UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

COLEGIADO DO CURSO DE ENGENHARIA DE COMPUTAÇÃO

Gabriel Ferrari Batista Martins

Estudo de acelerômetros para sistemas embarcados. Uma abordagem de instrumentação para estimação da velocidade

Vitória, ES

2021

Gabriel Ferrari Batista Martins

Estudo de acelerômetros para sistemas embarcados. Uma abordagem de instrumentação para estimação da velocidade

Anteprojeto apresentado ao Colegiado do Curso de Engenharia de Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito para aprovação na Disciplina Projeto de Graduação I.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Colegiado do Curso de Engenharia de Computação

Orientador: Prof. Camilo Arturo Rodriguez Diaz

Coorientador: Prof. Hans Jorg Andreas Schneebeli

Vitória, ES

2021

1 Introdução

A medição de aceleração e velocidade é importante em muitos sistemas principalmente veículos, onde essa informação pode ser usada para fazer predições da velocidade futura a curto prazo, que permitem escolha de rota otimizada, redução de tráfico e aumento da segurança.(ZHANG et al., 2021)

Um dos métodos mais simples de calcular a velocidade é a distância entre dois sensores dividida pelo tempo entre os sinais de cada um (ZHANG et al., 2021). Esse método no entanto é pouco prático no contexto de tempo real de veículos, pois assume que a distância percorrida pode ser medida a qualquer momento, e não ajuda a contextualizar os dados em 3 dimensões. No contexto de coleta e armazenamento de dados para redes de sensores, faria mais sentido utilizar um sensor acelerômetro para medições.

Um acelerômetro é um dispositivo que mede a aceleração linear, que é a tendência da velocidade aumentar ou diminuir na direção de um eixo. Já um giroscópio mede a aceleração angular que é como a aceleração linear, mas ao redor de um eixo ao invés de na direção do mesmo. Na Figura 1 temos os eixos da aceleração linear em vermelho e da aceleração angular em preto. Ambos os tipos de aceleração podem ser integradas para se obter as respectivas velocidades, e integradas novamente para se obter o deslocamento do referencial. Utilizando as medições de um acelerômetro, poderíamos então obter a velocidade e o deslocamento para dado período de tempo, porém há circunstancias do sensor que devem ser compreendidas para se obter dados de qualidade.

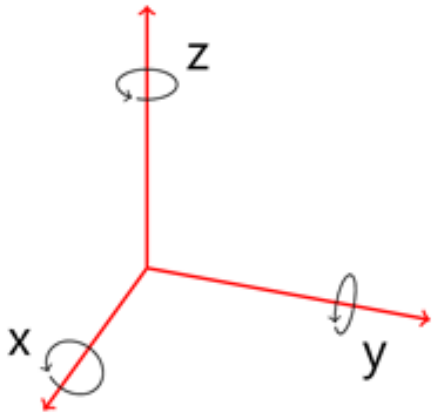

Figura 1 – Em vermelho, sentido linear dos eixos. Em preto sentido angular dos eixos.

O objetivo desse trabalho seria então demonstrar a precisão e compromissos de um acelerômetro comum de baixo custo para fim de coleta de dados, e o que pode ser feito

para melhorar a fidelidade das medidas para problemas como ruído e resolução e alguns algoritmos de integração para obter velocidade que podem ser usados.

1.1 Motivação e Justificativa

A motivação para esse trabalho foi a minha experiencia no projeto Vitória Baja, que me levou a aprender mais sobre sistemas embarcados, e onde comecei a implementar a medição e coleta de dados de acelerômetro em veículos baja. Isso me levou a ter mais interesse no assunto, que pode ser aplicado em diversos projetos.

A justificativa para tal é que na medição de velocidade por acelerômetros há vários detalhes como resolução e ruído, que levam a comprometimentos do projeto. Já há vários estudos sobre a precisão de tais métodos (CADE et al., 2018; SONG; UCHANSKI; HEDRICK, 2002; REIS, 2016) e como aumentar essa precisão e nesse trabalho seria comparado a eficiência de alguns desses métodos, como filtros de ruído e comparação de métodos diferentes de integração para obtenção da velocidade.

1.2 Objetivos

Objetivo geral: Estudar o potencial de utilização de acelerômetros para estimação de velocidade em sistemas embarcados.

Objetivos específicos:

- Montar o protótipo de coleta de dados;
- Obter medições de teste para calibração e estimação de velocidade;
- Implementação de algoritmos para estimação de velocidade em sistema embarcado em tempo real;
- Validar a proposta em testes de campo;
- Analisar dados obtidos;

1.3 Método de Desenvolvimento do Trabalho

O propósito desse trabalho é analisar o caso de uso de um acelerômetro de 6 eixos como acelerômetro embarcado de um veículo. Para isso serão obtidos dados de medição que serão comparados com padrões estabelecidos e analisados em sua eficácia. Os dados serão coletados por um circuito com microcontrolador e armazenado em cartão de memória, mas o mesmo principio pode ser aplicado à outras configurações.Pode se classificar então:

- Propósito: Pesquisa Exploratória;
- Abordagem: Quantitativa;
- Contexto: Veículo automotivo;
- Objetos: Sensor acelerômetro, circuito para leitura e registro de dados.
- Procedimentos: Pesquisa de Campo;
- Instrumentos de análise de dados: Ferramentas computacionais como Matlab e Octave e linguagens de programação como Python e C++.

1.4 Cronograma

- Atividade 1: Montar o circuito para registro da medição do MPU6050;
- Atividade 2: Utilizar o circuito montado em um veiculo automotivo e registrar a velocidade do veículo durante o percurso para calibração;
- Atividade 3: Implementar algoritmos que ajudem a processar as leituras em velocidade;
- Atividade 4: Validar a proposta calculando a velocidade em tempo real de um veiculo;
- Atividade 5: Verificar os dados obtidos e sua precisão. Entender o porquê dos resultados;

Tabela 1 – Cronograma das atividades.

