Coordenação do Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

Sistema Inteligente de Sinalização de Segurança em Veículos Automotores

Alfred Förster Junior
Henrique André da Silva

Orientador: Prof. Dr. Dalton Pedroso de Queiroz

Dourados – MS
2014

Coordenação do Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

Sistema Inteligente de Sinalização de Segurança em Veículos Automotores

Trabalho apresentado na disciplina de Projeto Final de Curso, pelos alunos Alfred Förster Junior e Henrique André da Silva, do Curso de Ciência da Computação, da Universidade Estadual de Mato Grosso do Sul, sob orientação do Prof. Dr. Dalton P. de Queiroz, como parte integrante dos requisitos para obtenção do título de Bacharel em Ciência da Computação.

Dourados – MS
2014

Sistema Inteligente de Sinalização de Segurança em Veículos Automotores

Alfred Förster Junior e Henrique André da Silva

BANCA EXAMINADORA:

_____

Profº Dr. Dalton Pedroso de Queiroz (Docente UEMS)
Orientador

_____

Prof. Dr. Antônio Aparecido Zanfolim (Docente UEMS)

_____

Profª Msc. Jessica Bassani de Oliveira (Docente UEMS)

Dourados, ____/____/____

Dourados-MS
2014

.

Dedico este Trabalho de Conclusão de Curso aos meus pais Alfred Förster e Matilde Jorge dos Reis Förster, à minha irmã Thais Jessica Reis Förster.

Alfred Förster Junior

Dedico este Trabalho de Conclusão de Curso aos meus pais Aloísio André e Maridete Pereira da Silva André

Henrique André da Silva

"Comece fazendo o necessário, depois o que é possível, e de repente você estará fazendo o impossível."

São Francisco de Assis

"Quando acho que cheguei ao ponto máximo, descubro que é possível superá-lo."

Ayrton Senna

"Somos insignificantes. Por mais que você programe sua vida, a qualquer momento tudo pode mudar."

Ayrton Senna

AGRADECIMENTOS

Primeiramente a Deus, pelo dom da sabedoria e principalmente nos mostrar o caminho do conhecimento.

Eu, Alfred, gostaria de agradecer particularmente aos meus pais, Alfred Förster e Matilde Jorge dos Reis Förster, a minha irmã, Thais Jessica Reis Förster, pela força e por nunca deixarem eu me desanimar no decorrer deste curso.

Eu, Henrique, gostaria de agradecer particularmente aos meus pais, Aloísio André e Maridete Pereira da Silva André, pelo apoio no decorrer deste curso, não deixando que os obstáculos que enfrentei fossem maiores que minha perseverança.

Ao nosso orientador, o Prof. Dr. Dalton Pedroso de Queiroz, pela orientação que nos foi prestada.

Enfim, agradecemos a todas as pessoas que de uma forma ou de outra contribuíram na consecução desse projeto, pois foram muitas. Especialmente, agradecemos ao Adilson Barison, pois sem sua grande contribuição este projeto não seria desenvolvido.

Alfred Förster Junior
Henrique André da Silva

RESUMO

Neste projeto foi desenvolvido um sistema inteligente para acionamento e desligamento automático da sinalização de segurança de uma motocicleta. Para criar essa funcionalidade, foi usado um giroscópio GY-521 e a plataforma Arduino, por ser uma plataforma de prototipagem barata que se adapta tanto a pequenos como a grandes projetos. A integração inicial deles foi realizada primeiramente a título de testes, utilizando o código de domínio público disponibilizado no site oficial da plataforma Arduino. Foi feita a verificação quanto à sensibilidade do giroscópio em um intervalo de consideração de giro. Após isso, desenvolveu-se a outra parte do sistema, que consistia basicamente de LEDs (representando as setas) e botões (que acionam os LEDs). Nessas conexões foi necessária a utilização de resistores por causa das diferenças de tensão em partes do sistema. Após os testes, o sistema eletrônico foi feito em uma placa de circuito impresso, onde houve necessidade de se temporizar o sistema com uma pausa na casa de milissegundos, para sincronização do mesmo e minimização de ruídos espúrios. A motivação para a realização deste trabalho foi de contribuir no sentido de minimizar acidentes de trânsito com motocicletas, pois, com o aumento da procura por motocicletas como alternativa para transportes mais rápidos e baratos, o índice de acidentes no trânsito causados por má sinalização das motocicletas aumentou acentuadamente. Uma das principais causas desses acidentes é que quando os motociclistas se esquecem de desligar as setas de suas motos, após suas conversões, elas continuam sinalizando uma intenção falsa de que a moto vai convergir em algum sentido e isso leva outros motoristas a interpretarem essa sinalização como válida, manobrando seus veículos de acordo com a sinalização indevida e aumentando então o risco de haver uma colisão.

Palavras-Chave: Segurança no Trânsito, Sistemas Inteligentes, Arduino, Giroscópio.

Sumário

LISTA DE FIGURAS

LISTA DE CÓDIGO

INTRODUÇÃO

Na busca por maior mobilidade urbana e principalmente devido à economia de combustível e preços mais acessíveis, os usuários buscam na motocicleta uma alternativa aos transportes coletivos lotados. Dados do Departamento Nacional de Transito (DENATRAN) apontam que entre os anos de 2001 e 2011 houve um aumento de 300% das motocicletas no trânsito no Brasil, fazendo com que esse número saltasse de 4.611.301 unidades em 2001 para 18. 442.413 unidades em 2011. Segundo dados da Federação Nacional da Distribuição de Veículos Automotores (FENABRAVE), as estatísticas recentes apontam que há um aumento constante nesses números, visto que em junho de 2014 o número de motocicletas saltou para 22.315.382 unidades. Especificamente em Mato Grosso do Sul, havia nesse último período aproximadamente 403.294 motocicletas, o que corresponde a aproximadamente 31,5% da frota total de veículos, que era de 1.280.394 veículos (DETRAN-MS, 2014).

Este forte aumento no número de motocicletas poderia ser comemorado, se não estivesse ligado a outro dado alarmante: o exponencial aumento do número de acidentes e mortes, envolvendo motocicletas e até mesmo sendo causados por elas.

Segundo dados do Sistema de Informações de Mortalidade (SIM), criado pelo Ministério da Saúde, em 10 anos o número de mortes de usuários de motos aumentou 263,5%, sendo que em 2011 foram 11.268 mortes contra 3.100 em 2001. Este aumento é maior que a porcentagem geral de mortos. Através de uma parceria entre a Faculdade de Medicina da Universidade de São Paulo (FMUSP), o Hospital das Clínicas da Faculdade de Medicina da Universidade de São Paulo (HCFMUSP) e a Associação Brasileira dos Fabricantes de Motocicletas, Ciclomotores, Motonetas, Bicicletas e Similares (ABRACICLO), foi realizado um estudo sobre o assunto na Zona Oeste do município de São Paulo, o qual apontou que, em 37% dos acidentes, a responsabilidade é do motociclista. Outro dado preocupante que este estudo, é que em 49% dos acidentes registrados o principal responsável pelo acidente foi o motociclista, e nesses casos 88% a causa foi imprudência.

Dados mais recentes divulgados pelo DETRAN – MS apontam que no município de Dourados – MS, no mês de julho de 2014 houve 172 acidentes com vítimas e destes, 82 foram com motociclistas, correspondendo a cerca de 48% do total de acidentes.

Buscando diminuir o número de acidentes, diversas entidades realizam campanhas para conscientizar os motociclistas e poder assim diminuir o número de acidentes. Outras procuram criar equipamentos e acessórios para que se possa diminuir pelo menos o impacto que sofre o motoqueiro em uma situação de acidente de trânsito. Um desses equipamentos é a jaqueta airbag, que é acionada após o motociclista ser "lançado" de sua moto. Temos nas motos, outros equipamentos, como é o caso dos freios ABS, Controle de Estabilidade, e Sensor de Inclinação [AirBag Denko,2014] [Bosh,2014].

Outro problema que ocorre em relação às motocicletas é com relação à sinalização de manobra, isto é, o desligamento das "setas" das motos que não são automatizadas, ficando a cargo do condutor da moto acioná-las e desativá-las quando inicia e termina manobras de conversão no trânsito. É muito comum o motoqueiro acionar a "seta" e depois esquecer-se de desligá-la, o que contribui para que mais acidentes ocorram, pois os outros motoristas, assim como os pedestres, não tem como saber que a sinalização é indevida, fazendo manobras de acordo com ela, o que geralmente leva a um acidente.

Nesse sentido, uma contribuição que se poderia dar para aumentar a segurança no trânsito com relação ao problema exposto anteriormente, seria desenvolver um sistema automático de desligamento das "setas" da moto como ocorre com os automóveis. O projeto ora desenvolvido pretendeu implementar um Sistema de desligamento Automatizado de Setas. Este sistema fará o desligamento automático da seta, após acionada pelo motoqueiro, quando terminada a conversão.

Para que esta ação ocorra, o sistema proposto foi feito usando um dispositivo chamado giroscópio. Este dispositivo permite que seja realizada a verificação do momento em que houve a finalização da conversão, ou seja, quando a motocicleta retornará a linha reta. Essa informação é repassada a um sistema de controle, que acionará o desligamento automático. Para consecução desse sistema automático foi usada a plataforma Arduino.

I – CONCEITOS TEÓRICOS

1.1 - Segurança no Trânsito com Motocicletas

Os equipamentos de trânsito preconizados para o motociclista pelo Código Trânsito Brasileiro (CTB) são equipamentos básicos e de proteção individual. Como equipamentos de proteção individual temos: capacete com viseira ou óculos de proteção, sapato fechado e roupa adequada, sendo que por "roupa adequada" entende-se jaqueta de couro, macacão e calça comprida. São considerados equipamentos básicos de segurança no trânsito os retrovisores, faixas reflexivas, sinalizadores laterais, alças laterais, revestimento para o cano da descarga e mata-cachorro, que é um equipamento utilizado na parte inferior da motocicleta que evita o contato do piloto ou do passageiro com o motor [TEIXEIRA et al., 2014].

Ainda constam outras medidas no Código Brasileiro de Trânsito sobre o uso seguro de motocicletas.

> *" Art. 57. Os ciclomotores devem ser conduzidos pela direita da pista de rolamento, preferencialmente no centro da faixa mais à direita ou no bordo direito da pista sempre que não houver acostamento ou faixa própria a eles destinada, proibida a sua circulação nas vias de trânsito rápido e sobre as calçadas das vias urbanas.*
>
> *Parágrafo único. Quando uma via comportar duas ou mais faixas de trânsito e a da direita for destinada ao uso exclusivo de outro tipo de veículo, os ciclomotores deverão circular pela faixa adjacente à da direita.*
>
> *Art. 58. Nas vias urbanas e nas rurais de pista dupla, a circulação de bicicletas deverá ocorrer, quando não houver ciclovia, ciclofaixa, ou acostamento, ou quando não for possível a utilização destes, nos bordos da pista de rolamento, no mesmo sentido de circulação regulamentado*

> *para a via, com preferência sobre os veículos automotores." (BRASIL, 1997).*

O descaso dos condutores e o uso inadequado de equipamentos estão associados com um alto índice de lesões, sendo o capacete é o único meio de minimizar traumas crânio encefálico sofrido por motociclistas. Roupas protetoras que incluem calçados fechados, protetores para as pernas, devem proporcionar certa proteção ao motociclista de maneira a minimizar lesões, principalmente de tecidos moles [DEBIEUX et al., 2010].

Em relação à sinalização de conversão, as motocicletas são as menos contempladas com a tecnologia de desligamento automático de setas. Por questões técnicas, não há implementado nesses veículos automotores a mesma facilidade que se encontra presente nos automóveis, ou seja, o desligamento automático da seta após uma conversão. Então, muitos motoqueiros se esquecem de desligar suas setas manualmente, o que pode levar a colisões no trânsito, pois os outros motoristas não tem como saber que aquela sinalização é indevida, fruto de esquecimento do condutor da moto. Nesse sentido, tecnologias que ajudem a implantar nas motos também esse sistema de desligamento automático de setas irão contribuir com uma maior segurança no trânsito.

1.2 - Giroscópio

> *"O giroscópio é um dispositivo que fornece a inclinação de um objeto em relação a um eixo definido. É um dispositivo de extrema importância na fabricação de foguetes, satélites, sondas espaciais, aplicações marítimas e outros." (SILVA JR, SANTOS, 2005, p. 1).*

O primeiro giroscópio foi o pião, brinquedo muito conhecido das crianças. No século XIX, na França, Jean-Bernard-León Foucault (1819-1868) criou experimentos que provaram a rotação da terra e dos objetos que sobre ela se movem. Já em 1850 ele conseguiu provar a manutenção da posição original de uma roda em uma estrutura de articulação mesmo com a rotação da terra. Mas somente após o ano de 1900 foi que os

cientistas encontraram utilidade para a sua ideia, empregando o giroscópio em um ambiente submarino, onde os sistemas de navegação são inúteis.

Anos depois, nos Estados Unidos, o inventor Elmer Ambrose Sperry (1860-1930) empregou o giroscópio no primeiro piloto automático, cuja função era de manter o curso de um navio.

Na Segunda Guerra Mundial, o giroscópio (até então de dois quadros) foi aperfeiçoado para 3 quadros de direção, sendo muito utilizados em foguetes, mísseis e aviões sem piloto [Encyclopedia.com, 2001].

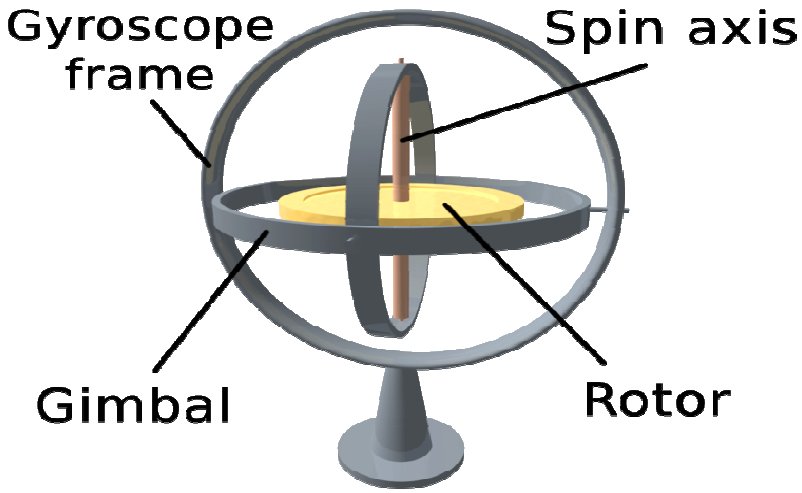

Figura 1. Esquema de um Giroscópio. Fonte: Encyclopedia.com.

De forma geral, o giroscópio contém uma roda, chamada de rotor, envolta de dois ou mais quadros circulares orientados ao longo de eixos direcionais diferentes entre si, podendo ser inclinado em um ângulo qualquer, conforme esquema mostrado na Figura 1.

Esses quadros são chamados de *gimbals* e permitem a livre movimentação da roda no centro da estrutura do giroscópio. Os quadros são montados fixos a uma base (chamada de *Gyroscope frame*) e a peça que liga o *rotor* com os quadros circulares é chamada *Spin axis*.

O eixo *rotor* aponta continuamente para sua posição original criando uma inércia giroscópica. O giroscópio só mantém sua inércia quando sua velocidade de giro é elevada e sua massa concentrada na borda do *rotor*.

Quando uma pressão é aplicada sobre um giroscópio provocando um giro no mesmo, isso é chamado de precessão. Mesmo mínima, uma força sempre será aplicada na construção de um giroscópio. A taxa de precessão varia de acordo com a força e sua

direção, no momento da inércia do *rotor* e da velocidade angular do *rotor*, a taxa é diretamente proporcional a força e inversamente proporcional ao momento e a velocidade. A resistência a força (precessão) irá continuar até que a direção do plano de rotação coincida com o da força.

Um giroscópio tem que ser construído de material rígido, ou seja, as moléculas que compõem seu corpo nunca alteram suas posições em relação uma das outras, não causando assim interferência nos movimento rotacionais do giroscópio. Outra característica do giroscópio é o torque, que é a força que gira um corpo qualquer em torno de algum de seus eixos [Encyclopedia.com, 2001].

Os giroscópios modernos são implementados em sistemas eletrônicos, como elementos de sensoriamento. Um exemplo desses sistemas é O MPU-6050, que é um sensor *InvenSense* (empresa fabricante do circuito integrado) que conta com um acelerômetro *MEMS* (Sistemas Micro Eletromecânico) e um giroscópio *MEMS*. Esse giroscópio, montando-se em um placa de circuito integrado, possui o nome de GY-521, que capta sinais de três canais ao mesmo tempo, denominados canais x, y e z, com uma precisão de 16 bits analógicos de conversão digital para cada canal. O MPU-6050 tem a funcionalidade de se comunicar com um sistema de controle através dos pinos SDA (Dados Seriais) e SDL (Clock Serial) com barramento I²C (Circuito Inter-Integrado). No caso do projeto ora proposto, o sistema de controle utilizado foi o Arduino.

Os CIs da *InvenSense* fornecem interface de sensores de movimento com algoritmos totalmente integrados e robustos de firmware *MotionFusion* ™. Os *MotionTracking* ™, usados no CI GY-521, oferecem fácil e direta integração de sensores de movimento com dispositivos, gerando baixo custo de desenvolvimento.

> *"O MPU-6050, é um dispositivo MotionTracking de 6 eixos, que combina 3 eixos de giroscópio, 3 eixos de acelerômetro e um Processador Digital de Movimento (DMP), tudo embutido em um pequeno espaço de 4x4x0.9 mm. Com o sensor I²C dedicado, ele aceita diretamente entradas de uma bússola de 3 eixos externo para fornecer uma completa saída de 9 eixos MotionFusion.*
>
> *O dispositivo MotionTracking MPU-6050, com o seus 6 eixos de integração calibra em tempo de*

> *execução o firmware, permitem que os fabricantes eliminem o custo e complexidade de seleção, qualificação a nível de sistema de dispositivos discretos, garantindo desempenho de movimento ideal para os consumidores. O MPU-6050 também é projetado para interagir com múltiplos sensores não-inerciais digitais, tais como sensores de pressão, em sua porta I²C auxiliar." [InvenSense, 2014]*.

A placa GY-521 possui um regulador de tensão, porque ao utilizar 3.3V para VCC, a tensão resultante pode ser baixa, o que pode provocar um mau desempenho do I2C-bus. É preferível aplicar 5V para o pino VCC do placa de sensores, pois a placa possui resistores pull-up no I²C-bus. O valor desses resistores pull-up são, por vezes, 10k e às vezes 2k2. O 2k2 é bastante baixo, se o resistor for ele, a tensão resultante será muito baixa, principalmente se a placa estiver combinada com outras placas [Playground, 2014].

A placa trabalha com tensões entre 2.375V-3.46V no pino VCC. Além de desse pino, o MPU6050 possui um pino que especifica os níveis lógicos do I2C-bus, esse pino é chamado de VLOGIC e ele trabalha a 1.8V, somada ou subtraída 5% da tensão vinda do pino VCC.

Um buffer de 1024 Byte FIFO (Primeiro que Entra, Primeiro que Sai) no chip economiza energia do sistema, com isso o processador do sistema lê os dados do sensor em rajadas e depois entra em um modo de baixa energia para, posteriormente, o MPU recolher mais dados. Como o MPU-6050 possui processadores e sensores que reconhecem os mais variados tipos de movimentos, ele acaba permitindo apenas aplicações *MotionInterface* (Interface de Movimento entre dispositivos e sensores) de baixa potência em aplicações portáteis com requisitos de processamento reduzidos para o processador do sistema. Como possui uma saída *MotionFusion* (algoritmos de Firmware) integrada, o DMP na MPU-60X0 descarrega os requisitos *MotionProcessing* (Algoritmos de Processamento de movimentos) de cálculo intensivo do processador do sistema, diminuindo a frequência de leitura da saída do sensor de movimento [InvenSense, 2014].

A funcionalidade de interrupção é configurada através do registo de configuração de interrupção. Itens que podem ser configurados incluem a configuração do pino INT, a interrupção de travamento e método de compensação dispara a interrupção. Os itens que pode provocar uma interrupção são: (1) Gerador de *clock*, que bloqueia para novo oscilador de referência (usado quando se muda de relógio); (2) os novos dados estão disponíveis para serem lidos (a partir dos registros FIFO e dados); (3) evento acelerômetro interrompe; e (4) o MPU-6050 não recebeu o reconhecimento de um sensor auxiliar no I2C-Bus secundário. O estado de interrupção pode ser lido a partir do registo de status de interrupção.

O I2C é composto por duas saídas seriais: os sinais de dados (SDA) e o sinal de relógio (SCL). Essas duas linhas de conexão tem transmissão bidirecionais e para um dispositivo ser o mestre basta ele colocar o endereço do dispositivo escravo no canal de comunicação [InvenSense, 2014].

> *"O MPU-6050 sempre atua como escravo do sistema e suas linhas SDA e SCL sempre precisam de resistores pull-up, funcionando assim a 400kHz. O endereço do escravo do MPU-60X0 é b110100X e é de 7 bits. A LSB bit do endereço de 7 bits é determinada pelo nível lógico no pino AD0. Isso permite que os dois MPU-60X0s possam ser conectados ao mesmo barramento I²C. Quando utilizado nesta configuração, o endereço de um dos dispositivos deve ser b1101000 (pino AD0 nível baixo) e o endereço da outra deve ser b1101001 (pino AD0 nível alto)". [InvenSense, 2014]*.

O MPU-6050 suporta Comunicações I2C tanto nos seus pinos primários (microprocessador) de interface serial quanto na sua Interface auxiliar (XDA e XCL). No caso de usar as duas interfaces, a tensão de entrada e saída usada é a VLOGIC, onde o AUX_VDDIO é alterado para 0 (valor power-on-reset). [InvenSense, 2014].

A Figura 2 mostra o esquema do CI GY-521 e seus pinos de comunicação externa.

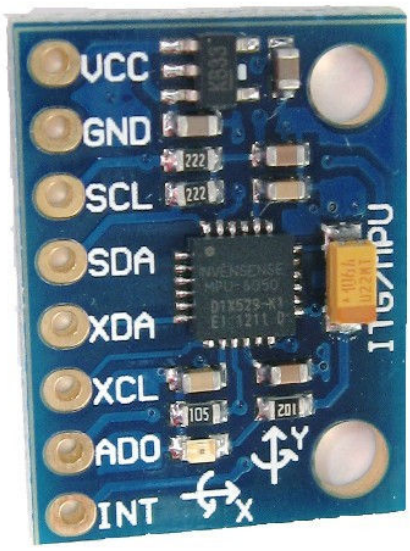

Figura 2. GY-521. Fonte: Arduino Playground.

1.3 – Arduino

O Arduino foi projetado no *Interaction Design Institute* na cidade italiana de **Ivrea**, no ano de 2005. Sua concepção se deu através da procura de um meio barato e de fácil utilização para estudantes de *Design*.

Os principais desenvolvedores do projeto foram além de Banzi e Cuartilles, David Mellis, que trabalhou no desenvolvimento do software e o Engenheiro Gianluca Martino, que auxiliou os estudantes no projeto e produziu a tiragem inicial que foram de aproximadamente 200 peças. As primeiras peças foram rapidamente vendidas e produziu-se mais peças melhorando e introduzindo novas versões. A versão oficial do Arduino superou a marca de 300 mil unidades vendidas.

O nome Arduino se deu em referência a um bar que frequentado por professores e acadêmicos do instituto.

O Arduino, segundo McRoberts (2011), é um pequeno computador que você pode programar para processar entradas e saídas entre o dispositivo e os componentes externos que são conectados a ele. Esses dispositivos podem ser conectados a ele tanto por meio de suas entradas digitais ou analógicas. O processamento das informações é realizado a partir do microprocessador *Atmel AT*. Além destes componentes também possui um cristal oscilador e um regulador linear de 5V.

Existem vários tipos de Arduino que foram produzidos, dentre eles temos o Arduino Uno que foi apresentado ao público em 2010 e sua grande diferença para os anteriores é o microprocessador, o *Duemilanove* é uma das placas mais populares, o

Mega é um dos mais novos produzidos, ele fornece um maior número de funções de entrada e saída. Ele gerou algumas outras variações como, por exemplo, o *Lilypad Arduino* que foi produzido para ser incorporado a tecidos e sua principal diferença para os demais é a sua velocidade de processamento que é inferior. Outro exemplo é o *Paperduino* que é um clone do Arduino onde não há placa de circuito impresso, seus componentes são ligados em uma folha de papel.

