lebih lanjut untuk mendapatkan data posisi dan kemiringan yang akurat. Untuk mendapatkan data posisi, pertama-tama data dari *accelerometer* (percepatan linear pada sumbu x, y dan z) diolah terlebih dahulu dengan algoritma kalibrasi dan digital filter, untuk mengeliminasi *noise* dan agar data percepatan yang didapat benar-benar akurat. Kemudian karena *accelerometer* juga menperhitungkan percepatan gravitasi, maka perlu adanya koreksi gravitasi yang membutuhkan data mengenai sudut kemiringan. Setelah koreksi gravitasi, maka data percepatan linear diintegrasikan sehingga didapatkan kecepatan linear. Dengan mengintegrasikan kecepatan linear maka didapatkan data posisi.

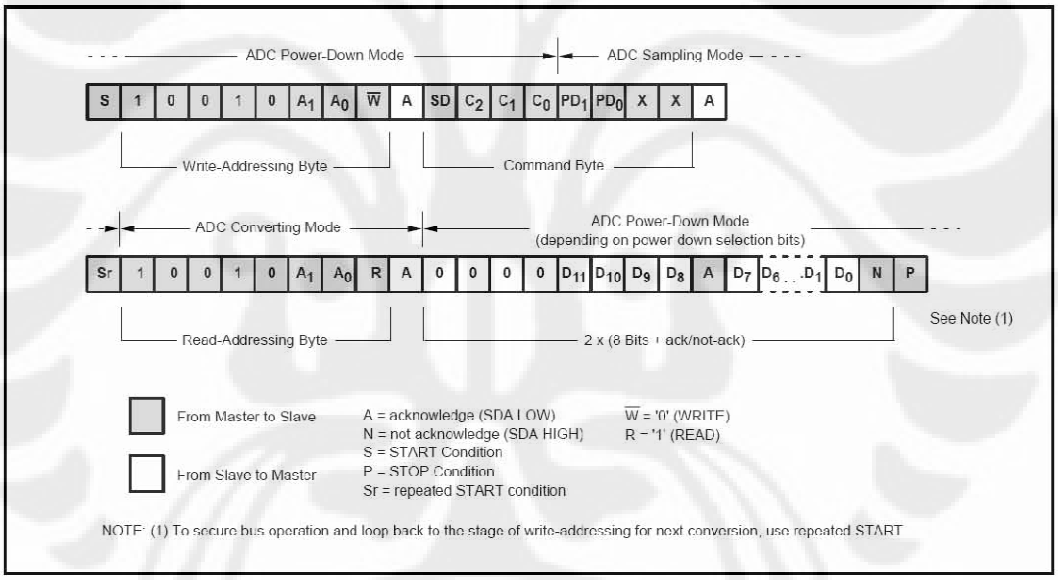
Sedangkan untuk mendapatkan sudut kemiringan, pertama-tama data dari *rate-gyroscope* (kecepatan putar pada sumbu x,y, dan z) juga diolah melalui kalibrasi dan digital filter. Kemudian data kecepatan putar diintegrasikan dan diolah melalui algoritma kalman filter. Kalman filter berfungsi untuk meningkatkan akurasi data dengan memanfaatkan data sudut kemiringan yang didapat dari percepatan *accelerometer* yang juga memperhitungkan percepatan

gravitasi bumi. Keluaran dari kalman filter akan menghasilkan data kemiringan yang lebih akurat.

Pada skripsi ini, karena IMU yang digunakan adalah 3-*axis accelerometer* dan 2-*axis rate-gyroscope*, maka data kemiringan yang didapatkan hanya pada dua sumbu, yaitu *roll* (perputaran pada sumbu x) dan *pitch* (perputaran pada sumbu y). Meskipun hanya dua sumbu putar, akan tetapi data ini cukup untuk menunjukkan kinerja kalman filter dan cukup untuk digunakan sebagai koreksi gravitasi.

3.2.1 Pengambilan Data dari IMU dengan I^2C *Interface*

Untuk mengambil data dari IMU menggunakan I^2C *interface* dengan urutan algoritma I^2C seperti pada gambar 3.12.



Gambar 3.12 Urutan algoritma I^2C untuk mengambil data dari IMU

Sumber : Datasheet ADS7828E

3.2.1.1 *Write-Addressing Byte*

Merupakan *address* ditambah bit *write* (0) yang harus dikirimkan dari mikrokontroler ke *slave*, untuk meminta "izin" melakukan transfer data pada divais yang ditunjuk *address*. Pada modul IMU VS-IX001 *address* yang digunakan tergantung dari switch 1-1 (A_1) dan switch 1-2 (A_2). Pengaturan *switch* ini akan mempengaruhi *address* dari modul, dan dapat dilihat pada tabel 3.2.

3.2.1.2 *Command Byte*

Merupakan *byte* yang perlu dituliskan ke *slave* untuk memilih *channel* yang ingin dikonversikan dari analog ke digital oleh mikrokontroler ADS7828. Dimana untuk percepatan x terhubung dengan channel 0, percepatan y dengan channel 1, percepatan z dengan channel 2, kecepatan putar *roll* dengan channel 4, dan kecepatan putar *pitch* dengan channel 5. Maka *byte* yang dikirimkan untuk mengambil data :

- Percepatan X adalah 0x84
- Percepatan Y adalah 0xC4
- Percepatan Z adalah 0x94
- Percepatan *Roll* adalah 0xA4
- Percepatan *Pitch* adalah 0xE4

3.2.1.3 *Read-Addressing Byte*

Sama seperti *write-addressing byte* akan tetapi bit terakhir merupakan bit read (1) atau dengan kata lain *write-addressing byte* ditambah 1.

3.2.1.4 D_{11}-D_0 (data)

Merupakan data 12 bit hasil *analog to digital converter*. Pada I2C dikirimkan dalam 2 byte, dimana byte pertama berisi 4 bit MSB (0 0 0 0 D_{11} D_{10} D_9 D_8) dan kemudian slave baru mengirimkan 8 bit LSB berikutnya (D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0).

3.2.2 Sistem Kalibrasi

Tujuan utama sistem kalibrasi adalah untuk meningkatkan keakuratan dari data mentah yang didapat dari IMU. Terdapat beberapa jenis kalibrasi yang diterapkan pada sistem navigasi inersia ini.

3.2.2.1 Penentuan Nilai *Offset*

Untuk menentukan nilai *offset* diperlukan kalibrasi *offset*, yang dilakukan ketika kondisi diam (statik). *Offset* perlu didapatkan secara akurat, sebab nilai ini akan digunakan sebagai referensi nilai nol pada data percepatan dan kecepatan

putar. Dimana nilai yang lebih rendah dari *offset* merupakan nilai negatif sementara apabila di atasnya adalah positif. Nilai *offset* ini penting untuk melihat arah percepatan dan arah kecepatan putar.

Gambar 3.13 Penentuan nilai offset

Sumber : Application note MMA7260Q

Seharusnya nilai *offset* dari IMU terdapat pada *datasheet* IMU yang digunakan. Pada skripsi ini, untuk *accelerometer* 1,65 V dan untuk *rate-gyroscope* 1,35 V, yang apabila dikonversikan ke digital dengan rumus 3.1 didapatkan nilai untuk *accelerometer* adalah 2048 dan untuk *rate-gyroscope* adalah 1675. Akan tetapi, pada kenyataannya, nilai titik nol tidak selalu sesuai dengan datasheet. Dimana akan juga dipengaruhi oleh keakuratan sensor, temperatur ruang, *noise*, *error* ADC, dan faktor lainnya.

Metode kalibrasi yang digunakan untuk menentukan nilai *offset*, menggunakan metode yang sederhana, agar tidak membebankan mikrokontroler yang digunakan. Metode tersebut yaitu, dengan mengambil beberapa contoh data (*sample*) sebanyak n dan dirata-ratakan. Pengambilan contoh data dilakukan ketika kondisi IMU dalam keadaan diam dan sejajar dengan permukaan bumi. Hasil rata-rata inilah yang akan digunakan sebagai nilai *offset*. Pada skripsi ini *sample* yang diambil adalah sebanyak 1024 data untuk masing-masing data yang diberikan *accelerometer* dan *rate-gyroscope.*

3.2.2.2 Konversi Data Digital

Setelah penentuan nilai *offset*, maka perlu dikonversikan data dari ADC menjadi data dalam satuan percepatan dan kecepatan putar. Oleh karena pada *accelerometer* digunakan sensitifitas 800 mv/g, maka untuk mendapatkan data dalam satuan *g* digunakan rumus 3.2.

$$percepatan = \frac{(data_{adc}\, accelerometer - \text{nilai offset})}{4096} \times \frac{3{,}3\ V}{0{,}8\ V/g} \qquad (3.2)$$

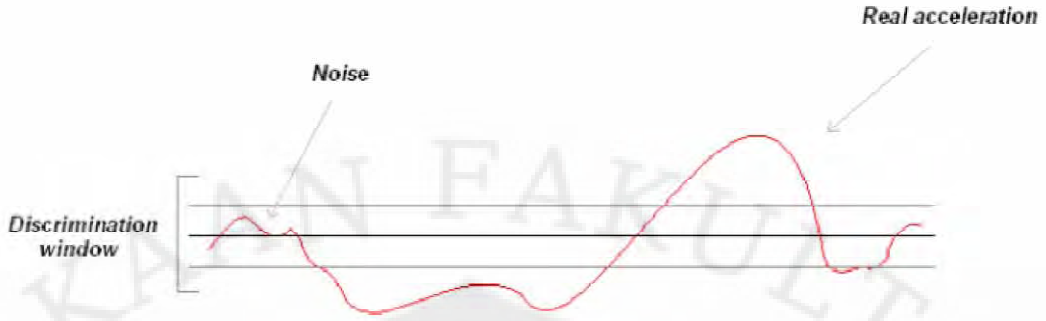
Sedangkan *rate-gyroscope* menggunakan sensitifitas 0.67 mV/deg/s, maka untuk mendapatkan data kecepatan putar dalam satuan *deg/s* digunakan rumus 3.3.

$$kecepatan\ putar = \frac{(data_{adc}\, gyroscope - \text{nilai offset})}{4096} \times \frac{3{,}3\ V}{0{,}00067\ V/deg/s} \qquad (3.3)$$

3.2.2.3 *Mechanical Filtering Window*

Ketika kondisi statik terjadi, kesalahan kecil pada data dari IMU akan mengakibatkan kesalahan yang cukup berarti pada hasil pengintegrasiannya. Misalnya pada kondisi statik terdapat kesalahan kecil pada percepatan sehingga percepatan tidak selalu nol, hal ini dapat diartikan sebagai adanya suatu kecepatan konstan. Sedangkan apabila terjadi kesalahan kecil pada kecepatan putar akan mengakibatkan bergesernya nilai sudut kemiringan, meskipun pada kenyataannya sensor IMU tidak bergerak sama sekali. Kesalahan-kesalahan kecil pada pembacaan sensor ini dapat diakibatkan getaran yang sangat kecil, *mechanical noise*, kesalahan sensor ataupun kesalahan ADC. Idealnya untuk kondisi statik, semua data yang didapatkan baik dari *accelerometer* maupun dari *rate-gyroscope* adalah bernilai nol.

Oleh karena itu untuk mencegahnya perlu dilakukan *filtering window* atau mendiskriminasikan nilai tertentu. Pada skripsi ini *filtering window* hanya diterapkan pada data percepatan dengan nilai diskriminasi antara 0,01 m/s^2 sampai -0,01 m/s^2, sedangkan pada kecepatan putar tidak perlu, sebab untuk mendapatkan data kemiringan dari kecepatan putar sudah menggunakan kalman filter yang dapat mengeliminasi *noise* seperti ini.

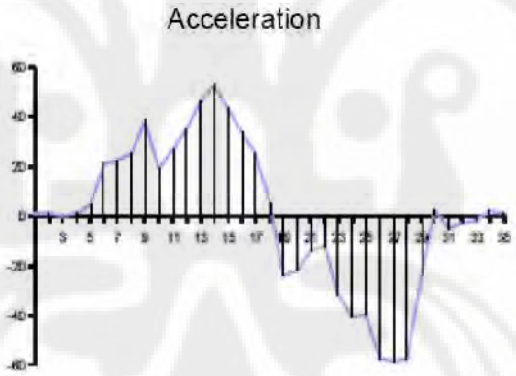


Gambar 3.14 *Discrimination window* untuk mengurangi efek mekanikal *noise*

Sumber : Application note MMA7260Q

3.2.2.4 Pemeriksaan "*Movement End*"

Pemeriksaan "*movement end*" dilakukan hanya pada data keluaran *accelerometer*. Idealnya, dari suatu kondisi IMU tak bergerak, kemudian digerakkan, kemudian kembali ke kondisi diam, akan membentuk kurva akselerasi seperti gambar di bawah ini.



Gambar 3.15 Keluaran *accelerometer* yang ideal

Sumber : Application Note MMA7260Q

Akan tetapi pada kenyataannya, keluaran dari *accelerometer* akan membentuk kurva daerah positif yang tidak sama dengan area negatif. Meskipun perbedaannya kecil, tetapi hal ini akan mengakibatkan hasil integrasi, yaitu kecepatan, tidak akan kembali ke nilai nol. Karena kecepatan akan konstan di nilai tertentu, maka menyebabkan terjadinya perubahan posisi terhadap waktu (tidak pernah stabil), meskipun sensor sudah tidak bergerak lagi.

Untuk mencegah hal ini terjadi, maka pada algoritma untuk mendapatkan data posisi diperlukan "pemaksaan" kecepatan menjadi nol. "Pemaksaan" ini

dilakukan apabila percepatan bernilai nol selama beberapa kali sampling. Pada skripsi ini apabila percepatan bernilai nol selama 20 kali sampling, maka kecepatan akan "dipaksakan" untuk bernilai nol. Metode ini sangat efektif untuk mendapatkan data posisi yang lebih akurat, akan tetapi mempunyai kelemahan apabila IMU bergerak pada kecepatan konstan (percepatan = 0) selama lebih dari 20 kali sampling. Adanya algoritma "pemaksaan" ini akan menyebakan kecepatan dianggap nol, meskipun sebenarnya sensor sedang bergerak dengan kecepatan konstan.

3.2.3 Digital Filter

Sinyal dari analog sensor seperti *rate-gyroscope* setelah diubah ke nilai digital oleh data akusisi mengandung random *noise*, sehingga perlu dilakukan proses eliminasi secara digital. Akan tetapi, karena keterbatasan CPU jenis mikrokontroler 8 bit yang digunakan, maka tidak dapat diimplementasikan algoritma filtering yang memerlukan waktu pemrosesan yang lama.