2021	**Junho**	**Julho**	**Agosto**	**Setembro**	**Outubro**
Atividade 1	X	X	X		
Atividade 2			X		
Atividade 3			X	X	X
Atividade 4					
Atividade 5					
2021-2022	**Novembro**	**Dezembro**	**Janeiro**	**Fevereiro**	**Março**
Atividade 1					
Atividade 2					
Atividade 3	X				
Atividade 4	X	X			
Atividade 5		X	X	X	

2 Fundamentação Teórica e Tecnologias Utilizadas

2.1 Introdução

O acelerômetro utilizado é o MPU-6050 (Figura 2), um tipico acelerômetro MEMS (do inglês, Micro-Electromechanical Sensor) (ZHANG et al., 2021) de baixo custo com 6 eixos. Apesar de pequeno, ele segue os mesmos padrões de outros sensores e pode ser substituído por alternativos. A designação de 6 eixos se deve por ele medir aceleração linear e aceleração angular em 3 eixos cada, com seus acelerômetros e giroscópios.

Figura 2 – Acelerômetro MPU-6050.

O funcionamento físico do acelerômetro se dá principalmente pelo efeito piezoelétrico, que é a propriedade que alguns materiais exibem de emitir corrente ao sofrer deformação (efeito direto) e sofrer deformação ao ser imersos em um campo elétrico (efeito reverso), e a força inercial de Coriolis, força inercial que para referenciais em movimento de rotação, é responsável pelo movimento aparentemente curvo (para o referencial) de um ponto inercial que se afasta do centro de rotação em linha reta.

Para entender o funcionamento dos acelerômetros, observe a Figura 3 de um cubo de materiais piezoelétricos com uma esfera dentro. Conforme o cubo acelera, a força pressiona a esfera contra as paredes do cubo, gerando uma corrente pelo efeito piezoelétrico que podemos medir. Já para o giroscópio, se utiliza o efeito piezoelétrico reverso, onde o material piezoelétrico é imerso em um campo elétrico, que causa uma corrente no material,

que por sua vez causa uma vibração. Ao rotacionar o material, a força da rotação causa uma vibração diferente daquela já estabelecida, que por sua vez causa uma corrente que pode ser medida (OLIVEIRA et al., 2018).

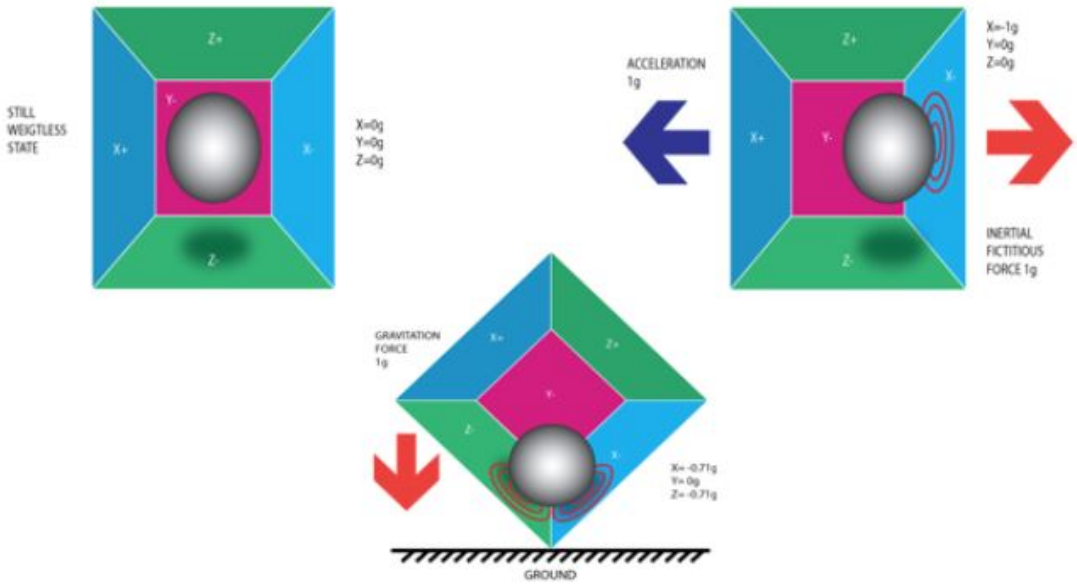

Figura 3 – Funcionamento físico de um acelerômetro piezoelétrico (OLIVEIRA et al., 2018).

2.2 Especificação Técnica

O acelerômetro utilizado é o modelo MPU6050 (INVENSENSE, 2013a) de 6 eixos, um acelerômetro tipico de baixo custo, assim chamado pois contém um giroscópio de 3 eixos e um acelerômetro de 3 eixos. Ele também tem um DMP (Digital Motion Processor - Processador Virtual de Movimento) que auxilia nos cálculos das leituras, e consegue trabalhar diretamente com sensores externos na porta auxiliar de comunicação I2C (protocolo de comunicação muito utilizado que requer apenas 2 fios (MANKAR et al., 2014)).

Há um giroscópio e acelerômetro para cada eixo (X ,Y, Z), todos com tamanho de 16 bits e com limite de valores (resolução) programáveis. Os giroscópios tem resolução de ±250, ±500, ±1000 e ±2000°/sec (degraus por segundo) e os acelerômetros tem resolução de ±2g, ±4g, ±8g e ±16g onde g é a aceleração da gravidade. O sensor de temperatura também é de 16 bits e mede de -40 a 85ºC com um offset de 35 º. Todos eles apresentam sensibilidade em LSB/medida, sendo então necessário multiplicar o valor que ele registra pela sensibilidade para obter o valor real.

