Instrumentação de um helicóptero não tripulado

Luciano Coutinho Caldas

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos, da Universidade de São Paulo. Curso de Engenharia Elétrica com ênfase em Eletrônica.

Orientador: Prof. Dr. Marco Henrique Terra

Co-orientador: Dr. Roberto Santos Inoue

São Carlos,
5 de dezembro de 2012

Coutinho Caldas, Luciano
C145i Instrumentação de um helicóptero não tripulado / Luciano Coutinho Caldas; orientador Marco Henrique Terra; coorientador Roberto Santos Inoue. São Carlos, 2012.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2012.

1. VANT. 2. helicóptero. 3. autônomo. 4. instrumentação. 5. inercial. 6. plataforma 3 graus de liberdade. 7. microcontrolador. I. Título.

FOLHA DE APROVAÇÃO

Nome: Luciano Coutinho Caldas

Título: "Instrumentação de um helicóptero não tripulado"

Trabalho de Conclusão de Curso defendido e aprovado em 28 / 11 / 2012,

com NOTA 9,5 (nove, meio), pela Comissão Julgadora:

Prof. Associado Marco Henrique Terra (Orientador)
SEL/EESC/USP

Prof. Dr. Valdir Grassi Júnior
SEL/EESC/USP

Prof. Dr. Adriano Almeida Gonçalves Siqueira
SEM/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

Dedicatória

Por meio deste pequeno espaço dedico este trabalho aos meus pais Henriques Caldas e Lucia Maria Coutinho Caldas que sempre me apoiaram durante minha graduação além dos outros momentos difícieis, mesmo que quase sempre a distância.

A Sra Cláudia, que Deus a tenha, que sempre esteve conosco nos suportando na república em que resido.

Agradecimentos

A toda minha família por estar sempre disposta a ajudar nos momentos difíceis e a minha segunda família, república "Nois q Tah", onde moro com verdadeiros irmãos que me ensinaram muito desde minha chegada à cidade, Leandro Poli, Juliana, Aruan, Douglas, Lukas, Paulo Mário, Bruno, Carlos Eduardo, Júnior, Caue, Orlando, Daniel, Vitor, Daniel Santos e a meu grande amigo Kelter.

Ao meu co-orientador e amigo Roberto Santos por me encaminhar e ensinar muito desde o período de iniciação científica, junto com meu orientador Marco Henrique Terra, Samuel, João e todo o pessoal do laboratório de sistemas inteligentes (LASI-EESC).

Aos senhores Odair e Rui funcionários do departamento de engenharia elétrica que com toda certeza dão suporte e salvam grande parte dos alunos deste curso.

Resumo

Este trabalho de conclusão de curso trata da instrumentação de uma plataforma de 3 graus de liberdade para realização de experimentos com um mini-helicóptero, a confecção de uma placa de circuito impresso do piloto automático e a realização de experimentos de estabilização em bancada do mini-helicóptero considerado nesta pesquisa. Além de familiarizar o aluno com a tecnologia utilizada em VANTs (Veículos aéreos não tripulados). A plataforma e o sistema do piloto automático encontram-se em desenvolvimento pelo grupo de pesquisa do Laboratório de Sistemas Inteligentes (LASI) da USP em São Carlos.

Palavras-chave: VANTs, helicóptero, autônomo, instrumentação, inercial, plataforma 3 graus de liberdade, microcontrolador.

Abstract

This project deals with the instrumentation of a 3-degrees-of-freedom platform to make experiments using a helicopter. After assembling its mainboard, tests of stability were performed with the helimodel. An important goal of this work is acquainting the student with VANT and helicopter technology. The platform and software of the autopilot have been developed in the Laboratory of Intelligent Systems (LASI) at USP São Carlos.

Keywords: VANTs, helicopter, autonomous, instrumentation, inertial, 3 freedom degrees of a platform, microcontroller.