Digital filter yang paling sederhana adalah dengan metode *moving average*. Algoritma *moving average* adalah dengan mengambil beberapa sample dan merata-ratakan nilainya. Data hasil rata-rata inilah yang merupakan keluaran dari filter tersebut, yaitu data setelah *noise* tereliminasi. Jenis *lowpass filter* seperti *moving average* cukup baik untuk diaplikasikan, karena tidak memerlukan pemrosesan yang rumit. Akan tetapi, mengambil terlalu banyak data sample dapat menyebabkan kehilangan data, sebaliknya jumlah data sample yang terlalu sedikit dapat menyebabkan nilainya menjadi kurang akurat. Selain itu, semakin banyak jumlah yang dirata-rata membuat waktu pemrosesan semakin lambat.

Pada skripsi ini digunakan metode digital filter yang berbeda, yaitu *double exponential*. *Exponential smoothing* biasa digunakan untuk memprediksi dalam dunia statistik, akan tetapi juga dapat digunakan untuk *filtering* sinyal dan dapat memproses secara cepat. Alasan utama penggunaan metode ini adalah tidak memerlukan proses yang rumit dan lebih optimal dibanding *moving average*, karena hanya dengan mengubah parameter eksponensial dapat digunakan untuk *filtering* dalam berbagai respon. Dan juga metode *double exponential* ini juga

dapat dikembangkan sebagai alternatif kalman filter untuk prediksi orientasi dinamik (J. Laviola Jr., 2003).

Dengan menggunakan metode *double exponential*, data sinyal digital $x_{(n)}$ dapat diproses untuk mengeliminasi *noise* secara sederhana dengan persamaan di bawah ini.

$$y_{(n)} = ax_{(n)} + (1-a)y_{(n-1)} \quad (3.4)$$

$$y_{(n)}^{[2]} = ay_{(n)} + (1-a)y_{(n-1)}^{[2]} \quad (3.5)$$

Persamaan 3.4 akan memperbaiki keluaran dari ADC sedangkan persamaan 3.5 akan memperbaiki keluaran dari persamaan 3.4. Dengan hasil *filtering* sinyal adalah $y_{(n)}^{[2]}$ dan parameter eksponensial adalah *a*. Nilai *a* adalah antara nol dan satu ($0 < a < 1$). Bila *noise* terlalu dominan maka nilai parameter *a* yang optimal adalah mendekati satu, jika sebaliknya maka nilai *a* lebih baik dekat ke nol. Selain itu, semakin besar nilai *a*, maka respon akan semakin terlambat. Penentuan parameter ini dapat secara langsung dicari sesuai dengan kondisi sinyal.

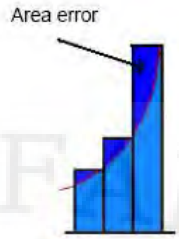
Pada skripsi ini digunakan parameter *a* bernilai 0,2. Penentuan nilai ini berdasarkan pengamatan dari sinyal yang dihasilkan, dimana parameter ini cukup optimal dalam mengeliminasi *noise* dan respon tidak lambat.

3.2.4 Metode Integrasi

Metode integrasi yang umum digunakan misalnya untuk mendapatkan data kecepatan dari percepatan adalah seperti rumus di bawah ini, yang merupakan penurunan dari persamaan 2.1.

$$v_{sekarang} = v_{sebelum} + a\,\Delta t \quad (3.4)$$

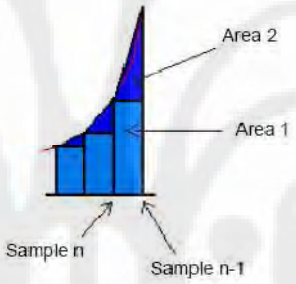
Dengan menggunakan metode ini maka akan menimbulkan *error sampling*. *Error sampling* terjadi seperti diilustrasikan pada gambar di bawah ini.



Gambar 3.16 Error sampling akibat proses integrasi

Sumber : Aplication Note MMA7260Q

Untuk mengurangi kesalahan yang terjadi akibat proses integrasi, maka di skripsi ini dilakukan pendekatan dengan metode integrasi yang berbeda, yaitu metode integrasi trapesium. Metode trapesium mengasumsikan bentuk trapesium dalam setiap pencuplikan pada proses integral, seperti pada gambar 3.17.



Gambar 3.17 Metode integrasi trapesium

Sumber : Application note MMA7260Q

Metode trapesium mengasumsikan luas daerah cuplik terdiri dari dua area, yaitu segitiga dan persegi panjang. Sehingga dapat dihitung luas daerah di bawah kurva (hasil integrasi) dengan menghitung luas segitiga dan persegi panjang tersebut. Sehingga kecepatan dapat diperoleh dari percepatan dengan rumus berikut ini.

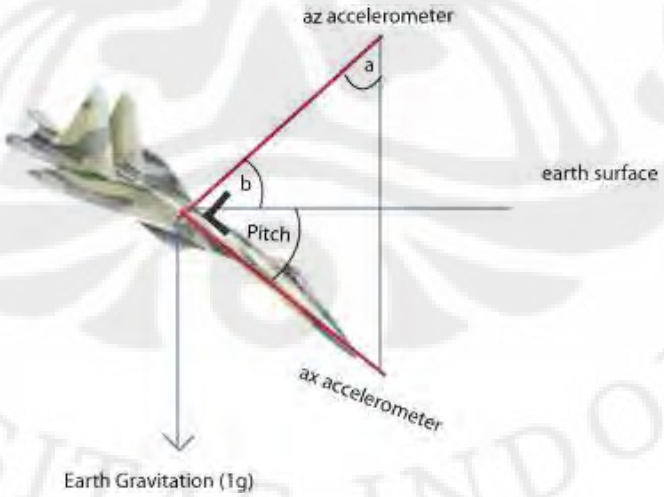
$$v_{sekarang} = v_{sebelum} + a_{sebelum}\Delta t + \frac{(a_{sekarang} - a_{sebelum})\Delta t}{2} \qquad (3.5)$$

Demikian juga proses integrasi untuk mendapatkan posisi dari kecepatan menggunakan metode integrasi trapesium.

3.2.5 Perancangan Kalman Filter dan Parameternya

Kalman filter digunakan untuk mendapatkan sudut kemiringan agar lebih akurat. Pada skripsi ini, kalman filter akan memanfaatkan dua data masukan. Masukan pertama adalah sejenis masukan yang dapat diubah ke prediksi dari keluaran yang diinginkan, hanya dengan menggunakan perhitungan linear. Atau dengan kata lain membutuhkan sebuah model linear dari suatu sistem. Di skripsi ini digunakan data dari *rate-gyroscope*. Masukan kedua merupakan sumber lain yang mempunyai perkiraan atau keakurasian yang lebih baik, dalam skripsi ini digunakan data dari *accelerometer*. Setiap iterasi, kalman filter akan mengubah variabel pada model linear sedikit demi sedikit, sehingga model linear ini akan menyamai data *accelerometer*.

Untuk mendapatkan data kemiringan dari *rate-gyroscope* digunakan persamaan 2.4. Sedangkan untuk mendapatkan data kemiringan dari *accelerometer* didapat dengan memanfaatkan percepatan gravitasi yang juga diperhitungkan oleh sensor ini. Data kemiringan yang bisa didapatkan dari *accelerometer* adalah *pitch* dan *roll*. Dimana sudut *pitch* dapat diperoleh dari percepatan gravitasi yang diukur pada sumbu x dan z dari IMU, sedangkan sudut *roll* dapat diperoleh dari percepatan gravitasi yang diukur pada sumbu y dan z dari IMU.



Gambar 3.18 Ilustrasi mendapatkan sudut dari percepatan *accelerometer*

Dari gambar 3.18 sudut a bisa didapatkan dari $tan^{-1}\left(\frac{ax\ accelerometer}{az\ accelerometer}\right)$, sehingga sudut b = 90° - a, kemudian *Pitch* = 90° - b, sehingga *Pitch* = a. Jadi persamaan pitch dapat dituliskan sebagai berikut ini.

$$Pitch = tan^{-1}\left(\frac{percepatan\ x\ imu}{percepatan\ z\ imu}\right) \qquad (3.6)$$

Demikian juga hal yang sama pada pada sudut *roll*, akan tetapi *roll* dipengaruhi percepatan yang dihitung *accelerometer* pada sumbu y.

$$Roll = tan^{-1}\left(\frac{percepatan\ y\ imu}{percepatan\ z\ imu}\right) \qquad (3.7)$$

Kelemahan utama data kemiringan yang didapat dari *accelerometer* adalah datanya tidak akurat apabila ada pengaruh dari pergerakan IMU, misalnya ketika kondisi dinamik atau terkena goncangan dan keuntungan utamanya adalah datanya sangat akurat (ketika kondisi statik). Sedangkan data kemiringan dari *rate-gyroscope* kelemahan utamanya adalah mudah terkena *noise*, adanya bias akibat suhu dan pertambahan waktu, serta datanya kurang akurat dibandingkan data dari *accelerometer*. Akan tetapi data *rate-gyroscope* akan tetap bagus, meskipun dalam kondisi dinamik. Kalman filter akan menggabungkan keuntungan dari kedua sumber data ini, sehingga data yang dihasilkan lebih handal dan akurat.

Bentuk model data *rate-gyroscope* yang digunakan adalah bentuk umum dari model linear, seperti pada persamaan 2.5, yaitu:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1}$$

$\hat{x}_k^-$ akan dianggap terdiri dari dua variabel state berupa data sudut kemiringan dari *rate-gyroscope* setelah dikurangi bias dan variabel state kedua adalah besar biasnya. Variabel u menunjukkan masukan, dalam kasus ini adalah data *rate-gyroscope*. Untuk mendapatkan sudut kemiringan dari rate-gyroscope setelah dikurangi bias dapat dituliskan seperti persamaan berikut ini:

$$alpha_k = alpha_{k-1} + (u_k - bias) \qquad (3.8)$$

Dan karena *rate-gyroscope* menghitung kecepatan putar (*deg/s*), maka perlu dikalikan dengan *dt* untuk mendapatkan data sudut (*deg*), sehingga persamaan 3.8 menjadi:

$$alpha_k = alpha_{k-1} + (u_k - bias)dt \tag{3.9}$$

$$alpha_k = alpha_{k-1} - (bias \times dt) + u_k dt \tag{3.10}$$

Dengan mengasumsikan bias adalah bernilai konstan maka persamaan umum model linear dari data *rate-gyroscope* adalah:

$$\begin{pmatrix} alpha \\ bias \end{pmatrix}_k = \begin{pmatrix} 1 & -dt \\ 0 & 1 \end{pmatrix} \begin{pmatrix} alpha \\ bias \end{pmatrix}_{k-1} + \begin{pmatrix} dt \\ 0 \end{pmatrix} u_k \tag{3.11}$$

Persamaan 3.11 ini yang digunakan sebagai *predicted state* pada kalman filter dan meskipun bias dianggap konstan pada pemodelan, kalman filter akan menyesuaikan biasnya pada setiap iterasi dengan membandingkan hasil dengan keluaran *accelerometer* (masukan kedua).

Z_k pada persamaan 2.7 merupakan data kemiringan yang didapatkan dari *accelerometer*. Sedangkan matrix *H*, yaitu matrix untuk mengekstraksi keluaran dari state matrix pemodelan keluaran *rate-gyroscope*.

$$alpha = H \times \hat{x}_k \tag{3.12}$$

Sehingga,

$$H = \begin{pmatrix} 1 & 0 \end{pmatrix} \tag{3.13}$$

Jadi *innovation* menghitung selisih atau perbedaan antara sudut kemiringan dari pemodelan keluaran *rate-gyroscope* dengan sudut kemiringan keluaran dari accelerometer.

Selain itu, pada skripsi ini digunakan kondisi inisial:

$$x_0 = \begin{pmatrix} 0 & 0 \end{pmatrix} \tag{3.14}$$

$$P_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (3.15)$$

Penentuan kondisi inisial ini tidak begitu penting, asalkan nilai P_0 tidak sama dengan nol, sebab kalman filter akan menyesuaikan dengan sendirinya, seperti yang telah disebutkan sebelumnya pada subbab 2.3. Sedangkan x_0 berdasarkan kondisi awal dimana IMU statik, sehingga diasumsikan adalah bernilai nol. Semakin x_0 mendekati kebenarannya, maka semakin cepat kalman filter akan menyesuaikan.

Kesulitan utama dalam membangun kalman filter adalah menentukan parameter *process noise covariance* Q dan *measurement noise covariance* R. Matrix Q pada persamaan 2.6 yang menunjukkan *process noise covariance*, mengadopsi parameter yang umum digunakan pada perangkat lunak *autopilot*, yaitu :

$$Q = \begin{pmatrix} 0{,}003 & 0 \\ 0 & 0{,}001 \end{pmatrix} \quad (3.16)$$

Sedangkan dalam *measurement noise covariance* R berdasarkan Welch dan Bishop (2006) sebaiknya memperhatikan sensor yang digunakan, sebab semakin nilai R mendekati nol, maka pengukuran aktual Z_k (dalam kasus ini adalah data kemiringan dari *accelerometer*) semakin "dipercaya" kebenarannya, sementara pengukuran prediksi $H\hat{x}_k$ (dalam kasus ini adalah data kemiringan dari *rate-gyroscope*) semakin "tidak dipercaya" kebenarannya oleh filter. Pada skripsi ini nilai R akan divariasikan untuk mendapatkan nilai yang optimal.

Algoritma kalman filter ditunjukkan pada gambar 2.4 dengan parameter dan model yang telah dijelaskan pada subbab ini.

3.2.6 Koreksi Gravitasi

Koreksi gravitasi perlu dilakukan untuk mendapatkan posisi, oleh karena *accelerometer* juga memperhitungkan percepatan gravitasi. Koreksi gravitasi ini disebut juga eliminasi percepatan gravitasi pada data percepatan *accelerometer*, dengan menggunakan data sudut kemiringan. Untuk dapat melakukan koreksi

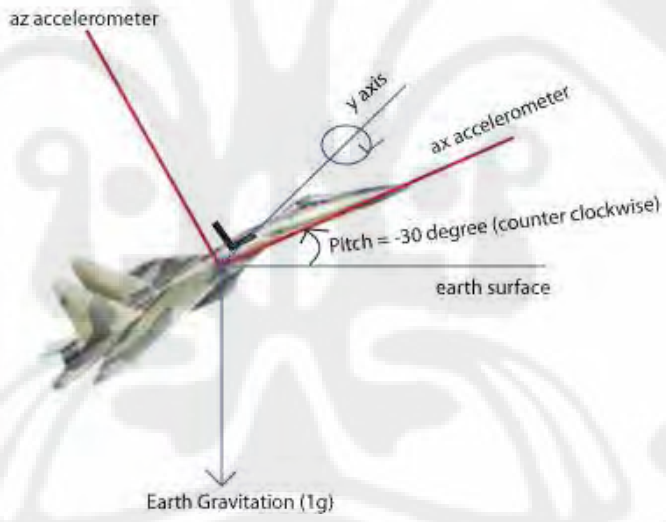
gravitasi dengan baik pada sistem navigasi inersia, maka dibutuhkan data sudut kemiringan yang sangat akurat, tanpa data yang akurat maka koreksi gravitasi tidak dapat dilakukan. Berikut ini adalah persamaan yang digunakan untuk koreksi gravitasi:

$$a_{x_{true}} = a_{x_{accelerometer}} - g\,(\sin(pitch)) \quad (3.17)$$

$$a_{y_{true}} = a_{y_{accelerometer}} + g\,(\sin(roll)) \quad (3.18)$$

$$a_{z_{true}} = a_{z_{accelerometer}} + g\sqrt{(1 - sin^2(pitch) - sin^2(roll)} \quad (3.19)$$

Dengan nilai $a_{x_{accelerometer}}$ dan $a_{y_{accelerometer}}$ adalah sama dengan nol dan $a_{z_{accelerometer}}$ bernilai - 1g ketika IMU sejajar dengan permukaan bumi. Untuk lebih jelasnya berikut ini adalah ilustrasinya.