A tensão de operação é entre 2.375 a 3.46 V e a corrente varia de acordo com o modo

de operação, onde o modo mais intensivo com giroscópio, acelerômetro e DMP ligados consome 3.9 mA e o menos intensivo consome 5 μA no modo passivo. Na Tabela 2 são resumidas as informações sobre o tipo de dados obtidos.

Tabela 2 – Especificações dos tipos de dados dos sensores.

Sensor	Precisão	Resolução	Sensitividade	Outros
Giroscópio	16 bits	±250 º/seg ±500 º/seg ±1000 º/seg ±2000 º/seg	131 LSB/(º/s) 65,5 LSB/(º/s) 32,8 LSB/(º/s) 16,4 LSB/(º/s)	Eixos: X Y Z
Acelerômetro	16 bits	±2g ±4g ±8g ±16g	16384 LSB/g 8192 LSB/g 4096 LSB/g 2048 LSB/g	Eixos: X Y Z
Temperatura	16 bits	-40 a 85 ºC	340 LSB/ºC	Offset de 35ºC

2.2.1 Uso dos registradores

A leitura e configuração do acelerômetro MPU-6050 deve ser realizada através da manipulação dos seus registradores (INVENSENSE, 2013b). Uma sequência para a inicialização do sensor utilizando o I2C é:

1. Ler o registrador WHO_AM_I no endereço hexadecimal 0x75. Se conseguir ler 0x68, a conexão foi estabelecida.

2. Escrever 0x00 para o registrador PWR_MGMT_1 em 0x6B , acordando o sensor e utilizando como relógio o relógio interno de 8MHz.

3. Configurar a taxa de amostragem desejada escrevendo 1 byte para SMPLRT_DIV em 0x19. A fórmula para amostragem é:

$$\text{Taxa de amostragem} = \frac{\text{8kHz se DLPF desligado senão 1kHz}}{1+SMPLRT\_DIV} \tag{2.1}$$

4. Configurar o Filtro Digital Passa Baixa (DLPF) se necessário para remover frequências abaixo do especificado (reduz ruído) escrevendo nos três bits menos significativos (DLPF_CFG) do registrador CONFIG em 1A de acordo com a Figura 4.

DLPF_CFG	Accelerometer (F_s = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

Figura 4 – Tabela de configuração de DLPF_CFG.

5. Configure se necessário os lmites do giroscópio e do acelerômetro, escrevendo nos bits 3 e 4 (FS_SEL[1:0]) dos registradores GYRO_CONFIG (0x1B) e ACCEL_CONFIG(1C) respectivamente. Refira-se a Tabela 3.

Tabela 3 – Configuração de resolução dos sensores.

FS_SEL[1:0]	**Resolução Giro**	**Resolução Acel**
00	±250 º/seg	±2g
01	±500 º/seg	±4g
10	±1000 º/seg	±8g
11	±2000 º/seg	±16g

Terminada a inicialização, pode-se ler as medições em binário direto nos registradores conforme a Tabela 4, sendo dividido por sensor em byte mais significativo (MSB) seguido pelo byte menos significativo (LSB).

Tabela 4 – Distribuição de registradores para sensores e eixos.

Sensor	Eixo	**MSB**	**LSB**
Acelerômetro	X	ACCEL_XOUT_H 0x3B	ACCEL_XOUT_L 0x3C
Acelerômetro	Y	ACCEL_YOUT_H 0x3D	ACCEL_YOUT_L 0x3E
Acelerômetro	Z	ACCEL_ZOUT_H 0x3F	ACCEL_ZOUT_L 0x40
Termômetro		TEMP_OUT_H 0x41	TEMP_OUT_L 0x42
Giroscópio	X	GYRO_XOUT_H 0x43	GYRO_XOUT_L 0x44
Giroscópio	Y	GYRO_YOUT_H 0x45	GYRO_YOUT_L 0x46
Giroscópio	Z	GYRO_ZOUT_H 0x47	GYRO_ZOUT_L 0x48

Outros métodos que podem ser mais uteis: Configurando o registrador FIFO de 1024 bytes para colocar automaticamente as medições na pilha, pode se configurar uma interrupção quando ele enche e capturar os dados, ou vigiar o o tamanho da pilha e ler

assim que tiver o tamanho certo (Ex: para os 7 sensores seria um pacote de 14 bytes). Para o DMP embarcado há bibliotecas em C++ com rotinas para utilizar o DMP e o MPU 6050 em geral (ELECTRONICCATS, 2019). Essas rotinas serão utilizadas durante o trabalho juntamente com a biblioteca do MPU 6050 do Arduíno (ELECTRONICCATS, 2019) e seu uso é melhor visto no Apêndice A.