Sumário

Lista de Figuras **xiii**

1 Introdução **1**

2 Análise dos componentes **5**

2.1 Visão global do autopiloto 5

2.2 Análise do funcionamento do rádio controle 6

2.3 Análise e testes dos *encoder* rotacional 9

2.3.1 O CI HCTL2017 10

2.4 *Rabbit RCM5400W* 12

2.5 IMU 6DOF v4 13

2.6 GPS 15

2.7 *Pololu micro serial 8 servo controller* 18

2.8 Sensor de pressão 20

2.8.1 Sensor MPXV5004DP 20

2.8.2 Sensor MPXA6115A 21

2.9 Sensor de temperatura - TMP35 23

2.10 Conversor analógico digital com saída serial - ADC0838 24

3 Resultados **27**

3.1 Reestruturação e instrumentação da plataforma 3D 27

3.2 Chaveamento entre microprocessador/rádio controle 29

3.3 Confecção da placa de circuito impresso da aquisição de dados dos *encoders* 30

3.4 Circuito do sensor de tensão e corrente 32

3.5 Circuito do sensor de pressão e temperatura 35

3.6 Controlador dos servos - Pololu 37

3.7 Circuito do piloto automático 40

4 Testes de bancada **45**

5 Conclusão **53**

6 Trabalhos futuros **55**

Referências Bibliográficas **57**

7 Anexos **59**

Lista de Figuras

1.1 Diagrama da origem dos sinais do autopiloto. 2

1.2 Esquema da base de testes desenvolvido pela equipe do LASI. . . 3

2.1 Periféricos que compõem o autopiloto. 6

2.2 Rádio Futaba 7CAP/7CHP utilizado no projeto. 7

2.3 Receptor de rádio que comanda os servo motores. 7

2.4 Esquema do rádio Futaba. 8

2.5 Diferentes formas de onda para as diferentes posições da chave. . . 8

2.6 *Encoders* que seriam utilizados. 9

2.7 Esquema de pinos dos *encoders* HEDS-5500. 10

2.8 Sequência de pulsos nos canais A e B, seguindo o padrão do código Gray. 10

2.9 Esquema de pinos do CI HCTL2017. 11

2.10 Leitura dos 3 *encoders* simultâneas feitas através do *Dynamic C*, *software* do Rabbit. 12

2.11 Módulo RCM5400W (figura retirada do manual do fabricante). . . 14

2.12 IMU 6DOF v4 (Figura retirada do manual do fabricante) 15

2.13 Receptor GPS da *San Jose Navigation* 32 canais, 5 Hz com antena (figuras retiradas do manual do fabricante). 16

2.14 *Pololu micro serial 8 servo controller* 18

2.15 Sensor MPXV5004DP (Foto retirada da internet). 20

2.16 Gráfico da resposta do sensor MPXV5004DP (gráfico retirado do *datasheet* do fabricante). 21

2.17 Sensor MPXA6115A (Foto retirada da internet). 22

2.18 Gráfico da resposta do sensor MPXA6115A (gráfico retirado do *datasheet* do fabricante). 22

2.19 Gráfico da resposta do sensor TMP35/36/37 (gráfico retirado do *datasheet* do fabricante). 23

2.20 Esquema de pinos do *ADC0838*. 24

3.1 Esq: Base de movimento 3D antiga. Dir: Base nova 3D. 27

3.2 Versão final da base para testes em bancada. 28

3.3 Diagrama elétrico do circuito desenvolvido para leitura dos *encoders*. 30

3.4 Placa de circuito impresso dupla face desenvolvida para leitura dos *encoders*. À esq. a face superior, e à dir. a face inferior. *Nota: Esse padrão será o mesmo para todas as outras placas mostradas aqui.* 31

3.5 Placa montada e funcionando para leitura dos *encoders*. 32

3.6 Esquemático do sensor de tensão e corrente. 33

3.7 Circuito impresso do sensor de tensão e corrente. 33

3.8 Circuito final montado. 34

3.9 Ensaio realizado com o sensor. 35

3.10 Esquema do circuito para aquisição dos dados dos sensores de pressão e temperatura. 36

3.11 Circuito criado a partir do esquema dos sensores. 37

3.12 Placa montada dos sensores de pressão e temperatura. 37

3.13 Tubo de Pitot usado para medição da pressão dinâmica. 38
3.14 Esquema elétrico do multiplexador controlador dos servos. 38
3.15 Circuito criado para o controle dos servos. 39
3.16 Placa montada do controlador com o Pololu. 39
3.17 Esquema de pinos do *Rabbit*. 40
3.18 Esquemático do piloto automático. 40
3.19 Esquema elétrico do piloto automático. 41
3.20 Esquema do piloto automático gerado a partir do esquema elétrico. 42
3.21 Vista de cima do piloto automático sem as placas externas. 43
3.22 Piloto automático finalizado. 44
3.23 Teste da alimentaçao do circuito e dos LEDs. 44

4.1 Montagem do sistema para testes. 45
4.2 Vista mais detalhada da instrumentação do helicóptero. 46
4.3 Gráfico dos valores lidos diretamente dos acelerômetros. 47
4.4 Gráfico dos valores lidos diretamente dos giroscópios. 48
4.5 Gráfico dos valores lidos diretamente do magnetômetro. 49
4.6 Gráfico do ângulo ϕ obtido no teste. 50
4.7 Gráfico do ângulo θ obtido no teste. 51
4.8 Gráfico do ângulo ψ obtido no teste. 52
4.9 Interface gráfica criada no MATLAB para comunicação com o piloto automático. 52

Capítulo 1

Introdução

O desenvolvimento do sistema de um helicóptero não tripulado não é algo trivial, pois requer conhecimentos interdisciplinares tais como engenharia espacial, engenharia elétrica, comunicações, ciência da computação, controle de sistema, operações em sistema em tempo real, entre outros, veja JUNG & TSIOTRAS (2007), JANG & LICCARDO (2006), ISIDORI et al. (2003), BOGDANOV et al. (2004), SARIPALLI & SUKHATME (2007) e INOUE et al. (2007). Por ser um dispositivo mecânico controlado eletronicamente que pode alcançar altas velocidades e altitudes, com altas velocidades de rotações de suas hélices, pode oferecer riscos consideráveis se não for projetado e construído com um alto grau de confiabilidade.

O principal componente de um helicóptero não tripulado é o piloto automático, o qual consiste de um computador de controle de voo, sensores, atuadores, dispositivos de comunicação e periféricos, juntamente com um programa para integrar os sinais de todos esses dispositivos. A Figura 1.1 exibe um diagrama de como é composto um autopiloto, em especial, o proposto neste projeto. Nela é visto a origem de cada sinal inercial e como eles se relacionam entre si.