Gambar 3.19 Ilustrasi koreksi gravitasi

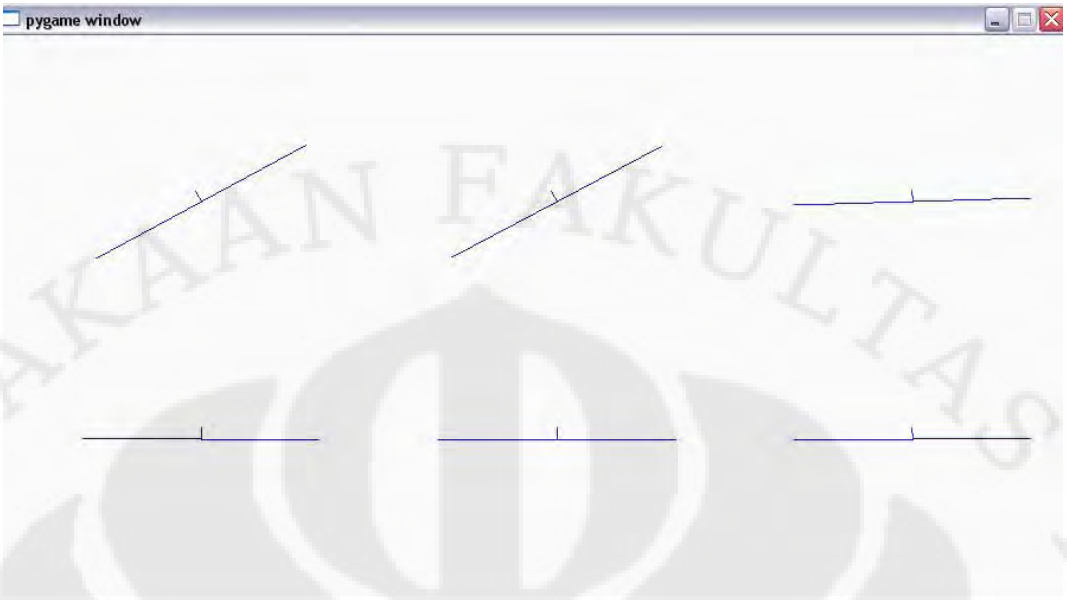
Misalnya IMU berada pada kondisi dengan sudut pitch sebesar 30° dan sudut roll 0°, maka karena pengaruh dari percepatan gravitasi sebesar 1 g, *accelerometer* juga akan memperhitungkan percepatan gravitasi pada sumbu x, yaitu sebesar -0,5 g, sedangkan pada sumbu z sebesar -0,866g, dan pada sumbu y sebesar 0g. Dengan menggunakan rumus 3.17, 3.18, dan 3.19 maka akan didapatkan percepatan yang telah dieliminasi percepatan gravitasi, sehingga pada keadaan

statik nilai percepatan sumbu x, y dan z akan selalu bernilai nol, berapapun sudut kemiringannya.

3.2.7 Perangkat Lunak *Phyton*

Perangkat lunak *phyton* pada PC digunakan untuk memperlihatkan gambaran visual mengenai kemiringan IMU. Untuk dapat melakukan hal ini *phyton* memerlukan program tambahan, yaitu perangkat lunak *pygame*. Gambaran visual ini juga mempermudah dalam mengamati perbedaan data kemiringan antara keluaran *rate-gyroscope, accelerometer* dengan kalman filter. Selain itu, *phyton* juga digunakan untuk mempermudah dalam melakukan pengubahan parameter pada kalman filter dan memperlihatkan secara visual pengaruh parameter tersebut. Pengubahan parameter hanya perlu dilakukan pada PC, tidak perlu memprogram ulang mikrokontroler. Algoritma pemrograman untuk menunjukkan gambaran visual kemiringan ini berbeda dengan algoritma sebelumnya. Pada algoritma sebelumnya semua pengolahan data dilakukan pada mikrokontroler sedangkan PC hanya untuk menampilkan data, sedangkan pada algoritma dengan *phyton*, sebagian pengolahan data juga dilakukan pada PC. Perbedaan algoritma dengan dan tanpa *phyton* dapat dilihat pada lampiran 1 dan 2. Algoritma dengan menggunakan *phyton* dapat dilihat pada lampiran 1. Sedangkan algoritma pemrograman tanpa menggunakan phyton dapat dilihat pada lampiran 2.

Gambar 3.18 menunjukkan gambar tampilan *phyton*. Gambar baris pertama menunjukkan tampilan IMU pada sumbu x dan baris kedua menunjukkan tampilan IMU pada sumbu y. Sedangkan kolom pertama adalah keluaran dari kalman filter, kolom kedua adalah keluaran dari *accelerometer* dan kolom ketiga adalah keluaran dari *rate-gyroscope*.

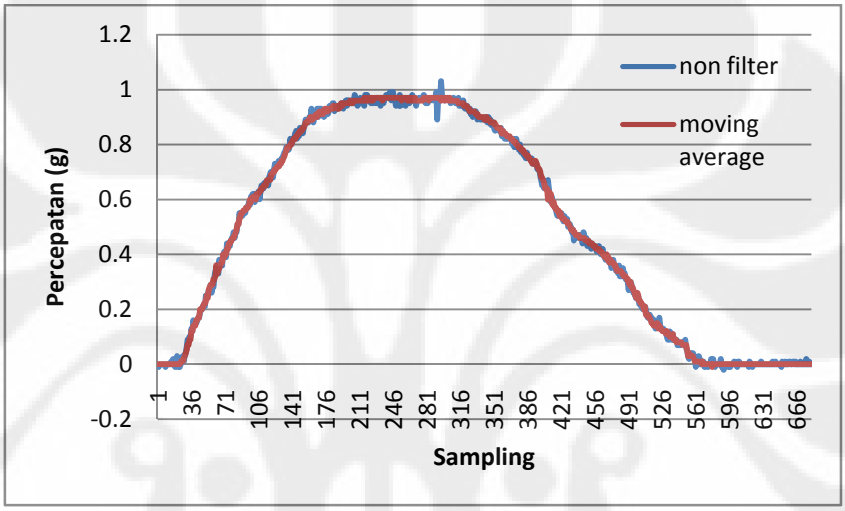


Gambar 3.19 Tampilan visual pada *phyton*

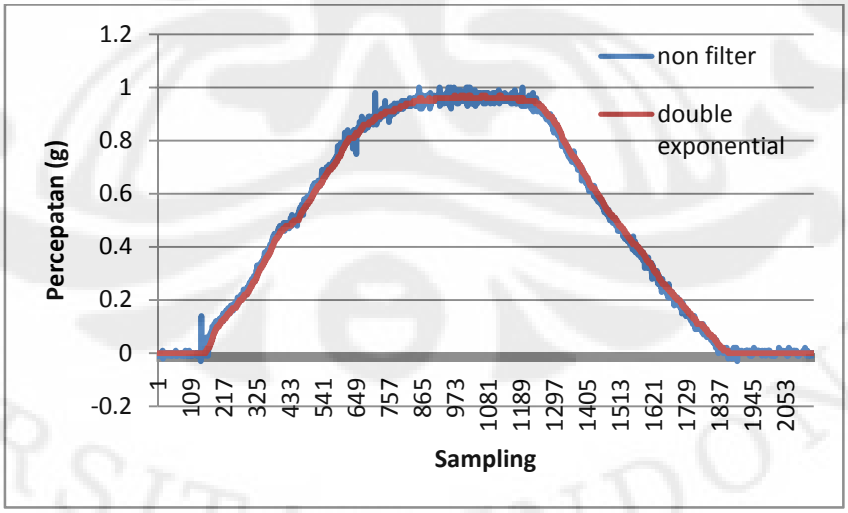
BAB 4
PENGUJIAN DAN ANALISA

4.1 Pengujian Digital Filter

Pengujian digital filter dilakukan untuk melihat kemampuan *double exponential* dalam mengeliminasi *noise* dibandingkan metode yang umum digunakan, yaitu metode *moving average*.



Gambar 4.1 Grafik perbandingan *moving average* dan tanpa filter



Gambar 4.2 Grafik perbandingan *double exponential* filter dan tanpa filter

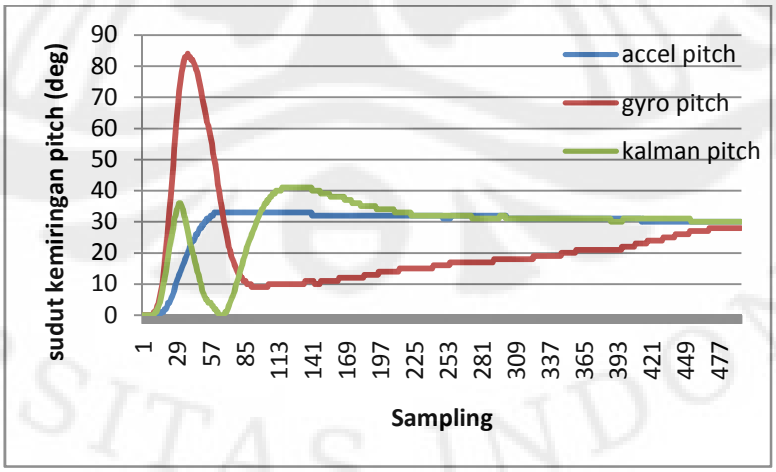
Gambar 4.1 menunjukkan perbandingan data percepatan pada sumbu x keluaran dari *accelerometer* menggunakan filter *moving average* (dengan 64

sampling) dan dengan tanpa *digital filter*. Gambar 4.2 menunjukkan perbandingan data percepatan pada sumbu x keluaran dari *accelerometer* menggunakan filter *double exponential* (dengan parameter *a* = 0,2) dan dengan tanpa *digital filter*. Dari kedua grafik di atas menunjukkan bahwa metode *moving average* dan *double exponential* sama-sama mampu mengeliminasi *noise* dengan baik. Akan tetapi untuk menjalankan algoritma pemrograman dari *moving average* membutuhkan waktu lebih lama ±3,84 ms (tidak terlihat dari grafik) dibandingkan metode *double exponential*. Pada skripsi ini digital filter akan digunakan untuk mengolah lima data (percepatan sumbu x, y, z, kecepatan putar *roll* dan *pitch*), sehingga perbedaan waktu dalam mengolah data dengan *moving average* akan lebih besar ±19,2 ms, sedangkan data akan semakin akurat apabila waktu sampling lebih cepat. Oleh karena itulah pada skripsi ini digunakan metode *double exponential*.

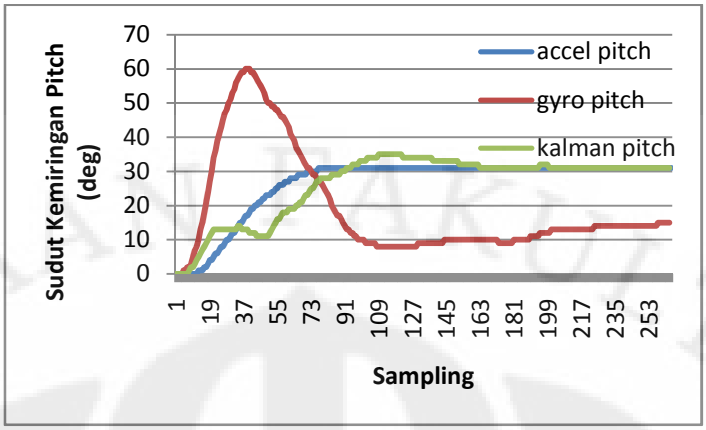
4.2 Pengujian Kalman Filter

4.2.1 Pengujian Parameter *Measurement Noise Coariance*

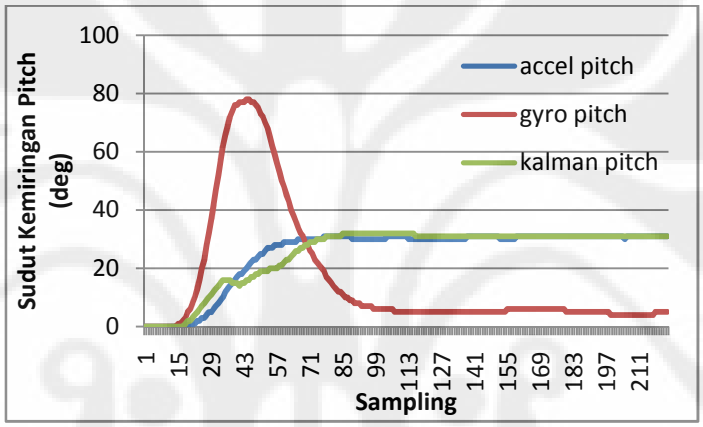
Pengujian pertama adalah menguji variasi parameter *measurement noise covariance* terhadap keluarannya. Parameter R akan divariasikan dengan nilai berbeda, yaitu 3, 0,3 dan 0,03. Pengujian dilakukan pada data sudut pitch, dengan periode sampling tiap 32 ms, dan dilakukan pergerakkan kurang lebih 30° ke arah positif (searah jarum jam). Berikut ini adalah grafik data hasil pengujian.



Gambar 4.3 Grafik perbandingan keluaran kalman filter dengan R = 3



Gambar 4.4 Grafik perbandingan keluaran kalman filter dengan R =0, 3



Gambar 4.5 Grafik perbandingan keluaran kalman filter dengan R =0, 03

Grafik di atas menunjukkan kurva data sudut kemiringan pitch keluaran dari *accelerometer* (kurva biru), *rate-gyroscope* (kurva merah) dan kalman filtter (kurva hijau). Tiap grafik menggunakan parameter *measurement noise covariance* R yang berbeda.