1.3.1 – Arduino Uno

O Arduino utilizado neste projeto foi o Arduino Uno, que utiliza o microprocessador ATmega238. Além deste microprocessador, possui as seguintes características:

- 14 pinos de entrada/saída digitais, dentre os quais 6 podem ser utilizados como saída PWM;
- 16 pinos de entrada/saída Analógicos;
- Ressonador Cerâmico de 16 MHZ;
- *Jack* para alimentação;
- Um cabeçalho ICSP;
- Botão para reset;
- Chip controlador *Atmega* 16U2 programado para converter USB para Serial.

A Figura 3 também encontrada no Anexo B, mostra um esquema dos pinos do Arduino Uno.

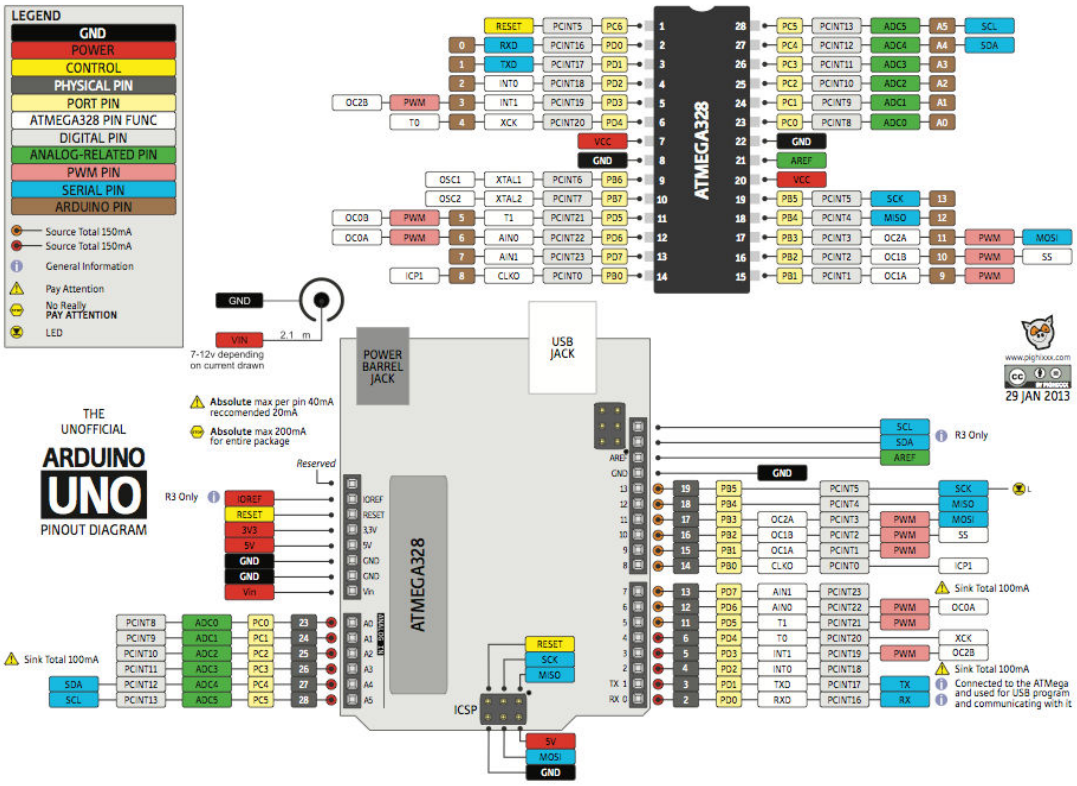

Figura 3: Esquema dos pinos do Arduino Uno. Fonte: PighiXXX.

Na figura 3 observamos a distribuição da pinagem do Arduino Uno, onde do lado esquerdo se localizam os pinos analógicos e a alimentação (5V, 3.3V, e 2 GNDs) e, no lado direito, os pinos digitais, onde alguns podem ser utilizados como PWM.

O Arduino uno pode ser ligado tanto através da sua entrada USB, bem como pela entrada externa que ele possui. Pela entrada externa, pode receber uma tensão entre 6V e 20V, no entanto segundo Souza (2014) em palestra ministrada durante a X LatinoWare (Conferencia Latino Americana de Software Livre) é recomendado a utilização de tensão entre 7V e 12V, visto que alguns testes apontaram que quando é fornecido tensão abaixo de 7V, o pino de saída de 5V pode fornecer tensão menor; e quanto a tensão é maior que 15V, há possibilidade do regulador de tensão superaquecer.

1.3.2 – A IDE Arduino

O Arduino traz juntamente com seu Hardware, um ambiente de desenvolvimento para a geração de *sketchs,* ou seja, os programas que serão que serão embarcadas no microcontrolador.

Esta IDE foi desenvolvida em Linguagem Java, baseando-se nas linguagem *Processing* e *Wiring*, e na biblioteca AVR-gcc, onde esta biblioteca serve para a programação de microcontroladores AVR.

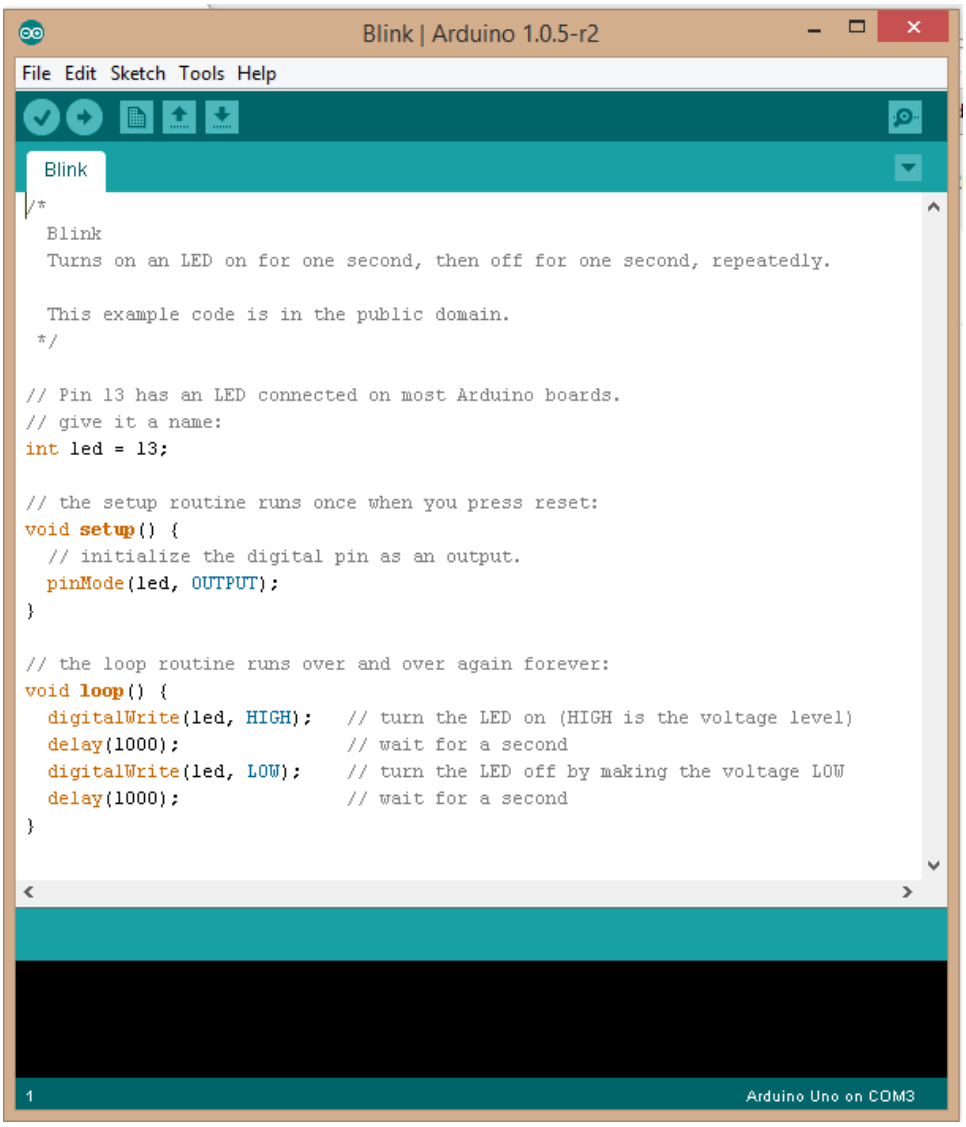

Figura 4: IDE Arduino. Fonte: autores.

A Figura 4 apresenta a IDE de programação com o exemplo do que é chamado de *"Hello Word do Arduino"*, o código que contém instruções para um led ficar piscando.

Nesta IDE, a aba *FILE*, pode se encontrar algumas opções como salvar o código, e alguns exemplos prontos, sendo que há exemplos de diversas aplicações como mostra a Figura 5.

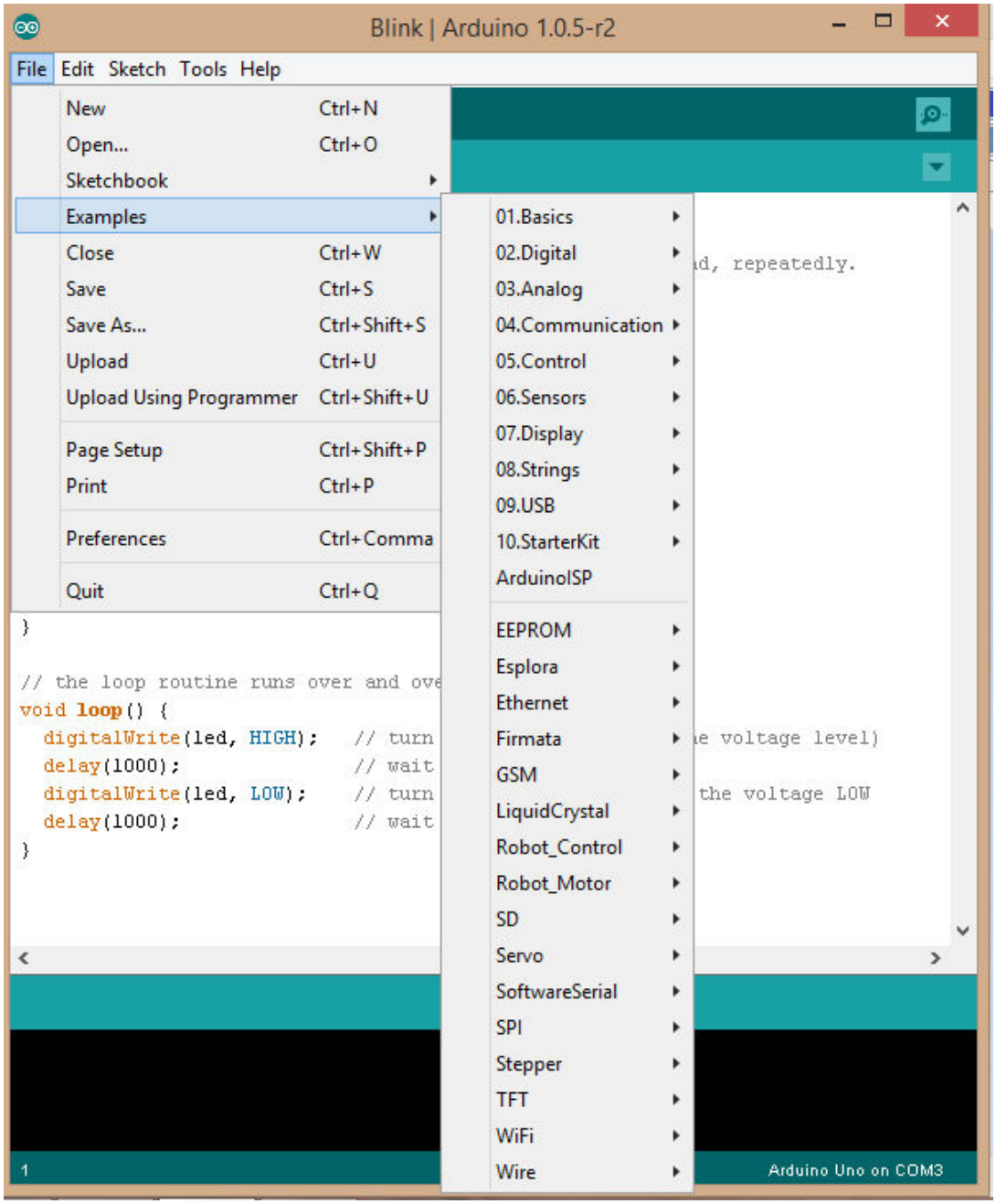

Figura 5: IDE aba FILE. Fonte: autores.

Na figura 5 podemos observar as categorias que possuem códigos exemplos, sendo os mesmos muito importantes para o aprendizado.

Outra aba principal é a *TOOLS* que contem dentre suas opções, opção para selecionar o modelo de Arduino e selecionar a porta de comunicação. Nesta aba também está presente a opção *Serial Monitor*. Este é um recurso para auxiliar na troca de dados entre o computador e a placa, sendo de bem simples compreensão, onde em sua interface deve-se apenas selecionar a taxa de transferência. A Figura 6 apresenta esta a tela do *Serial Monitor*.

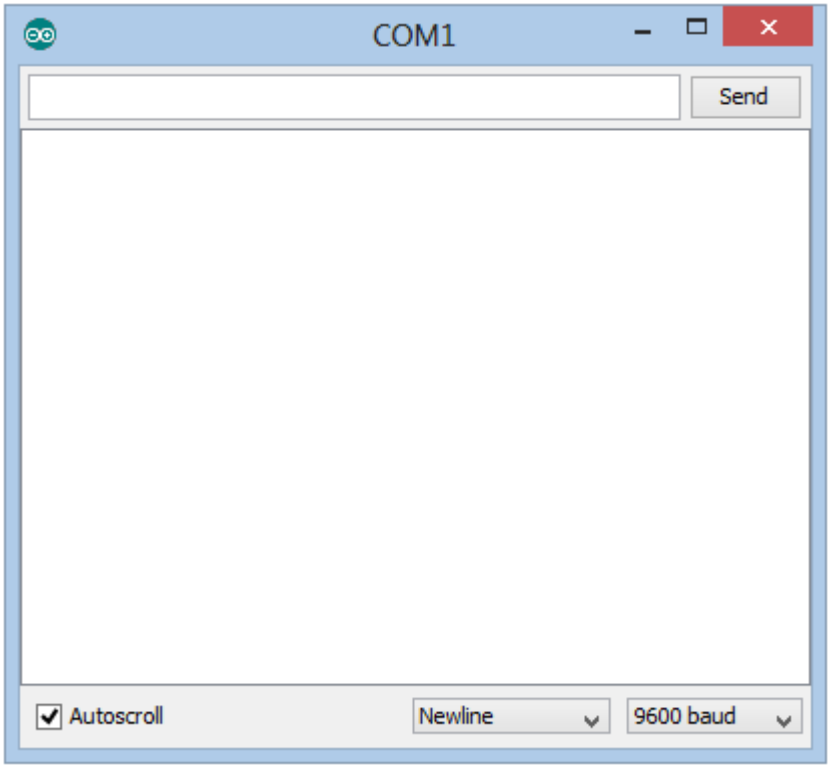

Figura 6: Tela do Serial Monitor. Fonte: os autores.

A codificação dos programas do Arduino, possuem 3 partes: estruturas, valores e funções.

A estrutura é dividida em duas funções: *Setup* e *Loop*.

A função *Setup* é chamada no inicio da execução, sendo usada para realizar a inicialização de variáveis, bibliotecas, etc. Ela é chamada apenas na inicialização ou quando pressiona o botão de *reset*.

A função *Loop* é chamada após a inicialização, realizando loops de execução do programa.

A estrutura está organizada desta forma para facilitar a programação, pois o foco em seu desenvolvimento não foi para pessoas já habituadas a ela, e sim para estudantes que muitas vezes nunca tiveram contato prévio com técnicas de programação de computadores [Souza, 2014]. Na figura 4, que apresenta a tela da IDE com uma codificação exemplo, pode-se ver esta estrutura, onde na função setup é inicializado o pino 13 como saída para o LED.

Outra parte presente, e muito necessária na codificação, são os valores. Estes valores são as definições de variáveis e de constantes.

Também as bibliotecas são elementos muito importantes, bastando que os desenvolvedores de componentes para Arduino criem uma biblioteca documentada para que a programação seja facilitada.

Na codificação usada neste projeto está presente a biblioteca *Wire* que realiza a comunicação com os componentes a partir do protocolo I2C, protocolo este criado pela *Philips* Semicondutores para transmissão de dados de forma digital, utilizando apenas duas linhas para realizar as transmissões: a SDA (linha de serial de dados) e SCL (linha serial de clock). Especificamente o Arduino Uno utiliza as portas A4 (SDA) e A5 (SCL) para realizar a comunicação com os dispositivos através deste protocolo. Esta biblioteca apresenta algumas funções utilizadas, dentre elas temos:

- *Wire.begin*(): Inicializa a comunicação I2C;
- *Wire.requestFrom*(): Utilizada para requisitar dados ao dispositivo de comunicação;
- *Wire.beginTransmission*(endereço): Começa a transmissão de dados ao dispositivo, sendo passado como parâmetro o endereço deste dispositivo;
- *Wire.endTransmission*(): Utilizada para finalizar a transmissão de dados com o protocolo I2C;
- *Wire.write*(): utilizada para gravar os dados do dispositivo, é utilizada entre as chamadas das funções *beginTransmission* e *endTransmission*.
- *Wire.read*(): Realiza a leitura dos bytes transmitidos pelo dispositivo.
- *Wire.available*(): Retorna o número de bytes disponíveis para a função read obter.

Outras funções foram utilizadas, dentre elas:

- *Serial.begin*(): Define a taxa de transferência em bits por segundo para a transmissão de dados serial, sendo utilizada a taxa de 9600 bits por segundo
- *Serial.print*(): É utilizada para impressão dos dados recebidos pelo Arduino na porta serial, transformando-os em caractere ASCII legível.
- *Serial.println*(): Possui característica semelhante à função *Serial.print*, sendo que após a finalização da impressão posiciona o cursor no inicio da linha seguinte.
- *abs*(): Retorna o valor absoluto do parâmetro
- *delay*(): Realiza a pausa do programa no tempo em milissegundos especificado como parâmetro.

- *pinMode*(): Define o pino especificado como entrada ou saída.
- *digitalRead*(): Realiza a leitura da tensão em valor booleano de determinado pino digital.
- *digitalWrite*(): Realiza a escrita de determinado valor. Estes valores por definição são escrito como *HIGH* (correspondente ao 1 booleano) e *LOW* (correspondente ao valor 0 booleano). O valor *HIGH* corresponde a 5V ou 3.3V e o *LOW* corresponde a 0V.

III. DESENVOLVIMENTO DO SISTEMA

Para o desenvolvimento do sistema, foi utilizado alguns componentes, dentre eles temos o Arduino que foi utilizado devido a sua facilidade de programação e utilização. A utilização do giroscópio deu-se em função a precisão na detecção do giro, e o CI GY-521 que contém o giroscópio inserido no MPU-6050, foi escolhido devido á facilidade de utilização e baixo custo.

3.1. Circuito Eletrônico

A figura 7 mostra o protótipo desenvolvido do circuito eletrônico a partir do aplicativo para desenho de protótipos de circuitos elétricos *Fritzing*.

Primeiramente foi realizada a conexão do Arduino com o GY-521, conforme esquema mostrado na figura 7 (1 e 2). Esta conexão ocorreu através do Protocolo I2C, protocolo desenvolvido pela Philips Semicondutores para conexão com poucos dispositivos. Para realizar a conexão é necessário apenas duas linhas: A linha serial de dados (SDA) e a linha serial de *clock* (SCL).

O Arduino possui dois pinos específicos para a conexão I2C, sendo que no Arduino Uno estas portas são a A4 para a SDA e a A5 para a SCL. Para a codificação é necessária a declaração da biblioteca *Wire*.

Com esta conexão foram realizados testes para comprovar a eficiência do giroscópio para o projeto ora proposto, sendo que, devido à sua alta sensibilidade, houve a necessidade de criar um laço de verificação para criar um *range*, onde se determinou uma variação de valores que não precisariam ser computados.

Feito isto, os LEDs foram conectados, conforme indicado na figura 7 (5 e 6), assim como os botões (3 e 4). Nesta conexão definiram-se os pinos do Arduino, onde o LED que representa a luz de seta para conversão à direita é acionado através do sinal do pino 7 e o que representa a luz para a conversão à esquerda é acionado pelo pino 8. O pino que recebe a informação do botão para acionamento do LED direito foi definido como o pino 10 e para o botão de acionamento do LED esquerdo definiu-se o pino 11. Após a conexão dos botões e LEDs, foi feita a programação das funcionalidades da seta, isto é: o acionamento, o desligamento manual e a troca de setas, ou seja, quando a seta

está ligada para um lado e o condutor deseja modificar o acionamento para o outro lado, basta para o mesmo pressionar o botão equivalente ao lado que quer acionar.

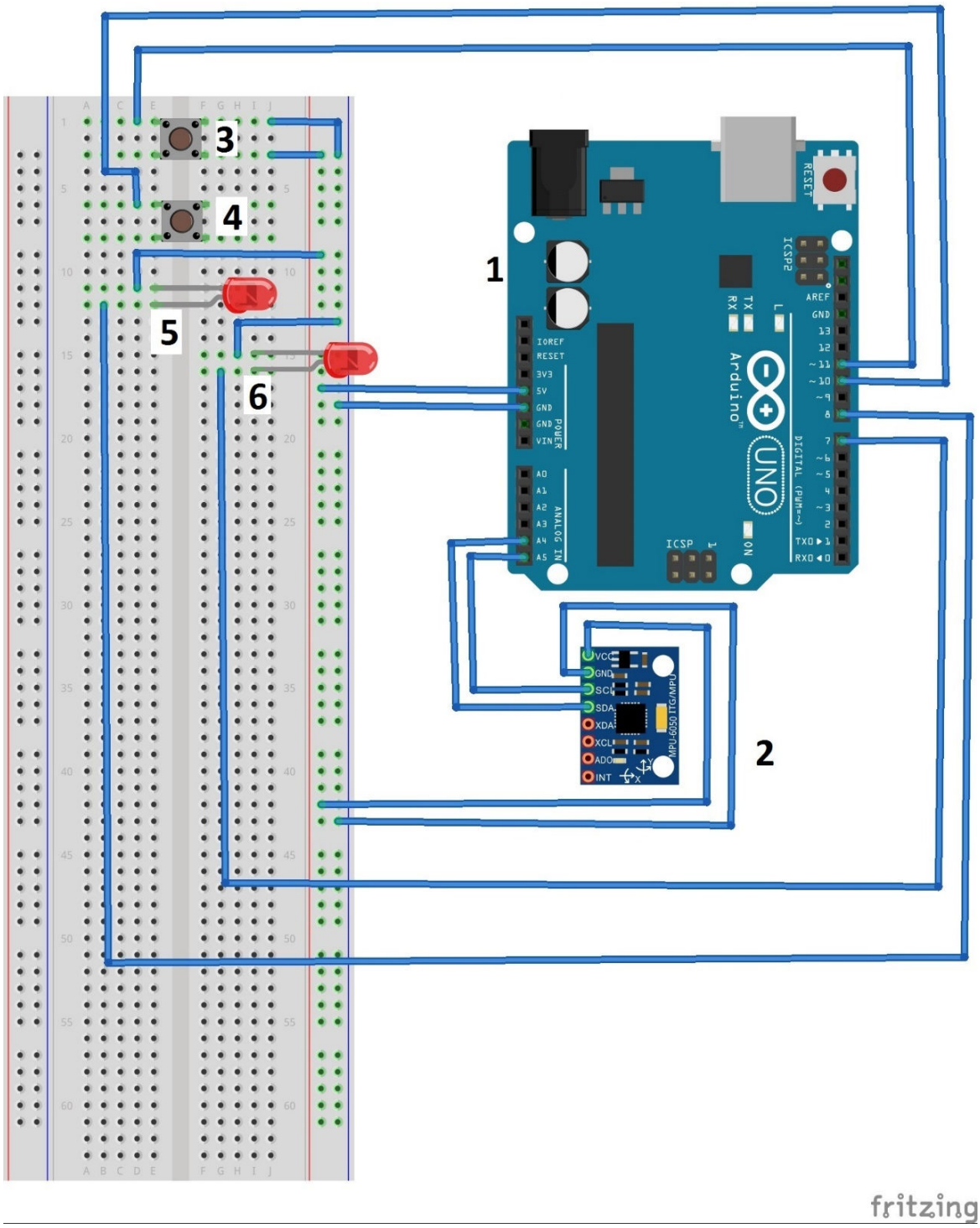

Figura 7: Protótipo do circuito elétrico. Fonte: os autores.