Existem vários pilotos automáticos no mercado porém, os códigos internos deles não são possíveis de serem acessados em geral, possibilitando apenas tarefas de mais alto nível. Assim, de maneira a obter maior flexibilidade na implementação

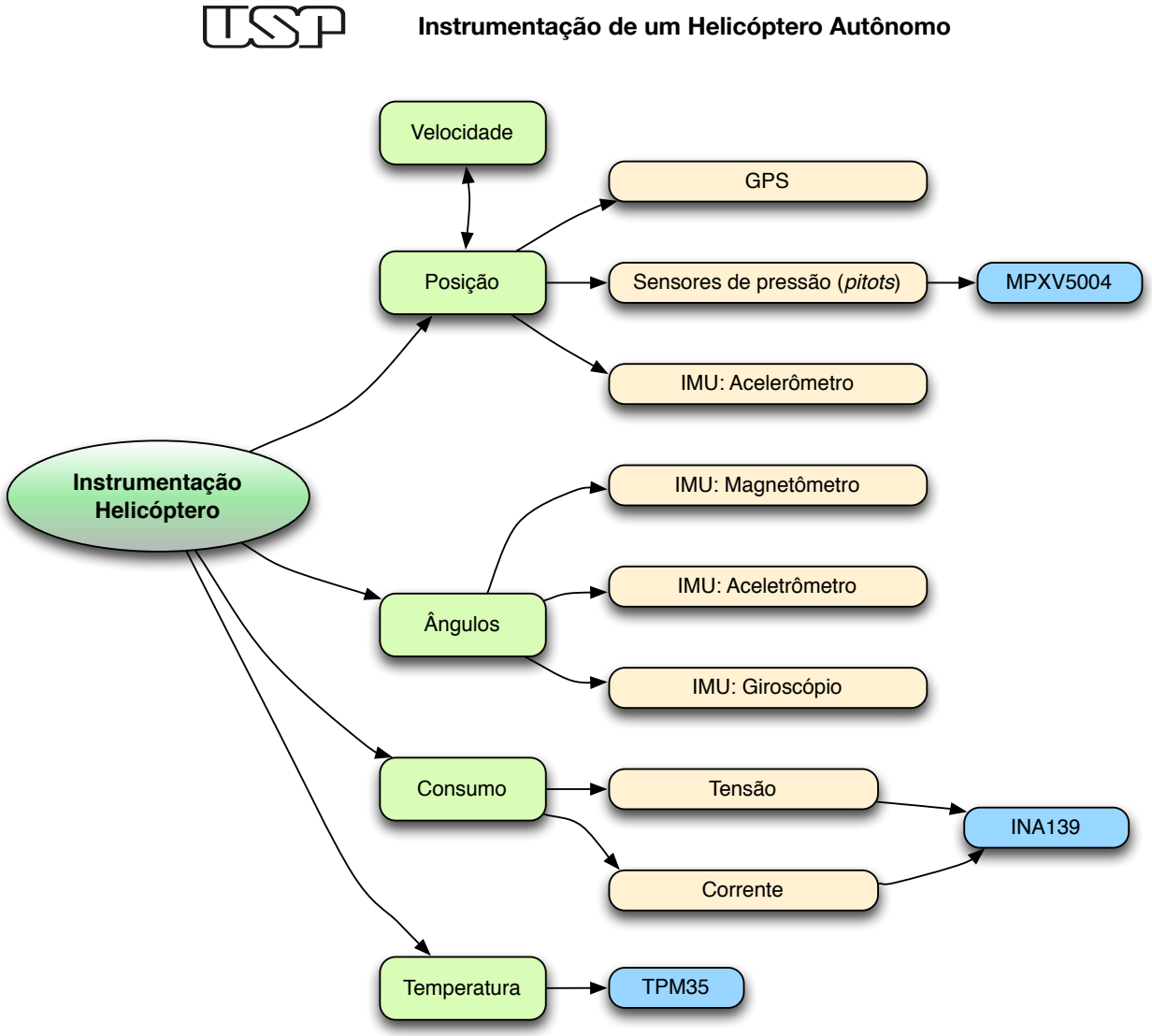


Figura 1.1: Diagrama da origem dos sinais do autopiloto.

de sistemas que determinem a posição, atitude e controle de um helicóptero não tripulado este projeto objetiva o desenvolvimento de um sistema de controle para um aparelho desse tipo.

Outro ponto importante e necessário para se construir um autopiloto são os testes que precisam ser realizados. Certamente devido ao risco envolvido não convém realizar testes em pleno voo, riscos estes relacionados ao perigo de se operar com um helicóptero, que possui um rotor que gira em alta velocidade e pode causar sérios danos se tocar em qualquer coisa. Uma alternativa para contornar este problema é utlizar uma plataforma de testes. Tal plataforma foi

desenvolvida no laboratório do LASI, Figura 1.2, e tem por objetivo servir de base para realização de testes de bancada. Nela é possível que o helicóptero se movimente de maneira angular, sem deslocamento linear, ou seja, nos seus ângulos naturais de arfagem, rolagem e guinada, por isso chamada de plataforma de 3 graus de liberdade. Os valores dos ângulos de inclinação podem então ser lidos via *encoders* (caixas pretas na Figura 1.2) e usados para as devidas finalidades. O helicóptero fica preso à última plataforma, podendo girar livremente e inclinar-se para os lados.