2.3 Circuito Utilizado

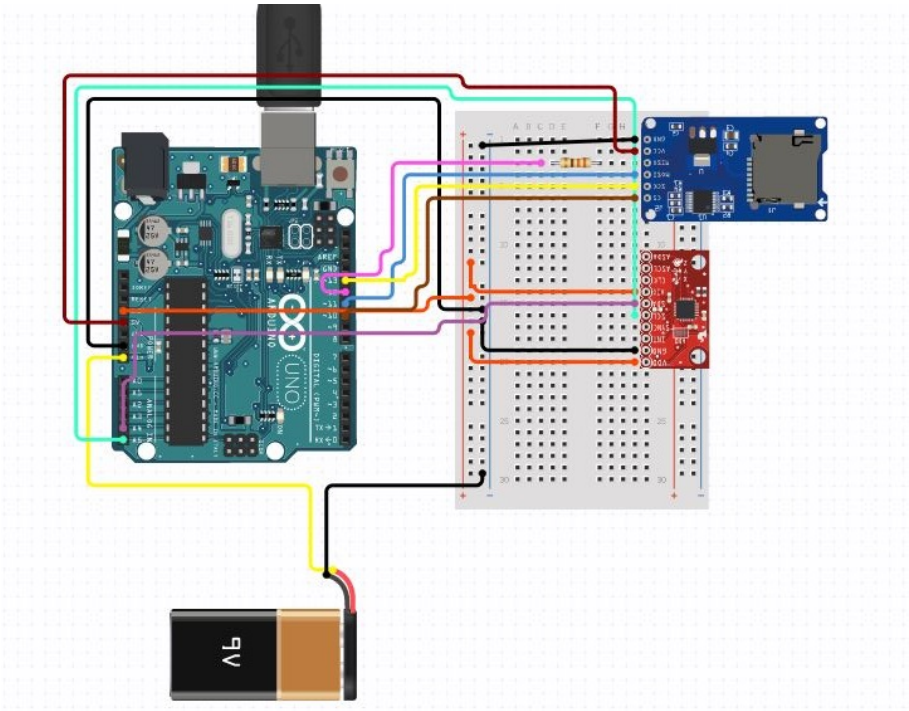

Figura 5 – Circuito utilizado para coleta de dados.

Para utilização do acelerômetro será utilizado o circuito da Figura 5 para leitura e armazenamento dos dados. O circuito consiste em:

- 1 Arduíno Uno;
- 1 Bateria 9V com encaixe para Arduíno;
- 1 Sensor MPU-6050;
- 1 Modulo de cartão Micro-SD Arduíno;
- Fios;
- Protoboard;

Esse é apenas o utilizado nesse trabalho, e outras formas de leitura, armazenamento, fornecimento de energia e microcontroladores poderiam ser usados, desde que cumprissem as

condições de funcionamento do MPU-6050 e as condições de amostragem e armazenamento de dados do projeto.

2.3.1 Código e Bibliotecas Usadas

O Arduíno é um ecossistema de desenvolvimento que contem uma IDE (do inglês, Integrated Development Environment) e um grande número de placas de desenvolvimento que são programadas em C/C++ com diversas bibliotecas feitas pela comunidade. Neste trabalho serão usadas as bibliotecas de SD e Wire do Arduino (ARDUINO-LIBRARIES, 2015), a biblioteca I2C-DEV (ROWBERG, 2014) , a biblioteca do MPU-6050 (ELECTRONICCATS, 2019) e uma versão ligeiramente alterada do projeto de registro de dados para MPU-6050 (ANMOLIO, 2016). O código se resume a iniciar o sensor e o módulo de cartão Micro-SD, configurar o sensor para gerar uma interrupção quando as leituras estiverem prontas, ler e escrever as leituras juntamente com o tempo decorrido desde o inicio do programa para um arquivo de texto no cartão Micro-SD. Por padrão as leituras são da primeira configuração de limite, como especificado na Tabela 2, mas isso pode ser mudado. A frequência de leitura é 400kHz. O código utilizado pode ser visto no Apêndice A.

2.4 Testes

Como o sensor apresenta mais de uma configuração de resolução (ver a Seção 2.2), serão feitos os mesmos testes para cada combinação de resolução dos acelerômetros e giroscópios. Seria feito o teste em um veículo com velocímetro além do sensor a ser testado, e ambos seriam registrados para comparação. Os percursos de teste seriam:

- Trilha reta no asfalto;
- Trilha reta em terra;
- Trilha curva no asfalto;
- Trilha curva na terra;

Para as 4 configurações de resolução do acelerômetro e do giroscópio do sensor teríamos então 4 trilhas x 4 resoluções do giroscópio x 4 resoluções do acelerômetro = 48 casos de teste. Cada caso deve levar em torno de 20 minutos, com taxa de amostragem inicialmente em torno de 35 ms entre amostras, podendo ser ajustado conforme necessário. Esses dados no entanto, necessitam tratamento para lidar com ruído de discretização e amostragem, problemas de resolução, além de métodos de integração para obter os dados de velocidade para comparar.

2.5 Tratamento de dados

Ao pegar os dados de um sensor, nem sempre é possível utilizar-los diretamente por diversos motivos, os mais relevantes são:

1. Formato dos dados;
2. Ruído(Erro) dos dados;

Conforme visto na Seção 2.2 o giroscópio e acelerômetro apresentam suas leituras mapeadas para 2 registradores de 8 bits em complemento de 2. Ao ler esses registradores, é necessário converte-los para um número com sinal (preferencialmente com precisão maior de 16 bits como o *double* do C++), dividir o resultado pela sensitividade da resolução configurada, e para o acelerômetro se necessário, converter de g para m/s^2 onde g é a gravidade do planeta Terra (INVENSENSE, 2013a). Agora os dados estão em ^{o}/s e m/s^2.

Tendo agora os dados em encontramos primeiramente com o problema de discretização, no qual para os 16 bits alocados, a resolução escolhida pode não ter a precisão certa para as medidas, onde uma resolução muito grande registra valores baixos ou com pouca variação, e uma resolução pequena frequentemente chega até o máximo, perdendo dados maiores que os limites. Para resolver isso é preciso estimar os limites do que será medido e escolher a resolução acordo, além de usar números com precisão maior que a original para guardar os dados. Tendo os dados em um formato trabalhável, podemos então integrá los para obter a velocidade no período, porém durante esse processo de integração enfrentamos o problema de ruído, que atrapalha a integração. É preciso então a aplicação de um filtro ou algoritmo que limpe esse ruído. O próprio MPU-6050 tem um filtro digital passa-baixa embutido, mas o filtro passa-baixa apenas elimina baixas frequências e introduz atraso, nos testes iremos explorar e comparar outras opções de obter a velocidade da aceleração medida:

- Filtro de Kalman: O filtro de Kalman é um sistema de duas etapas para sistemas discretos, predição e correção, baseado em técnicas recursivas do espaço de estados do sistema (OLIVEIRA; GONÇALVES, 2017). Dado variáveis iniciais do sistema, o filtro faz uma previsão dos próximos estados das variáveis. Essa estimativa é comparada com a estimativa baseada nas medições anteriores ao estado atual e a medição do estado atual em si, e usada para corrigir as previsões. Isso é então feito para os próximos estados sucessivamente. Após algumas aplicações do filtro, as predições ficam mais precisas e se torna possível isolar o ruído pelas medições que se afastam da predição do filtro e da predição feita com medidas anteriores. Sendo removido o ruido, pode-se integrar normalmente as medições para obter a velocidade.