A conexão completa do circuito se deu com a junção entre o GY-521, LEDs e botões, conforme esquema da figura 7, sendo implementada a codificação do mesmo para atuar como um sistema automático de desligamento de setas.

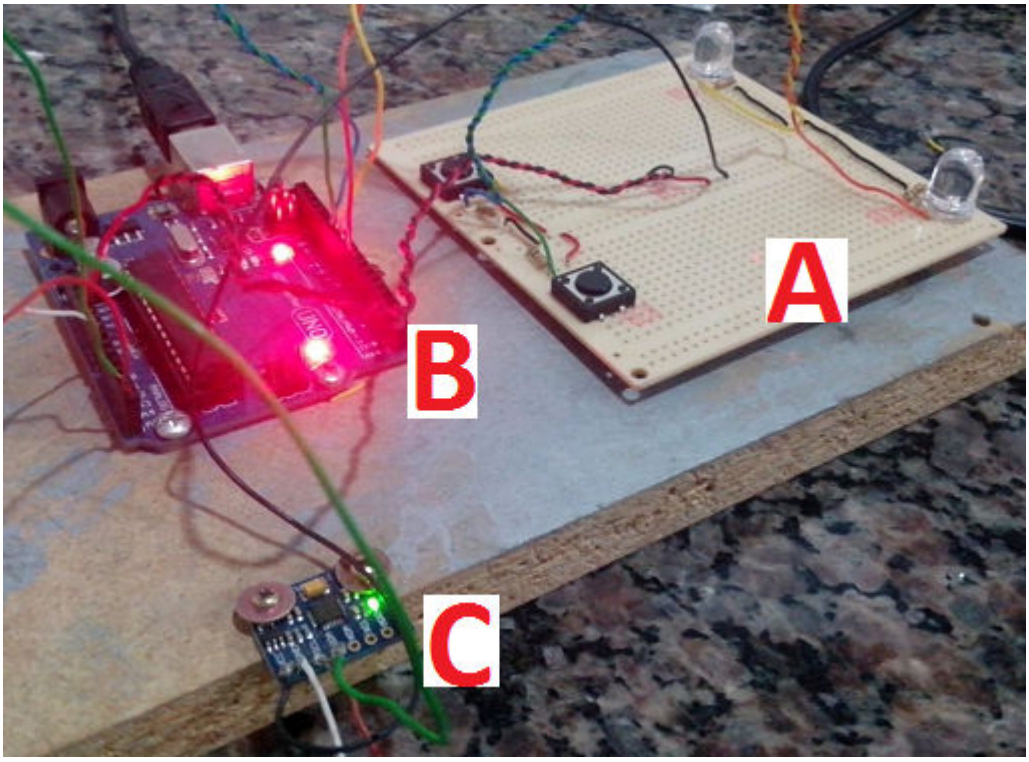

Figura 8: Protótipo implementado. Fonte: os autores.

A figura 8 é uma foto do circuito montado, onde temos: (A) mostra o circuito implementado contendo os LEDs e botões; (B) mostra o Arduino; (C) mostra o GY-521.

3.2 DESCRIÇÃO DO SOFTWARE

Para que o circuito atuasse como um sistema de controle de desligamento de setas, foi implementado um código fonte computacional na IDE do Arduino. As linhas iniciais desse código são baseadas em diretivas, que representam e renomeiam as portas de comunicação do Arduino com o GY-521 e com os registradores do GY-52. Isso para facilitar a manipulação dessas portas no código, porque as diretivas originais do GY-521 para códigos do Arduino possuem nomes muito extensos e pouco intuitivos, acarretando uma dificuldade de manipulação das mesmas. No código 1 são mostradas partes do código dessas diretivas de nomeação.

```
202  // O nome usa o intervalo de escala completa para o acelerômetro.
203  #define MPU6050_AFS_SEL_2G  MPU6050_AFS_SEL_0
204  #define MPU6050_AFS_SEL_4G  MPU6050_AFS_SEL_1
205  #define MPU6050_AFS_SEL_8G  MPU6050_AFS_SEL_2
206  #define MPU6050_AFS_SEL_16G MPU6050_AFS_SEL_3
207
208  #define MPU6050_SLV0_FIFO_EN  MPU6050_D0
209  #define MPU6050_SLV1_FIFO_EN  MPU6050_D1
210  #define MPU6050_SLV2_FIFO_EN  MPU6050_D2
211  #define MPU6050_ACCEL_FIFO_EN MPU6050_D3
212  #define MPU6050_ZG_FIFO_EN    MPU6050_D4
213  #define MPU6050_YG_FIFO_EN    MPU6050_D5
214  #define MPU6050_XG_FIFO_EN    MPU6050_D6
215  #define MPU6050_TEMP_FIFO_EN  MPU6050_D7
216
217  #define MPU6050_I2C_MST_CLK0  MPU6050_D0
218  #define MPU6050_I2C_MST_CLK1  MPU6050_D1
219  #define MPU6050_I2C_MST_CLK2  MPU6050_D2
220  #define MPU6050_I2C_MST_CLK3  MPU6050_D3
221  #define MPU6050_I2C_MST_P_NSR MPU6050_D4
222  #define MPU6050_SLV_3_FIFO_EN MPU6050_D5
223  #define MPU6050_WAIT_FOR_ES   MPU6050_D6
224  #define MPU6050_MULT_MST_EN   MPU6050_D7
```

Código 1: Diretivas.

Além das definições do GY-521, também foram definidas diretivas que renomeiam as portas ligadas aos componentes e informações mais relevantes ao projeto, que são os lados direito e esquerdo, os LEDs direito e esquerdo e os botões das setas direitas e esquerdas, variáveis globais de controle do sistema, tal como mostra o código 2.

```
598  #define DIREITA 1  //definindo o número do lado direito
599  #define ESQUERDA 2  //definindo o número do lado esquerdo
600  #define LEDDIREITO 7  //definindo o número do led direito
601  #define LEDESQUERDO 8  //definindo o número do led esquerdo
602  #define SETADIREITA 10  //definindo o número do botão direito
603  #define SETAESQUERDA 11  //definindo o número do botão esquerdo
```

Código 2: Diretivas de informações

Além de definir diretivas para as portas de comunicação do Arduino com o GY-521, foi definida também uma estrutura de dados para armazenamento dos valores que o giroscópio retorna para o Arduino, que são os valores de variação da aceleração e giro do dispositivo para cada um dos 3 eixos e o valor da temperatura ambiente. Conforme o código 3, os campos da estrutura que armazena os valores finais tem nomes sugestivos e intuitivos, diretamente relacionados às suas respectivas funções.

```
464  typedef union accel_t_gyro_union
465  {
466    struct
467    {
468      uint8_t x_accel_h;
469      uint8_t x_accel_l;
470      uint8_t y_accel_h;
471      uint8_t y_accel_l;
472      uint8_t z_accel_h;
473      uint8_t z_accel_l;
474      uint8_t t_h;
475      uint8_t t_l;
476      uint8_t x_gyro_h;
477      uint8_t x_gyro_l;
478      uint8_t y_gyro_h;
479      uint8_t y_gyro_l;
480      uint8_t z_gyro_h;
481      uint8_t z_gyro_l;
482    } reg;
483    struct
484    {
485      int16_t x_accel;
486      int16_t y_accel;
487      int16_t z_accel;
488      int16_t temperature;
489      int16_t x_gyro;
490      int16_t y_gyro;
491      int16_t z_gyro;
492    } value;
493  };
```

Código 3: Estrutura de dados.

Após a análise dessas diretivas foi criada a função *leGiroscópio(),* que efetua leitura dos dados do giroscópio. Esta rotina requisita a função que guarda os valores

retornados do giroscópio na estrutura de dados citada, utilizando a função *MPU6050_read()* de leitura e o protocolo I2C (codigo 4). Essa função inicia a transmissão com o GY-521, faz as leituras dos sensores e guarda na estrutura de dados, chamada *buffer* (que na verdade é a mesma estrutura da função *leGiroscópio()*). Após isto, a função *leGiroscópio()* ajusta o limite superior da leitura do intervalo do eixo z (eixo relevante para o projeto) em estruturas condicionais que consideram apenas leituras acima de 1000 e a diferença entre as leituras atual e anterior acima de 500 (Código 5).

```
538  // --------------------------------------------------------
539  // Função de leitura dos dados brutos do giroscópio
540  int MPU6050_read(int start, uint8_t *buffer, int size)
541  {
542    int i, n, error;
543
544    Wire.beginTransmission(MPU6050_I2C_ADDRESS);
545    n = Wire.write(start);
546    if (n != 1)
547      return (-10);
548
549    n = Wire.endTransmission(false);
550    if (n != 0)
551      return (n);
552
553    Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);
554    i = 0;
555    while(Wire.available() && i<size)
556    {
557      buffer[i++]=Wire.read();
558    }
559    if ( i != size)
560      return (-11);
561
562    return (0);
563  }
```

Código 4: Função de leitura

```
524  if(abs (accel_t_gyro.value.z_gyro - z_gyro_init) > 500 && (abs(accel_t_gyro.value.z_gyro ) > 1000))
525  {
526    cont = 1;
527  } else {
528    cont = 0;
529  }
```

Código 5: Função de limitação inferior de leitura

Algumas variáveis são criadas e inicializadas globalmente antes de iniciar a execução, que são elas: *seta* - que representa o estado das setas; *estado* - que representa o estado de conversão da moto; *estadoAnt* – representa o estado de conversão anterior ao atual; *temporizador* – que guarda e controla o tempo para a desativação das setas. Conforme código mostrado no código 6.

```
605  int seta = 0;  // variável que indica o estado da seta da moto (direita, esquerda ou desligada)
606  int estado = 0;  // variável que indica o estado do giroscópio (virando ou não virando)
607  int estadoAnt = 0;  // variável que indica o estado anterior ao atual do giroscópio (virando ou não virando)
608  int temporizador = 0; // variável que conta a quantidade de tempo em segundos
```

Código 6: Variáveis globais.

Na função de inicialização do software do Arduino, foi inicializada uma conexão com a biblioteca *Serial,* responsável pela porta serial do Arduino conectada ao computador, sendo definida uma frequência para a mesma (tanto essa função quanto todas as outras rotinas da biblioteca *Serial* só foram utilizadas para avaliação dos valores e depuração do código, o projeto final não utiliza as rotinas dessa biblioteca). Também foi iniciada uma conexão com a biblioteca *Wire,* responsável pela conexão entre o Arduino e o GY-521. A função *MPU6050_write_reg()* também é chamada para fazer a calibragem dos sensores do GY-521, configurar e estabelecer seus valores iniciais.

Como o Arduino precisa saber de forma antecipada quais de seus pinos serão tratados como entrada ou saída (no caso dos pinos digitais), foram definidos os pinos ligados aos LEDs como saída e os pinos ligados aos botões como entrada de sinais digitais, de acordo com os números definidos nas diretivas, conforme código 7.

```
610 // Função que configura todos os componentes do sistema sobre seus
    comportamentos e como serão utilizados pelo sistema
611 void setup() {
612   Serial.begin(9600);
613   uint8_t c;
614   accel_t_gyro_union accel_t_gyro_ini;
615
616   Wire.begin();
617
618   MPU6050_write_reg (MPU6050_PWR_MGMT_1, 0);
619
620   pinMode(LEDDIREITO, OUTPUT);
621   pinMode(LEDESQUERDO, OUTPUT);
622   pinMode(SETADIREITA, INPUT);
623   pinMode(SETAESQUERDA, INPUT);
624 }
```

Código 7: Função de inicialização do software do Arduino.

No bloco de execução principal do Arduino foi implementada a lógica geral do projeto, ela foi separada em estruturas condicionais que representam os estados que o sistema pode assumir.

A primeira estrutura condicional representa o estado de a seta estar ligada ou não para um dos lados, senão estiver neste estado, apenas as variáveis são reinicializadas. Se estiver ligada para um dos lados, é feita a leitura do giroscópio e é verificado se a conversão está em processo ou se já terminou. Se estiver em processo, o estado é guardado como anterior e uma nova leitura é feita. Se já estiver terminada a conversão e o estado anterior for uma conversão, um contador é disparado (uma unidade por segundo). O código 8 mostra este estado do sistema no código.

```
      // Bloco que reconhece uma conversão ou
631   estabilização do giroscópio de acordo com as
      setas
632    if (seta != 0) {
633      estado = leGiroscopio();
634      if (estado) {
635        estadoAnt = estado;
636      } else if (estadoAnt) {
637        delay(1000);
638        temporizador++;
639      }
640    } else {
641      estado = 0;
642      temporizador = 0;
643      estadoAnt = 0;
644    }
```

Código 8: Estrutura condicional do estado de ativação ou não da seta.

A segunda estrutura condicional representa o evento de pressionamento do botão da seta direita. Neste estado o Arduino efetua uma leitura do pino ligado ao botão da direita e, então, executa uma pequena pausa no código para sincronizar a execução com o hardware, daí verifica se a seta já estava ligada para a outra direção ou se estava desligada, se sim, o sistema desliga o LED da seta oposta e liga o da seta direita, indicando para a variável de estado que está havendo ou prestes a haver uma conversão a direita. Senão, é o caso então de desligar a própria seta direita, então o Arduino desliga o LED da seta direita. Esta situação está implementada conforme o código 9.

```
646  // Bloco que reconhece quando o botão que aciona a seta direita foi pressionado
647    if (digitalRead(SETADIREITA)) {
648      delay(250);
649      if (seta == 0 || seta == ESQUERDA) {
650        seta = DIREITA;
651        digitalWrite(LEDDIREITO, HIGH);
652        digitalWrite(LEDESQUERDO, LOW);
653      } else {
654        seta = 0;
655        digitalWrite(LEDDIREITO, LOW);
656      }
657      temporizador = 0;
658    }
```

Código 9: Estrutura que representa o acionamento do botão direito.

A terceira estrutura condicional representa o evento de pressionamento do botão da seta esquerda. Neste estado o Arduino efetua uma leitura do pino ligado ao botão da esquerda, então executa uma pequena pausa no código para sincronizar a execução com o hardware e, então, verifica se a seta já estava ligada para a outra direção ou se estava desligada. Se sim, o sistema desliga o LED da seta oposta e liga o da seta esquerda e indica para a variável de estado que está havendo ou prestes a haver uma conversão a esquerda. Senão, é o caso então de desligar a própria seta esquerda, então o Arduino desliga o LED da seta esquerda. Esta situação está implementada conforme o código 10.

```
660  // Bloco que reconhece quando o botão que aciona a seta esquerda foi pressionado
661    if (digitalRead(SETAESQUERDA)) {
662      delay(250);
663      if (seta == 0 || seta == DIREITA) {
664        seta = ESQUERDA;
665        digitalWrite(LEDESQUERDO, HIGH);
666        digitalWrite(LEDDIREITO, LOW);
667      } else {
668        seta = 0;
669        digitalWrite(LEDESQUERDO, LOW);
670      }
671      temporizador = 0;
672    }
```

Código 10: Estrutura que representa o acionamento do botão esquerdo

A quarta e última estrutura condicional do bloco principal, verifica se o contador de tempo chegou a 3 segundos, se chegou, as variáveis são zeradas e é enviado um sinal de desligamento para os dois LEDs. O código 11 é implementado nessa situação.

```
674    // Bloco que desliga as setas de acordo com a finalização de uma conversão
675    if (temporizador == 3) {
676      estado = 0;
677      estadoAnt = 0;
678      temporizador = 0;
679      digitalWrite(LEDESQUERDO, LOW);
680      digitalWrite(LEDDIREITO, LOW);
681      seta = 0;
682    }
```

Código 11: Estrutura que representa o desligamento de seta ao fim de uma conversão.

IV. TESTES REALIZADOS E RESULTADOS

Foram realizados testes visando verificar se o modo de funcionamento do sistema atendia aos requisitos do projeto. Estas verificações objetivaram também a otimização do protótipo.

Efetuou-se uma simulação do funcionamento do sistema em um ambiente real, ou seja, ruas pavimentadas e sem pavimentação do sistema urbano de vias de rolamento.

Os testes buscaram aperfeiçoar cada modulo em separado, primeiramente o teste de conexão entre o Arduino e o GY-521, depois a conexão entre o Arduino, botões e LEDs, e, posteriormente, o sistema completo em atuação.

4.1. Teste de conexão entre Arduino e GY-521

Foram feitos vários testes para se encontrar um valor que pudesse ser utilizado como parâmetro para que o sistema conseguisse saber se uma conversão foi ou não realizada.

Primeiramente foi construído um objeto com rodas e fixou-se o Arduino com o GY-521. Usando-se o código *Arduino Playground (retirado de: http://playground.arduino.cc/Main/MPU-6050)*, conforme apresentado no anexo A, realizou-se o primeiro teste, com o objetivo de observar a sensibilidade do Giroscópio presente no GY-521. Com os resultados obtidos, verificou-se que um conjunto de valores era registrado a partir de qualquer movimentação realizada com o dispositivo e, outros conjuntos de valores distintos apareciam apenas quando se simulava a movimentação para a conversão.

Com isto, modificou-se o código de exemplo, para que ele pudesse "ignorar" os valores que não correspondessem aos de movimentação para conversão.

A partir desta modificação foram realizados novos testes que apresentaram os resultados satisfatórios. No entanto, os testes foram realizados inicialmente em situações onde havia uma baixa trepidação. Então, como forma de otimizar o sistema, foram realizados testes em vias de rolamento mais reais, isto é, com alta trepidação.

Para simular a situação de alta trepidação, o sistema foi colocado em um automóvel e este automóvel trafegou por vários tipos de vias, onde foi observado o

comportamento do sistema nessas situações. O trajeto feito pelo automóvel é mostrado na figura 20.

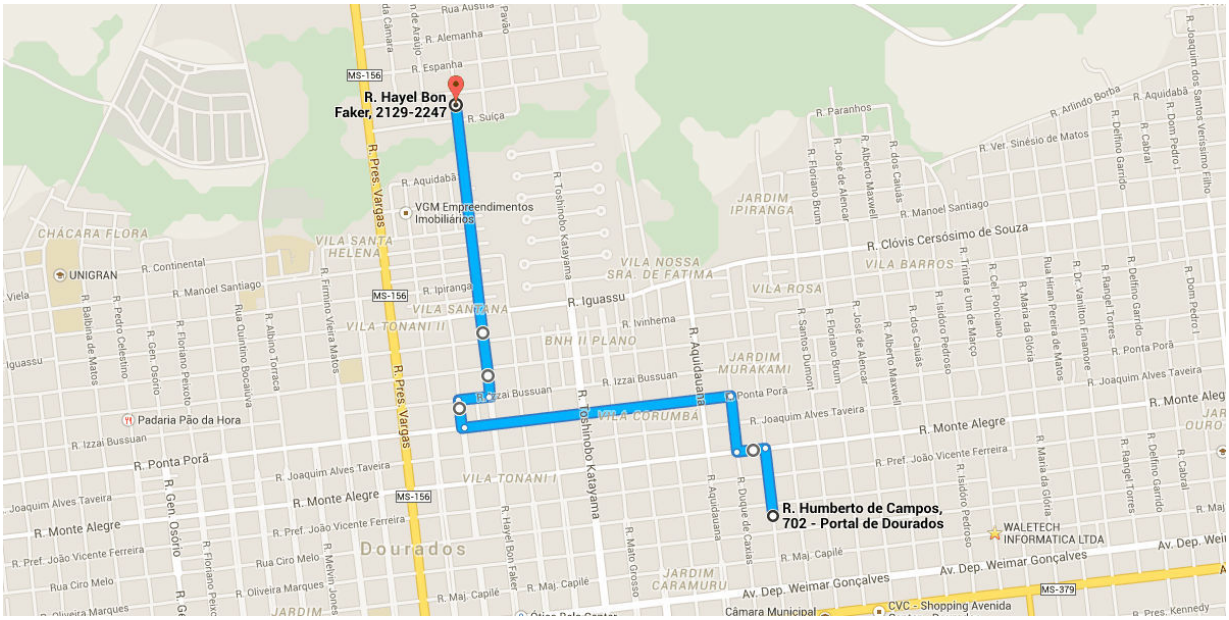

Figura 9: Trajeto realizado durante o primeiro teste. Fonte: autores.

Os resultados deste teste mostraram necessidade de se ajustar o conjunto de valores obtidos nos testes efetuados em situações de baixa trepidação, pois em alguns trechos da via, onde não era realizada uma conversão, o sistema acusava a conversão, isto devido à alta trepidação. Devido a isto, modificou-se o conjunto de valores de verificação de conversão e novamente o sistema foi testado nas mesmas situações de alta trepidação, sendo dessa vez os resultados satisfatórios. No apêndice A é mostrado a implementação gerada após a realização desses testes.

O teste não foi realizado em uma motocicleta porque era necessário observar os valores que o sistema registrava e isso exigia o uso de um computador. Também por uma questão de preservação do protótipo, pois o sistema não pode ser fixado com segurança na moto.

42. Teste de conexão entre o Arduino, botões e LEDs

O teste de conexão entre o Arduino, botões e LEDs foi preciso para garantir que o código implementado, conforme descrito no apêndice B, fosse capaz efetivamente de controlar o sistema quando implantado em motocicletas.

Neste teste foi verificou-se o acionamento do LED quando se pressionava seu respectivo botão. Esta etapa foi importante para a verificação dos pinos de entrada (pressionamento dos botões) e de saída (acendimento dos LEDs), definindo-se com exatidão os pinos a serem utilizados conforme se desejava acionamento para a esquerda ou para a direita. Os resultados foram satisfatórios, havendo indicação correta dos LEDs conforme o respectivo botão acionado.

4.3. Teste do sistema completo

Realizados os testes das partes em separado, partiu-se para os testes com o sistema global, onde realizou-se a codificação apresentada no Apêndice C. Esta codificação foi elaborada a partir das codificações apresentadas nos apêndices A e B.

O objetivo foi de encontrar possíveis erros que apareceriam com a junção das partes e também implementar agora o desligamento automático das setas.

Movimentando-se todo o sistema agora, observava-se o seguinte:

- Acionamento do sistema a partir o pressionamento do botão;
- Desligamento quando o usuário pressionar o botão;
- Modificação do sentido de conversão;
- Desligamento automatizado ao realizar a conversão.

Verificou-se que o sistema desligava quando o usuário pressionava o botão, ocorrendo uma falha. Este problema ocorreu devido à velocidade de processamento do microcontrolador que era maior do que o tempo fisiológico que um ser humano leva para aplicar pressão com os dedos. Para resolver este problema, foi utilizada a função Delay, que faz com que o programa realize uma pausa de alguns milissegundos, sendo testadas várias situações para encontrar o melhor tempo para essa pausa.

Também se verificou a sinalização de conversão quando se está em uma direção e o usuário deseja modificar sua rota para a direção oposta. Os resultados foram satisfatórios, tendo o sistema reagido prontamente ao comando.

O desligamento automático das setas após uma conversão foi testado e os resultados mostraram que o sistema respondia corretamente, ou seja, após ser realizada uma conversão, o sistema desligava automaticamente a respectiva seta.

V. CONCLUSÃO

A proposta foi conseguir desligar automaticamente a seta das motocicletas após o fim de suas conversões, para isso, o sistema criado é capaz de avaliar o estado de conversão das motos depois de ativada a seta da mesma. Foram também corrigidos valores relativos a sensibilidade de leitura do sensor do giroscópio e do botão que, combinados com o Arduino e os LEDS, formaram a simulação elétrica da seta das motos. Essa correção proporcionou uma otimização do sistema.

Nos testes realizados, tanto no carro quanto no laboratório, quando foi acionado o botão da seta, o LED correspondente acendia e só desligava após ocorrer uma estabilização posterior a uma variação considerável do valor de leitura do sensor de giro, fazendo, então, a automatização para todas as situações de conversões de trânsito da motocicleta.

Para facilitar a vida do motociclista, o sistema ainda proporciona a funcionalidade de que se ele deseja desativar uma seta ligada, basta então pressionar novamente o botão correspondente a seta e, ao ligar uma seta, a outra seta é desligada automaticamente. Se esse sistema for efetivamente implantado nas motos, o trânsito irá melhorar na questão de segurança, pois irá facilitar tanto a vida dos motociclistas (na utilização simplificada da seta e desligamento automático) quanto dos motoristas, pedestres e outros motociclistas que saberão qual manobra o motociclista realmente irá fazer, diminuindo consideravelmente acidentes de trânsito causados por motivo de confusões de manobras e desatenções.