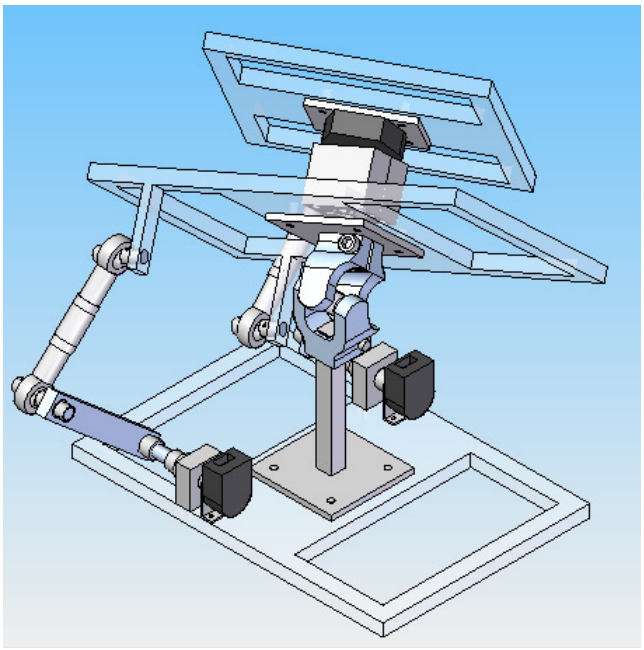


Figura 1.2: Esquema da base de testes desenvolvido pela equipe do LASI.

O foco do projeto foi em: aquisição dos ângulos desta plataforma de 3 graus de liberdade por um microprocessador utilizando *encoders*, a confecção da placa de circuito impresso do sistema do piloto automático, bem como os outros circuitos que o compõe, e a finalização do sistema do piloto automático que se encontra em desenvolvimento pelo grupo de pesquisa do LASI. O piloto automático será composto pelo microprocessador, pelo controle dos servos do mini-helicóptero que deve permitir alternar entre os comandos do microprocessador e os comandos de um receptor de rádio frequência, o sistema de posicionamento e atitude, compostos por sensores de baixo custo acelerômetros, giroscópios, magnetômetros, sensores de pressão estático (altitude) e dinâmico (velocidade do ar), e um recep-

tor GPS (*Global Positioning System*).

Capítulo 2

Análise dos componentes

2.1 Visão global do autopiloto

Antes de entrar em detalhes dos componentes do autopiloto propriamente ditos, esta seção servirá para introduzir o leitor dos componentes que compõe o autopiloto montado neste trabalho. Derivado da Figura 1.1 pode-se então montar um esquema mais refinado dos periféricos usados neste projeto. A Figura 2.1 exibe um diagrama das interfaces entre esses periféricos. Na parte superior, os balões de cor verde clara, estão os sensores fonte dos sinais inerciais e parâmetros físicos. Imediatamente abaixo, os que já possuem comunicação serial utilizam apenas *buffer* para compensar níveis diferentes de tensão. Outros como temperatura, pressão, sensor de tensão e corrente necessitam além do *buffer* conversores analógico digital para então enviar os dados para o microprocessador (*Rabbit*).

Ao centro encontra-se o processador responsável pelo monitoramento do sistema e tomada de decisões.

Abaixo em azul estão os atuadores. Antes igualmente aos sensores, é utilizado *buffer* para nivelamento de tensão. O atuador final, os servos, que agem na dinâmica do helicóptero ditando para onde será a direção e atitude do mesmo, seu comando é antecedido por um multiplexador, chamado de "Seleção"em vermelho no esquema. Os comandos dos servos podem vir de duas origens: Comando

manual, via rádio controle; ou pelo piloto automático, que é interfaceado pelo "Pololu"que é um receptor de comandos serial que os converte para os sinais dos servos.

Tal multiplexagem se dá necessária para caso haja algum problema com o piloto automático em voo, o usuário possa "retomar"manualmente o controle do helicóptero evitando assim que algum acidente venha a ocorrer.