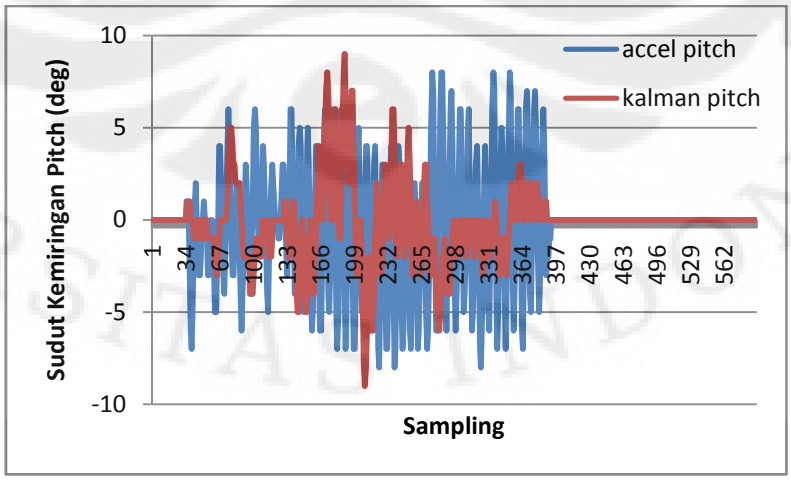
Dari ketiga grafik tersebut, terlihat bahwa semakin kecil nilai R, maka keluaran dari kalman filter akan semakin mendekati keluaran dari *accelerometer*. Dimana parameter R semakin mendekati nilai nol, maka kalman filter akan lebih cepat "percaya" dengan data *accelerometer*, dan semakin menjauhi nilai nol, maka kalman filter lebih lambat "percaya" dengan data *accelerometer*. Hal ini sesuai dengan teorinya, yang telah dibahas pada subbab 3.2.5.

Nilai parameter yang paling optimal dan yang digunakan dari ketiga nilai R tersebut adalah 0,3. Hal ini karena apabila nilai R terlalu kecil akan menyebabkan keluaran kalman filter semakin menyerupai keluaran sudut

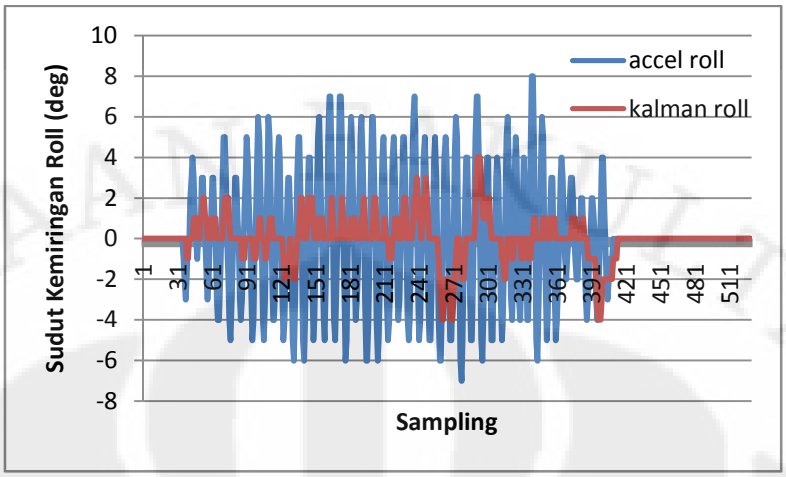
kemiringan dari *accelerometer*, yang akan bermasalah ketika adanya goncangan atau pergerakan (*accelerometer* memperhitungkan percepatan linear). Sedangkan apabila nilai R terlalu besar, maka menyebabkan data menjadi kurang akurat, oleh karena data sudut kemiringan yang diberikan oleh *rate-gyroscope* kurang akurat, rentan terhadap *noise*, adanya bias yang bertambah seiring dengan bertambahnya waktu, atau perubahan suhu. Pada gambar 4.3 terlihat keluaran kalman filter menjadi kurang akurat. Selain itu nilai parameter R 0,3 adalah nilai yang umum digunakan pada *autopilot* dan sistem navigasi inersia untuk mendapatkan data kemiringan dengan menggunakan kalman filter. Sebagai tambahan juga dilakukan pengujian dengan nilai parameter R yang dekat dengan 0,3 yaitu 0,2 dan 0,1. Hasil pengujian ini menunjukkan keluaran kalman filter tidak berbeda jauh dengan ketika parameter R 0,3.

4.2.2 Pengujian Keluaran Kalman Filter Ketika Pergerakan

Pengujian kedua adalah menguji keluaran kalman filter apabila ada pergerakan pada sumbu x dan y (goncangan) pada sudut kemiringan konstan, dimana seharusnya sudut kemiringan tidak berubah nilainya. Gambar 4.6 menunjukkan grafik sudut kemiringan *pitch* keluaran kalman filter dan keluaran *accelerometer* apabila diberikan goncangan pada sumbu x. Sedangkan gambar 4.7 menunjukkan grafik sudut kemiringan *roll* keluaran kalman filter dan keluaran *accelerometer* apabila diberikan goncangan pada sumbu y. Pengujian ini dilakukan ketika sudut kemiringan adalah konstan 0° dan menggunakan parameter R pada kalman filter sebesar 0,3 dengan periode sampling 32 ms.



Gambar 4.6 Grafik keluaran *pitch* kalman filter ketika diberi goncangan



Gambar 4.7 Grafik keluaran *roll* kalman filter ketika diberi goncangan

Dari kedua grafik tersebut tampak bahwa sudut kemiringan yang didapat dari *accelerometer* (kurva biru) sangat dipengaruhi oleh pergerakan atau goncangan, yaitu *pitch* dipengaruhi oleh pergerakan pada sumbu x dan *roll* dipengaruhi oleh pergerakkan pada sumbu y. Hal ini terjadi karena *accelerometer* juga memperhitungkan percepatan linear. Sedangkan keluaran dari kalman filter (kurva merah) menunjukkan kurva dengan amplitudo lebih kecil dibandingkan keluaran *accelerometer*. Hal ini membuktikan bahwa kalman filter mampu meredam pengaruh goncangan, meskipun kalman filter juga menggunakan data dari *accelerometer*.

4.2.3 Pengujian Keakuratan Keluaran Kalman Filter

Pengujian dilakukan sebanyak 10 kali setiap sudut dengan menggunakan nilai parameter R kalman filter 0,3 dan periode sampling 32 ms. Hasil rata-rata pengujian dapat dilihat pada tabel 4.1. Dari hasil percobaan menunjukkan data kemiringan yang didapat dari pengolahan kalman filter hampir mendekati 100% kebenarannya. Data kemiringan ini didapatkan ketika keluaran dari kalman filter sudah mencapai *steady state*. Perbedaan keluaran kalman filter dengan keadaan yang sebenarnya hanya sekitar ±1°. Sehingga terbukti bahwa sistem navigasi inersia yang dibangun untuk mendapatkan data sudut kemiringan sudah cukup akurat.

Tabel 4.1 Hasil pengujian keakuratan keluaran kalman filter

Besar Sudut	Pitch	Roll
Aktual	**Keluaran Kalman Filter**	**Keluaran Kalman Filter**
-120°	-120,0°	-120,0°
-90°	-90,0°	-90,2°
-60°	-59,8°	-60,0°
-30°	-30,0°	-30,0°
0°	0°	0°
30°	29,4°	30,2°
60°	60,0°	60,0°
90°	90,0°	89,7°
120°	119,9°	120,0°

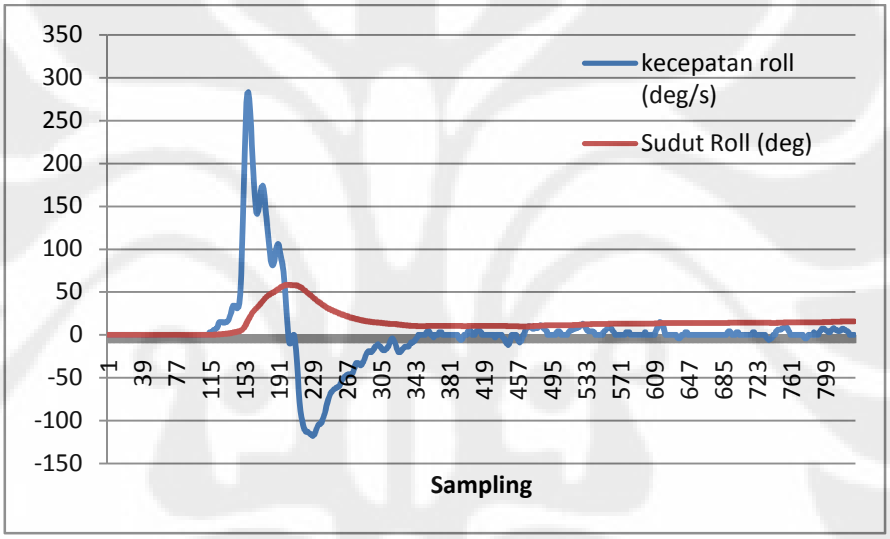
4.3 Pengujian Koreksi Gravitasi

Pengujian koreksi gravitasi dilakukan untuk melihat performa keluaran sudut kemiringan dari kalman filter untuk mengeliminasi percepatan gravitasi. Pengujian ini dilakukan dengan parameter R kalman filter 0,3 dan periode sampling 32 ms. Pengujian dilakukan pada posisi konstan dengan sudut kemiringan yang berbeda-beda. Pada kondisi ini keluaran dari *accelerometer* akan memperhitungkan percepatan gravitasi. Dengan koreksi gravitasi diharapkan percepatan tidak akan terpengaruh oleh percepatan gravitasi.

Hasil pengujian ketika dilakukan pengubahan pitch secara perlahan, menunjukkan koreksi gravitasi telah bekerja dengan baik, yaitu nilai keluaran percepatan pada sumbu x dan z tetap nol. Dimana seharusnya perubahan pitch akan mempengaruhi percepatan keluaran pada *accelerometer* sumbu x dan z. Dengan adanya koreksi gravitasi, meskipun terjadi perubahan *pitch*, percepatan keluaran pada sumbu x dan z akan tetap konstan di nol. Demikian pengujian hasil koreksi gravitasi dengan perubahan *roll*. Dimana seharusnya roll akan mempengaruhi percepatan keluaran *accelerometer* sumbu y dan z. Tetapi dengan adanya koreksi gravitasi, meskipun terjadi perubahan sudut *roll* percepatan akan tetap konstan di nol. Hal ini membuktikan koreksi gravitasi sudah sesuai.

Akan tetapi kondisi ini akan terjadi apabila perubahan sudut kemiringan berjalan lambat. Apabila perubahan sudut kemiringan yang terjadi cepat, nilai dari percepatan hasil koreksi gravitasi membutuhkan waktu untuk menyesuaikan hingga bernilai nol. Hal ini disebabkan karena respon kalman filter yang juga dipengaruhi oleh keluaran *rate-gyroscope.* Seperti yang dapat dilihat dari gambar 4.3, 4.4 dan 4.5, keluaran data kemiringan dari *rate-gyroscope* yang digunakan sangat tidak akurat.

4.4 Ketidakakuratan Keluaran *Rate-Gyroscope*

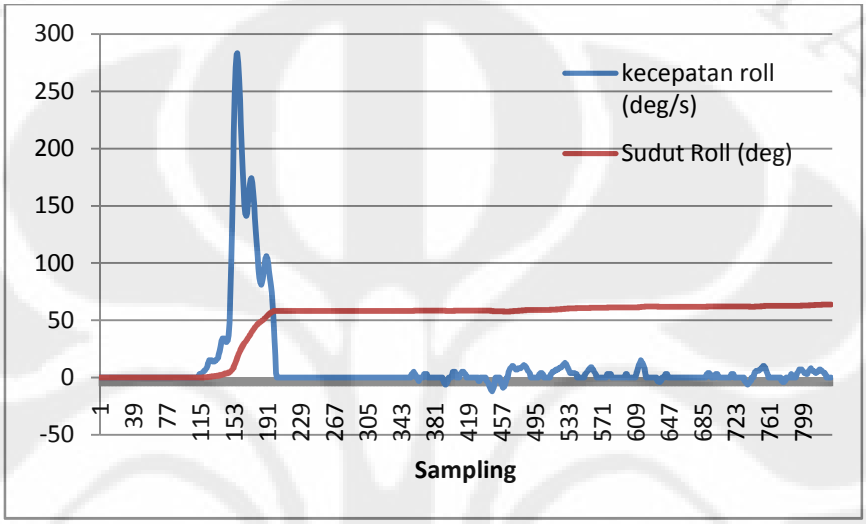


Gambar 4.8 Grafik kecepatan putar *roll* keluaran *rate-gyroscope* yang digunakan

Gambar 4.8 merupakan contoh keluaran dari *rate-gyroscope* yang digunakan pada skripsi ini. Dimana grafik tersebut merupakan kecepatan putar *roll* keluaran *rate-gyroscope* ketika dilakukan peputaran dan berhenti. Sedangkan data yang ideal adalah berbentuk seperti grafik pada gambar 4.9.

Dari gambar 4.8 terlihat adanya "*bouncing*" ke arah berlawanan dari arah putaran, kecenderungan juga terjadi pada *pitch* baik pada putaran ke arah positif maupun ke arah negatif. Akibat adanya "*bouncing*" ini akan mengubah sudut kemiringan. Penyebab dari adanya "*bouncing*" ini masih tidak diketahui. Ketika dilakukan pengujian dengan sensor lain dengan tipe yang sama, juga menunjukkan hal yang sama, meskipun terkadang didapatkan bentuk kurva kecepatan putar yang ideal ketika perputaran dilakukan dengan lambat dan dengan

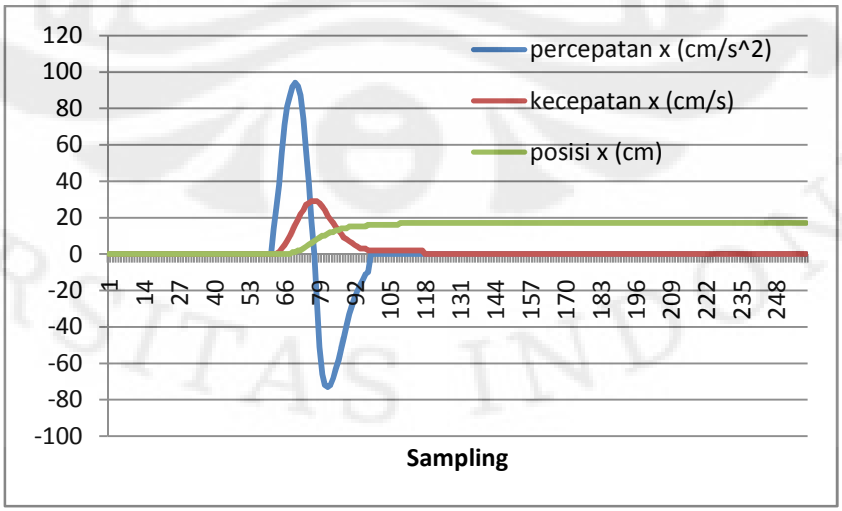
sudut yang sangat kecil. Karenanya dapat disimpulkan bahwa keluaran seperti itu memang karakteristik dari sensor *rate-gyroscope* enc-03R. Inilah penyebab utama data sudut kemiringan yang didapatkan dari *rate-gyroscope* sangat tidak akurat kebenarannya, oleh karena itulah dibutuhkan kalman filter untuk mengoreksi data tersebut.