Uma consideração importante a ser feita é que o sistema não funciona corretamente quando o motociclista aciona a seta para realizar ultrapassagem, por causa da pouca variação de direção da moto, não havendo desligamento da seta quando a manobra já tenha sido totalmente realizada, necessitando que o motociclista desative manualmente a seta. Outro caso de falha no sistema é quando o motociclista aciona a seta para uma conversão e efetua um grande desvio de algum obstáculo na pista antes de efetuar a manobra para qual a seta foi acionada, neste caso o sistema desliga a seta incorretamente por causa da alta variação de direção da moto. Então, esses são pontos que ainda podem ser melhorados no sistema, gerando assim a possibilidade de trabalhos futuros para sanar essas deficiências. Uma possibilidade para ser estudada seria adicionar ao sistema um GPS, que aumentaria a precisão no reconhecimento das

manobras da moto e talvez resolvesse o problema de funcionamento do sistema nas situações em que ele se comporta incorretamente.

Outro ponto importante é que, notou-se a possibilidade do sistema ser usado futuramente em carros, pois seria mais eficiente que o atual sistema de desligamento automático da seta dos mesmos, que tem como referência o volante, diferente do sistema desenvolvido neste trabalho que se baseia na direção do chassi do veículo.

Por fim, o Sistema Inteligente de Sinalização de Segurança em Veículos Automotores desenvolvido, proporcionou a nós que o desenvolvemos um aprofundamento no conhecimento sobre sistemas embarcados, visão e conscientização sobre segurança no trânsito e pesquisa científica no meio acadêmico, sendo de vital importância para complementar nossa formação acadêmica. Ao finalizar este projeto, acreditamos que ele possa trazer significativa contribuição na área de segurança no transito para veículos automotores.

Referências Bibliográficas

AIRBAG, Denko. Uma reputação construída em centésimos de segundos. Disponível em: <http://www.compredenko.com.br/dna-denko-de-seguranca>. Acesso em: 10 jul. 2014.

BANZI, Massimo. Getting started with arduino. 2. ed. Make, 2011.

BAYLE, Julien. C Programming for Arduino. Birmingham: Packt Publishing, 2013.

BOSH. Sistema de segurança para motocicletas. 2014. Disponível em: <http://www.bosch-moto.com.br/pt/br/fahrsicherheit_fuer_zweiraeder_1/sicherheitssysteme_fuer_zweiraeder_1/sicherheitssysteme_fuer_zweiraeder.html>. Acesso em: 10 jul. 2014.

BRASIL. Lei nº 9503, de 23 de setembro de 1997. Código de Trânsito Brasileiro. Disponível em: <http://www.planalto.gov.br/ccivil_03/leis/l9503.htm>. Acesso em: 22 out. 2014.

CARROS, Revista Pense. Entenda como funciona o Airbag e conheça alguns mitos sobre o item de segurança. 2014. Disponível em: <http://revista.pensecarros.com.br/noticia/2010/09/entenda-como-funciona-o-airbag-e-conheca-alguns-mitos-sobre-o-item-de-seguranca-3028314.html>. Acesso em: 12 jul. 2014.

DEBIEUX, Pedro et al. Lesões do aparelho locomotor nos acidentes com motocicleta. Acta Ortopédica Brasileira, São Paulo, v. 6, n. 2, p.353-356, 2010. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1413-78522010000600010>. Acesso em: 22 out. 2014.

DETRAN-MS. Estatísticas-Dourados. 2014. Disponível em: <http://www.detran.ms.gov.br/institucional/207/estatistica>. Acesso em: 26 jun. 2014.

ENCYCLOPEDIA.COM. Gyroscope.: How Products Are Made.. 2001. Disponível em: <http://www.encyclopedia.com/doc/1G2-2897000050.html>. Acesso em: 13 set. 2014.

EVANS, Martin; NOBLE, Joshua; HOCHENBAUM, Jordan. Arduíno em ação. São Paulo: Novatec, 2013.

FENABRAVE. Emplacamentos - Junho. Disponível em: <http://www3.fenabrave.org.br:8082/plus_fenabrave/plus/>. Acesso em: 5 jul. 2014.

FUNCIONA, Blog Como Tudo. Como funcionam os indicadores de direção. Disponível em: <http://carros.hsw.uol.com.br/sinalizadores-de-direcao.htm>. Acesso em: 10 jul. 2014.

INVENSENSE. Motion Interface – a Transformational Technology. Disponível em: <http://www.invensense.com/mems/motioninterface.html>. Acesso em: 20 set. 2014.

INVENSENSE. MPU-6000 and MPU-6050 Product Specification Revision 3.4. 2013. Disponível em: <http://www.invensense.com/mems/gyro/documents/PS-MPU-6000A-00v3.4.pdf>. Acesso em: 10 jun. 2014.

JACKET, Blog Airbag. Tudo sobre a jaqueta Airbag. Disponível em: <http://www.airbagjacket.eu/documentos_por.html>. Acesso em: 10 jul. 2014.

MCROBERTS, Michael. Arduíno Básico. São Paulo: Novatec, 2011.

MIOTTO, Rafael. Número de mortes em acidente com moto sobe 263,5% em 10 anos. 2013. Disponível em: <http://g1.globo.com/carros/motos/noticia/2013/06/numero-de-mortes-em-acidente-com-moto-sobe-2635-em-10-anos.html>. Acesso em: 15 jul. 2014.

PIGHIXXX. Pinouts Boards. 2013. Disponível em: <http://www.pighixxx.com/test/pinoutspg/boards/>. Acesso em: 10 jul. 2014.

PLAYGROUND, Arduino. MPU-6050 Accelerometer + Gyro. Disponível em: <http://playground.arduino.cc/Main/MPU-6050>. Acesso em: 10 jun. 2014.

PROGRAMAR, Revista. INTRODUÇÃO AO ARDUINO. Disponível em: <http://www.revista-programar.info/artigos/introducao-ao-arduino/>. Acesso em: 20 jul. 2014.

REFERENCE, Arduino. Serial. Disponível em: <http://arduino.cc/en/Reference/Serial>. Acesso em: 30 jul. 2014.

REFERENCE, Arduino. Wire Library. Disponível em: <http://arduino.cc/en/reference/wire>. Acesso em: 30 jul. 2014.

REVISTA PROGRAMAR. Online: Portugal A Programar, v. 17, dez. 2018. Disponível em: <http://www.revista-programar.info/>. Acesso em: 10 jul. 2014.

SCHMIDT, Maik. Arduíno: A Quick-Start Guide. The Pragmatic Bookshelf, 2011.

SEGURAS, Vias. Acidentes com pedestres. 2012. Disponível em: <http://www.vias-seguras.com/os_acidentes/acidentes_com_pedestres>. Acesso em: 12 jul. 2014.

SILVA JR, Gilson J. da; SANTOS, Edval J. P. Aspectos de funcionamento e fabricação do giroscópio a ondas acusticas de superfície. XXXIII – Congresso Brasileiro de Ensino de Engenharia. 12 a 15 de setembro – Campina Grande. Paraíba, 2005. Disponível em: http://www.abenge.org.br/CobengeAnteriores/2005/artigos/PE-9-01227907400-1118726403502.pdf

SOUZA, Thalis Antunes de, Plataforma Open Hardware para Robótica, LATINOWARE, 2014.

TEIXEIRA, Jules Ramon Brito et al. Utilização dos equipamentos de proteção individual por mototaxistas: percepção dos fatores de risco e associados. Cad. Saúde Pública, Rio de Janeiro, v. 4, n. 30, p.885-890, abr. 2014. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0102-311X2014000400885&lng=pt&nrm=iso&tlng=en>. Acesso em: 22 out. 2014.