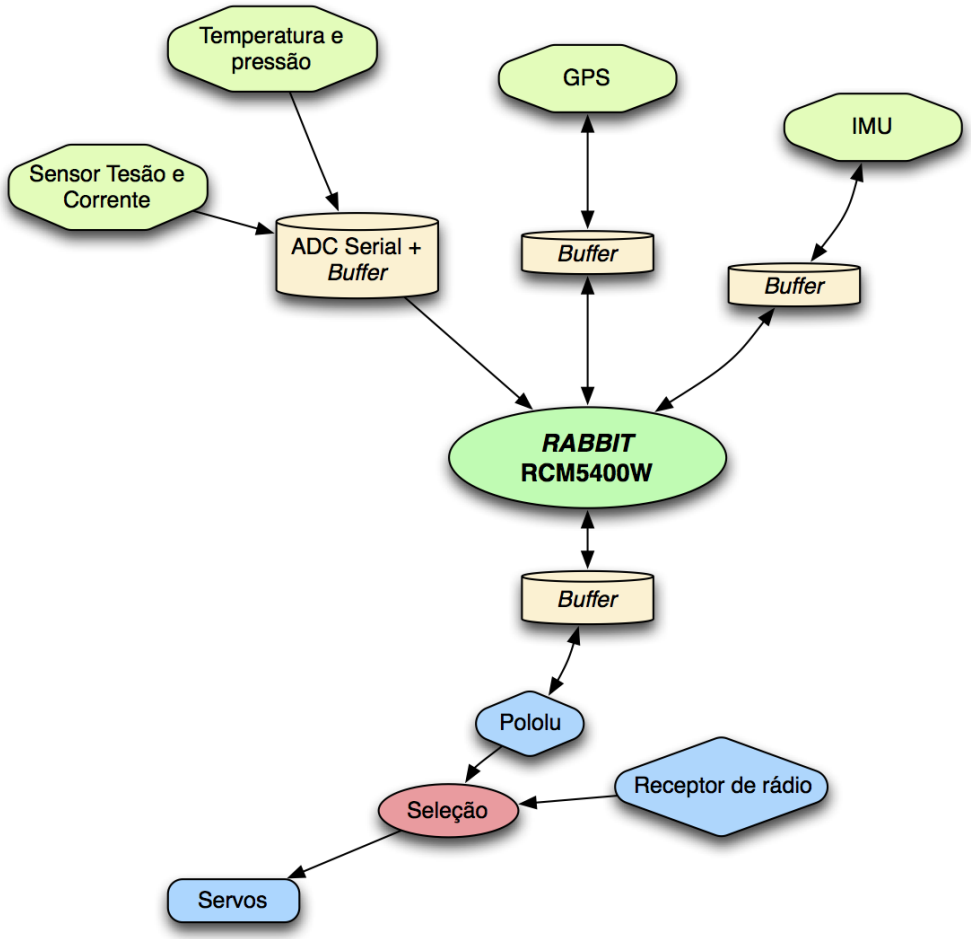


Figura 2.1: Periféricos que compõem o autopiloto.

2.2 Análise do funcionamento do rádio controle

Dada a necessidade de se trabalhar com os sinais de acionamento dos servo motores gerados pelo receptor de rádio do helimodelo, foi feito um estudo dis-

pondo de um osciloscópio para avaliar a forma de onda do sinal desse receptor bem como a relação entre os canais do receptor e os comandos do rádio. As figuras 2.2 e 2.3 mostram respectivamente o rádio controle que acompanha o helicóptero e o receptor que envia os comandos de posição para os servo motores. Na Figura 2.4 pode ser vista relação entre os comandos de rádio e em quais canais eles atuam.

Figura 2.2: Rádio Futaba 7CAP/7CHP utilizado no projeto.

Figura 2.3: Receptor de rádio que comanda os servo motores.

A análise via osciloscópio dos sinais enviados pelo receptor do rádio para os servos são da forma de PWM onde a posição angular dos servos é determinada pela largura dos pulsos gerados, que é proporcional a posição da alavanca do rádio controle. A Figura 2.5 exibe os dois extremos captados para um canal qualquer.

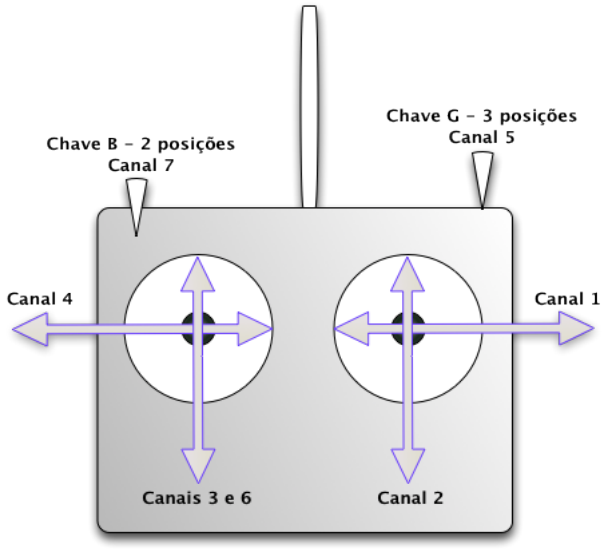


Figura 2.4: Esquema do rádio Futaba.

Quando o rádio está em "repouso", isto é, sem influencia da mão do usuário, o sinal de PWM possui período de duração de $1,52ms$ de duração. Esse pulso pode variar em até cerca de $0,40ms$ para mais ou para menos conforme a posição das alavancas do rádio.

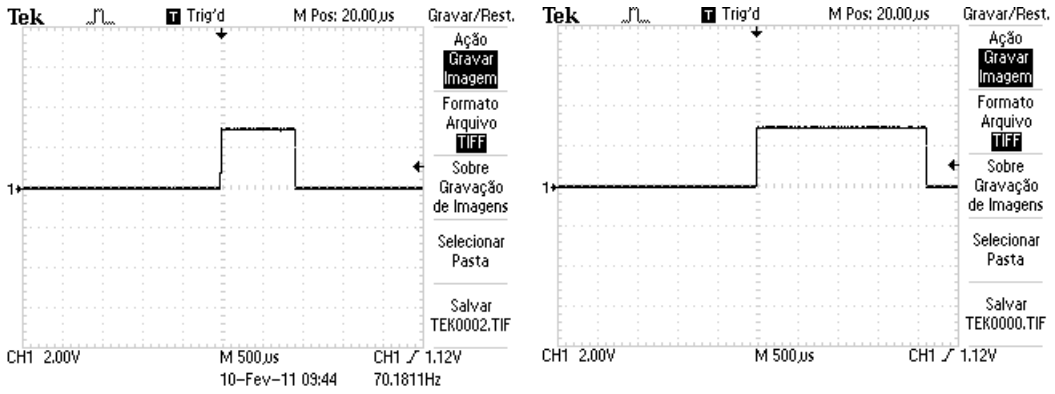


Figura 2.5: Diferentes formas de onda para as diferentes posições da chave.