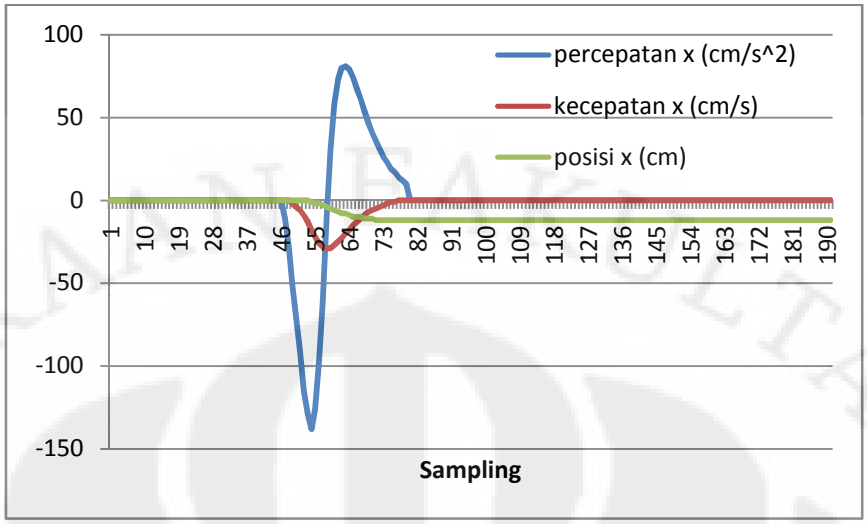
Gambar 4.9 Grafik keluaran *rate-gyroscope* yang ideal

4.5 Pengujian Data Posisi Keluaran Accelerometer

Berikut ini adalah contoh grafik percepatan, kecepatan dan posisi keluaran dari *accelerometer* ketika dilakukan perubahan posisi ke arah negatif dan positif pada sumbu x, dengan periode sampling 32 ms.



Gambar 4.10 Grafik keluaran *accelerometer* dengan perubahan posisi pada sumbu x positif



Gambar 4.11 Grafik keluaran *accelerometer* dengan perubahan posisi pada sumbu x negatif

Dari kedua grafik di atas, *accelerometer* menghasilkan keluaran berbentuk kurva percepatan, kecepatan dan posisi yang ideal. Demikian juga keluaran ketika dilakukan perubahan posisi pada sumbu y dan sumbu z, juga menunjukkan kurva yang mirip. Hal ini menunjukkan bahwa sistem kalibrasi telah bekerja dengan baik.

Selain itu dilakukan juga pengujian keakuratan data posisi dengan koreksi gravitasi dan tanpa koreksi gravitasi. Dengan koreksi gravitasi, data posisi yang didapatkan masih tidak akurat, hal ini terutama disebabkan karena pengaruh keluaran sudut kemiringan *rate-gyroscope* yang tidak akurat, sehingga menyebabkan keluaran kalman filter kurang optimal dalam mengeliminasi percepatan gravitasi.

Sedangkan data posisi tanpa menggunakan koreksi gravitasi didapatkan data yang cukup akurat. Dengan melakukan pengujian sebanyak 100 kali terhadap pengubahan posisi ke sumbu x positif, x negatif, y positif dan y negatif sebesar 20 cm. Nilai rata-rata dari pengubahan posisi ke sumbu x positif adalah 22,60 cm dengan standar deviasi ±7,60 cm, sedangkan ke sumbu x negatif adalah -20,64 cm dengan standar deviasi ±7,99 cm. Dan pengubahan posisi ke sumbu y positif didapatkan nilai rata-rata 20,56 cm dengan standar deviasi ±9,29 cm, sedangkan pengubahan posisi ke sumbu y negatif didapatkan nilai rata-rata -20,39 cm dengan standar deviasi ±8,59 cm.

Hasil pengujian ini menunjukkan rata-rata yang mendekati dengan nilai sebenarnya yaitu 20 cm. Hal ini juga membuktikan bahwa sistem kalibrasi yang dilakukan telah bekerja dengan baik. Meskipun rata-rata yang didapatkan mendekati nilai sebenarnya, nilai standar deviasinya cukup besar. Hal ini menunjukkan bahwa data yang didapat tidak konsisten (kadang-kadang mendekati kebenarannya). Meskipun data yang didapatkan tidak konsisten, hasil pengujian data posisi tanpa koreksi gravitasi menunjukkan arah pergerakkan sudah sesuai dengan gerakan yang dilakukan, bahkan dengan rata-rata mendekati nilai kebenaran. Penyebab utama dari ketidakakuratan ini adalah karena modul VS-IX001 memang didesain untuk mendapatkan data sudut kemiringan, dan tidak didesain untuk pelacakan jejak (mendapatkan data posisi). Selain itu dapat juga disebabkan oleh karena hal lain, yaitu kecepatan sampling yang terlalu lambat (semakin cepat kecepatan sampling semakin akurat data yang didapatkan) ataupun pengaruh goncangan dan *noise*.

BAB 5
KESIMPULAN

Dari keseluruhan pembahasan dalam skripsi ini dapat disimpulkan beberapa hal, yaitu :

1. Komponen dasar dari sistem navigasi inersia adalah *Inertial Measurement Unit* (IMU). Biasanya IMU merupakan rangkaian yang terdiri dari 3 buah *accelerometers* dan 3 buah *rate-gyroscope*.
2. Untuk mendapatkan data posisi, keluaran dari *accelerometer* perlu melewati sistem kalibrasi dan digital filter, serta koreksi gravitasi yang membutuhkan data sudut kemiringan, kemudian dilakukan pengintegrasian sebanyak dua kali.
3. Sudut kemiringan yang akurat bisa didapatkan dengan menggunakan kalman filter yang memanfaatkan dua masukkan, yaitu hasil pengintegrasian keluaran *rate-gyroscope* dan sudut kemiringan dari *accelerometer*. Dimana kalman filter akan mengambil kelebihan masing-masing data dari sumber yang berbeda tersebut.
4. Hasil pengujian menunjukkan metode *double exponential* filter lebih handal dalam mengeliminasi *noise* dibandingkan metode *moving average* karena pengolahan algoritmanya yang lebih cepat ± 3,84 ms.
5. Semakin mendekati nol parameter *measurement noise covariance* R pada kalman filter maka kalman filter akan lebih cepat "percaya" dengan keluaran *accelerometer,* dan lebih "tidak percaya" pada keluaran *rate-gyroscope*.
6. Kalman filter mampu meredam perubahan sudut yang terjadi akibat goncangan.
7. Hasil pengujian keakuratan data kemiringan dengan menggunakan kalman filter mendekati 100%.

DAFTAR REFERENSI

Gedex (2008, Mei 17). *Menggunakan jalur I^2C*. Desember 25, 2008. http://gedex.web.id/archives/2008/05/17/menggunakan-jalur-i2c/

J. Laviola Jr., Joseph (2003). *Double Exponential Smoothing: An Alternative to Kalman Filter-Based Predictive Tracking*. Providence: Brown University Technology Center for Advanced Scientific Computing and Visualization.

Kadir, Abdul (2001). *Pemograman C++*. Yogyakarata: Andi Yogyakarta.

Kumar N., Vikas (2004, Juli). *Integration of Inertial Navigation System and Global Positioning System Using Kalman Filtering.* Mumbai: M.tech dissertation in Department of Aerospace Engineering Indian Institute of Technology Bombay.

Lobo, Jorge, et al. *Inertial Navigation System for Mobiles Land Vehicle*. Coimbra, Portugal: Instituto de Sistemas e Robótica, Departamento de Engenharia Electrotécnica, Universidade de Coimbra.

National Instrument. (2008, Desember 10). *Inertial Measurement Unit*. Desember 22, 2008. http://zone.ni.com/devzone/cda/tut/p/id/8163

NONGNU. *Example Using the two-wire interface (TWI)*. Desember 2, 2008. http://www.nongnu.org/avr-libc/user-manual/group__twi__demo.html.

Özgür Kivrak, Arda (2006, Desember). *Design of Control Systems for a Quadrotor Flight Vehicle Equipped with Inertial Sensor*. Mastes's Thesis in Mechatronics Engineering Atılım University.

Pycke, Tom (2006, 22 Mei). *Kalman Filtering of IMU data*. Maret 20, 2009. http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data

Procyon AVRlib. *I2C.c*. November 25, 2008. http://www.mil.ufl.edu/~chrisarnold/components/microcontrollerBoard/AVR/avrlib/docs/html/i2c_8c-source.html

Rönnbäck, Sven (2000, Februari). *Development of a INS/GPS Navigation Loop.* Mastes's Thesis in Computer Science and Electrical Engineering Luleâ University of Technology.

Seifert, Kurt & Camacho, Oscar (2007, Februari). *Implementing Position Algorithm Using Accelerometers.* Freescale Semiconductor Application Note AN3397.

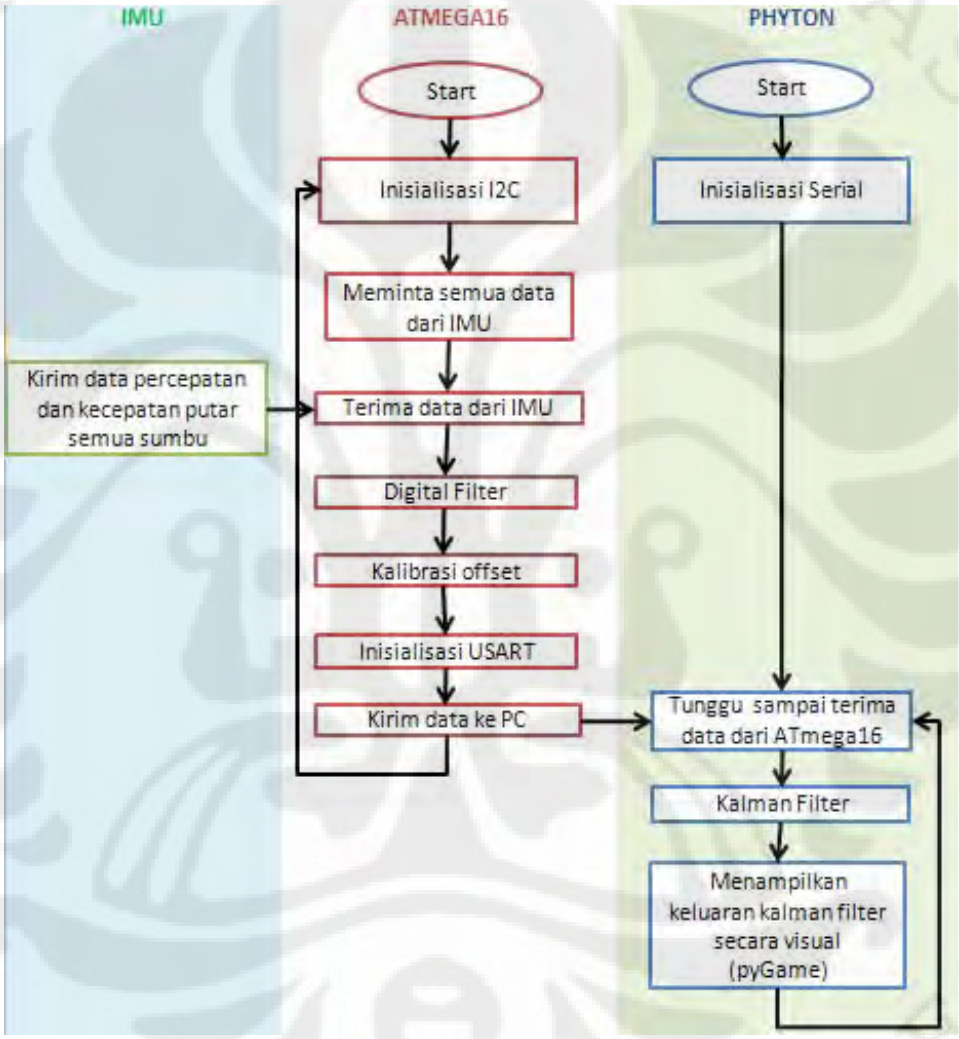
Welch, Greg & Bishop, Gary (2006). *An Introduction to the Kalman Filter*. Chappel Hill: Department of Computer Science University of North Carolina.

Widada, Wahyu (2005). *Aplikasi Digital Exponensial Filtering Untuk Embedded Sensor Payload Roket*. Bogor: Lembaga Penerbangan dan Antariksa Nasional Pusat Teknologi Wahana Dirgantara.

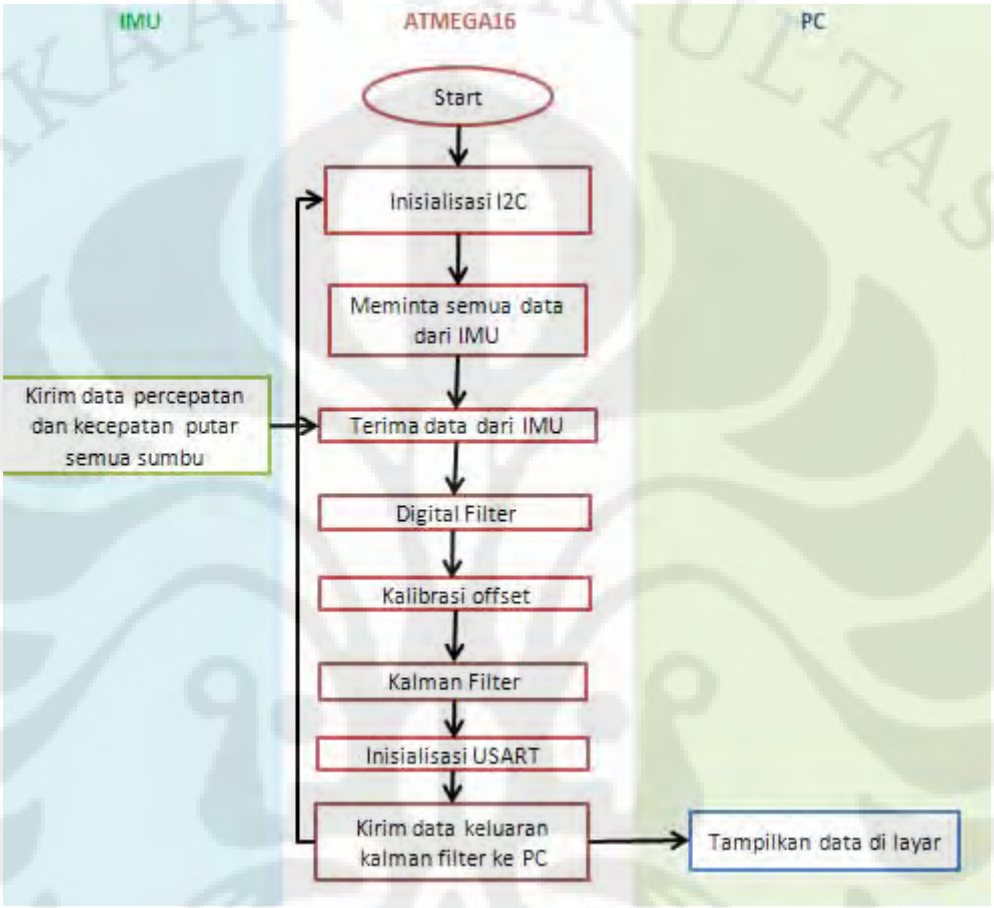
Winoto, Ardi (2008). *Mikrokontroler AVR Atmega8/32/16/8535 dan Pemrogramannya dengan bahasa C pada WinAVR*. Bandung: Informatika Bandung.