- Integração com trapézios: Como o nome diz, funciona avaliando a área do gráfico da função como uma serie de trapézios. Para um sistema discreto como o desse projeto, isso significa que a integral da aceleração entre os pontos a e b medidos é a diferença entre a medição em a e b dividida pelo tempo entre os dois. Obtém se então o gráfico da velocidade aplicando esse método as medições sequencialmente.

- Integração por triângulos (BROM, 2013): Utiliza se um ponto intermediário xi, e se integra somando as áreas dos triângulos formados por $[(xi, 0), (xi, f(xi)), (xn, f(xn))]$ para cada ponto xn. A Figura 6 ajuda a visualizar o processo.

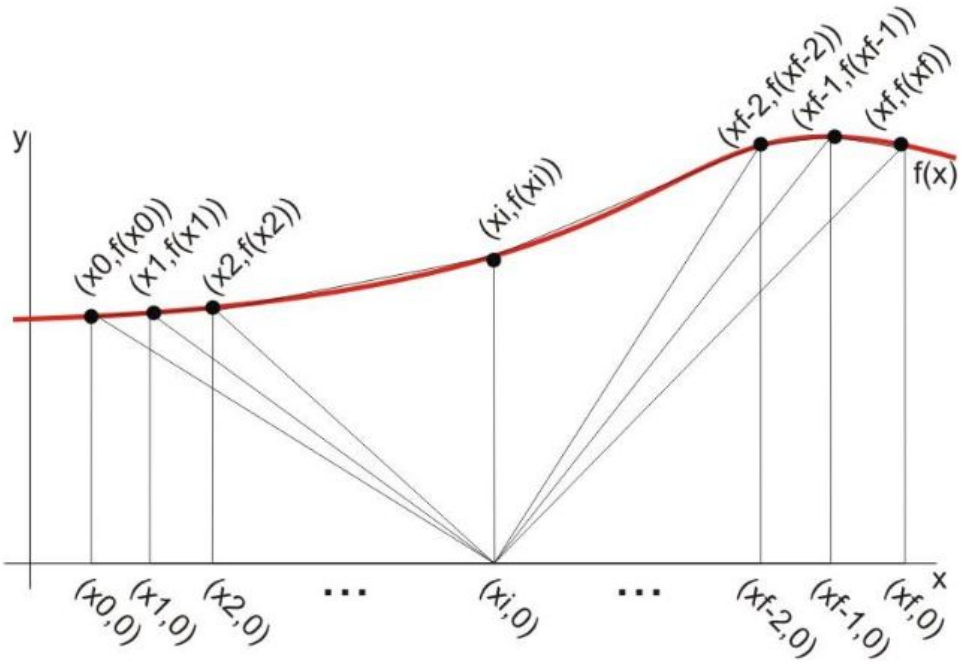

Figura 6 – Triângulos usados na integração por triângulos.

Todos os três algoritmos podem ser aplicados através de ferramentas de matemática computacional como Matlab. Será então comparado a eficiência desses métodos para tratamento dos casos de teste.

Referências

ANMOLIO. *Anmolio/Mpu6050-Datalogging*. 2016. Disponível em: <https://github.com/anmolio/mpu6050-datalogging>. Citado na página 10.

ARDUINO-LIBRARIES. *SD library for Arduino*. Arduino, 2015. Disponível em: <https://github.com/arduino-libraries/SD>. Citado na página 10.

BROM, P. C. Integração numérica por soma de áreas de triângulos: Regra dos triângulos repetidos. *REVISTA EIXO*, v. 2, n. 1, p. 53–68, 2013. Citado na página 12.

CADE, D. E. et al. Determining forward speed from accelerometer jiggle in aquatic environments. *Journal of Experimental Biology*, v. 221, n. 2, 01 2018. ISSN 0022-0949. Jeb170449. Disponível em: <https://doi.org/10.1242/jeb.170449>. Citado na página 3.

ELECTRONICCATS. *MPU6050 Arduino Library*. 2019. Disponível em: <https://github.com/ElectronicCats/mpu6050>. Citado 2 vezes nas páginas 9 e 10.

INVENSENSE, I. Mpu-6000 and mpu 6050 product specification revision 3.4. *United States*, 2013. Citado 2 vezes nas páginas 6 e 11.

INVENSENSE, I. Mpu-6000 and mpu-6050 register map and descriptions revision 4.2. *United States*, 2013. Citado na página 7.

MANKAR, J. et al. Review of i2c protocol. *International Journal of Research in Advent Technology*, Citeseer, v. 2, n. 1, 2014. Citado na página 6.

OLIVEIRA, V. et al. Proposta de reconstrução de trajetória de pigs utilizando componentes não convencionais. In: . [S.l.: s.n.], 2018. Citado na página 6.

OLIVEIRA, W. dos S.; GONÇALVES, E. N. Implementação em c: filtro de kalman, fusão de sensores para determinação de ângulos. *ForScience*, v. 5, n. 3, 2017. Citado na página 11.

REIS, E. L. dos. Velocímetros–quando o travamento da agulha pode retratar a velocidade de colisão. *Revista Brasileira de Criminalística*, v. 5, n. 3, p. 39–48, 2016. Citado na página 3.