Uma análise um pouco mais atenta nota-se que no rádio controle há apenas 2 alavancas de comando com 2 eixos cada, o que enviaria comando para apenas 4 canais, mas, o receptor possui 7 canais. Isso se deve a, como visto na Figura 2.4, há 2 alavancas que comandam 2 canais, bem como a alavanca esquerda vertical envia comandos para 2 canais, que na prática funcionam em oposição, ou seja, enquanto um aumenta o outro diminui a largura do pulso, trabalhando então em

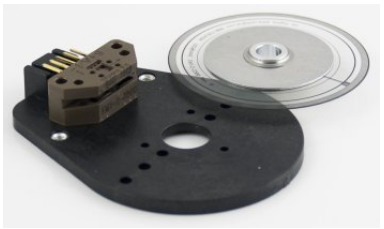
complemento.

Será discutido na seção 3.2 o uso do canal 7, que é comandado pela chave B, para o chaveamento entre o controle dos servo motores.

2.3 Análise e testes dos *encoder* rotacional

O *encoder* usado (HEDS-5500) possui 2 pinos de saída referidos como canal A e canal B, de acordo com a Figura 2.7. Ao girar o eixo do *encoder* uma sequência de pulsos é gerada nesses pinos, veja a Figura 2.8, esses pulsos estão defasados de 90 graus nos canais A e B, formando uma sequência que segue o padrão do código Gray. Note que para determinar o sentido de rotação do eixo, basta observar qual sinal varia primeiro de nível lógico baixo para nível lógico alto. Considerando o sentido anti-horário, o canal A varia antes do canal B, e vice-versa. Para determinar a posição angular do eixo é preciso utilizar um dispositivo que seja capaz de decodificar os sinais digitais do encoder. Essa decodificação consiste basicamente na contagem dos pulsos enviados pelo *encoder*. Ela pode ser crescente ou decrescente dependendo do sentido de rotação do eixo, e envia na saída o valor atual desta contagem. Este dispositivo é chamado de decodificador de quadratura. O Rabbit possui internamente dois desses decodificadores já prontos para uso. Um código simples foi utilizado (ver Anexos) apenas para entendimento do funcionamento deste componente.

(a)



(b)

Figura 2.6: *Encoders* que seriam utilizados.

Para leitura do terceiro *encoder* foi necessário então utilizar um C.I. externo para realizar tal tarefa. O componente escolhido foi o CI HCTL2017 fabricado

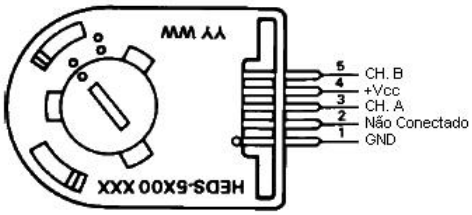


Figura 2.7: Esquema de pinos dos *encoders* HEDS-5500.

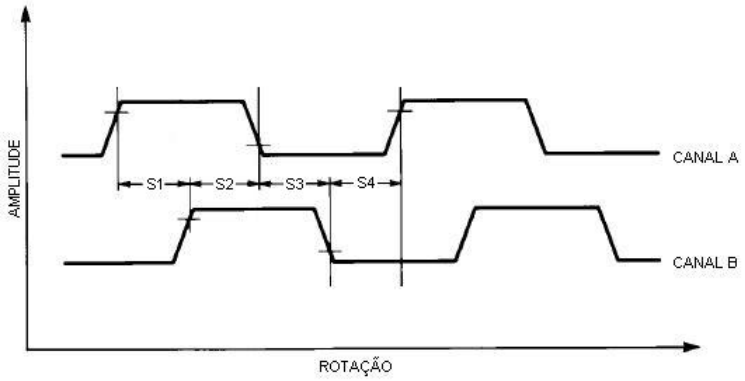


Figura 2.8: Sequência de pulsos nos canais A e B, seguindo o padrão do código Gray.

pela *Avago Technologies*. Este CI é um decodificador de quadratura e possui um contador de 16 bits. Para o projeto, pretende-se utilizar este CI, juntamente com os 2 decodificadores de quadratura presentes no módulo Rabbit RCM4500W, possibilitando, desta forma, a leitura de 3 encoders ao mesmo tempo.

2.3.1 O CI HCTL2017

O HCTL2017 é alimentado com uma faixa de tensão de +4,5V a +5,5V, é sincronizado por um clock externo com uma frequência de no máximo 14MHz, possui 2 entradas (CH A e CH B) que são conectadas ao encoder, um barramento de 8 bits na saída (D0 a D7) controlado por um latch, uma porta de habilitação (OE), uma porta de seleção (SEL) e uma porta de reset (RST). A Figura 2.9 ilustra como devem ser feitas as ligações dos pinos.