LAMPIRAN

Lampiran 1:Algoritma Pemrograman Sistem Navigasi Inersia dengan *Phyton*



Lampiran 2: Algoritma Pemrograman Sistem Navigasi Inersia tanpa *Phyton*



Lampiran 3:Pemrograman Sistem Navigasi Inersia AVR dengan koreksi gravitasi

```
#include <avr/io.h>
#include <stdlib.h>
#include <string.h>
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <util/twi.h>
#include <math.h>

/*microcontroller clock*/
#ifndef F_CPU
#define F_CPU 8000000UL
#endif

/* I2C clock in Hz */
#define SCL_CLOCK  100000L

/*address*/
#define ads7828  0x90

/*channel*/
#define x 0x84
#define y 0xC4
#define z 0x94
#define roll 0xA4
#define pitch 0xE4

#define          pi  3.14159265358979323846264 3
float ax[2], vx[2], px[2],ay[2], vy[2], py[2],az[2], vz[2], pz[2],vroll[2],
proll[2],vpitch[2], ppitch[2];
long sstatex,sstatey,sstatez,sstateroll,sstatepitch;
int data_ximu, data_yimu, data_zimu,data_rollimu, data_pitchimu;
int time,datakirim,countx,county,countz,count2;
float rollsp1s, rollsp2s, xsp1s, xsp2s,ysp1s, ysp2s,zsp1s, zsp2s, pitchsp1s,
pitchsp2s, theta, psi, taw;

/*parameter for kalman filter*/
float xroll[2][1]={{0},{0}};
float xpitch [2][1]={{0},{0}};
float Pkalmanroll [2][2]= {{1,0},{0,1}};
float Pkalmanpitch [2][2]= {{1,0},{0,1}};
float gyro_rate, accel_angle;
float P[2][2];
```

```
float xkalman[2][1];
float A_01;
float B_00;
float Sz_00;
float Sw_00;
float Sw_11;
float dt;

void kalman_init(void);
void kalman_update(void);
void movement_end_check(void);
void get_data(void);
void calibrate(void) ;
int baca_imu(int reg);
float smoothing1 (int data, float sp1s);
float smoothing2 (float sp1, float sp2s);
unsigned char i2c_readNak(void);
unsigned char i2c_readAck(void);
unsigned char i2c_rep_start(unsigned char address);
unsigned char i2c_write( unsigned char data );
void i2c_stop(void);
unsigned char i2c_start(unsigned char address);
void USART_Init(void);
unsigned char USART_Receive( void );
void USART_Transmit( unsigned char data );

int main() {
   char kirim[40];

        char* test_x = "x : ";
        char* test_y = "y : ";
        char* test_z = "z : ";
        char* test_roll = "roll : ";
        char* test_pitch = "pitch : ";
        char* test_time = "time : ";
        int i;

//calibration
calibrate();

//initial for digital filter
xsp1s = xsp2s = sstatex;
ysp1s = ysp2s = sstatey;
zsp1s = zsp2s = sstatez;
rollsp1s = rollsp2s = sstateroll;
pitchsp1s= pitchsp2s= sstatepitch;
time=0;
```

```
//USART initialization
USART_Init();

while (1) {
        // clock timer = 1/1024 clock sistem
        TCCR1B = 0b00000101;
        TCNT1 = 0;

        //Kalman filter initialization
        kalman_init();

        //get data from IMU
        get_data();

        //digital filter data acceleration z
        zsp1s = smoothing1 (data_zimu, zsp1s);
        zsp2s = smoothing2 (zsp1s, zsp2s);

        //calculate acceleration z in g
        az[1]=((zsp2s-sstatez)*0.10071)-100;

        //get acceleration z to calculate tilt
        taw= az[1];

        //prevent data not over the range
        //so it can be calculate using tan
        if (taw>100) {taw=100;}
        if (taw<-100) {taw=-100;}

        //digital filter data acceleration x
        xsp1s = smoothing1 (data_ximu, xsp1s);
        xsp2s = smoothing2 (xsp1s, xsp2s);

        //calculate acceleration x in g
        ax[1]=(xsp2s-sstatex)*0.10071;

        //get acceleration x to calculate tilt
        theta=ax[1];

        //prevent data not over the range
        //so it can be calculate using tan
        if (theta>100) {theta=100;}
        if (theta<-100) {theta=-100;}

        //calculate pitch angle from accelerometer
        theta=atan2(theta/100,taw/-100)*180/pi;

        //digital filter pitch rate from gyroscope
```

```
pitchsp1s = smoothing1 (data_pitchimu, pitchsp1s);
pitchsp2s = smoothing2 (pitchsp1s, pitchsp2s);

//calculate the pitch rate from gyroscope in deg/s
vpitch[1]=(pitchsp2s-sstatepitch)*1.2025;

//prevent the data not over the sensor range
if (vpitch[1]>=300)
{vpitch[1]=300;}
if (vpitch[1]<=-300)
{vpitch[1]=-300;}

//The current value must be sent to the previous value
//vpitch[0]=vpitch[1];
//ppitch[0]=ppitch[1];

//start Kalman Filter algorithm to calculate pitch angle
P[0][0]=Pkalmanpitch[0][0];
P[0][1]=Pkalmanpitch[0][1];
P[1][0]=Pkalmanpitch[1][0];
P[1][1]=Pkalmanpitch[1][1];

gyro_rate=vpitch[1];
accel_angle=theta;

xkalman[0][0]=xpitch[0][0];
xkalman[0][1]=xpitch[0][1];

kalman_update();

Pkalmanpitch[0][0]=P[0][0];
Pkalmanpitch[0][1]=P[0][1];
Pkalmanpitch[1][0]=P[1][0];
Pkalmanpitch[1][1]=P[1][1];

xpitch[0][0]=xkalman[0][0];
xpitch[0][1]=xkalman[0][1];

//send output pitch angle from kalman filter via USART
datakirim= xpitch[0][0];
sprintf(kirim, "%d  ",datakirim);
for (i = 0; test_pitch[i]!='\0'; i++) {
        USART_Transmit(test_pitch[i]); }
for (i = 0; kirim[i] != '\0'; i++) {
        USART_Transmit(kirim[i]); }

//digital filter data acceleration y
ysp1s = smoothing1 (data_yimu, ysp1s);
```

```
ysp2s = smoothing2 (ysp1s, ysp2s);
ay[1]=(ysp2s-sstatey)*0.10071;

//store acceleration y to calculate tilt angle
psi=ay[1];

//prevent data not over the range
//so it can be calculate using tan
if (psi>100) {psi=100;}
if (psi<-100) {psi=-100;}

//calculate roll angle from accelerometer
psi=atan2(psi/-100,taw/-100)*180/pi;

//digital filter roll data from gyroscope
rollsp1s = smoothing1 (data_rollimu, rollsp1s);
rollsp2s = smoothing2 (rollsp1s, rollsp2s);

//calculate the roll rate from gyroscope in deg/s
vroll[1]=(rollsp2s-sstateroll)*1.2025;

//prevent the data not over the sensor range
if (vroll[1]>=300)
{vroll[1]=300;}
if (vroll[1]<=-300)
{vroll[1]=-300;}

//The current value must be sent to the previous value
//vroll[0]=vroll[1];
//proll[0]=proll[1];

//Start Kalman Filter Algorithm to calculate roll angle
P[0][0]=Pkalmanroll[0][0];
P[0][1]=Pkalmanroll[0][1];
P[1][0]=Pkalmanroll[1][0];
P[1][1]=Pkalmanroll[1][1];

gyro_rate=vroll[1];
accel_angle=psi;

xkalman[0][0]=xroll[0][0];
xkalman[0][1]=xroll[0][1];

kalman_update();

Pkalmanroll[0][0]=P[0][0];
Pkalmanroll[0][1]=P[0][1];
Pkalmanroll[1][0]=P[1][0];
```

```
Pkalmanroll[1][1]=P[1][1];

xroll[0][0]=xkalman[0][0];
xroll[0][1]=xkalman[0][1];

//send output roll angle from kalman filter via USART
datakirim= xroll[0][0];
sprintf(kirim, "%d  ",datakirim);
for (i = 0; test_roll[i]!='\0'; i++) {
        USART_Transmit(test_roll[i]); }
for (i = 0; kirim[i] != '\0'; i++) {
        USART_Transmit(kirim[i]); }

//gravity correction in x axis
ax[1]=ax[1]-(sin((xpitch[0][0]/180)*pi)*100);

//discrimination window applied to variable acceleration x
if ((ax[1] <=3)&&(ax[1] >= -3))
        {ax[1] = 0;}

//calculate acceleration in cm^2/s
ax[1]=ax[1]*9.8;

//first X integration:
vx[1]= vx[0]+ ((ax[0]+ ((ax[1] - ax[0])/2))*time*0.000128);

//second X integration:
px[1]= px[0]+ ((vx[0]+ ((vx[1] - vx[0])/2))*time*0.000128);

//The current value must be sent to the previous value
vx[0]=vx[1];
ax[0]=ax[1];
px[0]=px[1];

//send data position in x axis via USART
datakirim=px[1];
sprintf(kirim, "%d  ",datakirim);
for (i = 0; test_x[i]!='\0'; i++) {
        USART_Transmit(test_x[i]); }
for (i = 0; kirim[i] != '\0'; i++) {
        USART_Transmit(kirim[i]); }

//gravity corretion in y axis
ay[1]=ay[1]+(sin((xroll[0][0]/180)*pi)*100);

//discrimination window applied to variable acceleration y
if ((ay[1] <=3)&&(ay[1] >= -3))
        {ay[1] = 0;}
```

```
//calculate acceleration in cm^2/s
ay[1]=ay[1]*9.8;

//first Y integration:
vy[1]= vy[0]+ ((ay[0]+ ((ay[1] - ay[0])/2))*time*0.000128);

//second Y integration:
py[1]= py[0]+ ((vy[0]+ ((vy[1] - vy[0])/2))*time*0.000128);

//The current value must be sent to the previous value
vy[0]=vy[1];
ay[0]=ay[1];
py[0]=py[1];

//send data position in y axis via USART
datakirim=py[1];
sprintf(kirim, "%d  ",datakirim);
for (i = 0; test_y[i]!='\0'; i++) {
        USART_Transmit(test_y[i]); }
for (i = 0; kirim[i] != '\0'; i++) {
        USART_Transmit(kirim[i]); }

//gravity correction in Z axis
az[1]=az[1]+(sqrt(1-square(sin((xroll[0][0]/180)*pi))-
square(sin((xpitch[0][0]/180*pi))))*100);

//discrimination window applied to variable acceleration z
if ((az[1] <=3)&&(az[1] >= -3))
{az[1] = 0;}

//calculate acceleration in cm^2/s
az[1]=az[1]*9.8;

//first Z integration:
vz[1]= vz[0]+ ((az[0]+ ((az[1] - az[0])/2))*time*0.000128);

//second Z integration:
pz[1]= pz[0]+ ((vz[0]+ ((vz[1] - vz[0])/2))*time*0.000128);

//The current value must be sent to the previous value
vz[0]=vz[1];
az[0]=az[1];
pz[0]=pz[1];

//send data position in y axis via USART
datakirim=pz[1];
sprintf(kirim, "%d  ",datakirim);
```

```
        for (i = 0; test_z[i]!='\0'; i++) {
                USART_Transmit(test_z[i]);}
        for (i = 0; kirim[i] != '\0'; i++) {
                USART_Transmit(kirim[i]); }

        //send clock to do the process
        sprintf(kirim, "%d  ",time);
        for (i = 0; test_time[i]!='\0'; i++) {
                USART_Transmit(test_time[i]); }
        for (i = 0; kirim[i] != '\0'; i++) {
                USART_Transmit(kirim[i]); }

        //movement end check algorithm
        movement_end_check();

        USART_Transmit('\r');

        //delay until clock reach 250
        while (TCNT1<250);
        //store clock in varible time
        time=TCNT1;
        //stop time
        TCCR1B = 0b00000000; }

return 0;
}

/*USART initialization*/
void USART_Init(void) {
        /* Set baud rate */
        UBRRH = 0;
        UBRRL = 25;
        UCSRA = (1<<U2X);
        /* Enable receiver and transmitter */
        UCSRB = (1<<RXEN)|(1<<TXEN);
        /* Set frame format: 8data, 2stop bit */
        UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0); }

/*receive data from USART*/
unsigned char USART_Receive( void )        {
        /* Wait for data to be received */
        while ( !(UCSRA & (1<<RXC)) );
        /* Get and return received data from buffer */
        return UDR; }

/*transmit data via USART*/
void USART_Transmit( unsigned char data ) {
/* Wait for empty transmit buffer */
```

```
while ( !( UCSRA & (1<<UDRE)) );
/* Put data into buffer, sends the data */
UDR = data; }

/*Send start sinyal in I2C*/
unsigned char i2c_start(unsigned char address)
{uint8_t   twst;
        // send START condition
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));
        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;
        // send device address
        TWDR = address;
        TWCR = (1<<TWINT) | (1<<TWEN);
        // wail until transmission completed and ACK/NACK has been received
        while(!(TWCR & (1<<TWINT)));
        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) )
return 1;
        return 0; }

/*send stop sinyal in I2C*/
void i2c_stop(void) {
   /* send stop condition */
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
        // wait until stop condition is executed and bus released
        while(TWCR & (1<<TWSTO)); }

/*write data to slave in I2c*/
unsigned char i2c_write( unsigned char data ) {
   uint8_t   twst;
        // send data to the previously addressed device
        TWDR = data;
        TWCR = (1<<TWINT) | (1<<TWEN);
        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));
        // check value of TWI Status Register. Mask prescaler bits
        twst = TW_STATUS & 0xF8;
        if( twst != TW_MT_DATA_ACK) return 1;
        return 0;}

/*send start signal again*/
unsigned char i2c_rep_start(unsigned char address) {
   return i2c_start( address ); }
```

```
/*read data from slave need ACK*/
unsigned char i2c_readAck(void) {
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
        while(!(TWCR & (1<<TWINT)));
   return TWDR; }

/*read data from slave without ACK*/
unsigned char i2c_readNak(void) {
        TWCR = (1<<TWINT) | (1<<TWEN);
        while(!(TWCR & (1<<TWINT)));
   return TWDR; }

/*to read from IMU*/
int baca_imu(int reg) {
unsigned char ret;
unsigned char retw;
unsigned char dataA;
unsigned char dataB;
unsigned char retr;
char* error1 = "device not found";
char* error2 = "write failed";
char* error3 = "read failed";
int i;

//intialization I2C
TWSR = 0;                                    // no prescaler
TWBR = ((F_CPU/SCL_CLOCK)-16)/2;     // must be > 10 for stable operation

//send start signal with "write" command
ret = i2c_start(ads7828+TW_WRITE);

//if failed or no ack send stop signal and send "device not found" via USART
if ( ret ) { i2c_stop();
                USART_Transmit('\r');
                for (i = 0; error1[i]!='\0'; i++) {
                USART_Transmit(error1[i]);}}
//if success then write reg to slave
else { retw=i2c_write(reg);
        //if failed to write send stop signal and send "write failed" via USART
        if (retw) { i2c_stop();
                USART_Transmit('\r');
                for (i = 0; error2[i]!='\0'; i++) {
                USART_Transmit(error2[i]);} }
        //if success to write
        else {
        //read value back, wait until the device is no longer busy from the previous
        write operation //
```

```
        retr= i2c_rep_start(ads7828+TW_READ);
        // set device address and read command
        //if failed send stop command and send "read failed" via USART
                if (retr) { i2c_stop();
                        USART_Transmit('\r');
                        for (i = 0; error3[i]!='\0'; i++) {
                USART_Transmit(error3[i]);} }
                //if success read the data send by slave
                else { dataA = i2c_readAck();           //store MSB in dataA
                        dataB = i2c_readNak();          //store LSB in dataB
                        i2c_stop();       //send stop signal }
                } }
 return ((dataA<<8) + dataB); //mix the MSB and LSB
}

/*double exponential filter algorithm*/
float smoothing1 (int data, float sp1s) {
float sp1;
sp1 = (0.2*data)+(0.8*sp1s);
return sp1; }

 float smoothing2 (float sp1, float sp2s) {
 float sp2;
 sp2 = (0.2*sp1)+(0.8*sp2s);
 return sp2; }

/*calibration function*/
void calibrate(void) {
  unsigned int count1;
  count1 = 0;
  sstatex=sstatey=0;
  do{
   //get data from imu
   get_data();
   // Accumulate Samples
  sstatex = sstatex + data_ximu;
  sstatey = sstatey + data_yimu;
  sstatez = sstatez+ data_zimu;
  sstateroll = sstateroll + data_rollimu;
  sstatepitch = sstatepitch + data_pitchimu;
  count1++;
  }while(count1!=0x0400);                  // loop until 1024 times

  //divide by 1024 and store offset value
  sstatex=sstatex>>10;
  sstatey=sstatey>>10;
  sstatez=sstatez>>10;
  sstateroll=sstateroll>>10;
```

```
 sstatepitch=sstatepitch>>10; }

/*function to get all data from IMU*/
void get_data(void){
        data_ximu = baca_imu(x);
        data_yimu = baca_imu(y);
        data_zimu = baca_imu(z);
        data_rollimu = baca_imu(roll);
        data_rollimu = baca_imu(roll);
        data_pitchimu = baca_imu(pitch);}

/*function to movement end check*/
void movement_end_check(void) {
 //count the number of acceleration samples that equals cero
 if (ax[1]==0)
   { countx++;}
 else { countx =0;}
 //if this number exceeds 20, assume that velocity is zero
 if (countx>=20) {
  vx[1]=0;
  vx[0]=0;}

 // do the same for the Y axis
 if (ay[1]==0)
  { county++;}
 else { county =0;}
 if (county>=20) {
  vy[1]=0;
  vy[0]=0; }

  //do the same for the Z axis
  if (az[1]==0)
  { countz++;}
 else { countz =0;}
 if (countz>=20) {
  vz[1]=0;
  vz[0]=0;}}

/*Kalman Filter Initialization*/
void kalman_init(void) {
   // Set the delta in seconds between samples.
   A_01 = -(time*0.000128);
   B_00 = time*0.000128;

   // Set the measurement noise covariance matrix values.
   Sz_00 = 0.3;

   // Set the process noise covariance matrix values.
```

```
  Sw_00 = 0.001;
  Sw_11 = 0.003;}

/*Kalman Filter Algorithm*/
void kalman_update(void) {
// Update the state estimation and compute the Kalman gain.
// The estimated angle from the output matrix is returned.
float s_00;
float inn_00;
float K_00;
float K_10;
float AP_00;
float AP_01;
float AP_10;
float AP_11;
float KCPAT_00;
float KCPAT_01;
float KCPAT_10;
float KCPAT_11;

// Update the state estimate by extrapolating current state estimate with input u.
// x = A * x + B * u
xkalman[0][0] += (A_01 * xkalman[1][0]) + (B_00 * gyro_rate);

// Compute AP matrix for use below.
// AP = A * P
AP_00 = P[0][0] + A_01 * P[1][0];
AP_01 = P[0][1] + A_01 * P[1][1];
AP_10 = P[1][0];
AP_11 = P[1][1];

// Project the error covariance ahead
// P= A * P * A' + Sw
P[0][0] = (AP_00 + (AP_01 * A_01))+Sw_00;
P[0][1] = AP_01;
P[1][0] = AP_10 + (AP_11 * A_01);
P[1][1] = AP_11+ Sw_11;

 // Compute the innovation -- error between measured value and state estimate.
 // inn = y - c * x
inn_00 = accel_angle - xkalman[0][0];

// Compute the covariance of the innovation.
// s = C * P * C' + Sz
s_00 = P[0][0] + Sz_00;

// Compute AP matrix for use below.
// AP = A * P
```

```
AP_00 = P[0][0] + A_01 * P[1][0];
AP_01 = P[0][1] + A_01 * P[1][1];
AP_10 = P[1][0];
AP_11 = P[1][1];

// Compute the kalman gain matrix.
// K = A * P * C' * inv(s)
K_00 = AP_00 / s_00;
K_10 = AP_10 / s_00;

// Update the state estimate.
// x = x + K * inn
xkalman[0][0] += K_00 * inn_00;
xkalman[1][0] += K_10 * inn_00;

// Compute the new covariance of the estimation error.
// P = P - K * C * P * A'
KCPAT_00 = (K_00 * P[0][0]) + (K_00 * P[0][1]);
KCPAT_01 = (K_00 * P[0][1]);
KCPAT_10 = (K_10 * P[0][0]) + (K_10 * P[0][1]);
KCPAT_11 = (K_10 * P[0][1]);
P[0][0] = P[0][0] - KCPAT_00;
P[0][1] = P[0][1] - KCPAT_01;
P[1][0] = P[1][0] - KCPAT_10;
P[1][1] = P[1][1] - KCPAT_11; }
```