Anexo A: Código exemplo para teste do GY-521

```
// MPU-6050 Accelerometer + Gyro
// -----------------------------
//
// By arduino.cc user "Krodal".
//
// June 2012
//      first version
// July 2013
//      The 'int' in the union for the x,y,z
//      changed into int16_t to be compatible
//      with Arduino Due.
//
// Open Source / Public Domain
//
// Using Arduino 1.0.1
// It will not work with an older version,
// since Wire.endTransmission() uses a parameter
// to hold or release the I2C bus.
//
// Documentation:
// - The InvenSense documents:
//   - "MPU-6000 and MPU-6050 Product Specification",
//     PS-MPU-6000A.pdf
//   - "MPU-6000 and MPU-6050 Register Map and Descriptions",
//     RM-MPU-6000A.pdf or RS-MPU-6000A.pdf
//   - "MPU-6000/MPU-6050 9-Axis Evaluation Board User Guide"
//     AN-MPU-6000EVB.pdf
//


// The accuracy is 16-bits.
```

```
//
// Temperature sensor from -40 to +85 degrees Celsius
//   340 per degrees, -512 at 35 degrees.
//
// At power-up, all registers are zero, except these two:
//      Register 0x6B (PWR_MGMT_2) = 0x40  (I read zero).
//      Register 0x75 (WHO_AM_I)   = 0x68.
//
 #include <Wire.h>


// The name of the sensor is "MPU-6050".
// For program code, I omit the '-',
// therefor I use the name "MPU6050...."
// Register names according to the datasheet.
// According to the InvenSense document
// "MPU-6000 and MPU-6050 Register Map
// and Descriptions Revision 3.2", there are no registers
// at 0x02 ... 0x18, but according other information
// the registers in that unknown area are for gain
// and offsets.
//
#define MPU6050_AUX_VDDIO          0x01   // R/W
#define MPU6050_SMPLRT_DIV         0x19   // R/W
#define MPU6050_CONFIG             0x1A   // R/W
#define MPU6050_GYRO_CONFIG        0x1B   // R/W
#define MPU6050_ACCEL_CONFIG       0x1C   // R/W
#define MPU6050_FF_THR             0x1D   // R/W
#define MPU6050_FF_DUR             0x1E   // R/W
#define MPU6050_MOT_THR            0x1F   // R/W
#define MPU6050_MOT_DUR            0x20   // R/W
#define MPU6050_ZRMOT_THR          0x21   // R/W
#define MPU6050_ZRMOT_DUR          0x22   // R/W
#define MPU6050_FIFO_EN            0x23   // R/W
```

```
#define MPU6050_I2C_MST_CTRL       0x24   // R/W
#define MPU6050_I2C_SLV0_ADDR      0x25   // R/W
#define MPU6050_I2C_SLV0_REG       0x26   // R/W
#define MPU6050_I2C_SLV0_CTRL      0x27   // R/W
#define MPU6050_I2C_SLV1_ADDR      0x28   // R/W
#define MPU6050_I2C_SLV1_REG       0x29   // R/W
#define MPU6050_I2C_SLV1_CTRL      0x2A   // R/W
#define MPU6050_I2C_SLV2_ADDR      0x2B   // R/W
#define MPU6050_I2C_SLV2_REG       0x2C   // R/W
#define MPU6050_I2C_SLV2_CTRL      0x2D   // R/W
#define MPU6050_I2C_SLV3_ADDR      0x2E   // R/W
#define MPU6050_I2C_SLV3_REG       0x2F   // R/W
#define MPU6050_I2C_SLV3_CTRL      0x30   // R/W
#define MPU6050_I2C_SLV4_ADDR      0x31   // R/W
#define MPU6050_I2C_SLV4_REG       0x32   // R/W
#define MPU6050_I2C_SLV4_DO        0x33   // R/W
#define MPU6050_I2C_SLV4_CTRL      0x34   // R/W
#define MPU6050_I2C_SLV4_DI        0x35   // R
#define MPU6050_I2C_MST_STATUS     0x36   // R
#define MPU6050_INT_PIN_CFG        0x37   // R/W
#define MPU6050_INT_ENABLE         0x38   // R/W
#define MPU6050_INT_STATUS         0x3A   // R
#define MPU6050_ACCEL_XOUT_H       0x3B   // R
#define MPU6050_ACCEL_XOUT_L       0x3C   // R
#define MPU6050_ACCEL_YOUT_H       0x3D   // R
#define MPU6050_ACCEL_YOUT_L       0x3E   // R
#define MPU6050_ACCEL_ZOUT_H       0x3F   // R
#define MPU6050_ACCEL_ZOUT_L       0x40   // R
#define MPU6050_TEMP_OUT_H         0x41   // R
#define MPU6050_TEMP_OUT_L         0x42   // R
#define MPU6050_GYRO_XOUT_H        0x43   // R
#define MPU6050_GYRO_XOUT_L        0x44   // R
#define MPU6050_GYRO_YOUT_H        0x45   // R
#define MPU6050_GYRO_YOUT_L        0x46   // R
```

```
#define MPU6050_GYRO_ZOUT_H       0x47   // R
#define MPU6050_GYRO_ZOUT_L       0x48   // R
#define MPU6050_EXT_SENS_DATA_00   0x49   // R
#define MPU6050_EXT_SENS_DATA_01   0x4A   // R
#define MPU6050_EXT_SENS_DATA_02   0x4B   // R
#define MPU6050_EXT_SENS_DATA_03   0x4C   // R
#define MPU6050_EXT_SENS_DATA_04   0x4D   // R
#define MPU6050_EXT_SENS_DATA_05   0x4E   // R
#define MPU6050_EXT_SENS_DATA_06   0x4F   // R
#define MPU6050_EXT_SENS_DATA_07   0x50   // R
#define MPU6050_EXT_SENS_DATA_08   0x51   // R
#define MPU6050_EXT_SENS_DATA_09   0x52   // R
#define MPU6050_EXT_SENS_DATA_10   0x53   // R
#define MPU6050_EXT_SENS_DATA_11   0x54   // R
#define MPU6050_EXT_SENS_DATA_12   0x55   // R
#define MPU6050_EXT_SENS_DATA_13   0x56   // R
#define MPU6050_EXT_SENS_DATA_14   0x57   // R
#define MPU6050_EXT_SENS_DATA_15   0x58   // R
#define MPU6050_EXT_SENS_DATA_16   0x59   // R
#define MPU6050_EXT_SENS_DATA_17   0x5A   // R
#define MPU6050_EXT_SENS_DATA_18   0x5B   // R
#define MPU6050_EXT_SENS_DATA_19   0x5C   // R
#define MPU6050_EXT_SENS_DATA_20   0x5D   // R
#define MPU6050_EXT_SENS_DATA_21   0x5E   // R
#define MPU6050_EXT_SENS_DATA_22   0x5F   // R
#define MPU6050_EXT_SENS_DATA_23   0x60   // R
#define MPU6050_MOT_DETECT_STATUS  0x61   // R
#define MPU6050_I2C_SLV0_DO       0x63   // R/W
#define MPU6050_I2C_SLV1_DO       0x64   // R/W
#define MPU6050_I2C_SLV2_DO       0x65   // R/W
#define MPU6050_I2C_SLV3_DO       0x66   // R/W
#define MPU6050_I2C_MST_DELAY_CTRL 0x67   // R/W
#define MPU6050_SIGNAL_PATH_RESET  0x68   // R/W
#define MPU6050_MOT_DETECT_CTRL    0x69   // R/W
```

```
#define MPU6050_USER_CTRL        0x6A  // R/W
#define MPU6050_PWR_MGMT_1       0x6B  // R/W
#define MPU6050_PWR_MGMT_2       0x6C  // R/W
#define MPU6050_FIFO_COUNTH      0x72  // R/W
#define MPU6050_FIFO_COUNTL      0x73  // R/W
#define MPU6050_FIFO_R_W         0x74  // R/W
#define MPU6050_WHO_AM_I         0x75  // R


// Defines for the bits, to be able to change
// between bit number and binary definition.
// By using the bit number, programming the sensor
// is like programming the AVR microcontroller.
// But instead of using "(1<<X)", or "_BV(X)",
// the Arduino "bit(X)" is used.
#define MPU6050_D0 0
#define MPU6050_D1 1
#define MPU6050_D2 2
#define MPU6050_D3 3
#define MPU6050_D4 4
#define MPU6050_D5 5
#define MPU6050_D6 6
#define MPU6050_D7 7

// AUX_VDDIO Register
#define MPU6050_AUX_VDDIO MPU6050_D7    // I2C high: 1=VDD, 0=VLOGIC

// CONFIG Register
// DLPF is Digital Low Pass Filter for both gyro and accelerometers.
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_DLPF_CFG0    MPU6050_D0
#define MPU6050_DLPF_CFG1    MPU6050_D1
```

```
#define MPU6050_DLPF_CFG2     MPU6050_D2
#define MPU6050_EXT_SYNC_SET0 MPU6050_D3
#define MPU6050_EXT_SYNC_SET1 MPU6050_D4
#define MPU6050_EXT_SYNC_SET2 MPU6050_D5

// Combined definitions for the EXT_SYNC_SET values
#define MPU6050_EXT_SYNC_SET_0 (0)
#define MPU6050_EXT_SYNC_SET_1 (bit(MPU6050_EXT_SYNC_SET0))
#define MPU6050_EXT_SYNC_SET_2 (bit(MPU6050_EXT_SYNC_SET1))
#define MPU6050_EXT_SYNC_SET_3 (bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT_SYNC_SET0))
#define MPU6050_EXT_SYNC_SET_4 (bit(MPU6050_EXT_SYNC_SET2))
#define MPU6050_EXT_SYNC_SET_5 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET0))
#define MPU6050_EXT_SYNC_SET_6 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1))
#define MPU6050_EXT_SYNC_SET_7 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT_SYNC_SET0))

// Alternative names for the combined definitions.
#define MPU6050_EXT_SYNC_DISABLED MPU6050_EXT_SYNC_SET_0
#define MPU6050_EXT_SYNC_TEMP_OUT_L MPU6050_EXT_SYNC_SET_1
#define MPU6050_EXT_SYNC_GYRO_XOUT_L MPU6050_EXT_SYNC_SET_2
#define MPU6050_EXT_SYNC_GYRO_YOUT_L MPU6050_EXT_SYNC_SET_3
#define MPU6050_EXT_SYNC_GYRO_ZOUT_L MPU6050_EXT_SYNC_SET_4
#define MPU6050_EXT_SYNC_ACCEL_XOUT_L MPU6050_EXT_SYNC_SET_5
```

```
        #define                         MPU6050_EXT_SYNC_ACCEL_YOUT_L
MPU6050_EXT_SYNC_SET_6
        #define                         MPU6050_EXT_SYNC_ACCEL_ZOUT_L
MPU6050_EXT_SYNC_SET_7

        // Combined definitions for the DLPF_CFG values
        #define MPU6050_DLPF_CFG_0 (0)
        #define MPU6050_DLPF_CFG_1 (bit(MPU6050_DLPF_CFG0))
        #define MPU6050_DLPF_CFG_2 (bit(MPU6050_DLPF_CFG1))
        #define                                         MPU6050_DLPF_CFG_3
(bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG0))
        #define MPU6050_DLPF_CFG_4 (bit(MPU6050_DLPF_CFG2))
        #define                                         MPU6050_DLPF_CFG_5
(bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG0))
        #define                                         MPU6050_DLPF_CFG_6
(bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1))
        #define                                         MPU6050_DLPF_CFG_7
(bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG
0))

        // Alternative names for the combined definitions
        // This name uses the bandwidth (Hz) for the accelometer,
        // for the gyro the bandwidth is almost the same.
        #define MPU6050_DLPF_260HZ    MPU6050_DLPF_CFG_0
        #define MPU6050_DLPF_184HZ    MPU6050_DLPF_CFG_1
        #define MPU6050_DLPF_94HZ     MPU6050_DLPF_CFG_2
        #define MPU6050_DLPF_44HZ     MPU6050_DLPF_CFG_3
        #define MPU6050_DLPF_21HZ     MPU6050_DLPF_CFG_4
        #define MPU6050_DLPF_10HZ     MPU6050_DLPF_CFG_5
        #define MPU6050_DLPF_5HZ      MPU6050_DLPF_CFG_6
        #define MPU6050_DLPF_RESERVED MPU6050_DLPF_CFG_7

        // GYRO_CONFIG Register
        // The XG_ST, YG_ST, ZG_ST are bits for selftest.
```

```
// The FS_SEL sets the range for the gyro.
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_FS_SEL0 MPU6050_D3
#define MPU6050_FS_SEL1 MPU6050_D4
#define MPU6050_ZG_ST   MPU6050_D5
#define MPU6050_YG_ST   MPU6050_D6
#define MPU6050_XG_ST   MPU6050_D7

// Combined definitions for the FS_SEL values
#define MPU6050_FS_SEL_0 (0)
#define MPU6050_FS_SEL_1 (bit(MPU6050_FS_SEL0))
#define MPU6050_FS_SEL_2 (bit(MPU6050_FS_SEL1))
#define MPU6050_FS_SEL_3 (bit(MPU6050_FS_SEL1)|bit(MPU6050_FS_SEL0))

// Alternative names for the combined definitions
// The name uses the range in degrees per second.
#define MPU6050_FS_SEL_250  MPU6050_FS_SEL_0
#define MPU6050_FS_SEL_500  MPU6050_FS_SEL_1
#define MPU6050_FS_SEL_1000 MPU6050_FS_SEL_2
#define MPU6050_FS_SEL_2000 MPU6050_FS_SEL_3

// ACCEL_CONFIG Register
// The XA_ST, YA_ST, ZA_ST are bits for selftest.
// The AFS_SEL sets the range for the accelerometer.
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_ACCEL_HPF0 MPU6050_D0
#define MPU6050_ACCEL_HPF1 MPU6050_D1
#define MPU6050_ACCEL_HPF2 MPU6050_D2
#define MPU6050_AFS_SEL0   MPU6050_D3
#define MPU6050_AFS_SEL1   MPU6050_D4
#define MPU6050_ZA_ST      MPU6050_D5
```

```
#define MPU6050_YA_ST      MPU6050_D6
#define MPU6050_XA_ST      MPU6050_D7

// Combined definitions for the ACCEL_HPF values
#define MPU6050_ACCEL_HPF_0 (0)
#define MPU6050_ACCEL_HPF_1 (bit(MPU6050_ACCEL_HPF0))
#define MPU6050_ACCEL_HPF_2 (bit(MPU6050_ACCEL_HPF1))
#define MPU6050_ACCEL_HPF_3 (bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL_HPF0))
#define MPU6050_ACCEL_HPF_4 (bit(MPU6050_ACCEL_HPF2))
#define MPU6050_ACCEL_HPF_7 (bit(MPU6050_ACCEL_HPF2)|bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL_HPF0))

// Alternative names for the combined definitions
// The name uses the Cut-off frequency.
#define MPU6050_ACCEL_HPF_RESET  MPU6050_ACCEL_HPF_0
#define MPU6050_ACCEL_HPF_5HZ    MPU6050_ACCEL_HPF_1
#define MPU6050_ACCEL_HPF_2_5HZ  MPU6050_ACCEL_HPF_2
#define MPU6050_ACCEL_HPF_1_25HZ MPU6050_ACCEL_HPF_3
#define MPU6050_ACCEL_HPF_0_63HZ MPU6050_ACCEL_HPF_4
#define MPU6050_ACCEL_HPF_HOLD   MPU6050_ACCEL_HPF_7

// Combined definitions for the AFS_SEL values
#define MPU6050_AFS_SEL_0 (0)
#define MPU6050_AFS_SEL_1 (bit(MPU6050_AFS_SEL0))
#define MPU6050_AFS_SEL_2 (bit(MPU6050_AFS_SEL1))
#define MPU6050_AFS_SEL_3 (bit(MPU6050_AFS_SEL1)|bit(MPU6050_AFS_SEL0))

// Alternative names for the combined definitions
// The name uses the full scale range for the accelerometer.
#define MPU6050_AFS_SEL_2G  MPU6050_AFS_SEL_0
#define MPU6050_AFS_SEL_4G  MPU6050_AFS_SEL_1
```

```
#define MPU6050_AFS_SEL_8G  MPU6050_AFS_SEL_2
#define MPU6050_AFS_SEL_16G MPU6050_AFS_SEL_3

// FIFO_EN Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_SLV0_FIFO_EN  MPU6050_D0
#define MPU6050_SLV1_FIFO_EN  MPU6050_D1
#define MPU6050_SLV2_FIFO_EN  MPU6050_D2
#define MPU6050_ACCEL_FIFO_EN MPU6050_D3
#define MPU6050_ZG_FIFO_EN    MPU6050_D4
#define MPU6050_YG_FIFO_EN    MPU6050_D5
#define MPU6050_XG_FIFO_EN    MPU6050_D6
#define MPU6050_TEMP_FIFO_EN  MPU6050_D7

// I2C_MST_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_MST_CLK0  MPU6050_D0
#define MPU6050_I2C_MST_CLK1  MPU6050_D1
#define MPU6050_I2C_MST_CLK2  MPU6050_D2
#define MPU6050_I2C_MST_CLK3  MPU6050_D3
#define MPU6050_I2C_MST_P_NSR MPU6050_D4
#define MPU6050_SLV_3_FIFO_EN MPU6050_D5
#define MPU6050_WAIT_FOR_ES   MPU6050_D6
#define MPU6050_MULT_MST_EN   MPU6050_D7

// Combined definitions for the I2C_MST_CLK
#define MPU6050_I2C_MST_CLK_0 (0)
#define MPU6050_I2C_MST_CLK_1  (bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_2  (bit(MPU6050_I2C_MST_CLK1))
#define MPU6050_I2C_MST_CLK_3
(bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_4  (bit(MPU6050_I2C_MST_CLK2))
```

```
        #define MPU6050_I2C_MST_CLK_5 (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK0))
        #define MPU6050_I2C_MST_CLK_6 (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1))
        #define MPU6050_I2C_MST_CLK_7 (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
        #define MPU6050_I2C_MST_CLK_8  (bit(MPU6050_I2C_MST_CLK3))
        #define MPU6050_I2C_MST_CLK_9 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK0))
        #define MPU6050_I2C_MST_CLK_10 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1))
        #define MPU6050_I2C_MST_CLK_11 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
        #define MPU6050_I2C_MST_CLK_12 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2))
        #define MPU6050_I2C_MST_CLK_13 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK0))
        #define MPU6050_I2C_MST_CLK_14 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1))
        #define MPU6050_I2C_MST_CLK_15 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))

        // Alternative names for the combined definitions
        // The names uses I2C Master Clock Speed in kHz.
        #define MPU6050_I2C_MST_CLK_348KHZ MPU6050_I2C_MST_CLK_0
        #define MPU6050_I2C_MST_CLK_333KHZ MPU6050_I2C_MST_CLK_1
        #define MPU6050_I2C_MST_CLK_320KHZ MPU6050_I2C_MST_CLK_2
        #define MPU6050_I2C_MST_CLK_308KHZ MPU6050_I2C_MST_CLK_3
        #define MPU6050_I2C_MST_CLK_296KHZ MPU6050_I2C_MST_CLK_4
```

```
#define MPU6050_I2C_MST_CLK_286KHZ MPU6050_I2C_MST_CLK_5
#define MPU6050_I2C_MST_CLK_276KHZ MPU6050_I2C_MST_CLK_6
#define MPU6050_I2C_MST_CLK_267KHZ MPU6050_I2C_MST_CLK_7
#define MPU6050_I2C_MST_CLK_258KHZ MPU6050_I2C_MST_CLK_8
#define MPU6050_I2C_MST_CLK_500KHZ MPU6050_I2C_MST_CLK_9
#define MPU6050_I2C_MST_CLK_471KHZ MPU6050_I2C_MST_CLK_10
#define MPU6050_I2C_MST_CLK_444KHZ MPU6050_I2C_MST_CLK_11
#define MPU6050_I2C_MST_CLK_421KHZ MPU6050_I2C_MST_CLK_12
#define MPU6050_I2C_MST_CLK_400KHZ MPU6050_I2C_MST_CLK_13
#define MPU6050_I2C_MST_CLK_381KHZ MPU6050_I2C_MST_CLK_14
#define MPU6050_I2C_MST_CLK_364KHZ MPU6050_I2C_MST_CLK_15

// I2C_SLV0_ADDR Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV0_RW MPU6050_D7

// I2C_SLV0_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV0_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV0_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV0_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV0_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV0_GRP     MPU6050_D4
#define MPU6050_I2C_SLV0_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV0_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV0_EN      MPU6050_D7

// A mask for the length
#define MPU6050_I2C_SLV0_LEN_MASK 0x0F

// I2C_SLV1_ADDR Register
// These are the names for the bits.
```

```
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV1_RW MPU6050_D7

// I2C_SLV1_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV1_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV1_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV1_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV1_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV1_GRP     MPU6050_D4
#define MPU6050_I2C_SLV1_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV1_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV1_EN      MPU6050_D7

// A mask for the length
#define MPU6050_I2C_SLV1_LEN_MASK 0x0F

// I2C_SLV2_ADDR Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV2_RW MPU6050_D7

// I2C_SLV2_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV2_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV2_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV2_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV2_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV2_GRP     MPU6050_D4
#define MPU6050_I2C_SLV2_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV2_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV2_EN      MPU6050_D7
```

```
// A mask for the length
#define MPU6050_I2C_SLV2_LEN_MASK 0x0F

// I2C_SLV3_ADDR Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV3_RW MPU6050_D7

// I2C_SLV3_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV3_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV3_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV3_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV3_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV3_GRP     MPU6050_D4
#define MPU6050_I2C_SLV3_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV3_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV3_EN      MPU6050_D7

// A mask for the length
#define MPU6050_I2C_SLV3_LEN_MASK 0x0F

// I2C_SLV4_ADDR Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV4_RW MPU6050_D7

// I2C_SLV4_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_MST_DLY0     MPU6050_D0
#define MPU6050_I2C_MST_DLY1     MPU6050_D1
```

```
#define MPU6050_I2C_MST_DLY2     MPU6050_D2
#define MPU6050_I2C_MST_DLY3     MPU6050_D3
#define MPU6050_I2C_MST_DLY4     MPU6050_D4
#define MPU6050_I2C_SLV4_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV4_INT_EN  MPU6050_D6
#define MPU6050_I2C_SLV4_EN      MPU6050_D7

// A mask for the delay
#define MPU6050_I2C_MST_DLY_MASK 0x1F

// I2C_MST_STATUS Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV0_NACK MPU6050_D0
#define MPU6050_I2C_SLV1_NACK MPU6050_D1
#define MPU6050_I2C_SLV2_NACK MPU6050_D2
#define MPU6050_I2C_SLV3_NACK MPU6050_D3
#define MPU6050_I2C_SLV4_NACK MPU6050_D4
#define MPU6050_I2C_LOST_ARB  MPU6050_D5
#define MPU6050_I2C_SLV4_DONE MPU6050_D6
#define MPU6050_PASS_THROUGH  MPU6050_D7

// I2C_PIN_CFG Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_CLKOUT_EN       MPU6050_D0
#define MPU6050_I2C_BYPASS_EN   MPU6050_D1
#define MPU6050_FSYNC_INT_EN    MPU6050_D2
#define MPU6050_FSYNC_INT_LEVEL MPU6050_D3
#define MPU6050_INT_RD_CLEAR    MPU6050_D4
#define MPU6050_LATCH_INT_EN    MPU6050_D5
#define MPU6050_INT_OPEN        MPU6050_D6
#define MPU6050_INT_LEVEL       MPU6050_D7
```

```
// INT_ENABLE Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_DATA_RDY_EN    MPU6050_D0
#define MPU6050_I2C_MST_INT_EN MPU6050_D3
#define MPU6050_FIFO_OFLOW_EN  MPU6050_D4
#define MPU6050_ZMOT_EN        MPU6050_D5
#define MPU6050_MOT_EN         MPU6050_D6
#define MPU6050_FF_EN          MPU6050_D7

// INT_STATUS Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_DATA_RDY_INT   MPU6050_D0
#define MPU6050_I2C_MST_INT    MPU6050_D3
#define MPU6050_FIFO_OFLOW_INT MPU6050_D4
#define MPU6050_ZMOT_INT       MPU6050_D5
#define MPU6050_MOT_INT        MPU6050_D6
#define MPU6050_FF_INT         MPU6050_D7

// MOT_DETECT_STATUS Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_MOT_ZRMOT MPU6050_D0
#define MPU6050_MOT_ZPOS  MPU6050_D2
#define MPU6050_MOT_ZNEG  MPU6050_D3
#define MPU6050_MOT_YPOS  MPU6050_D4
#define MPU6050_MOT_YNEG  MPU6050_D5
#define MPU6050_MOT_XPOS  MPU6050_D6
#define MPU6050_MOT_XNEG  MPU6050_D7

// IC2_MST_DELAY_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
```

```
#define MPU6050_I2C_SLV0_DLY_EN MPU6050_D0
#define MPU6050_I2C_SLV1_DLY_EN MPU6050_D1
#define MPU6050_I2C_SLV2_DLY_EN MPU6050_D2
#define MPU6050_I2C_SLV3_DLY_EN MPU6050_D3
#define MPU6050_I2C_SLV4_DLY_EN MPU6050_D4
#define MPU6050_DELAY_ES_SHADOW MPU6050_D7

// SIGNAL_PATH_RESET Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_TEMP_RESET  MPU6050_D0
#define MPU6050_ACCEL_RESET MPU6050_D1
#define MPU6050_GYRO_RESET  MPU6050_D2

// MOT_DETECT_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_MOT_COUNT0      MPU6050_D0
#define MPU6050_MOT_COUNT1      MPU6050_D1
#define MPU6050_FF_COUNT0       MPU6050_D2
#define MPU6050_FF_COUNT1       MPU6050_D3
#define MPU6050_ACCEL_ON_DELAY0 MPU6050_D4
#define MPU6050_ACCEL_ON_DELAY1 MPU6050_D5

// Combined definitions for the MOT_COUNT
#define MPU6050_MOT_COUNT_0 (0)
#define MPU6050_MOT_COUNT_1 (bit(MPU6050_MOT_COUNT0))
#define MPU6050_MOT_COUNT_2 (bit(MPU6050_MOT_COUNT1))
#define MPU6050_MOT_COUNT_3
(bit(MPU6050_MOT_COUNT1)|bit(MPU6050_MOT_COUNT0))

// Alternative names for the combined definitions
#define MPU6050_MOT_COUNT_RESET MPU6050_MOT_COUNT_0
```

```
        // Combined definitions for the FF_COUNT
        #define MPU6050_FF_COUNT_0 (0)
        #define MPU6050_FF_COUNT_1 (bit(MPU6050_FF_COUNT0))
        #define MPU6050_FF_COUNT_2 (bit(MPU6050_FF_COUNT1))
        #define MPU6050_FF_COUNT_3
(bit(MPU6050_FF_COUNT1)|bit(MPU6050_FF_COUNT0))

        // Alternative names for the combined definitions
        #define MPU6050_FF_COUNT_RESET MPU6050_FF_COUNT_0

        // Combined definitions for the ACCEL_ON_DELAY
        #define MPU6050_ACCEL_ON_DELAY_0 (0)
        #define MPU6050_ACCEL_ON_DELAY_1
(bit(MPU6050_ACCEL_ON_DELAY0))
        #define MPU6050_ACCEL_ON_DELAY_2
(bit(MPU6050_ACCEL_ON_DELAY1))
        #define MPU6050_ACCEL_ON_DELAY_3
(bit(MPU6050_ACCEL_ON_DELAY1)|bit(MPU6050_ACCEL_ON_DELAY0))

        // Alternative names for the ACCEL_ON_DELAY
        #define MPU6050_ACCEL_ON_DELAY_0MS
MPU6050_ACCEL_ON_DELAY_0
        #define MPU6050_ACCEL_ON_DELAY_1MS
MPU6050_ACCEL_ON_DELAY_1
        #define MPU6050_ACCEL_ON_DELAY_2MS
MPU6050_ACCEL_ON_DELAY_2
        #define MPU6050_ACCEL_ON_DELAY_3MS
MPU6050_ACCEL_ON_DELAY_3

        // USER_CTRL Register
        // These are the names for the bits.
        // Use these only with the bit() macro.
        #define MPU6050_SIG_COND_RESET MPU6050_D0
        #define MPU6050_I2C_MST_RESET  MPU6050_D1
```

```
#define MPU6050_FIFO_RESET     MPU6050_D2
#define MPU6050_I2C_IF_DIS     MPU6050_D4   // must be 0 for MPU-6050
#define MPU6050_I2C_MST_EN     MPU6050_D5
#define MPU6050_FIFO_EN        MPU6050_D6

// PWR_MGMT_1 Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_CLKSEL0      MPU6050_D0
#define MPU6050_CLKSEL1      MPU6050_D1
#define MPU6050_CLKSEL2      MPU6050_D2
#define MPU6050_TEMP_DIS       MPU6050_D3      // 1: disable temperature sensor
#define MPU6050_CYCLE        MPU6050_D5    // 1: sample and sleep
#define MPU6050_SLEEP        MPU6050_D6    // 1: sleep mode
#define MPU6050_DEVICE_RESET MPU6050_D7      // 1: reset to default values

// Combined definitions for the CLKSEL
#define MPU6050_CLKSEL_0 (0)
#define MPU6050_CLKSEL_1 (bit(MPU6050_CLKSEL0))
#define MPU6050_CLKSEL_2 (bit(MPU6050_CLKSEL1))
#define                                                MPU6050_CLKSEL_3 (bit(MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))
#define MPU6050_CLKSEL_4 (bit(MPU6050_CLKSEL2))
#define                                                MPU6050_CLKSEL_5 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL0))
#define                                                MPU6050_CLKSEL_6 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL1))
#define                                                MPU6050_CLKSEL_7 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))

// Alternative names for the combined definitions
#define MPU6050_CLKSEL_INTERNAL    MPU6050_CLKSEL_0
```

```
#define MPU6050_CLKSEL_X          MPU6050_CLKSEL_1
#define MPU6050_CLKSEL_Y          MPU6050_CLKSEL_2
#define MPU6050_CLKSEL_Z          MPU6050_CLKSEL_3
#define MPU6050_CLKSEL_EXT_32KHZ   MPU6050_CLKSEL_4
#define MPU6050_CLKSEL_EXT_19_2MHZ MPU6050_CLKSEL_5
#define MPU6050_CLKSEL_RESERVED    MPU6050_CLKSEL_6
#define MPU6050_CLKSEL_STOP        MPU6050_CLKSEL_7

// PWR_MGMT_2 Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_STBY_ZG       MPU6050_D0
#define MPU6050_STBY_YG       MPU6050_D1
#define MPU6050_STBY_XG       MPU6050_D2
#define MPU6050_STBY_ZA       MPU6050_D3
#define MPU6050_STBY_YA       MPU6050_D4
#define MPU6050_STBY_XA       MPU6050_D5
#define MPU6050_LP_WAKE_CTRL0 MPU6050_D6
#define MPU6050_LP_WAKE_CTRL1 MPU6050_D7

// Combined definitions for the LP_WAKE_CTRL
#define MPU6050_LP_WAKE_CTRL_0 (0)
#define MPU6050_LP_WAKE_CTRL_1 (bit(MPU6050_LP_WAKE_CTRL0))
#define MPU6050_LP_WAKE_CTRL_2 (bit(MPU6050_LP_WAKE_CTRL1))
#define                                    MPU6050_LP_WAKE_CTRL_3
(bit(MPU6050_LP_WAKE_CTRL1)|bit(MPU6050_LP_WAKE_CTRL0))

// Alternative names for the combined definitions
// The names uses the Wake-up Frequency.
#define MPU6050_LP_WAKE_1_25HZ MPU6050_LP_WAKE_CTRL_0
#define MPU6050_LP_WAKE_2_5HZ  MPU6050_LP_WAKE_CTRL_1
#define MPU6050_LP_WAKE_5HZ    MPU6050_LP_WAKE_CTRL_2
#define MPU6050_LP_WAKE_10HZ   MPU6050_LP_WAKE_CTRL_3
```

```
// Default I2C address for the MPU-6050 is 0x68.
// But only if the AD0 pin is low.
// Some sensor boards have AD0 high, and the
// I2C address thus becomes 0x69.
#define MPU6050_I2C_ADDRESS 0x68


// Declaring an union for the registers and the axis values.
// The byte order does not match the byte order of
// the compiler and AVR chip.
// The AVR chip (on the Arduino board) has the Low Byte
// at the lower address.
// But the MPU-6050 has a different order: High Byte at
// lower address, so that has to be corrected.
// The register part "reg" is only used internally,
// and are swapped in code.
typedef union accel_t_gyro_union
{
  struct
  {
    uint8_t x_accel_h;
    uint8_t x_accel_l;
    uint8_t y_accel_h;
    uint8_t y_accel_l;
    uint8_t z_accel_h;
    uint8_t z_accel_l;
    uint8_t t_h;
    uint8_t t_l;
    uint8_t x_gyro_h;
    uint8_t x_gyro_l;
    uint8_t y_gyro_h;
    uint8_t y_gyro_l;
    uint8_t z_gyro_h;
```

```
  uint8_t z_gyro_l;
 } reg;
 struct
 {
  int16_t x_accel;
  int16_t y_accel;
  int16_t z_accel;
  int16_t temperature;
  int16_t x_gyro;
  int16_t y_gyro;
  int16_t z_gyro;
 } value;
};


void setup()
{
 int error;
 uint8_t c;


 Serial.begin(9600);
 Serial.println(F("InvenSense MPU-6050"));
 Serial.println(F("June 2012"));

 // Initialize the 'Wire' class for the I2C-bus.
 Wire.begin();


 // default at power-up:
 //    Gyro at 250 degrees second
 //    Acceleration at 2g
 //    Clock source at internal 8MHz
 //    The device is in sleep mode.
```

```
  //

  error = MPU6050_read (MPU6050_WHO_AM_I, &c, 1);
  Serial.print(F("WHO_AM_I : "));
  Serial.print(c,HEX);
  Serial.print(F(", error = "));
  Serial.println(error,DEC);

  // According to the datasheet, the 'sleep' bit
  // should read a '1'.
  // That bit has to be cleared, since the sensor
  // is in sleep mode at power-up.
  error = MPU6050_read (MPU6050_PWR_MGMT_1, &c, 1);
  Serial.print(F("PWR_MGMT_1 : "));
  Serial.print(c,HEX);
  Serial.print(F(", error = "));
  Serial.println(error,DEC);


  // Clear the 'sleep' bit to start the sensor.
  MPU6050_write_reg (MPU6050_PWR_MGMT_1, 0);
}


void loop()
{
  int error;
  double dT;
  accel_t_gyro_union accel_t_gyro;


  Serial.println(F(""));
  Serial.println(F("MPU-6050"));
```

```
        // Read the raw values.
        // Read 14 bytes at once,
        // containing acceleration, temperature and gyro.
        // With the default settings of the MPU-6050,
        // there is no filter enabled, and the values
        // are not very stable.
        error = MPU6050_read (MPU6050_ACCEL_XOUT_H, (uint8_t *)
&accel_t_gyro,sizeof(accel_t_gyro));
        Serial.print(F("Read accel, temp and gyro, error = "));
        Serial.println(error,DEC);


        // Swap all high and low bytes.
        // After this, the registers values are swapped,
        // so the structure name like x_accel_l does no
        // longer contain the lower byte.
        uint8_t swap;
        #define SWAP(x,y) swap = x; x = y; y = swap

        SWAP (accel_t_gyro.reg.x_accel_h, accel_t_gyro.reg.x_accel_l);
        SWAP (accel_t_gyro.reg.y_accel_h, accel_t_gyro.reg.y_accel_l);
        SWAP (accel_t_gyro.reg.z_accel_h, accel_t_gyro.reg.z_accel_l);
        SWAP (accel_t_gyro.reg.t_h, accel_t_gyro.reg.t_l);
        SWAP (accel_t_gyro.reg.x_gyro_h, accel_t_gyro.reg.x_gyro_l);
        SWAP (accel_t_gyro.reg.y_gyro_h, accel_t_gyro.reg.y_gyro_l);
        SWAP (accel_t_gyro.reg.z_gyro_h, accel_t_gyro.reg.z_gyro_l);


        // Print the raw acceleration values

        Serial.print(F("accel x,y,z: "));
        Serial.print(accel_t_gyro.value.x_accel, DEC);
        Serial.print(F(", "));
        Serial.print(accel_t_gyro.value.y_accel, DEC);
```

```
  Serial.print(F(", "));
  Serial.print(accel_t_gyro.value.z_accel, DEC);
  Serial.println(F(""));


  // The temperature sensor is -40 to +85 degrees Celsius.
  // It is a signed integer.
  // According to the datasheet:
  //   340 per degrees Celsius, -512 at 35 degrees.
  // At 0 degrees: -512 - (340 * 35) = -12412

  Serial.print(F("temperature: "));
  dT = ( (double) accel_t_gyro.value.temperature + 12412.0) / 340.0;
  Serial.print(dT, 3);
  Serial.print(F(" degrees Celsius"));
  Serial.println(F(""));


  // Print the raw gyro values.

  Serial.print(F("gyro x,y,z : "));
  Serial.print(accel_t_gyro.value.x_gyro, DEC);
  Serial.print(F(", "));
  Serial.print(accel_t_gyro.value.y_gyro, DEC);
  Serial.print(F(", "));
  Serial.print(accel_t_gyro.value.z_gyro, DEC);
  Serial.print(F(", "));
  Serial.println(F(""));

  delay(1000);
}


// ----------------------------------------------------------
```

```
// MPU6050_read
//
// This is a common function to read multiple bytes
// from an I2C device.
//
// It uses the boolean parameter for Wire.endTransMission()
// to be able to hold or release the I2C-bus.
// This is implemented in Arduino 1.0.1.
//
// Only this function is used to read.
// There is no function for a single byte.
//
int MPU6050_read(int start, uint8_t *buffer, int size)
{
 int i, n, error;

 Wire.beginTransmission(MPU6050_I2C_ADDRESS);
 n = Wire.write(start);
 if (n != 1)
  return (-10);

 n = Wire.endTransmission(false);    // hold the I2C-bus
 if (n != 0)
  return (n);

 // Third parameter is true: relase I2C-bus after data is read.
 Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);
 i = 0;
 while(Wire.available() && i<size)
 {
  buffer[i++]=Wire.read();
 }
 if ( i != size)
  return (-11);
```

```
  return (0);  // return : no error
}


// ----------------------------------------------------
// MPU6050_write
//
// This is a common function to write multiple bytes to an I2C device.
//
// If only a single register is written,
// use the function MPU_6050_write_reg().
//
// Parameters:
//   start : Start address, use a define for the register
//   pData : A pointer to the data to write.
//   size  : The number of bytes to write.
//
// If only a single register is written, a pointer
// to the data has to be used, and the size is
// a single byte:
//   int data = 0;        // the data to write
//   MPU6050_write (MPU6050_PWR_MGMT_1, &c, 1);
//
int MPU6050_write(int start, const uint8_t *pData, int size)
{
  int n, error;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
  n = Wire.write(start);        // write the start address
  if (n != 1)
    return (-20);

  n = Wire.write(pData, size);  // write data bytes
```

```
  if (n != size)
    return (-21);

  error = Wire.endTransmission(true); // release the I2C-bus
  if (error != 0)
    return (error);

  return (0);         // return : no error
}

// ----------------------------------------------------
// MPU6050_write_reg
//
// An extra function to write a single register.
// It is just a wrapper around the MPU_6050_write()
// function, and it is only a convenient function
// to make it easier to write a single register.
//
int MPU6050_write_reg(int reg, uint8_t data)
{
  int error;

  error = MPU6050_write(reg, &data, 1);

  return (error);
}
```

Anexo B: Esquema dos pinos do Arduino UNO

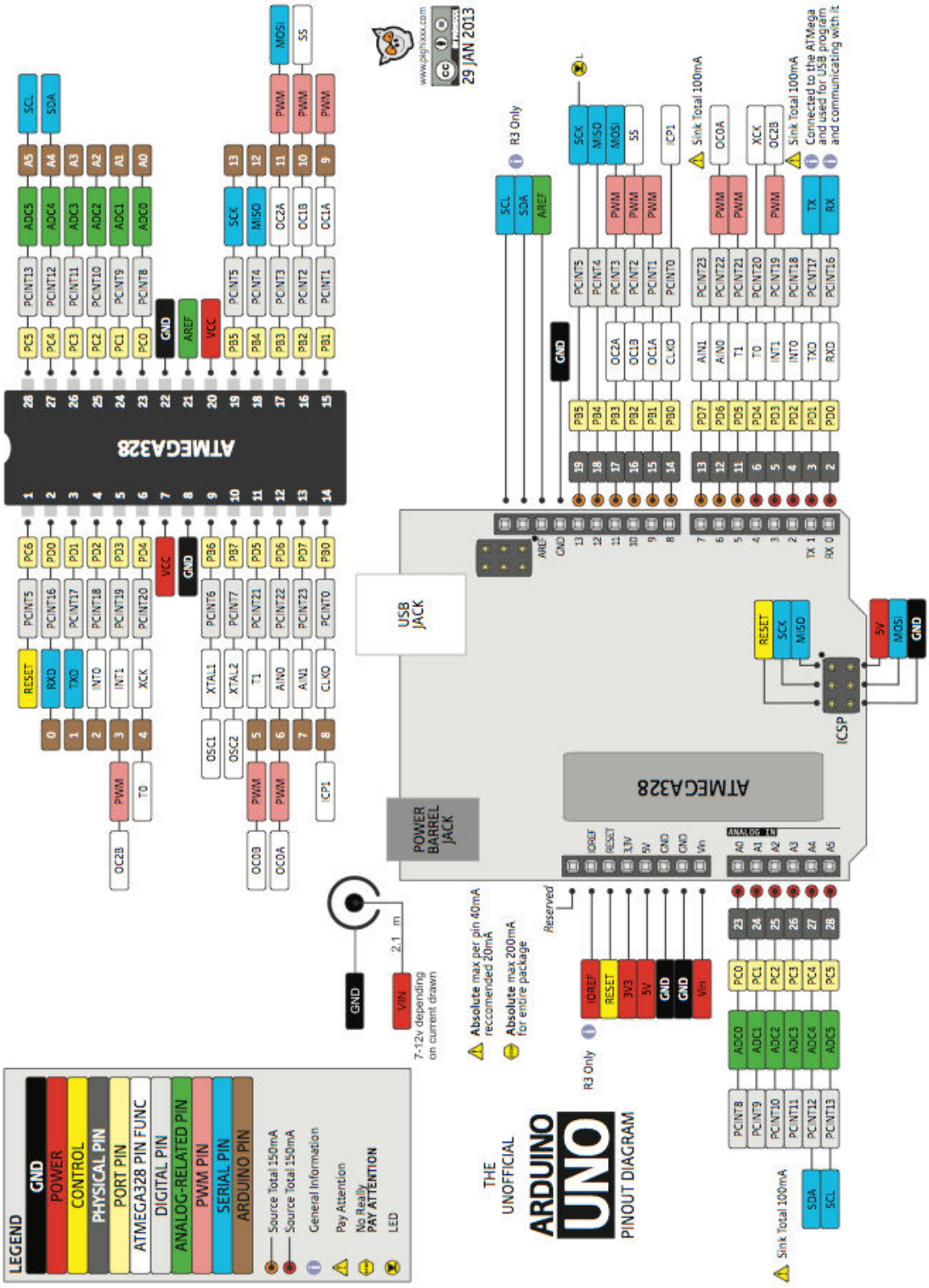

Figura 10: Esquema de Pinagem do Arduino. Fonte: PighiXXX.