Como o CI possui somente 8 saídas, os 16 bits da contagem devem ser lidos por partes, separados em 8 bits mais significativos e 8 bits menos significativos.

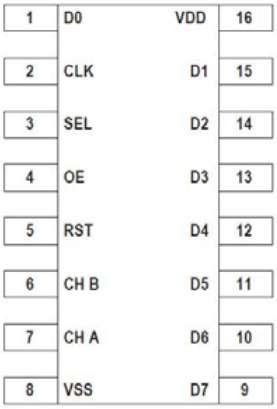


Figura 2.9: Esquema de pinos do CI HCTL2017.

Esse controle de leitura é feito pela porta SEL, sendo que os 8 bits mais significativos e os 8 bits menos significativos são selecionados, respectivamente, para SEL=0 e SEL=1. Já a porta OE é utilizada para habilitar ou desabilitar os buffers tristate presentes nessas saídas, colocando-as no estado de alta impedância quando OE=1. A porta RST é utilizada para zerar a contagem. A leitura dessas portas de controle é realizada somente na borda de descida do pulso do clock. Este procedimento de leitura foi feito e é mostrado no código anexo ao final deste relatório e está comentado para melhor entendimento.

O *datasheet* deste CI recomenda uma frequência mínima para o clock externo, que deve ser maior que pelo menos 6 vezes a frequência do encoder. Sabe-se que a freqüência máxima de operação recomendada pelo datasheet do encoder é de 100kHz, portanto, a freqüência de clock deve ser maior que 600kHz.

Com auxílio de *buffers* para reduzir/aumentar níveis de tensão devido a incompatibilidade entre Rabbit (funciona com 3.3V) e *encoders* (funcionam com 5V) junto com o C.I. HCTL2017 *Avago* (funciona com 5V) foi possível então montar o circuito e desta forma realizar a leitura dos *encoders*. O resultado obtido pode ser visto na Figura 2.10.

O código deste programa é visto nos Anexos ao final deste relatório e está todo comentado para facilitar o entendimento. Vale comentar sobre algumas funções chave utilizadas no programa, tais como: *BitWrPortI()*, usada para escrever apenas um bit na posição desejada numa porta desejada; *qd_read()*, função muito

útil pois ela que faz a leitura no decodificador de quadratura interna do Rabbit. Por fim, a função *RdPortI()* faz a leitura do valor gerado pelo decodificador de quadratura implementado em *hardware* para, assim, mostrar na tela.

Esta etapa foi de fundamental importância pois será utilizada para os testes de estabilidade na segunda etapa deste projeto.

Figura 2.10: Leitura dos 3 *encoders* simultâneas feitas através do *Dynamic C*, *software* do Rabbit.

2.4 *Rabbit RCM5400W*

A partir das necessidades para a automação de um VANT, inicialmente foi proposto um estudo do ambiente de desenvolvimento do Rabbit a fim de se familiarizar com este ambiente e sua linguagem de programação *Dynamic C* bem como suas interfaces com *hardware* tais como comunicação serial, portas de entrada e saída, decodificadores de quadratura, dentre outros.

O módulo RabbitCore RCM5400W (Figure 2.11) possui comunicação Wi-Fi 802.11b/g que possibilita o desenvolvimento de sistemas de embarcados de baixo custo, baixo consumo, com controle *wireless* embarcado. Dentre suas características, o módulo inclui hardware DMA, velocidade clock superior a 100 MHz, conexões I/O compartilhadas com seis portas seriais e quatro níveis de funções de pinos alternados que incluem fase variável PWM, I/O auxiliares, decodificadores de quadratura, e captura de entrada. Seu ambiente de programação, *Dynamic C*, possui códigos já prontos que permitem reduzir o tamanho do código

escrito pelo programador. As principais características do módulo RCM5400W são (para maiores detalhes consultar o manual do fabricante):

- Microprocessador Rabbit 5000 @ 73.73 MHz;
- 512K de dados SRAM;
- 512K de memória Flash;
- 1M de memória serial Flash;
- Conectividade Wi-Fi 802.11b/g padrão, ISM 2.4 GHz;
- 35 portas digitais I/O configuráveis para quatro funções diferentes;
- 6 portas seriais de alta velocidade, CMOS compatível;
- Possui relógio de tempo real;
- Supervisor *Watchdog*;
- 10 temporizadores de 8 bit, 1 de 10 bit, e 1 de 16 bit;
- 4 canais PWM;
- Alimentação e portas I/O de 3,3 V;
- Pequeno tamanho: 47x72x14 mm;

Inicialmente foi proposto um estudo a fim de se familiarizar com o kit. O foco principal deste estudo foi a leitura dos *encoders* que são do tipo rotacional e incremental, veja a Figura 2.6, que será descrito no próximo tópico.

2.5 IMU 6DOF v4

A unidade inercial de medida utilizada neste projeto foi a 6DOF v4 fabricado pela *sparkfun* vista na Figura 2.12. Ela possui acelerômetros de três eixos MMA7260Q com sensibilidades configuráveis em 1,5g, 2g, 4g ou 6g, 2 giroscópios IDG300 e magnetômetros HMC1052L e HMC1051Z. A frequência de transmissão

Figura 2.11: Módulo RCM5400W (figura retirada do manual do fabricante).

dos dados é escolhida pelo programador, e essa transmissão pode ser feita em ASCII ou em formato binário por *Bluetooth* ou via serial (TTL).