ROWBERG, J. I2cdevlib. mpu-6050 6-axis accelerometer/gyroscope. *Publicación electrónica: http://www. i2cdevlib. com/devices/mpu6050 Consultada*, v. 12, n. 01, 2014. Citado na página 10.

SONG, C. K.; UCHANSKI, M.; HEDRICK, J. K. *Vehicle speed estimation using accelerometer and wheel speed measurements*. [S.l.], 2002. Citado na página 3.

ZHANG, C. et al. Estimation of the vehicle speed using cross-correlation algorithms and mems wireless sensors. *Sensors*, v. 21, n. 5, 2021. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/21/5/1721>. Citado 2 vezes nas páginas 2 e 5.

A Código Utilizado

Listagem A.1 – Código utilizado no projeto.

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
//
//
// Changelog:
//      2013-05-08 - added seamless Fastwire support
//                 - added note about gyro calibration
//      2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility error
//      2012-06-20 - improved FIFO overflow handling and simplified read process
//      2012-06-19 - completely rearranged DMP initialization code and simplification
//      2012-06-13 - pull gyro and accel data from FIFO packet instead of reading directly
//      2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING
//      2012-06-05 - add gravity-compensated initial reference frame acceleration output
//                 - add 3D math helper file to DMP6 example sketch
//                 - add Euler output and Yaw/Pitch/Roll output formats
//      2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)
//      2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250
//      2012-05-30 - basic DMP initialization working

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/
```

```
43
44 // I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
45 // for both classes must be in the include path of your project
46 #include "I2Cdev.h"
47 #include <SPI.h>
48 #include <SD.h>
49 #include "MPU6050_6Axis_MotionApps20.h"
50 #include "Servo.h"
51 #include "Wire.h"
52 //#include "MPU6050.h" // not necessary if using MotionApps include file
53 Servo myservo;
54 int pos=0;
55 //#include "MPU6050.h" // not necessary if using MotionApps include file
56
57 // Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
58 // is used in I2Cdev.h
59 #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
60     #include "Wire.h"
61 #endif
62
63 // class default I2C address is 0x68
64 // specific I2C addresses may be passed as a parameter here
65 // AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
66 // AD0 high = 0x69
67 MPU6050 mpu;
68 //MPU6050 mpu(0x69); // <-- use for AD0 high
69
70 /* =========================================================================
71    NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
72    depends on the MPU-6050's INT pin being connected to the Arduino's
73    external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
74    digital I/O pin 2.
75  * ========================================================================= */
76
77 /* =========================================================================
78    NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
79    when using Serial.write(buf, len). The Teapot output uses this method.
80    The solution requires a modification to the Arduino USBAPI.h file, which
81    is fortunately simple, but annoying. This will be fixed in the next IDE
82    release. For more info, see these links:
83
84    http://arduino.cc/forum/index.php/topic,109987.0.html
85    http://code.google.com/p/arduino/issues/detail?id=958
86  * ========================================================================= */
87
88
89
90 // uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
91 // quaternion components in a [w, x, y, z] format (not best for parsing
92 // on a remote host such as Processing or something though)
93 //#define OUTPUT_READABLE_QUATERNION
94
95 // uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
96 // (in degrees) calculated from the quaternions coming from the FIFO.
97 // Note that Euler angles suffer from gimbal lock (for more info, see
98 // http://en.wikipedia.org/wiki/Gimbal_lock)
99 //#define OUTPUT_READABLE_EULER
```

```
100
101 // uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
102 // pitch/roll angles (in degrees) calculated from the quaternions coming
103 // from the FIFO. Note this also requires gravity vector calculations.
104 // Also note that yaw/pitch/roll angles suffer from gimbal lock (for
105 // more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
106 #define OUTPUT_READABLE_YAWPITCHROLL
107
108 // uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
109 // components with gravity removed. This acceleration reference frame is
110 // not compensated for orientation, so +X is always +X according to the
111 // sensor, just without the effects of gravity. If you want acceleration
112 // compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
113 //#define OUTPUT_READABLE_REALACCEL
114
115 // uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
116 // components with gravity removed and adjusted for the world frame of
117 // reference (yaw is relative to initial orientation, since no magnetometer
118 // is present in this case). Could be quite handy in some cases.
119 #define OUTPUT_READABLE_WORLDACCEL
120
121 // uncomment "OUTPUT_TEAPOT" if you want output that matches the
122 // format used for the InvenSense teapot demo
123 //#define OUTPUT_TEAPOT
124
125
126 const int chipSelect = 4;
127 #define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
128 bool blinkState = false;
129
130
131 // for calculating the time since when the arduino started
132  double timing;
133  double timing1;
134 // MPU control/status vars
135 bool dmpReady = false;  // set true if DMP init was successful
136 uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
137 uint8_t devStatus;      // return status after each device operation (0 = success
      , !0 = error)
138 uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
139 uint16_t fifoCount;     // count of all bytes currently in FIFO
140 uint8_t fifoBuffer[64]; // FIFO storage buffer
141
142 // orientation/motion vars
143 Quaternion q;           // [w, x, y, z]         quaternion container
144 VectorInt16 aa;         // [x, y, z]            accel sensor measurements
145 VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor
      measurements
146 VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor
      measurements
147 VectorFloat gravity;    // [x, y, z]            gravity vector
148 float euler[3];         // [psi, theta, phi]    Euler angle container
149 float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and
      gravity vector
150 boolean start=0;        // used for the first attempt to calculate elapsed time
151 // packet structure for InvenSense teapot demo
152 uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n
```