Lampiran 4: Pemrograman Sistem Navigasi Inersia AVR dengan *Phyton*

```
#include <avr/io.h>
#include <stdlib.h>
#include <string.h>
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <util/twi.h>
#include <math.h>

/*microcontroller clock*/
#define F_CPU 8000000UL

/* I2C clock in Hz */
#define SCL_CLOCK  100000L

/*address*/
#define ads7828  0x90

/*channel*/
#define x 0x84
#define y 0xC4
#define z 0x94
#define roll 0xA4
#define pitch 0xE4

int baca_imu(int reg);
unsigned char i2c_readNak(void);
unsigned char i2c_readAck(void);
unsigned char i2c_rep_start(unsigned char address);
unsigned char i2c_write( unsigned char data );
void i2c_stop(void);
unsigned char i2c_start(unsigned char address);
float smoothing1 (int data, float sp1s);
float smoothing2 (float sp1, float sp2s);
void USART_Init(void);
unsigned char USART_Receive( void );
void USART_Transmit( unsigned char data );
void calibrate (void);

int data_ximu,data_yimu,data_zimu,data_rollimu,data_pitchimu;
long sstatex,sstatey,sstateroll,sstatepitch, sstatez;
float rollsp1s, rollsp2s, xsp1s, xsp2s,ysp1s, ysp2s,zsp1s, zsp2s, pitchsp1s,
pitchsp2s;
```

```
int main(void) {

char kirim[50];
int i;

/*initiation USART*/
USART_Init();

/*calibration function*/
calibrate();

/*initial for digital filter*/
xsp1s = xsp2s = sstatex;
ysp1s = ysp2s = sstatey;
zsp1s = zsp2s = sstatez;
rollsp1s = rollsp2s = sstateroll;
pitchsp1s= pitchsp2s= sstatepitch;

while (1) {
        // clock timer = 1/1024 clock sistem
        TCCR1B = 0b00000101;
        TCNT1 = 0;

        //send offset x
        data_ximu=sstatex;
        sprintf(kirim, "%ub",data_ximu);
        for (i = 0; kirim[i] != '\0'; i++) {
                USART_Transmit(kirim[i]);}

        //send offset y
        data_yimu=sstatey;
        sprintf(kirim, "%u;",data_yimu);
        for (i = 0; kirim[i] != '\0'; i++) {
                USART_Transmit(kirim[i]);}

        //send offset z
        data_zimu=sstatez;
        sprintf(kirim, "%u;",data_zimu);
        for (i = 0; kirim[i] != '\0'; i++) {
                USART_Transmit(kirim[i]);}

        //send offset roll
        data_rollimu=sstateroll;
        sprintf(kirim, "%u;",data_rollimu);
        for (i = 0; kirim[i] != '\0'; i++) {
                USART_Transmit(kirim[i]);}

        //send offset pitch
```

```
data_pitchimu=sstatepitch;
sprintf(kirim, "%u;",data_pitchimu);
for (i = 0; kirim[i] != '\0'; i++) {
        USART_Transmit(kirim[i]);}

//get data acceleration x
data_ximu = baca_imu(x);

//digital filter algorithm
xsp1s = smoothing1 (data_ximu, xsp1s);
xsp2s = smoothing2 (xsp1s, xsp2s);
data_ximu=xsp2s;

//send data acceleration x after filter
sprintf(kirim, "%u;",data_ximu);
for (i = 0; kirim[i] != '\0'; i++) {
        USART_Transmit(kirim[i]);}

//get data acceleration y
data_yimu  = baca_imu(y);

//digital filter algorithm
ysp1s = smoothing1 (data_yimu, ysp1s);
ysp2s = smoothing2 (ysp1s, ysp2s);
data_yimu=ysp2s;

//send data acceleration y after filter
sprintf(kirim, "%u;",data_yimu);
for (i = 0; kirim[i] != '\0'; i++) {
        USART_Transmit(kirim[i]);}

//get data acceleration z
data_zimu = baca_imu(z);

//digital filter algorithm
zsp1s = smoothing1 (data_zimu, zsp1s);
zsp2s = smoothing2 (zsp1s, zsp2s);
data_zimu=zsp2s;

//send data acceleration z after filter
sprintf(kirim, "%u;",data_zimu);
for (i = 0; kirim[i] != '\0'; i++) {
        USART_Transmit(kirim[i]);}

//get data angular rate roll twice to correct the data
data_rollimu = baca_imu(roll);
data_rollimu = baca_imu(roll);
```

```
        //digital filter algorithm
        rollsp1s = smoothing1 (data_rollimu, rollsp1s);
        rollsp2s = smoothing2 (rollsp1s, rollsp2s);
        data_rollimu= rollsp2s;

        //send data angular rate roll after filter
        sprintf(kirim, "%u;",data_rollimu);
        for (i = 0; kirim[i] != '\0'; i++) {
                USART_Transmit(kirim[i]);}

        //get data angular rate pitch
        data_pitchimu = baca_imu(pitch);

        //digital filter algorithm
        pitchsp1s = smoothing1 (data_pitchimu, pitchsp1s);
        pitchsp2s = smoothing2 (pitchsp1s, pitchsp2s);
        data_pitchimu= pitchsp2s;

        //send data angular rate pitch after filter
        sprintf(kirim, "%ua\n\r",data_pitchimu);
        for (i = 0; kirim[i] != '\0'; i++) {
                USART_Transmit(kirim[i]);}

        // delay until clock 250
        while (TCNT1 < 250);

        //stop timer
        TCCR1B = 0b00000000;}
  return 0;}

/*USART initialisation*/
void USART_Init(void) {
        /* Set baud rate */
        UBRRH = 0;
        UBRRL = 25;
        UCSRA = (1<<U2X);
        /* Enable receiver and transmitter */
        UCSRB = (1<<RXEN)|(1<<TXEN);
        /* Set frame format: 8data, 2 stop bit*/
        UCSRC = (1<<URSEL)|(3<<UCSZ0)|(1<<USBS);}

/*receive data from USART*/
unsigned char USART_Receive( void )     {
        /* Wait for data to be received */
        while ( !(UCSRA & (1<<RXC)) );
        /* Get and return received data from buffer */
        return UDR;}
```

```
/*transmit data via USART*/
void USART_Transmit( unsigned char data ){
/* Wait for empty transmit buffer */
while ( !( UCSRA & (1<<UDRE)) );
/* Put data into buffer, sends the data */
UDR = data;}

/*Send start sinyal in I2C*/
unsigned char i2c_start(unsigned char address)
{uint8_t   twst;
        // send START condition
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));
        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;
        // send device address
        TWDR = address;
        TWCR = (1<<TWINT) | (1<<TWEN);
        // wail until transmission completed and ACK/NACK has been received
        while(!(TWCR & (1<<TWINT)));
        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) )
return 1;
        return 0;}


/*send stop sinyal in I2C*/
void i2c_stop(void){
   // send stop condition
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
        // wait until stop condition is executed and bus released
        while(TWCR & (1<<TWSTO));}

/*write data to slave in I2c*/
unsigned char i2c_write( unsigned char data ){
   uint8_t   twst;
        // send data to the previously addressed device
        TWDR = data;
        TWCR = (1<<TWINT) | (1<<TWEN);
        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));
        // check value of TWI Status Register. Mask prescaler bits
        twst = TW_STATUS & 0xF8;
        if( twst != TW_MT_DATA_ACK) return 1;
        return 0;}
```

```
/*send start signal again*/
unsigned char i2c_rep_start(unsigned char address){
   return i2c_start( address );}

/*read data from slave need ACK*/
unsigned char i2c_readAck(void){
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
        while(!(TWCR & (1<<TWINT)));
   return TWDR;}

/*read data from slave without ACK*/
unsigned char i2c_readNak(void){
        TWCR = (1<<TWINT) | (1<<TWEN);
        while(!(TWCR & (1<<TWINT)));
   return TWDR;}

/*to read from IMU*/
int baca_imu(int reg) {
unsigned char ret;
unsigned char retw;
unsigned char dataA;
unsigned char dataB;
unsigned char retr;

//intialization I2C
TWSR = 0;                                   // no prescaler
TWBR = ((F_CPU/SCL_CLOCK)-16)/2;    // must be > 10 for stable operation
//send start signal with "write" command
ret = i2c_start(ads7828+TW_WRITE);

//if failed or no ack send stop signal
if ( ret ) {
      i2c_stop();}

//if success then write reg to slave
else {   retw=i2c_write(reg);
        //if failed to write send stop signal
        if (retw) {
                i2c_stop();}
        //if success to write
        else {
        //read value back, wait until the device is no longer busy from the previous
        //write operation
      retr= i2c_rep_start(ads7828+TW_READ);
                // set device address and read command
                //if failed send stop command
                if (retw) {
```

```
                i2c_stop();}
            //if success read the data send by slave
            else {
                dataA = i2c_readAck();      //store MSB in dataA
                dataB = i2c_readNak();      //store LSB in dataB
                i2c_stop();}    //send stop signal
            }}
 return ((dataA<<8) + dataB);//mix the MSB and LSB}

/*calibration function*/
void calibrate(void) {
  unsigned int count1;
  count1 = 0;
  sstatex=sstatey=sstatez=0;
  do{   //get data from imu
  data_ximu=baca_imu(x);
  data_yimu=baca_imu(y);
  data_zimu=baca_imu(z);
  data_rollimu=baca_imu(roll);
  data_rollimu=baca_imu(roll);
  data_pitchimu=baca_imu(pitch);

  // Accumulate Samples
  sstatex = sstatex + data_ximu;
  sstatey = sstatey + data_yimu;
  sstatez = sstatez+ data_zimu;
  sstateroll = sstateroll + data_rollimu;
  sstatepitch = sstatepitch + data_pitchimu;
  count1++;}
  while(count1!=0x0400);              // loop until 1024 times

  //divide by 1024 and store offset value
  sstatex=sstatex>>10;
  sstatey=sstatey>>10;
  sstatez=sstatez>>10;
  sstateroll=sstateroll>>10;
  sstatepitch=sstatepitch>>10;}

/*double exponential filter algorithm*/
float smoothing1 (int data, float sp1s) {
float sp1;
sp1 = (0.2*data)+(0.8*sp1s);
return sp1;}

 float smoothing2 (float sp1, float sp2s) {
 float sp2;
 sp2 = (0.2*sp1)+(0.8*sp2s);
 return sp2;}
```