Apêndice A: Código modificado para testes do GY-521

```
#include <Wire.h>

#define MPU6050_AUX_VDDIO          0x01
#define MPU6050_SMPLRT_DIV         0x19
#define MPU6050_CONFIG             0x1A
#define MPU6050_GYRO_CONFIG        0x1B
#define MPU6050_ACCEL_CONFIG       0x1C
#define MPU6050_FF_THR             0x1D
#define MPU6050_FF_DUR             0x1E
#define MPU6050_MOT_THR            0x1F
#define MPU6050_MOT_DUR            0x20
#define MPU6050_ZRMOT_THR          0x21
#define MPU6050_ZRMOT_DUR          0x22
#define MPU6050_FIFO_EN            0x23
#define MPU6050_I2C_MST_CTRL       0x24
#define MPU6050_I2C_SLV0_ADDR      0x25
#define MPU6050_I2C_SLV0_REG       0x26
#define MPU6050_I2C_SLV0_CTRL      0x27
#define MPU6050_I2C_SLV1_ADDR      0x28
#define MPU6050_I2C_SLV1_REG       0x29
#define MPU6050_I2C_SLV1_CTRL      0x2A
#define MPU6050_I2C_SLV2_ADDR      0x2B
#define MPU6050_I2C_SLV2_REG       0x2C
#define MPU6050_I2C_SLV2_CTRL      0x2D
#define MPU6050_I2C_SLV3_ADDR      0x2E
#define MPU6050_I2C_SLV3_REG       0x2F
#define MPU6050_I2C_SLV3_CTRL      0x30
#define MPU6050_I2C_SLV4_ADDR      0x31
#define MPU6050_I2C_SLV4_REG       0x32
#define MPU6050_I2C_SLV4_DO        0x33
#define MPU6050_I2C_SLV4_CTRL      0x34
```

```
#define MPU6050_I2C_SLV4_DI       0x35
#define MPU6050_I2C_MST_STATUS    0x36
#define MPU6050_INT_PIN_CFG       0x37
#define MPU6050_INT_ENABLE        0x38
#define MPU6050_INT_STATUS        0x3A
#define MPU6050_ACCEL_XOUT_H      0x3B
#define MPU6050_ACCEL_XOUT_L      0x3C
#define MPU6050_ACCEL_YOUT_H      0x3D
#define MPU6050_ACCEL_YOUT_L      0x3E
#define MPU6050_ACCEL_ZOUT_H      0x3F
#define MPU6050_ACCEL_ZOUT_L      0x40
#define MPU6050_TEMP_OUT_H        0x41
#define MPU6050_TEMP_OUT_L        0x42
#define MPU6050_GYRO_XOUT_H       0x43
#define MPU6050_GYRO_XOUT_L       0x44
#define MPU6050_GYRO_YOUT_H       0x45
#define MPU6050_GYRO_YOUT_L       0x46
#define MPU6050_GYRO_ZOUT_H       0x47
#define MPU6050_GYRO_ZOUT_L       0x48
#define MPU6050_EXT_SENS_DATA_00  0x49
#define MPU6050_EXT_SENS_DATA_01  0x4A
#define MPU6050_EXT_SENS_DATA_02  0x4B
#define MPU6050_EXT_SENS_DATA_03  0x4C
#define MPU6050_EXT_SENS_DATA_04  0x4D
#define MPU6050_EXT_SENS_DATA_05  0x4E
#define MPU6050_EXT_SENS_DATA_06  0x4F
#define MPU6050_EXT_SENS_DATA_07  0x50
#define MPU6050_EXT_SENS_DATA_08  0x51
#define MPU6050_EXT_SENS_DATA_09  0x52
#define MPU6050_EXT_SENS_DATA_10  0x53
#define MPU6050_EXT_SENS_DATA_11  0x54
#define MPU6050_EXT_SENS_DATA_12  0x55
#define MPU6050_EXT_SENS_DATA_13  0x56
#define MPU6050_EXT_SENS_DATA_14  0x57
```

```
#define MPU6050_EXT_SENS_DATA_15   0x58
#define MPU6050_EXT_SENS_DATA_16   0x59
#define MPU6050_EXT_SENS_DATA_17   0x5A
#define MPU6050_EXT_SENS_DATA_18   0x5B
#define MPU6050_EXT_SENS_DATA_19   0x5C
#define MPU6050_EXT_SENS_DATA_20   0x5D
#define MPU6050_EXT_SENS_DATA_21   0x5E
#define MPU6050_EXT_SENS_DATA_22   0x5F
#define MPU6050_EXT_SENS_DATA_23   0x60
#define MPU6050_MOT_DETECT_STATUS  0x61
#define MPU6050_I2C_SLV0_DO        0x63
#define MPU6050_I2C_SLV1_DO        0x64
#define MPU6050_I2C_SLV2_DO        0x65
#define MPU6050_I2C_SLV3_DO        0x66
#define MPU6050_I2C_MST_DELAY_CTRL 0x67
#define MPU6050_SIGNAL_PATH_RESET  0x68
#define MPU6050_MOT_DETECT_CTRL    0x69
#define MPU6050_USER_CTRL          0x6A
#define MPU6050_PWR_MGMT_1         0x6B
#define MPU6050_PWR_MGMT_2         0x6C
#define MPU6050_FIFO_COUNTH        0x72
#define MPU6050_FIFO_COUNTL        0x73
#define MPU6050_FIFO_R_W           0x74
#define MPU6050_WHO_AM_I           0x75


#define MPU6050_D0 0
#define MPU6050_D1 1
#define MPU6050_D2 2
#define MPU6050_D3 3
#define MPU6050_D4 4
#define MPU6050_D5 5
#define MPU6050_D6 6
#define MPU6050_D7 7
```

```
#define MPU6050_AUX_VDDIO MPU6050_D7

#define MPU6050_DLPF_CFG0     MPU6050_D0
#define MPU6050_DLPF_CFG1     MPU6050_D1
#define MPU6050_DLPF_CFG2     MPU6050_D2
#define MPU6050_EXT_SYNC_SET0 MPU6050_D3
#define MPU6050_EXT_SYNC_SET1 MPU6050_D4
#define MPU6050_EXT_SYNC_SET2 MPU6050_D5

#define MPU6050_EXT_SYNC_SET_0 (0)
#define MPU6050_EXT_SYNC_SET_1 (bit(MPU6050_EXT_SYNC_SET0))
#define MPU6050_EXT_SYNC_SET_2 (bit(MPU6050_EXT_SYNC_SET1))
#define MPU6050_EXT_SYNC_SET_3 (bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT_SYNC_SET0))
#define MPU6050_EXT_SYNC_SET_4 (bit(MPU6050_EXT_SYNC_SET2))
#define MPU6050_EXT_SYNC_SET_5 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET0))
#define MPU6050_EXT_SYNC_SET_6 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1))
#define MPU6050_EXT_SYNC_SET_7 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT_SYNC_SET0))

#define MPU6050_EXT_SYNC_DISABLED MPU6050_EXT_SYNC_SET_0
#define MPU6050_EXT_SYNC_TEMP_OUT_L MPU6050_EXT_SYNC_SET_1
#define MPU6050_EXT_SYNC_GYRO_XOUT_L MPU6050_EXT_SYNC_SET_2
#define MPU6050_EXT_SYNC_GYRO_YOUT_L MPU6050_EXT_SYNC_SET_3
#define MPU6050_EXT_SYNC_GYRO_ZOUT_L MPU6050_EXT_SYNC_SET_4
```

```
        #define                          MPU6050_EXT_SYNC_ACCEL_XOUT_L
MPU6050_EXT_SYNC_SET_5
        #define                          MPU6050_EXT_SYNC_ACCEL_YOUT_L
MPU6050_EXT_SYNC_SET_6
        #define                          MPU6050_EXT_SYNC_ACCEL_ZOUT_L
MPU6050_EXT_SYNC_SET_7

        #define MPU6050_DLPF_CFG_0 (0)
        #define MPU6050_DLPF_CFG_1 (bit(MPU6050_DLPF_CFG0))
        #define MPU6050_DLPF_CFG_2 (bit(MPU6050_DLPF_CFG1))
        #define                                  MPU6050_DLPF_CFG_3
(bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG0))
        #define MPU6050_DLPF_CFG_4 (bit(MPU6050_DLPF_CFG2))
        #define                                  MPU6050_DLPF_CFG_5
(bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG0))
        #define                                  MPU6050_DLPF_CFG_6
(bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1))
        #define                                  MPU6050_DLPF_CFG_7
(bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG
0))

        #define MPU6050_DLPF_260HZ    MPU6050_DLPF_CFG_0
        #define MPU6050_DLPF_184HZ    MPU6050_DLPF_CFG_1
        #define MPU6050_DLPF_94HZ     MPU6050_DLPF_CFG_2
        #define MPU6050_DLPF_44HZ     MPU6050_DLPF_CFG_3
        #define MPU6050_DLPF_21HZ     MPU6050_DLPF_CFG_4
        #define MPU6050_DLPF_10HZ     MPU6050_DLPF_CFG_5
        #define MPU6050_DLPF_5HZ      MPU6050_DLPF_CFG_6
        #define MPU6050_DLPF_RESERVED MPU6050_DLPF_CFG_7

        #define MPU6050_FS_SEL0 MPU6050_D3
        #define MPU6050_FS_SEL1 MPU6050_D4
        #define MPU6050_ZG_ST   MPU6050_D5
        #define MPU6050_YG_ST   MPU6050_D6
```

```
#define MPU6050_XG_ST   MPU6050_D7

#define MPU6050_FS_SEL_0 (0)
#define MPU6050_FS_SEL_1 (bit(MPU6050_FS_SEL0))
#define MPU6050_FS_SEL_2 (bit(MPU6050_FS_SEL1))
#define MPU6050_FS_SEL_3 (bit(MPU6050_FS_SEL1)|bit(MPU6050_FS_SEL0))

#define MPU6050_FS_SEL_250  MPU6050_FS_SEL_0
#define MPU6050_FS_SEL_500  MPU6050_FS_SEL_1
#define MPU6050_FS_SEL_1000 MPU6050_FS_SEL_2
#define MPU6050_FS_SEL_2000 MPU6050_FS_SEL_3

#define MPU6050_ACCEL_HPF0 MPU6050_D0
#define MPU6050_ACCEL_HPF1 MPU6050_D1
#define MPU6050_ACCEL_HPF2 MPU6050_D2
#define MPU6050_AFS_SEL0   MPU6050_D3
#define MPU6050_AFS_SEL1   MPU6050_D4
#define MPU6050_ZA_ST      MPU6050_D5
#define MPU6050_YA_ST      MPU6050_D6
#define MPU6050_XA_ST      MPU6050_D7

#define MPU6050_ACCEL_HPF_0 (0)
#define MPU6050_ACCEL_HPF_1 (bit(MPU6050_ACCEL_HPF0))
#define MPU6050_ACCEL_HPF_2 (bit(MPU6050_ACCEL_HPF1))
#define MPU6050_ACCEL_HPF_3 (bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL_HPF0))
#define MPU6050_ACCEL_HPF_4 (bit(MPU6050_ACCEL_HPF2))
#define MPU6050_ACCEL_HPF_7 (bit(MPU6050_ACCEL_HPF2)|bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL
_HPF0))

#define MPU6050_ACCEL_HPF_RESET  MPU6050_ACCEL_HPF_0
#define MPU6050_ACCEL_HPF_5HZ    MPU6050_ACCEL_HPF_1
```

```
#define MPU6050_ACCEL_HPF_2_5HZ  MPU6050_ACCEL_HPF_2
#define MPU6050_ACCEL_HPF_1_25HZ MPU6050_ACCEL_HPF_3
#define MPU6050_ACCEL_HPF_0_63HZ MPU6050_ACCEL_HPF_4
#define MPU6050_ACCEL_HPF_HOLD   MPU6050_ACCEL_HPF_7

#define MPU6050_AFS_SEL_0 (0)
#define MPU6050_AFS_SEL_1 (bit(MPU6050_AFS_SEL0))
#define MPU6050_AFS_SEL_2 (bit(MPU6050_AFS_SEL1))
#define                                                MPU6050_AFS_SEL_3
(bit(MPU6050_AFS_SEL1)|bit(MPU6050_AFS_SEL0))

#define MPU6050_AFS_SEL_2G  MPU6050_AFS_SEL_0
#define MPU6050_AFS_SEL_4G  MPU6050_AFS_SEL_1
#define MPU6050_AFS_SEL_8G  MPU6050_AFS_SEL_2
#define MPU6050_AFS_SEL_16G MPU6050_AFS_SEL_3

#define MPU6050_SLV0_FIFO_EN  MPU6050_D0
#define MPU6050_SLV1_FIFO_EN  MPU6050_D1
#define MPU6050_SLV2_FIFO_EN  MPU6050_D2
#define MPU6050_ACCEL_FIFO_EN MPU6050_D3
#define MPU6050_ZG_FIFO_EN    MPU6050_D4
#define MPU6050_YG_FIFO_EN    MPU6050_D5
#define MPU6050_XG_FIFO_EN    MPU6050_D6
#define MPU6050_TEMP_FIFO_EN  MPU6050_D7

#define MPU6050_I2C_MST_CLK0  MPU6050_D0
#define MPU6050_I2C_MST_CLK1  MPU6050_D1
#define MPU6050_I2C_MST_CLK2  MPU6050_D2
#define MPU6050_I2C_MST_CLK3  MPU6050_D3
#define MPU6050_I2C_MST_P_NSR MPU6050_D4
#define MPU6050_SLV_3_FIFO_EN MPU6050_D5
#define MPU6050_WAIT_FOR_ES   MPU6050_D6
#define MPU6050_MULT_MST_EN   MPU6050_D7
```

```
#define MPU6050_I2C_MST_CLK_0 (0)
#define MPU6050_I2C_MST_CLK_1  (bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_2  (bit(MPU6050_I2C_MST_CLK1))
#define MPU6050_I2C_MST_CLK_3 (bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_4  (bit(MPU6050_I2C_MST_CLK2))
#define MPU6050_I2C_MST_CLK_5 (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_6 (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1))
#define MPU6050_I2C_MST_CLK_7 (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_8  (bit(MPU6050_I2C_MST_CLK3))
#define MPU6050_I2C_MST_CLK_9 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_10 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1))
#define MPU6050_I2C_MST_CLK_11 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_12 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2))
#define MPU6050_I2C_MST_CLK_13 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_14 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1))
#define MPU6050_I2C_MST_CLK_15 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))

#define MPU6050_I2C_MST_CLK_348KHZ MPU6050_I2C_MST_CLK_0
```

```
#define MPU6050_I2C_MST_CLK_333KHZ MPU6050_I2C_MST_CLK_1
#define MPU6050_I2C_MST_CLK_320KHZ MPU6050_I2C_MST_CLK_2
#define MPU6050_I2C_MST_CLK_308KHZ MPU6050_I2C_MST_CLK_3
#define MPU6050_I2C_MST_CLK_296KHZ MPU6050_I2C_MST_CLK_4
#define MPU6050_I2C_MST_CLK_286KHZ MPU6050_I2C_MST_CLK_5
#define MPU6050_I2C_MST_CLK_276KHZ MPU6050_I2C_MST_CLK_6
#define MPU6050_I2C_MST_CLK_267KHZ MPU6050_I2C_MST_CLK_7
#define MPU6050_I2C_MST_CLK_258KHZ MPU6050_I2C_MST_CLK_8
#define MPU6050_I2C_MST_CLK_500KHZ MPU6050_I2C_MST_CLK_9
#define MPU6050_I2C_MST_CLK_471KHZ MPU6050_I2C_MST_CLK_10
#define MPU6050_I2C_MST_CLK_444KHZ MPU6050_I2C_MST_CLK_11
#define MPU6050_I2C_MST_CLK_421KHZ MPU6050_I2C_MST_CLK_12
#define MPU6050_I2C_MST_CLK_400KHZ MPU6050_I2C_MST_CLK_13
#define MPU6050_I2C_MST_CLK_381KHZ MPU6050_I2C_MST_CLK_14
#define MPU6050_I2C_MST_CLK_364KHZ MPU6050_I2C_MST_CLK_15

#define MPU6050_I2C_SLV0_RW MPU6050_D7

#define MPU6050_I2C_SLV0_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV0_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV0_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV0_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV0_GRP     MPU6050_D4
#define MPU6050_I2C_SLV0_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV0_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV0_EN      MPU6050_D7

#define MPU6050_I2C_SLV0_LEN_MASK 0x0F

#define MPU6050_I2C_SLV1_RW MPU6050_D7

#define MPU6050_I2C_SLV1_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV1_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV1_LEN2    MPU6050_D2
```

```
#define MPU6050_I2C_SLV1_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV1_GRP     MPU6050_D4
#define MPU6050_I2C_SLV1_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV1_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV1_EN      MPU6050_D7

#define MPU6050_I2C_SLV1_LEN_MASK 0x0F

#define MPU6050_I2C_SLV2_RW MPU6050_D7

#define MPU6050_I2C_SLV2_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV2_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV2_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV2_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV2_GRP     MPU6050_D4
#define MPU6050_I2C_SLV2_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV2_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV2_EN      MPU6050_D7

#define MPU6050_I2C_SLV2_LEN_MASK 0x0F

#define MPU6050_I2C_SLV3_RW MPU6050_D7

#define MPU6050_I2C_SLV3_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV3_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV3_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV3_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV3_GRP     MPU6050_D4
#define MPU6050_I2C_SLV3_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV3_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV3_EN      MPU6050_D7

#define MPU6050_I2C_SLV3_LEN_MASK 0x0F
```

```
#define MPU6050_I2C_SLV4_RW MPU6050_D7

#define MPU6050_I2C_MST_DLY0     MPU6050_D0
#define MPU6050_I2C_MST_DLY1     MPU6050_D1
#define MPU6050_I2C_MST_DLY2     MPU6050_D2
#define MPU6050_I2C_MST_DLY3     MPU6050_D3
#define MPU6050_I2C_MST_DLY4     MPU6050_D4
#define MPU6050_I2C_SLV4_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV4_INT_EN  MPU6050_D6
#define MPU6050_I2C_SLV4_EN      MPU6050_D7

#define MPU6050_I2C_MST_DLY_MASK 0x1F

#define MPU6050_I2C_SLV0_NACK MPU6050_D0
#define MPU6050_I2C_SLV1_NACK MPU6050_D1
#define MPU6050_I2C_SLV2_NACK MPU6050_D2
#define MPU6050_I2C_SLV3_NACK MPU6050_D3
#define MPU6050_I2C_SLV4_NACK MPU6050_D4
#define MPU6050_I2C_LOST_ARB  MPU6050_D5
#define MPU6050_I2C_SLV4_DONE MPU6050_D6
#define MPU6050_PASS_THROUGH  MPU6050_D7

#define MPU6050_CLKOUT_EN       MPU6050_D0
#define MPU6050_I2C_BYPASS_EN   MPU6050_D1
#define MPU6050_FSYNC_INT_EN    MPU6050_D2
#define MPU6050_FSYNC_INT_LEVEL MPU6050_D3
#define MPU6050_INT_RD_CLEAR    MPU6050_D4
#define MPU6050_LATCH_INT_EN    MPU6050_D5
#define MPU6050_INT_OPEN        MPU6050_D6
#define MPU6050_INT_LEVEL       MPU6050_D7

#define MPU6050_DATA_RDY_EN    MPU6050_D0
#define MPU6050_I2C_MST_INT_EN MPU6050_D3
#define MPU6050_FIFO_OFLOW_EN  MPU6050_D4
```

```
#define MPU6050_ZMOT_EN        MPU6050_D5
#define MPU6050_MOT_EN         MPU6050_D6
#define MPU6050_FF_EN          MPU6050_D7

#define MPU6050_DATA_RDY_INT   MPU6050_D0
#define MPU6050_I2C_MST_INT    MPU6050_D3
#define MPU6050_FIFO_OFLOW_INT MPU6050_D4
#define MPU6050_ZMOT_INT       MPU6050_D5
#define MPU6050_MOT_INT        MPU6050_D6
#define MPU6050_FF_INT         MPU6050_D7

#define MPU6050_MOT_ZRMOT MPU6050_D0
#define MPU6050_MOT_ZPOS  MPU6050_D2
#define MPU6050_MOT_ZNEG  MPU6050_D3
#define MPU6050_MOT_YPOS  MPU6050_D4
#define MPU6050_MOT_YNEG  MPU6050_D5
#define MPU6050_MOT_XPOS  MPU6050_D6
#define MPU6050_MOT_XNEG  MPU6050_D7

#define MPU6050_I2C_SLV0_DLY_EN MPU6050_D0
#define MPU6050_I2C_SLV1_DLY_EN MPU6050_D1
#define MPU6050_I2C_SLV2_DLY_EN MPU6050_D2
#define MPU6050_I2C_SLV3_DLY_EN MPU6050_D3
#define MPU6050_I2C_SLV4_DLY_EN MPU6050_D4
#define MPU6050_DELAY_ES_SHADOW MPU6050_D7

#define MPU6050_TEMP_RESET  MPU6050_D0
#define MPU6050_ACCEL_RESET MPU6050_D1
#define MPU6050_GYRO_RESET  MPU6050_D2

#define MPU6050_MOT_COUNT0      MPU6050_D0
#define MPU6050_MOT_COUNT1      MPU6050_D1
#define MPU6050_FF_COUNT0       MPU6050_D2
#define MPU6050_FF_COUNT1       MPU6050_D3
```

```
#define MPU6050_ACCEL_ON_DELAY0 MPU6050_D4
#define MPU6050_ACCEL_ON_DELAY1 MPU6050_D5

#define MPU6050_MOT_COUNT_0 (0)
#define MPU6050_MOT_COUNT_1 (bit(MPU6050_MOT_COUNT0))
#define MPU6050_MOT_COUNT_2 (bit(MPU6050_MOT_COUNT1))
#define MPU6050_MOT_COUNT_3 (bit(MPU6050_MOT_COUNT1)|bit(MPU6050_MOT_COUNT0))

#define MPU6050_MOT_COUNT_RESET MPU6050_MOT_COUNT_0

#define MPU6050_FF_COUNT_0 (0)
#define MPU6050_FF_COUNT_1 (bit(MPU6050_FF_COUNT0))
#define MPU6050_FF_COUNT_2 (bit(MPU6050_FF_COUNT1))
#define MPU6050_FF_COUNT_3 (bit(MPU6050_FF_COUNT1)|bit(MPU6050_FF_COUNT0))

#define MPU6050_FF_COUNT_RESET MPU6050_FF_COUNT_0

#define MPU6050_ACCEL_ON_DELAY_0 (0)
#define MPU6050_ACCEL_ON_DELAY_1 (bit(MPU6050_ACCEL_ON_DELAY0))
#define MPU6050_ACCEL_ON_DELAY_2 (bit(MPU6050_ACCEL_ON_DELAY1))
#define MPU6050_ACCEL_ON_DELAY_3 (bit(MPU6050_ACCEL_ON_DELAY1)|bit(MPU6050_ACCEL_ON_DELAY0))

#define MPU6050_ACCEL_ON_DELAY_0MS MPU6050_ACCEL_ON_DELAY_0
#define MPU6050_ACCEL_ON_DELAY_1MS MPU6050_ACCEL_ON_DELAY_1
#define MPU6050_ACCEL_ON_DELAY_2MS MPU6050_ACCEL_ON_DELAY_2
```

```
#define MPU6050_ACCEL_ON_DELAY_3MS MPU6050_ACCEL_ON_DELAY_3

#define MPU6050_SIG_COND_RESET MPU6050_D0
#define MPU6050_I2C_MST_RESET  MPU6050_D1
#define MPU6050_FIFO_RESET     MPU6050_D2
#define MPU6050_I2C_IF_DIS     MPU6050_D4
#define MPU6050_I2C_MST_EN     MPU6050_D5
#define MPU6050_FIFO_EN        MPU6050_D6

#define MPU6050_CLKSEL0      MPU6050_D0
#define MPU6050_CLKSEL1      MPU6050_D1
#define MPU6050_CLKSEL2      MPU6050_D2
#define MPU6050_TEMP_DIS     MPU6050_D3
#define MPU6050_CYCLE        MPU6050_D5
#define MPU6050_SLEEP        MPU6050_D6
#define MPU6050_DEVICE_RESET MPU6050_D7

#define MPU6050_CLKSEL_0 (0)
#define MPU6050_CLKSEL_1 (bit(MPU6050_CLKSEL0))
#define MPU6050_CLKSEL_2 (bit(MPU6050_CLKSEL1))
#define MPU6050_CLKSEL_3 (bit(MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))
#define MPU6050_CLKSEL_4 (bit(MPU6050_CLKSEL2))
#define MPU6050_CLKSEL_5 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL0))
#define MPU6050_CLKSEL_6 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL1))
#define MPU6050_CLKSEL_7 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))

#define MPU6050_CLKSEL_INTERNAL    MPU6050_CLKSEL_0
#define MPU6050_CLKSEL_X           MPU6050_CLKSEL_1
#define MPU6050_CLKSEL_Y           MPU6050_CLKSEL_2
```

```
#define MPU6050_CLKSEL_Z          MPU6050_CLKSEL_3
#define MPU6050_CLKSEL_EXT_32KHZ   MPU6050_CLKSEL_4
#define MPU6050_CLKSEL_EXT_19_2MHZ MPU6050_CLKSEL_5
#define MPU6050_CLKSEL_RESERVED    MPU6050_CLKSEL_6
#define MPU6050_CLKSEL_STOP        MPU6050_CLKSEL_7

#define MPU6050_STBY_ZG       MPU6050_D0
#define MPU6050_STBY_YG       MPU6050_D1
#define MPU6050_STBY_XG       MPU6050_D2
#define MPU6050_STBY_ZA       MPU6050_D3
#define MPU6050_STBY_YA       MPU6050_D4
#define MPU6050_STBY_XA       MPU6050_D5
#define MPU6050_LP_WAKE_CTRL0 MPU6050_D6
#define MPU6050_LP_WAKE_CTRL1 MPU6050_D7

#define MPU6050_LP_WAKE_CTRL_0 (0)
#define MPU6050_LP_WAKE_CTRL_1 (bit(MPU6050_LP_WAKE_CTRL0))
#define MPU6050_LP_WAKE_CTRL_2 (bit(MPU6050_LP_WAKE_CTRL1))
#define                                          MPU6050_LP_WAKE_CTRL_3
(bit(MPU6050_LP_WAKE_CTRL1)|bit(MPU6050_LP_WAKE_CTRL0))

#define MPU6050_LP_WAKE_1_25HZ MPU6050_LP_WAKE_CTRL_0
#define MPU6050_LP_WAKE_2_5HZ  MPU6050_LP_WAKE_CTRL_1
#define MPU6050_LP_WAKE_5HZ    MPU6050_LP_WAKE_CTRL_2
#define MPU6050_LP_WAKE_10HZ   MPU6050_LP_WAKE_CTRL_3


#define MPU6050_I2C_ADDRESS 0x68


typedef union accel_t_gyro_union
{
  struct
  {
```

```
    uint8_t x_accel_h;
    uint8_t x_accel_l;
    uint8_t y_accel_h;
    uint8_t y_accel_l;
    uint8_t z_accel_h;
    uint8_t z_accel_l;
    uint8_t t_h;
    uint8_t t_l;
    uint8_t x_gyro_h;
    uint8_t x_gyro_l;
    uint8_t y_gyro_h;
    uint8_t y_gyro_l;
    uint8_t z_gyro_h;
    uint8_t z_gyro_l;
  }
  reg;
  struct
  {
    int16_t x_accel;
    int16_t y_accel;
    int16_t z_accel;
    int16_t temperature;
    int16_t x_gyro;
    int16_t y_gyro;
    int16_t z_gyro;
  }
  value;
};

//Declaração das variaveis que guardarão os valores anteriores para que possa ser comparados
int16_t x_accel_init;
int16_t y_accel_init;
int16_t z_accel_init;
```

```
int16_t x_gyro_init;
int16_t y_gyro_init;
int16_t z_gyro_init;

int cont, cont0;
void setup()
{
 int error;
 uint8_t c;
 accel_t_gyro_union accel_t_gyro_ini;

 Serial.begin(9600);
 Serial.println(F("InvenSense MPU-6050"));
 Serial.println(F("June 2012"));

 Wire.begin();

 error = MPU6050_read (MPU6050_WHO_AM_I, &c, 1);
 Serial.print(F("WHO_AM_I : "));
 Serial.print(c,HEX);
 Serial.print(F(", error = "));
 Serial.println(error,DEC);

 error = MPU6050_read (MPU6050_PWR_MGMT_1, &c, 1);
 Serial.print(F("PWR_MGMT_1 : "));
 Serial.print(c,HEX);
 Serial.print(F(", error = "));
 Serial.println(error,DEC);

 error = MPU6050_read (MPU6050_ACCEL_XOUT_H, (uint8_t *)
&accel_t_gyro_ini, sizeof(accel_t_gyro_ini));
 x_accel_init = accel_t_gyro_ini.value.x_accel;
 y_accel_init = accel_t_gyro_ini.value.y_accel;
```

```
  z_accel_init = accel_t_gyro_ini.value.z_accel;
  x_gyro_init = accel_t_gyro_ini.value.x_gyro;
  y_gyro_init = accel_t_gyro_ini.value.y_gyro;
  z_gyro_init = accel_t_gyro_ini.value.z_gyro;

  Serial.print(F("Valores iniciais, aceleracao: "));
  Serial.print(x_accel_init,DEC);
  Serial.print(F(", "));
  Serial.print(y_accel_init,DEC);
  Serial.print(F(", "));
  Serial.println(z_accel_init,DEC);
  Serial.print(F("Giro"));
  Serial.print(x_gyro_init,DEC);
  Serial.print(F(", "));
  Serial.print(y_gyro_init,DEC);
  Serial.print(F(", "));
  Serial.println(z_gyro_init,DEC);


  MPU6050_write_reg (MPU6050_PWR_MGMT_1, 0);
  cont = 0;
  cont0 = 0;
 }



 void loop()
 {
  int error;
  double dT;
  accel_t_gyro_union accel_t_gyro;

  error = MPU6050_read (MPU6050_ACCEL_XOUT_H, (uint8_t *)
&accel_t_gyro, sizeof(accel_t_gyro));
```

```
  uint8_t swap;
#define SWAP(x,y) swap = x; x = y; y = swap

   SWAP (accel_t_gyro.reg.x_accel_h, accel_t_gyro.reg.x_accel_l);
  SWAP (accel_t_gyro.reg.y_accel_h, accel_t_gyro.reg.y_accel_l);
  SWAP (accel_t_gyro.reg.z_accel_h, accel_t_gyro.reg.z_accel_l);
  SWAP (accel_t_gyro.reg.t_h, accel_t_gyro.reg.t_l);
  SWAP (accel_t_gyro.reg.x_gyro_h, accel_t_gyro.reg.x_gyro_l);
  SWAP (accel_t_gyro.reg.y_gyro_h, accel_t_gyro.reg.y_gyro_l);
  SWAP (accel_t_gyro.reg.z_gyro_h, accel_t_gyro.reg.z_gyro_l);

  // Laços para a verificação da sensibilidade

  if(abs (accel_t_gyro.value.x_gyro - x_gyro_init) > 500 &&
(abs(accel_t_gyro.value.x_gyro ) > 1000))
  {
   Serial.println(F("girou em X"));
  }
  if(abs (accel_t_gyro.value.y_gyro - y_gyro_init) > 500 &&
(abs(accel_t_gyro.value.y_gyro ) > 1000))
  {
   Serial.println(F("girou em Y"));
  }
  if(abs (accel_t_gyro.value.z_gyro - z_gyro_init) > 500 &&
(abs(accel_t_gyro.value.z_gyro ) > 1000))
  {
   Serial.println(F("girou em Z"));
   cont = cont0;
  }
  x_accel_init = accel_t_gyro.value.x_accel;
  x_gyro_init = accel_t_gyro.value.x_gyro;
  y_gyro_init = accel_t_gyro.value.y_gyro;
  z_gyro_init = accel_t_gyro.value.z_gyro;
```

```
  Serial.println(F("cont "));
  Serial.println(cont,DEC);

  if(cont0 == cont+3)
  {
    Serial.println(F("Desliga"));
  }

  // Impressão da Leitura da Temperatura
  Serial.print(F("temperature: "));
  dT = ( (double) accel_t_gyro.value.temperature + 12412.0) / 340.0;
  Serial.print(dT, 3);
  Serial.print(F(" degrees Celsius"));
  Serial.println(F(""));

  //Impressão dos valores lidos pelo giroscopio

  Serial.print(F("gyro x,y,z : "));
  Serial.print(accel_t_gyro.value.x_gyro, DEC);
  Serial.print(F(", "));
  Serial.print(accel_t_gyro.value.y_gyro, DEC);
  Serial.print(F(", "));
  Serial.print(accel_t_gyro.value.z_gyro, DEC);
  Serial.print(F(", "));
  Serial.println(F(""));

  delay(1000);
}

int MPU6050_read(int start, uint8_t *buffer, int size)
{
  int i, n, error;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
```

```
  n = Wire.write(start);
  if (n != 1)
    return (-10);

  n = Wire.endTransmission(false);
  if (n != 0)
    return (n);

  Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);
  i = 0;
  while(Wire.available() && i<size)
  {
    buffer[i++]=Wire.read();
  }
  if ( i != size)
    return (-11);

  return (0);
}

int MPU6050_write(int start, const uint8_t *pData, int size)
{
  int n, error;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
  n = Wire.write(start);
  if (n != 1)
    return (-20);

  n = Wire.write(pData, size);
  if (n != size)
    return (-21);

  error = Wire.endTransmission(true);
```

```
  if (error != 0)
    return (error);

  return (0);
}


int MPU6050_write_reg(int reg, uint8_t data)
{
  int error;

  error = MPU6050_write(reg, &data, 1);

  return (error);
}
```