```
    ' };
153
154 ////////////////////////////////////  RC Transmitter
    ////////////////////////////////////
155
156 int pin1 = 7;
157 int pin2 = 8;
158 int pin3 = 9;
159 int pin4 = 10;
160 unsigned long duration1, duration2, duration3,duration4;
161
162 // ================================================================
163 // ===                  INTERRUPT DETECTION ROUTINE               ===
164 // ================================================================
165
166 volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin
    has gone high
167 void dmpDataReady() {
168     mpuInterrupt = true;
169 }
170
171
172
173 // ================================================================
174 // ===                      INITIAL SETUP                         ===
175 // ================================================================
176
177 void setup() {
178
179      pinMode(pin1, INPUT);
180      pinMode(pin2, INPUT);
181      pinMode(pin3, INPUT);
182      pinMode(pin4, INPUT);
183     // join I2C bus (I2Cdev library doesn't do this automatically)
184     #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
185         Wire.begin();
186         TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz). Comment this line
                if having compilation difficulties with TWBR.
187     #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
188         Fastwire::setup(400, true);
189     #endif
190
191     // initialize serial communication
192     // (115200 chosen because it is required for Teapot Demo output, but it's
193     // really up to you depending on your project)
194     Serial.begin(115200);
195     while (!Serial); // wait for Leonardo enumeration, others continue
            immediately
196
197     // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio
198     // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
199     // the baud timing being too misaligned with processor ticks. You must use
200     // 38400 or slower in these cases, or use some kind of external separate
201     // crystal solution for the UART timer.
202
203     // initialize device
204     Serial.println(F("Initializing I2C devices..."));
```

```
205     mpu.initialize();
206
207     // verify connection
208     Serial.println(F("Testing device connections..."));
209     Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F
            ("MPU6050 connection failed"));
210
211     delay(150);
212
213     // load and configure the DMP
214     Serial.println(F("Initializing DMP..."));
215     devStatus = mpu.dmpInitialize();
216 ////////////////////////////////////////////////////////////////
        ////////////////////////////////////////////////////////////////
217     Serial.print("Initializing SD card...");
218
219   // see if the card is present and can be initialized:
220   if (!SD.begin(chipSelect)) {
221     Serial.println("Card failed, or not present");
222     // don't do anything more:
223     return;
224   }
225   Serial.println("card initialized.");
226 ////////////////////////////////////////////////////////////////
        ////////////////////////////////////////////////////////////////
227 //valores dos offsers em ac gy xyz -1607, -1055, 357, 111, 100, -63
228     // supply your own gyro offsets here, scaled for min sensitivity
229     mpu.setXAccelOffset(-1607);
230     mpu.setYAccelOffset(-1055);
231     mpu.setZAccelOffset(357); // 1688 factory default for my test chip
232     mpu.setXGyroOffset(111);
233     mpu.setYGyroOffset(100);
234     mpu.setZGyroOffset(-63);
235
236     // make sure it worked (returns 0 if so)
237     if (devStatus == 0) {
238         // turn on the DMP, now that it's ready
239         Serial.println(F("Enabling DMP..."));
240         mpu.setDMPEnabled(true);
241
242         // enable Arduino interrupt detection
243         Serial.println(F("Enabling interrupt detection (Arduino external
                interrupt 0)..."));
244         attachInterrupt(0, dmpDataReady, RISING);
245         mpuIntStatus = mpu.getIntStatus();
246
247         // set our DMP Ready flag so the main loop() function knows it's okay to
                use it
248         Serial.println(F("DMP ready! Waiting for first interrupt..."));
249         dmpReady = true;
250
251         // get expected DMP packet size for later comparison
252         packetSize = mpu.dmpGetFIFOPacketSize();
253     } else {
254         // ERROR!
255         // 1 = initial memory load failed
256         // 2 = DMP configuration updates failed
```

```
257         // (if it's going to break, usually the code will be 1)
258         Serial.print(F("DMP Initialization failed (code "));
259         Serial.print(devStatus);
260         Serial.println(F(")"));
261     }
262 
263     // configure LED for output
264     pinMode(LED_PIN, OUTPUT);
265     myservo.attach(9);
266 }
267 
268 int complete=0;
269 double timing_sec;
270 
271 // ================================================================
272 // ===                      MAIN PROGRAM LOOP                     ===
273 // ================================================================
274 
275 void loop() {
276 
277       ////////////////////////////////////// RC Transmitter Values
            ////////////////////////////
278 
279 
280 
281 
282 
283 
284     // if programming failed, don't try to do anything
285     if (!dmpReady) return;
286 
287     // wait for MPU interrupt or extra packet(s) available
288     while (!mpuInterrupt && fifoCount < packetSize) {
289         // other program behavior stuff here
290         // .
291         // .
292         // .
293         // if you are really paranoid you can frequently test in between other
294         // stuff to see if mpuInterrupt is true, and if so, "break;" from the
295         // while() loop to immediately process the MPU data
296         // .
297         // .
298         // .
299     }
300 
301     // reset interrupt flag and get INT_STATUS byte
302     mpuInterrupt = false;
303     mpuIntStatus = mpu.getIntStatus();
304 
305     // get current FIFO count
306     fifoCount = mpu.getFIFOCount();
307 
308     // check for overflow (this should never happen unless our code is too
        inefficient)
309     if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
310         // reset so we can continue cleanly
311         mpu.resetFIFO();
```