Lampiran 5: Pemrograman Sistem Navigasi Inersia pada *Phyton*

```
import pygame, sys
import time
import serial
import numpy
import math

#inisialisasi untuk komunikasi serial
ser = serial.Serial(0,38400,stopbits=2)

#waktu pencuplikan
dt = 0.032

# Bentuk umum State-space
# x. = A*x+B*u  ==>     x. = F*x+B*x
# y  = C*x      ==>     y  = H*x
#
# Bentuk ini diubah untuk memudahkan pembacaan dari algoritma kalman filter
# umum

F = numpy.array([[1, -dt], [0, 1]],"float32")
B = numpy.array([dt, 0],"float32")
H = numpy.array([1, 0],"float32")
Q = numpy.array([[0.003, 0],[0.001, 1]],"float32")
P = numpy.array([[1, 0],[0, 1]],"float32")
Ppsi = numpy.array([[1, 0],[0, 1]],"float32")
R = 0.3
x = [0, 0]
xpsi = [0, 0]
roll_p = 0
rolls = 0
pitch_p = 0
pitchs = 0

screen = pygame.display.set_mode((900, 480))
running = 1

tengah_x_kalman = 170
tengah_y_kalman = 340
tengah_x_acc = 470
tengah_y_acc = 340
tengah_x_perc = 770
tengah_y_perc = 340

tengah_x_kalman_psi = 170
tengah_y_kalman_psi = 140
```

```
tengah_x_acc_psi = 470
tengah_y_acc_psi = 140
tengah_x_perc_psi = 770
tengah_y_perc_psi = 140

while running:

        data = ser.readline()
        error = 0

        #Algoritma kasar untuk ekstraksi data
        i = data.find('b')
        if (i != -1):
                try:
                        sstatex = float(data[:i])
                        data = data[i+1:]
                except ValueError:
                        error = 1
        else:
                error = 1
        i = data.find(';')
        if (i != -1):
                try:
                        sstatey = float(data[:i])
                        data = data[i+1:]
                except ValueError:
                        error = 1
        else:
                error = 1
        i = data.find(';')
        if (i != -1):
                try:
                        sstatez = float(data[:i])
                        data = data[i+1:]
                except ValueError:
                        error = 1
        else:
                error = 1
        i = data.find(';')
        if (i != -1):
                try:
                        sstateroll = float(data[:i])
                        data = data[i+1:]
                except ValueError:
                        error = 1
        else:
                error = 1
        i = data.find(';')
```

```
if (i != -1):
        try:
                sstatepitch = float(data[:i])
                data = data[i+1:]
        except ValueError:
                error = 1
else:
        error = 1

i = data.find(';')
if (i != -1):
        try:
                data_x = float(data[:i])
                data = data[i+1:]
        except ValueError:
                error = 1
else:
        error = 1
i = data.find(';')
if (i != -1 and error == 0):
        try:
                data_y = float(data[:i])
                data = data[i+1:]
        except ValueError:
                error = 1
else:
        error = 1
i = data.find(';')
if (i != -1 and error == 0):
        try:
                z = float(data[:i])
                data = data[i+1:]
        except ValueError:
                error = 1
else:
        error = 1
i = data.find(';')
if (i != -1 and error == 0):
        try:
                roll = float(data[:i])
                data = data[i+1:]
        except ValueError:
                error = 1
else:
        error = 1
i = data.find('a')
if (i != -1 and error == 0 and i < 7):
        try:
```

```
                pitch = float(data[:i])
                data = data[i+1:]
            except ValueError:
                error = 1
    else:
        error = 1

    #Mulai algoritma menghitung sudut
    if error == 0:

        #Hitung kecepatan roll dari gyroscope
        roll = ((roll-sstateroll)*3.3/4096)/0.00067

        #windowing agar tak melewati range sensor
        if (roll>300):
            roll=300
        elif (roll<-300):
            roll=-300

        #integrasi kecepatan roll
        roll_p += (rolls+((roll-rolls)/2))*dt #=roll

        #hitung akselerasi pada sumbu y
        theta2 = (data_y-sstatey)/4096.*3.3/.8

        #windowing
        if (theta2<0.02 and theta2>-0.02):
            theta2=0
        if (theta2 < -1.):
            theta2 = -1.
        elif (theta2 > 1.):
            theta2 = 1.

# hitung akselerasi pada sumbu Z
        taw=(((z-sstatez)/4096)*3.3/.8)-1

        #windowing
        if (taw<0.02 and taw>-0.02):
            taw=0
        if (taw < -1.):
            taw = -1.
        elif (taw > 1.):
            taw = 1.

        #hitung sudut roll dari data accelerometer
        theta2 = math.atan2(-theta2,-taw)
        theta2 = math.degrees(theta2)
```

```
#Mulai Algorima kalman filter
x = numpy.dot(F,x)+numpy.dot(B,roll)
P = numpy.dot(numpy.dot(F,P),numpy.transpose(F))+Q

err = theta2 - x[0]
s = numpy.dot(H,numpy.dot(P,numpy.transpose(H)))+R
K = numpy.dot(P,numpy.transpose(H))/s
ok = numpy.dot(K,err)
x += numpy.dot(K,err)
P = numpy.dot(1.0-numpy.dot(K,H),P)

#untuk keperluan visualisasi
theta = math.radians(-x[0])
theta_acc = theta2
theta_perc = roll_p

mulai_x_kalman = tengah_x_kalman - 100*math.cos(theta)
mulai_y_kalman = tengah_y_kalman - 100*math.sin(theta)
akhir_x_kalman = tengah_x_kalman +10*math.cos(theta+
        math.radians(-90))
akhir_y_kalman = tengah_y_kalman + 10*math.sin(theta+
        math.radians(-90))
akhir_x_kalman2 = tengah_x_kalman + 100*math.cos(theta)
akhir_y_kalman2 = tengah_y_kalman + 100*math.sin(theta)
mulai_x_acc = tengah_x_acc - 100*math.cos(math.radians
        (-theta_acc))
mulai_y_acc = tengah_y_acc - 100*math.sin(math.radians
        (-theta_acc))
akhir_x_acc = tengah_x_acc + 10*math.cos(math.radians
        (-theta_acc-90))
akhir_y_acc = tengah_y_acc + 10*math.sin(math.radians
        (-theta_acc-90))
akhir_x_acc2 = tengah_x_acc + 100*math.cos(math.radians
        (-theta_acc))
akhir_y_acc2 = tengah_y_acc + 100*math.sin(math.radians
        (-theta_acc))
mulai_x_perc = tengah_x_perc - 100*math.cos(math.radians
        (-theta_perc))
mulai_y_perc = tengah_y_perc - 100*math.sin(math.radians
        (-theta_perc))
akhir_x_perc = tengah_x_perc + 10*math.cos(math.radians
        (-theta_perc-90))
akhir_y_perc = tengah_y_perc + 10*math.sin(math.radians
        (-theta_perc-90))
akhir_x_perc2 = tengah_x_perc + 100*math.cos(math.radians
        (-theta_perc))
akhir_y_perc2 = tengah_y_perc + 100*math.sin(math.radians
        (-theta_perc))
```

```
screen.fill((250,250,250))
pygame.draw.line(screen, (0,0,255), (tengah_x_kalman,
        tengah_y_kalman), (akhir_x_kalman, akhir_y_kalman))
pygame.draw.line(screen, (0,0,255), (mulai_x_kalman,
        mulai_y_kalman), (akhir_x_kalman2, akhir_y_kalman2))
pygame.draw.line(screen, (0,0,255), (tengah_x_acc,tengah_y_acc),
        (akhir_x_acc, akhir_y_acc))
pygame.draw.line(screen, (0,0,255), (mulai_x_acc,mulai_y_acc),
        (akhir_x_acc2, akhir_y_acc2))
pygame.draw.line(screen, (0,0,255), (tengah_x_perc,
        tengah_y_perc), (akhir_x_perc, akhir_y_perc))
pygame.draw.line(screen, (0,0,255), (mulai_x_perc,mulai_y_perc),
        (akhir_x_perc2, akhir_y_perc2))

#hitung kecepatan pitch dari gyroscope
pitch = (pitch-sstatepitch)/4096.0*3.3/.00067

#windowing untuk mencegah data lebih dari range sensor
if (pitch>300):
        pitch=300
elif (pitch<-300):
        pitch=-300

#integrasi kecepatan roll
pitch_p += (pitchs+((pitch-pitchs)/2))*dt #=roll

#hitung percepatan pada sumbu x
psi2 = (data_x-sstatex)/4096*3.3/.8

#windowing
if (psi2<0.02 and psi2>-0.02):
        psi2=0
if (psi2 < -1.):
        psi2 = -1.
elif (psi2 > 1.):
        psi2 = 1.

#hitung sudut pitch dari keluaran accelerometer
psi2 = math.atan2(psi2,-taw)
psi2 = math.degrees(psi2)

#mulai algoritma kalman filter
xpsi = numpy.dot(F,xpsi)+numpy.dot(B,pitch)
Ppsi = numpy.dot(numpy.dot(F,Ppsi),numpy.transpose(F))+Q

err = psi2 - xpsi[0]
s = numpy.dot(H,numpy.dot(Ppsi,numpy.transpose(H)))+R
```

```
K = numpy.dot(Ppsi,numpy.transpose(H))/s
ok = numpy.dot(K,err)
xpsi += numpy.dot(K,err)
Ppsi = numpy.dot(1.0-numpy.dot(K,H),Ppsi)

#untuk keperluan visualisasi
psi = math.radians(-xpsi[0])
psi_acc = psi2
psi_perc = pitch_p

mulai_x_kalman_psi = tengah_x_kalman_psi - 100*math.cos(psi)
mulai_y_kalman_psi = tengah_y_kalman_psi - 100*math.sin(psi)
akhir_x_kalman_psi = tengah_x_kalman_psi + 10*math.cos(psi+
        math.radians(-90))
akhir_y_kalman_psi = tengah_y_kalman_psi + 10*math.sin(psi+
        math.radians(-90))
akhir_x_kalman2_psi = tengah_x_kalman_psi + 100*math.cos(psi)
akhir_y_kalman2_psi = tengah_y_kalman_psi + 100*math.sin(psi)
mulai_x_acc_psi = tengah_x_acc_psi - 100*math.cos(math.radians
        (-psi_acc))
mulai_y_acc_psi = tengah_y_acc_psi - 100*math.sin(math.radians
        (-psi_acc))
akhir_x_acc_psi = tengah_x_acc_psi + 10*math.cos(math.radians
        (-psi_acc-90))
akhir_y_acc_psi = tengah_y_acc_psi + 10*math.sin(math.radians
        (-psi_acc-90))
akhir_x_acc2_psi = tengah_x_acc_psi + 100*math.cos
        (math.radians(-psi_acc))
akhir_y_acc2_psi = tengah_y_acc_psi + 100*math.sin
        (math.radians(-psi_acc))
mulai_x_perc_psi = tengah_x_perc_psi - 100*math.cos
        (math.radians(-psi_perc))
mulai_y_perc_psi = tengah_y_perc_psi - 100*math.sin
        (math.radians(-psi_perc))
akhir_x_perc_psi = tengah_x_perc_psi + 10*math.cos
        (math.radians(-psi_perc-90))
akhir_y_perc_psi = tengah_y_perc_psi + 10*math.sin
        (math.radians(-psi_perc-90))
akhir_x_perc2_psi = tengah_x_perc_psi + 100*math.cos
        (math.radians(-psi_perc))
akhir_y_perc2_psi = tengah_y_perc_psi + 100*math.sin
        (math.radians(-psi_perc))

pygame.draw.line(screen, (0,0,255), (tengah_x_kalman_psi,
        tengah_y_kalman_psi), (akhir_x_kalman_psi,
        akhir_y_kalman_psi))
pygame.draw.line(screen, (0,0,255), (mulai_x_kalman_psi,
        mulai_y_kalman_psi), (akhir_x_kalman2_psi,
```

```
            akhir_y_kalman2_psi))
pygame.draw.line(screen, (0,0,255), (tengah_x_acc_psi,
            tengah_y_acc_psi), (akhir_x_acc_psi, akhir_y_acc_psi))
pygame.draw.line(screen, (0,0,255), (mulai_x_acc_psi,
            mulai_y_acc_psi), (akhir_x_acc2_psi, akhir_y_acc2_psi))
pygame.draw.line(screen, (0,0,255), (tengah_x_perc_psi,
            tengah_y_perc_psi), (akhir_x_perc_psi, akhir_y_perc_psi))
pygame.draw.line(screen, (0,0,255), (mulai_x_perc_psi,
            mulai_y_perc_psi), (akhir_x_perc2_psi,
            akhir_y_perc2_psi))
pygame.display.flip()

event = pygame.event.poll()
if event.type == pygame.QUIT:
            ser.close()
            pygame.quit()
            sys.exit()
```