Apêndice B: Código para teste dos LEDS e Botões

```
#define DIREITA 1
#define ESQUERDA 2

#include <Wire.h>

int ledDireita = 7;
int ledEsquerda = 8;
int setaDireita = 10;
int setaEsquerda = 11;
int seta = 0;
int temporizador = 0;

void setup() {
  Serial.begin(9600);
  pinMode(ledDireita, OUTPUT);
  pinMode(ledEsquerda, OUTPUT);
  pinMode(setaDireita, INPUT);
  pinMode(setaEsquerda, INPUT);
}
void loop() {

  if (seta != 0) {
    //Serial.println(F("entrou aki"));
    delay(1000);
    temporizador++;
  } else {
    temporizador = 0;
  }

  if (digitalRead(setaDireita)) {
```

```
if (seta == 0 || seta == ESQUERDA) {
     seta = DIREITA;
     digitalWrite(ledDireita, HIGH);
     digitalWrite(ledEsquerda, LOW);
   } else {
     digitalWrite(ledDireita, LOW);
   }
   temporizador = 0;
  }

  if (digitalRead(setaEsquerda)) {
   if (seta == 0 || seta == DIREITA) {
     seta = ESQUERDA;
     digitalWrite(ledEsquerda, HIGH);
     digitalWrite(ledDireita, LOW);
   } else {
     digitalWrite(ledEsquerda, LOW);
   }
   temporizador = 0;
  }
  if (temporizador == 3) {
   temporizador = 0;
   digitalWrite(ledEsquerda, LOW);
   digitalWrite(ledDireita, LOW);
   seta = 0;
  }
}
```

Apêndice C: Código final do projeto

```
// Definindo os nomes de a Acordo com o datasheet.
// De a Acordo com o Documento da Invensense
// "MPU-6000 e MPU-6050 Register Mapa e descrições Revisão 3.2",
// não há registros entre 0x02 e 0x18,
// mas de acordo com outras informações os registros
// na área desconhecida são para ganhos e compensações.

#define MPU6050_AUX_VDDIO          0x01   // R/W
#define MPU6050_SMPLRT_DIV         0x19   // R/W
#define MPU6050_CONFIG             0x1A   // R/W
#define MPU6050_GYRO_CONFIG        0x1B   // R/W
#define MPU6050_ACCEL_CONFIG       0x1C   // R/W
#define MPU6050_FF_THR             0x1D   // R/W
#define MPU6050_FF_DUR             0x1E   // R/W
#define MPU6050_MOT_THR            0x1F   // R/W
#define MPU6050_MOT_DUR            0x20   // R/W
#define MPU6050_ZRMOT_THR          0x21   // R/W
#define MPU6050_ZRMOT_DUR          0x22   // R/W
#define MPU6050_FIFO_EN            0x23   // R/W
#define MPU6050_I2C_MST_CTRL       0x24   // R/W
#define MPU6050_I2C_SLV0_ADDR      0x25   // R/W
#define MPU6050_I2C_SLV0_REG       0x26   // R/W
#define MPU6050_I2C_SLV0_CTRL      0x27   // R/W
#define MPU6050_I2C_SLV1_ADDR      0x28   // R/W
#define MPU6050_I2C_SLV1_REG       0x29   // R/W
#define MPU6050_I2C_SLV1_CTRL      0x2A   // R/W
#define MPU6050_I2C_SLV2_ADDR      0x2B   // R/W
#define MPU6050_I2C_SLV2_REG       0x2C   // R/W
#define MPU6050_I2C_SLV2_CTRL      0x2D   // R/W
#define MPU6050_I2C_SLV3_ADDR      0x2E   // R/W
```

```
#define MPU6050_I2C_SLV3_REG       0x2F   // R/W
#define MPU6050_I2C_SLV3_CTRL      0x30   // R/W
#define MPU6050_I2C_SLV4_ADDR      0x31   // R/W
#define MPU6050_I2C_SLV4_REG       0x32   // R/W
#define MPU6050_I2C_SLV4_DO        0x33   // R/W
#define MPU6050_I2C_SLV4_CTRL      0x34   // R/W
#define MPU6050_I2C_SLV4_DI        0x35   // R
#define MPU6050_I2C_MST_STATUS     0x36   // R
#define MPU6050_INT_PIN_CFG        0x37   // R/W
#define MPU6050_INT_ENABLE         0x38   // R/W
#define MPU6050_INT_STATUS         0x3A   // R
#define MPU6050_ACCEL_XOUT_H       0x3B   // R
#define MPU6050_ACCEL_XOUT_L       0x3C   // R
#define MPU6050_ACCEL_YOUT_H       0x3D   // R
#define MPU6050_ACCEL_YOUT_L       0x3E   // R
#define MPU6050_ACCEL_ZOUT_H       0x3F   // R
#define MPU6050_ACCEL_ZOUT_L       0x40   // R
#define MPU6050_TEMP_OUT_H         0x41   // R
#define MPU6050_TEMP_OUT_L         0x42   // R
#define MPU6050_GYRO_XOUT_H        0x43   // R
#define MPU6050_GYRO_XOUT_L        0x44   // R
#define MPU6050_GYRO_YOUT_H        0x45   // R
#define MPU6050_GYRO_YOUT_L        0x46   // R
#define MPU6050_GYRO_ZOUT_H        0x47   // R
#define MPU6050_GYRO_ZOUT_L        0x48   // R
#define MPU6050_EXT_SENS_DATA_00   0x49   // R
#define MPU6050_EXT_SENS_DATA_01   0x4A   // R
#define MPU6050_EXT_SENS_DATA_02   0x4B   // R
#define MPU6050_EXT_SENS_DATA_03   0x4C   // R
#define MPU6050_EXT_SENS_DATA_04   0x4D   // R
#define MPU6050_EXT_SENS_DATA_05   0x4E   // R
#define MPU6050_EXT_SENS_DATA_06   0x4F   // R
#define MPU6050_EXT_SENS_DATA_07   0x50   // R
#define MPU6050_EXT_SENS_DATA_08   0x51   // R
```

```
#define MPU6050_EXT_SENS_DATA_09   0x52   // R
#define MPU6050_EXT_SENS_DATA_10   0x53   // R
#define MPU6050_EXT_SENS_DATA_11   0x54   // R
#define MPU6050_EXT_SENS_DATA_12   0x55   // R
#define MPU6050_EXT_SENS_DATA_13   0x56   // R
#define MPU6050_EXT_SENS_DATA_14   0x57   // R
#define MPU6050_EXT_SENS_DATA_15   0x58   // R
#define MPU6050_EXT_SENS_DATA_16   0x59   // R
#define MPU6050_EXT_SENS_DATA_17   0x5A   // R
#define MPU6050_EXT_SENS_DATA_18   0x5B   // R
#define MPU6050_EXT_SENS_DATA_19   0x5C   // R
#define MPU6050_EXT_SENS_DATA_20   0x5D   // R
#define MPU6050_EXT_SENS_DATA_21   0x5E   // R
#define MPU6050_EXT_SENS_DATA_22   0x5F   // R
#define MPU6050_EXT_SENS_DATA_23   0x60   // R
#define MPU6050_MOT_DETECT_STATUS  0x61   // R
#define MPU6050_I2C_SLV0_DO        0x63   // R/W
#define MPU6050_I2C_SLV1_DO        0x64   // R/W
#define MPU6050_I2C_SLV2_DO        0x65   // R/W
#define MPU6050_I2C_SLV3_DO        0x66   // R/W
#define MPU6050_I2C_MST_DELAY_CTRL 0x67   // R/W
#define MPU6050_SIGNAL_PATH_RESET  0x68   // R/W
#define MPU6050_MOT_DETECT_CTRL    0x69   // R/W
#define MPU6050_USER_CTRL          0x6A   // R/W
#define MPU6050_PWR_MGMT_1         0x6B   // R/W
#define MPU6050_PWR_MGMT_2         0x6C   // R/W
#define MPU6050_FIFO_COUNTH        0x72   // R/W
#define MPU6050_FIFO_COUNTL        0x73   // R/W
#define MPU6050_FIFO_R_W           0x74   // R/W
#define MPU6050_WHO_AM_I           0x75   // R
```