```
312         // otherwise, check for DMP data ready interrupt (this should happen
                frequently)
313     } else if (mpuIntStatus & 0x02) {
314         // wait for correct available data length, should be a VERY short wait
315         while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
316
317         // read a packet from FIFO
318         mpu.getFIFOBytes(fifoBuffer, packetSize);
319
320         // track FIFO count here in case there is > 1 packet available
321         // (this lets us immediately read more without waiting for an interrupt)
322         fifoCount -= packetSize;
323
324
325
326
327
328
329
330
331
332       /*  #ifdef OUTPUT_READABLE_QUATERNION
333             // display quaternion values in easy matrix form: w x y z
334             mpu.dmpGetQuaternion(&q, fifoBuffer);
335             Serial.print("quat\t");
336             Serial.print(q.w);
337             Serial.print("\t");
338             Serial.print(q.x);
339             Serial.print("\t");
340             Serial.print(q.y);
341             Serial.print("\t");
342             Serial.println(q.z);
343         #endif
344
345         #ifdef OUTPUT_READABLE_EULER
346             // display Euler angles in degrees
347             mpu.dmpGetQuaternion(&q, fifoBuffer);
348             mpu.dmpGetEuler(euler, &q);
349             Serial.print("euler\t");
350             Serial.print(euler[0] * 180/M_PI);
351             Serial.print("\t");
352             Serial.print(euler[1] * 180/M_PI);
353             Serial.print("\t");
354             Serial.println(euler[2] * 180/M_PI);
355         #endif
356 */
357         #ifdef OUTPUT_READABLE_YAWPITCHROLL
358             // display Euler angles in degrees
359
360             if (complete==0)
361             myservo.write(20);
362
363             if (complete==1)
364             myservo.write(90);
365
366
367             mpu.dmpGetQuaternion(&q, fifoBuffer);
```

```
368             mpu.dmpGetGravity(&gravity, &q);
369             mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
370
371             Serial.print("ypr  time\t");
372             Serial.print(ypr[0] * 180/M_PI);
373             Serial.print("\t");
374             Serial.print(ypr[1] * 180/M_PI);
375             Serial.print("\t");
376             Serial.print(ypr[2] * 180/M_PI);
377             Serial.print("\t");
378             timing = millis();
379             if (start == 0)
380             { timing1=timing;
381               start=1;
382             }
383             timing=timing-timing1;
384             timing_sec = timing/1000;
385             Serial.println(timing_sec);
386             if((timing_sec)/5.0 == 0.0)  complete=1;
387             if((timing_sec)/5.0 == 0.0 && (timing_sec)/10.0 == 0.0)  complete=0;
388         #endif
389
390   /*      #ifdef OUTPUT_READABLE_REALACCEL
391             // display real acceleration, adjusted to remove gravity
392             mpu.dmpGetQuaternion(&q, fifoBuffer);
393             mpu.dmpGetAccel(&aa, fifoBuffer);
394             mpu.dmpGetGravity(&gravity, &q);
395             mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
396             Serial.print("areal\t");
397             Serial.print(aaReal.x);
398             Serial.print("\t");
399             Serial.print(aaReal.y);
400             Serial.print("\t");
401             Serial.println(aaReal.z);
402         #endif
403 */
404         #ifdef OUTPUT_READABLE_WORLDACCEL
405             // display initial world-frame acceleration, adjusted to remove
                    gravity
406             // and rotated based on known orientation from quaternion
407             mpu.dmpGetQuaternion(&q, fifoBuffer);
408             mpu.dmpGetAccel(&aa, fifoBuffer);
409             mpu.dmpGetGravity(&gravity, &q);
410             mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
411             mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
412             Serial.print("aworld\t");
413             Serial.print(aaWorld.x);
414             Serial.print("\t");
415             Serial.print(aaWorld.y);
416             Serial.print("\t");
417             Serial.println(aaWorld.z);
418         #endif
419 /*
420         /*#ifdef OUTPUT_TEAPOT
421             // display quaternion values in InvenSense Teapot demo format:
422             teapotPacket[2] = fifoBuffer[0];
423             teapotPacket[3] = fifoBuffer[1];
```

```
424             teapotPacket[4] = fifoBuffer[4];
425             teapotPacket[5] = fifoBuffer[5];
426             teapotPacket[6] = fifoBuffer[8];
427             teapotPacket[7] = fifoBuffer[9];
428             teapotPacket[8] = fifoBuffer[12];
429             teapotPacket[9] = fifoBuffer[13];
430             Serial.write(teapotPacket, 14);
431             teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
432         #endif
433 */
434         // blink LED to indicate activity
435         blinkState = !blinkState;
436         digitalWrite(LED_PIN, blinkState);
437 
438       /*  duration1 = pulseIn(pin1, HIGH);
439 duration2 = pulseIn(pin2, HIGH);
440         duration3 = pulseIn(pin3, HIGH);
441         duration4 = pulseIn(pin4, HIGH);
442   Serial.print("R-P-T-Y");
443               Serial.print("\t \t");
444               Serial.print(duration1);
445               Serial.print("\t");
446               Serial.print(duration2);
447               Serial.print("\t");
448               Serial.print(duration3);
449               Serial.print("\t");
450               Serial.print(duration4);
451               Serial.print("\t");
452         */
453     }
454 
455 ////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////
456 // make a string for assembling the data to log:
457   String dataString = "";
458 
459   // real world accel e roll pitch yaw
460   dataString+=String(aaWorld.x);
461   dataString += " ";
462   dataString+=String(aaWorld.y);
463   dataString += " ";
464   dataString+=String(aaWorld.z);
465   dataString += " ";
466   for (int count = 0; count < 3; count++) {
467     //int sensor = analogRead(analogPin);
468     dataString += String(ypr[count]);
469     if (count < 3) {
470       dataString += " ";
471     }
472   }
473    dataString += String(timing_sec);
474 
475   // open the file. note that only one file can be open at a time,
476   // so you have to close this one before opening another.
477   File dataFile = SD.open("datalog.txt", FILE_WRITE);
478 
479   // if the file is available, write to it:
```

```
480   if (dataFile) {
481     dataFile.println(dataString);
482     dataFile.close();
483
484   }
485   // if the file isn't open, pop up an error:
486   else {
487     Serial.println("error opening datalog.txt");
488   }
489
490 }
```