// Define para os bits, para serem capazes de mudar entre o número de bits e definição binária.

```
// Usando o número de bits, a programação do sensor é como programar o microcontrolador AVR.
// Mas em vez de usar "(1 << X)", ou "_BV (X)", o Arduino usa "bit (X)".

#define MPU6050_D0 0
#define MPU6050_D1 1
#define MPU6050_D2 2
#define MPU6050_D3 3
#define MPU6050_D4 4
#define MPU6050_D5 5
#define MPU6050_D6 6
#define MPU6050_D7 7

// Regsitrador AUX_VDDIO
#define MPU6050_AUX_VDDIO MPU6050_D7   // I2C high: 1=VDD, 0=VLOGIC

#define MPU6050_DLPF_CFG0    MPU6050_D0
#define MPU6050_DLPF_CFG1    MPU6050_D1
#define MPU6050_DLPF_CFG2    MPU6050_D2
#define MPU6050_EXT_SYNC_SET0 MPU6050_D3
#define MPU6050_EXT_SYNC_SET1 MPU6050_D4
#define MPU6050_EXT_SYNC_SET2 MPU6050_D5

#define MPU6050_EXT_SYNC_SET_0 (0)
#define MPU6050_EXT_SYNC_SET_1 (bit(MPU6050_EXT_SYNC_SET0))
#define MPU6050_EXT_SYNC_SET_2 (bit(MPU6050_EXT_SYNC_SET1))
#define MPU6050_EXT_SYNC_SET_3 (bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT_SYNC_SET0))
#define MPU6050_EXT_SYNC_SET_4 (bit(MPU6050_EXT_SYNC_SET2))
#define MPU6050_EXT_SYNC_SET_5 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET0))
#define MPU6050_EXT_SYNC_SET_6 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1))
```

```
#define MPU6050_EXT_SYNC_SET_7 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT_SYNC_SET0))

#define MPU6050_EXT_SYNC_DISABLED MPU6050_EXT_SYNC_SET_0
#define MPU6050_EXT_SYNC_TEMP_OUT_L MPU6050_EXT_SYNC_SET_1
#define MPU6050_EXT_SYNC_GYRO_XOUT_L MPU6050_EXT_SYNC_SET_2
#define MPU6050_EXT_SYNC_GYRO_YOUT_L MPU6050_EXT_SYNC_SET_3
#define MPU6050_EXT_SYNC_GYRO_ZOUT_L MPU6050_EXT_SYNC_SET_4
#define MPU6050_EXT_SYNC_ACCEL_XOUT_L MPU6050_EXT_SYNC_SET_5
#define MPU6050_EXT_SYNC_ACCEL_YOUT_L MPU6050_EXT_SYNC_SET_6
#define MPU6050_EXT_SYNC_ACCEL_ZOUT_L MPU6050_EXT_SYNC_SET_7

#define MPU6050_DLPF_CFG_0 (0)
#define MPU6050_DLPF_CFG_1 (bit(MPU6050_DLPF_CFG0))
#define MPU6050_DLPF_CFG_2 (bit(MPU6050_DLPF_CFG1))
#define MPU6050_DLPF_CFG_3 (bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG0))
#define MPU6050_DLPF_CFG_4 (bit(MPU6050_DLPF_CFG2))
#define MPU6050_DLPF_CFG_5 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG0))
#define MPU6050_DLPF_CFG_6 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1))
#define MPU6050_DLPF_CFG_7 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG0))
```

```
// Largura de banda para o acelerômetro é quase a mesma para o giroscópio
#define MPU6050_DLPF_260HZ    MPU6050_DLPF_CFG_0
#define MPU6050_DLPF_184HZ    MPU6050_DLPF_CFG_1
#define MPU6050_DLPF_94HZ     MPU6050_DLPF_CFG_2
#define MPU6050_DLPF_44HZ     MPU6050_DLPF_CFG_3
#define MPU6050_DLPF_21HZ     MPU6050_DLPF_CFG_4
#define MPU6050_DLPF_10HZ     MPU6050_DLPF_CFG_5
#define MPU6050_DLPF_5HZ      MPU6050_DLPF_CFG_6
#define MPU6050_DLPF_RESERVED MPU6050_DLPF_CFG_7

#define MPU6050_FS_SEL0 MPU6050_D3
#define MPU6050_FS_SEL1 MPU6050_D4
#define MPU6050_ZG_ST   MPU6050_D5
#define MPU6050_YG_ST   MPU6050_D6
#define MPU6050_XG_ST   MPU6050_D7

#define MPU6050_FS_SEL_0 (0)
#define MPU6050_FS_SEL_1 (bit(MPU6050_FS_SEL0))
#define MPU6050_FS_SEL_2 (bit(MPU6050_FS_SEL1))
#define                                                    MPU6050_FS_SEL_3
(bit(MPU6050_FS_SEL1)|bit(MPU6050_FS_SEL0))

// Os nomes usam o intervalo em graus por segundo.
#define MPU6050_FS_SEL_250  MPU6050_FS_SEL_0
#define MPU6050_FS_SEL_500  MPU6050_FS_SEL_1
#define MPU6050_FS_SEL_1000 MPU6050_FS_SEL_2
#define MPU6050_FS_SEL_2000 MPU6050_FS_SEL_3

#define MPU6050_ACCEL_HPF0 MPU6050_D0
#define MPU6050_ACCEL_HPF1 MPU6050_D1
#define MPU6050_ACCEL_HPF2 MPU6050_D2
#define MPU6050_AFS_SEL0   MPU6050_D3
#define MPU6050_AFS_SEL1   MPU6050_D4
```

```
#define MPU6050_ZA_ST      MPU6050_D5
#define MPU6050_YA_ST      MPU6050_D6
#define MPU6050_XA_ST      MPU6050_D7

#define MPU6050_ACCEL_HPF_0 (0)
#define MPU6050_ACCEL_HPF_1 (bit(MPU6050_ACCEL_HPF0))
#define MPU6050_ACCEL_HPF_2 (bit(MPU6050_ACCEL_HPF1))
#define MPU6050_ACCEL_HPF_3 (bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL_HPF0))
#define MPU6050_ACCEL_HPF_4 (bit(MPU6050_ACCEL_HPF2))
#define MPU6050_ACCEL_HPF_7 (bit(MPU6050_ACCEL_HPF2)|bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL_HPF0))

#define MPU6050_ACCEL_HPF_RESET  MPU6050_ACCEL_HPF_0
#define MPU6050_ACCEL_HPF_5HZ    MPU6050_ACCEL_HPF_1
#define MPU6050_ACCEL_HPF_2_5HZ  MPU6050_ACCEL_HPF_2
#define MPU6050_ACCEL_HPF_1_25HZ MPU6050_ACCEL_HPF_3
#define MPU6050_ACCEL_HPF_0_63HZ MPU6050_ACCEL_HPF_4
#define MPU6050_ACCEL_HPF_HOLD   MPU6050_ACCEL_HPF_7

#define MPU6050_AFS_SEL_0 (0)
#define MPU6050_AFS_SEL_1 (bit(MPU6050_AFS_SEL0))
#define MPU6050_AFS_SEL_2 (bit(MPU6050_AFS_SEL1))
#define MPU6050_AFS_SEL_3 (bit(MPU6050_AFS_SEL1)|bit(MPU6050_AFS_SEL0))

// O nome usa o intervalo de escala completa para o acelerômetro.
#define MPU6050_AFS_SEL_2G  MPU6050_AFS_SEL_0
#define MPU6050_AFS_SEL_4G  MPU6050_AFS_SEL_1
#define MPU6050_AFS_SEL_8G  MPU6050_AFS_SEL_2
#define MPU6050_AFS_SEL_16G MPU6050_AFS_SEL_3

#define MPU6050_SLV0_FIFO_EN  MPU6050_D0
```

```
#define MPU6050_SLV1_FIFO_EN  MPU6050_D1
#define MPU6050_SLV2_FIFO_EN  MPU6050_D2
#define MPU6050_ACCEL_FIFO_EN MPU6050_D3
#define MPU6050_ZG_FIFO_EN    MPU6050_D4
#define MPU6050_YG_FIFO_EN    MPU6050_D5
#define MPU6050_XG_FIFO_EN    MPU6050_D6
#define MPU6050_TEMP_FIFO_EN  MPU6050_D7

#define MPU6050_I2C_MST_CLK0  MPU6050_D0
#define MPU6050_I2C_MST_CLK1  MPU6050_D1
#define MPU6050_I2C_MST_CLK2  MPU6050_D2
#define MPU6050_I2C_MST_CLK3  MPU6050_D3
#define MPU6050_I2C_MST_P_NSR MPU6050_D4
#define MPU6050_SLV_3_FIFO_EN MPU6050_D5
#define MPU6050_WAIT_FOR_ES   MPU6050_D6
#define MPU6050_MULT_MST_EN   MPU6050_D7

#define MPU6050_I2C_MST_CLK_0 (0)
#define MPU6050_I2C_MST_CLK_1  (bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_2  (bit(MPU6050_I2C_MST_CLK1))
#define                                          MPU6050_I2C_MST_CLK_3
(bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_4  (bit(MPU6050_I2C_MST_CLK2))
#define                                          MPU6050_I2C_MST_CLK_5
(bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK0))
#define                                          MPU6050_I2C_MST_CLK_6
(bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1))
#define                                          MPU6050_I2C_MST_CLK_7
(bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C
_MST_CLK0))
#define MPU6050_I2C_MST_CLK_8  (bit(MPU6050_I2C_MST_CLK3))
#define                                          MPU6050_I2C_MST_CLK_9
(bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK0))
```

```
        #define                                     MPU6050_I2C_MST_CLK_10
(bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1))
        #define                                     MPU6050_I2C_MST_CLK_11
(bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C
_MST_CLK0))
        #define                                     MPU6050_I2C_MST_CLK_12
(bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2))
        #define                                     MPU6050_I2C_MST_CLK_13
(bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C
_MST_CLK0))
        #define                                     MPU6050_I2C_MST_CLK_14
(bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C
_MST_CLK1))
        #define                                     MPU6050_I2C_MST_CLK_15
(bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C
_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))

        // Os nomes usam I2C Master Clock Speed em kHz.
        #define MPU6050_I2C_MST_CLK_348KHZ MPU6050_I2C_MST_CLK_0
        #define MPU6050_I2C_MST_CLK_333KHZ MPU6050_I2C_MST_CLK_1
        #define MPU6050_I2C_MST_CLK_320KHZ MPU6050_I2C_MST_CLK_2
        #define MPU6050_I2C_MST_CLK_308KHZ MPU6050_I2C_MST_CLK_3
        #define MPU6050_I2C_MST_CLK_296KHZ MPU6050_I2C_MST_CLK_4
        #define MPU6050_I2C_MST_CLK_286KHZ MPU6050_I2C_MST_CLK_5
        #define MPU6050_I2C_MST_CLK_276KHZ MPU6050_I2C_MST_CLK_6
        #define MPU6050_I2C_MST_CLK_267KHZ MPU6050_I2C_MST_CLK_7
        #define MPU6050_I2C_MST_CLK_258KHZ MPU6050_I2C_MST_CLK_8
        #define MPU6050_I2C_MST_CLK_500KHZ MPU6050_I2C_MST_CLK_9
        #define MPU6050_I2C_MST_CLK_471KHZ MPU6050_I2C_MST_CLK_10
        #define MPU6050_I2C_MST_CLK_444KHZ MPU6050_I2C_MST_CLK_11
        #define MPU6050_I2C_MST_CLK_421KHZ MPU6050_I2C_MST_CLK_12
        #define MPU6050_I2C_MST_CLK_400KHZ MPU6050_I2C_MST_CLK_13
        #define MPU6050_I2C_MST_CLK_381KHZ MPU6050_I2C_MST_CLK_14
        #define MPU6050_I2C_MST_CLK_364KHZ MPU6050_I2C_MST_CLK_15
```

```
#define MPU6050_I2C_SLV0_RW MPU6050_D7

#define MPU6050_I2C_SLV0_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV0_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV0_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV0_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV0_GRP     MPU6050_D4
#define MPU6050_I2C_SLV0_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV0_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV0_EN      MPU6050_D7

#define MPU6050_I2C_SLV0_LEN_MASK 0x0F

#define MPU6050_I2C_SLV1_RW MPU6050_D7

#define MPU6050_I2C_SLV1_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV1_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV1_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV1_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV1_GRP     MPU6050_D4
#define MPU6050_I2C_SLV1_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV1_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV1_EN      MPU6050_D7

#define MPU6050_I2C_SLV1_LEN_MASK 0x0F

#define MPU6050_I2C_SLV2_RW MPU6050_D7

#define MPU6050_I2C_SLV2_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV2_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV2_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV2_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV2_GRP     MPU6050_D4
```

```
#define MPU6050_I2C_SLV2_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV2_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV2_EN      MPU6050_D7

#define MPU6050_I2C_SLV2_LEN_MASK 0x0F

#define MPU6050_I2C_SLV3_RW MPU6050_D7

#define MPU6050_I2C_SLV3_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV3_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV3_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV3_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV3_GRP     MPU6050_D4
#define MPU6050_I2C_SLV3_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV3_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV3_EN      MPU6050_D7

#define MPU6050_I2C_SLV3_LEN_MASK 0x0F

#define MPU6050_I2C_SLV4_RW MPU6050_D7

#define MPU6050_I2C_MST_DLY0     MPU6050_D0
#define MPU6050_I2C_MST_DLY1     MPU6050_D1
#define MPU6050_I2C_MST_DLY2     MPU6050_D2
#define MPU6050_I2C_MST_DLY3     MPU6050_D3
#define MPU6050_I2C_MST_DLY4     MPU6050_D4
#define MPU6050_I2C_SLV4_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV4_INT_EN  MPU6050_D6
#define MPU6050_I2C_SLV4_EN      MPU6050_D7

#define MPU6050_I2C_MST_DLY_MASK 0x1F

#define MPU6050_I2C_SLV0_NACK MPU6050_D0
#define MPU6050_I2C_SLV1_NACK MPU6050_D1
```

```
#define MPU6050_I2C_SLV2_NACK MPU6050_D2
#define MPU6050_I2C_SLV3_NACK MPU6050_D3
#define MPU6050_I2C_SLV4_NACK MPU6050_D4
#define MPU6050_I2C_LOST_ARB  MPU6050_D5
#define MPU6050_I2C_SLV4_DONE MPU6050_D6
#define MPU6050_PASS_THROUGH  MPU6050_D7

#define MPU6050_CLKOUT_EN       MPU6050_D0
#define MPU6050_I2C_BYPASS_EN   MPU6050_D1
#define MPU6050_FSYNC_INT_EN    MPU6050_D2
#define MPU6050_FSYNC_INT_LEVEL MPU6050_D3
#define MPU6050_INT_RD_CLEAR    MPU6050_D4
#define MPU6050_LATCH_INT_EN    MPU6050_D5
#define MPU6050_INT_OPEN        MPU6050_D6
#define MPU6050_INT_LEVEL       MPU6050_D7

#define MPU6050_DATA_RDY_EN    MPU6050_D0
#define MPU6050_I2C_MST_INT_EN MPU6050_D3
#define MPU6050_FIFO_OFLOW_EN  MPU6050_D4
#define MPU6050_ZMOT_EN        MPU6050_D5
#define MPU6050_MOT_EN         MPU6050_D6
#define MPU6050_FF_EN          MPU6050_D7

#define MPU6050_DATA_RDY_INT   MPU6050_D0
#define MPU6050_I2C_MST_INT    MPU6050_D3
#define MPU6050_FIFO_OFLOW_INT MPU6050_D4
#define MPU6050_ZMOT_INT       MPU6050_D5
#define MPU6050_MOT_INT        MPU6050_D6
#define MPU6050_FF_INT         MPU6050_D7

#define MPU6050_MOT_ZRMOT MPU6050_D0
#define MPU6050_MOT_ZPOS  MPU6050_D2
#define MPU6050_MOT_ZNEG  MPU6050_D3
#define MPU6050_MOT_YPOS  MPU6050_D4
```

```
#define MPU6050_MOT_YNEG  MPU6050_D5
#define MPU6050_MOT_XPOS  MPU6050_D6
#define MPU6050_MOT_XNEG  MPU6050_D7

#define MPU6050_I2C_SLV0_DLY_EN MPU6050_D0
#define MPU6050_I2C_SLV1_DLY_EN MPU6050_D1
#define MPU6050_I2C_SLV2_DLY_EN MPU6050_D2
#define MPU6050_I2C_SLV3_DLY_EN MPU6050_D3
#define MPU6050_I2C_SLV4_DLY_EN MPU6050_D4
#define MPU6050_DELAY_ES_SHADOW MPU6050_D7

#define MPU6050_TEMP_RESET  MPU6050_D0
#define MPU6050_ACCEL_RESET MPU6050_D1
#define MPU6050_GYRO_RESET  MPU6050_D2

#define MPU6050_MOT_COUNT0      MPU6050_D0
#define MPU6050_MOT_COUNT1      MPU6050_D1
#define MPU6050_FF_COUNT0       MPU6050_D2
#define MPU6050_FF_COUNT1       MPU6050_D3
#define MPU6050_ACCEL_ON_DELAY0 MPU6050_D4
#define MPU6050_ACCEL_ON_DELAY1 MPU6050_D5

#define MPU6050_MOT_COUNT_0 (0)
#define MPU6050_MOT_COUNT_1 (bit(MPU6050_MOT_COUNT0))
#define MPU6050_MOT_COUNT_2 (bit(MPU6050_MOT_COUNT1))
#define                                          MPU6050_MOT_COUNT_3
(bit(MPU6050_MOT_COUNT1)|bit(MPU6050_MOT_COUNT0))

#define MPU6050_MOT_COUNT_RESET MPU6050_MOT_COUNT_0

#define MPU6050_FF_COUNT_0 (0)
#define MPU6050_FF_COUNT_1 (bit(MPU6050_FF_COUNT0))
#define MPU6050_FF_COUNT_2 (bit(MPU6050_FF_COUNT1))
```

```
#define MPU6050_FF_COUNT_3 (bit(MPU6050_FF_COUNT1)|bit(MPU6050_FF_COUNT0))

#define MPU6050_FF_COUNT_RESET MPU6050_FF_COUNT_0

#define MPU6050_ACCEL_ON_DELAY_0 (0)
#define MPU6050_ACCEL_ON_DELAY_1 (bit(MPU6050_ACCEL_ON_DELAY0))
#define MPU6050_ACCEL_ON_DELAY_2 (bit(MPU6050_ACCEL_ON_DELAY1))
#define MPU6050_ACCEL_ON_DELAY_3 (bit(MPU6050_ACCEL_ON_DELAY1)|bit(MPU6050_ACCEL_ON_DELAY0))

#define MPU6050_ACCEL_ON_DELAY_0MS MPU6050_ACCEL_ON_DELAY_0
#define MPU6050_ACCEL_ON_DELAY_1MS MPU6050_ACCEL_ON_DELAY_1
#define MPU6050_ACCEL_ON_DELAY_2MS MPU6050_ACCEL_ON_DELAY_2
#define MPU6050_ACCEL_ON_DELAY_3MS MPU6050_ACCEL_ON_DELAY_3

#define MPU6050_SIG_COND_RESET MPU6050_D0
#define MPU6050_I2C_MST_RESET  MPU6050_D1
#define MPU6050_FIFO_RESET     MPU6050_D2
#define MPU6050_I2C_IF_DIS     MPU6050_D4
#define MPU6050_I2C_MST_EN     MPU6050_D5
#define MPU6050_FIFO_EN        MPU6050_D6

#define MPU6050_CLKSEL0      MPU6050_D0
#define MPU6050_CLKSEL1      MPU6050_D1
#define MPU6050_CLKSEL2      MPU6050_D2
#define MPU6050_TEMP_DIS     MPU6050_D3
#define MPU6050_CYCLE        MPU6050_D5
```

```
#define MPU6050_SLEEP        MPU6050_D6
#define MPU6050_DEVICE_RESET MPU6050_D7

#define MPU6050_CLKSEL_0 (0)
#define MPU6050_CLKSEL_1 (bit(MPU6050_CLKSEL0))
#define MPU6050_CLKSEL_2 (bit(MPU6050_CLKSEL1))
#define MPU6050_CLKSEL_3 (bit(MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))
#define MPU6050_CLKSEL_4 (bit(MPU6050_CLKSEL2))
#define MPU6050_CLKSEL_5 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL0))
#define MPU6050_CLKSEL_6 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL1))
#define MPU6050_CLKSEL_7 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))

#define MPU6050_CLKSEL_INTERNAL    MPU6050_CLKSEL_0
#define MPU6050_CLKSEL_X           MPU6050_CLKSEL_1
#define MPU6050_CLKSEL_Y           MPU6050_CLKSEL_2
#define MPU6050_CLKSEL_Z           MPU6050_CLKSEL_3
#define MPU6050_CLKSEL_EXT_32KHZ   MPU6050_CLKSEL_4
#define MPU6050_CLKSEL_EXT_19_2MHZ MPU6050_CLKSEL_5
#define MPU6050_CLKSEL_RESERVED    MPU6050_CLKSEL_6
#define MPU6050_CLKSEL_STOP        MPU6050_CLKSEL_7

#define MPU6050_STBY_ZG       MPU6050_D0
#define MPU6050_STBY_YG       MPU6050_D1
#define MPU6050_STBY_XG       MPU6050_D2
#define MPU6050_STBY_ZA       MPU6050_D3
#define MPU6050_STBY_YA       MPU6050_D4
#define MPU6050_STBY_XA       MPU6050_D5
#define MPU6050_LP_WAKE_CTRL0 MPU6050_D6
#define MPU6050_LP_WAKE_CTRL1 MPU6050_D7
```

```
#define MPU6050_LP_WAKE_CTRL_0 (0)
#define MPU6050_LP_WAKE_CTRL_1 (bit(MPU6050_LP_WAKE_CTRL0))
#define MPU6050_LP_WAKE_CTRL_2 (bit(MPU6050_LP_WAKE_CTRL1))
#define MPU6050_LP_WAKE_CTRL_3 (bit(MPU6050_LP_WAKE_CTRL1)|bit(MPU6050_LP_WAKE_CTRL0))

#define MPU6050_LP_WAKE_1_25HZ MPU6050_LP_WAKE_CTRL_0
#define MPU6050_LP_WAKE_2_5HZ  MPU6050_LP_WAKE_CTRL_1
#define MPU6050_LP_WAKE_5HZ    MPU6050_LP_WAKE_CTRL_2
#define MPU6050_LP_WAKE_10HZ   MPU6050_LP_WAKE_CTRL_3

#include <Wire.h>

#define MPU6050_I2C_ADDRESS 0x68

typedef union accel_t_gyro_union
{
 struct
 {
  uint8_t x_accel_h;
  uint8_t x_accel_l;
  uint8_t y_accel_h;
  uint8_t y_accel_l;
  uint8_t z_accel_h;
  uint8_t z_accel_l;
  uint8_t t_h;
  uint8_t t_l;
  uint8_t x_gyro_h;
  uint8_t x_gyro_l;
  uint8_t y_gyro_h;
  uint8_t y_gyro_l;
  uint8_t z_gyro_h;
  uint8_t z_gyro_l;
 } reg;
```

```
  struct
  {
    int16_t x_accel;
    int16_t y_accel;
    int16_t z_accel;
    int16_t temperature;
    int16_t x_gyro;
    int16_t y_gyro;
    int16_t z_gyro;
  } value;
};

//Declaração das variáveis que guardarão os valores anteriores para que possa ser comparados
    int16_t x_accel_init;
    int16_t y_accel_init;
    int16_t z_accel_init;
    int16_t x_gyro_init;
    int16_t y_gyro_init;
    int16_t z_gyro_init;

//-----------------------------------------------
// Função de análise dos dados do giroscópio
int leGiroscopio()
{
  int error, cont;
  double dT;
  accel_t_gyro_union accel_t_gyro;

  error = MPU6050_read (MPU6050_ACCEL_XOUT_H, (uint8_t *) &accel_t_gyro, sizeof(accel_t_gyro));

  uint8_t swap;
  #define SWAP(x,y) swap = x; x = y; y = swap
```

```
  SWAP (accel_t_gyro.reg.x_accel_h, accel_t_gyro.reg.x_accel_l);
  SWAP (accel_t_gyro.reg.y_accel_h, accel_t_gyro.reg.y_accel_l);
  SWAP (accel_t_gyro.reg.z_accel_h, accel_t_gyro.reg.z_accel_l);
  SWAP (accel_t_gyro.reg.t_h, accel_t_gyro.reg.t_l);
  SWAP (accel_t_gyro.reg.x_gyro_h, accel_t_gyro.reg.x_gyro_l);
  SWAP (accel_t_gyro.reg.y_gyro_h, accel_t_gyro.reg.y_gyro_l);
  SWAP (accel_t_gyro.reg.z_gyro_h, accel_t_gyro.reg.z_gyro_l);

 if(abs (accel_t_gyro.value.z_gyro - z_gyro_init) > 500 &&
(abs(accel_t_gyro.value.z_gyro ) > 1000))
 {
  cont = 1;
 } else {
  cont = 0;
 }
  x_accel_init = accel_t_gyro.value.x_accel;
  x_gyro_init = accel_t_gyro.value.x_gyro;
  y_gyro_init = accel_t_gyro.value.y_gyro;
  z_gyro_init = accel_t_gyro.value.z_gyro;

  return cont;
}

// ---------------------------------------------------------
// Função de leitura dos dados brutos do giroscópio
int MPU6050_read(int start, uint8_t *buffer, int size)
{
  int i, n, error;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
  n = Wire.write(start);
  if (n != 1)
    return (-10);
```

```
  n = Wire.endTransmission(false);
  if (n != 0)
    return (n);

  Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);
  i = 0;
  while(Wire.available() && i<size)
  {
    buffer[i++]=Wire.read();
  }
  if ( i != size)
    return (-11);

  return (0);
}

// ---------------------------------------------------------
// Função de configuração do giroscópio
int MPU6050_write(int start, const uint8_t *pData, int size)
{
  int n, error;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
  n = Wire.write(start);
  if (n != 1)
    return (-20);

  n = Wire.write(pData, size);
  if (n != size)
    return (-21);

  error = Wire.endTransmission(true);
  if (error != 0)
```

```
    return (error);

  return (0);
}

// ------------------------------------------------------
// Função que solicita a configuração do giroscópio
int MPU6050_write_reg(int reg, uint8_t data)
{
  int error;

  error = MPU6050_write(reg, &data, 1);

  return (error);
}

#define DIREITA 1  //definindo o número do lado direito
#define ESQUERDA 2  //definindo o número do lado esquerdo
#define LEDDIREITO 7  //definindo o número do led direito
#define LEDESQUERDO 8  //definindo o número do led esquerdo
#define SETADIREITA 10  //definindo o número do botão direito
#define SETAESQUERDA 11  //definindo o número do botão esquerdo

int seta = 0;  // variável que indica o estado da seta da moto (direita, esquerda ou desligada)
int estado = 0;  // variável que indica o estado do giroscópio (virando ou não virando)
int estadoAnt = 0;  // variável que indica o estado anterior ao atual do giroscópio (virando ou não virando)
int temporizador = 0; // variável que conta a quantidade de tempo em segundos

//-----------------------------------------------------
// Função que configura todos os componentes do sistema sobre seus comportamentos e como serão utilizados pelo sistema
```

```
void setup() {
  Serial.begin(9600);
  uint8_t c;
  accel_t_gyro_union accel_t_gyro_ini;

  Wire.begin();

  MPU6050_write_reg (MPU6050_PWR_MGMT_1, 0);

  pinMode(LEDDIREITO, OUTPUT);
  pinMode(LEDESQUERDO, OUTPUT);
  pinMode(SETADIREITA, INPUT);
  pinMode(SETAESQUERDA, INPUT);
}

//--------------------------------------------------------
// Função de iteração principal do sistema
void loop() {

  // Bloco que reconhece uma conversão ou estabilização do giroscópio de acordo com as setas
  if (seta != 0) {
    estado = leGiroscopio();
    if (estado) {
      estadoAnt = estado;
    } else if (estadoAnt) {
      delay(1000);
      temporizador++;
    }
  } else {
    estado = 0;
    temporizador = 0;
    estadoAnt = 0;
  }
```

```
// Bloco que reconhece quando o botão que aciona a seta direita foi pressionado
if (digitalRead(SETADIREITA)) {
  delay(250);
  if (seta == 0 || seta == ESQUERDA) {
    seta = DIREITA;
    digitalWrite(LEDDIREITO, HIGH);
    digitalWrite(LEDESQUERDO, LOW);
  } else {
    seta = 0;
    digitalWrite(LEDDIREITO, LOW);
  }
  temporizador = 0;
}

// Bloco que reconhece quando o botão que aciona a seta esquerda foi pressionado
if (digitalRead(SETAESQUERDA)) {
  delay(250);
  if (seta == 0 || seta == DIREITA) {
    seta = ESQUERDA;
    digitalWrite(LEDESQUERDO, HIGH);
    digitalWrite(LEDDIREITO, LOW);
  } else {
    seta = 0;
    digitalWrite(LEDESQUERDO, LOW);
  }
  temporizador = 0;
}

// Bloco que desliga as setas de acordo com a finalização de uma conversão
if (temporizador == 3) {
  estado = 0;
  estadoAnt = 0;
```

```
    temporizador = 0;
    digitalWrite(LEDESQUERDO, LOW);
    digitalWrite(LEDDIREITO, LOW);
    seta = 0;
  }
}
```