As características do IMU 6DOF v4 são:

- Tensão de alimentação: 4,2V a 7V d.c.;
- Tensão Tx e Rx de 3,3V;
- Frequências de resposta: Sensores magnéticos: 312Hz; Giroscópios IDG300: 120 Hz; Acelerômetros MMA7260Q 350Hz (eixos X e Y) e 150Hz (eixo Z).

Os dados enviados pela IMU são compostos pelas medidas ativas e são iniciadas por um “A” (65 em ASCII) terminadas por “Z” (90 em ASCII). Cada canal é informado conforme a sequência abaixo:

1. Contador;
2. Magnetômetros (x,y,z);
5. Acelerômetros (x,y,z);

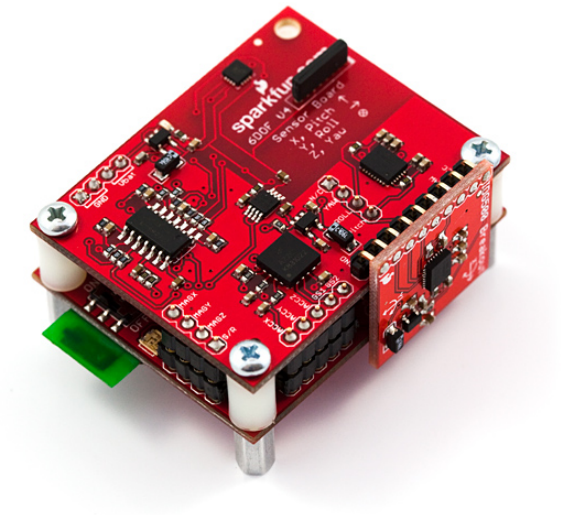


Figura 2.12: IMU 6DOF v4 (Figura retirada do manual do fabricante)

8. Giroscópio (x,y,z);

O contador é composto por dois *bytes* que chega como MSB-LSB, e irá variar de 0 para 32767. Se algum dos outros canais estiver selecionado como inativo, o dado será omitido da saída e o dado posterior subirá e será informado na sequência. No modo binário, cada canal ativo é informado com 2 *bytes*: MSB e LSB, nessa sequência, e eles estarão sempre entre 0 e 1023 por causa do analógico digital de 10-bit. O tamanho da estrutura de dados será de 4 *bytes* ("A", "Z" e o contador) mais dois *bytes* de cada medida ativa. Com todos os canais ativos a estrutura de dados terá o tamanho de 22 *bytes*.

2.6 GPS

Em geral veículos autônomos necessitam de um GPS para sua localização global. O GPS usado nesse projeto foi do fabricante *San Jose Navigation* exibido na Figura 2.13 que fará parte do piloto automático. Abaixo é citado algumas de suas especificações.

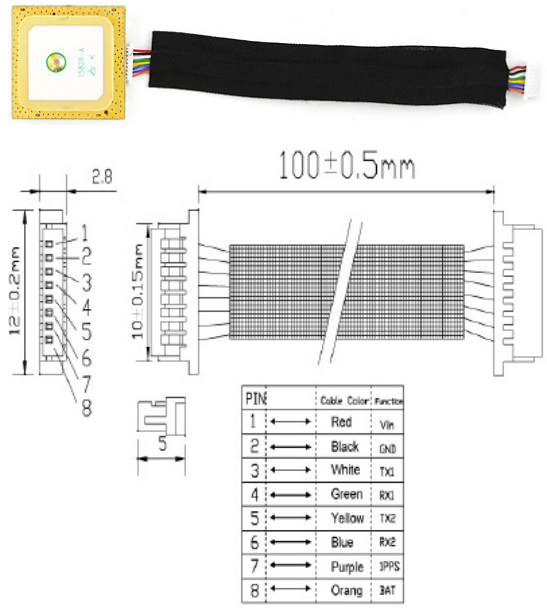


Figura 2.13: Receptor GPS da *San Jose Navigation* 32 canais, 5 Hz com antena (figuras retiradas do manual do fabricante).

O esquema de ligação dos pinos do GPS da Fig. 2.13 é o seguinte:

1. Tensão de entrada 3,3∼5V dc ± 10%;
2. Terra;
3. Transmissor da porta serial 1, saída TTL, 0∼2,8V (deixar aberto se não for usado);
4. Receptor da porta serial 1, entrada TTL, 0∼2,8V (deixar aberto se não for usado);
5. Transmissor da porta serial 2, saída TTL, 0∼2,8V (deixar aberto se não for usado);
6. Reservado (não utilizar e deixar aberto);
7. Pulso de tempo, pode ser usado para ligar e desligar LED, saída TTL, 0∼2,8V;
8. Tensão de entrada da bateria de *backup* 2∼5V DC ± 10%.

Algumas de suas características são:

- Taxa de atualização de 1 ∼ 5Hz;

- *Baud Rate* de 4800 a 115200 bps;
- Suporte a DGPS, WAAS, EGNOS, MSAS;
- Arquitetura do receptor de 32 canais paralelos;
- -158dBm de sensibilidade;
- 3,3m de precisão de posição;
- 2,6m de precisão de posição com DGPS;
- 0,1 Knot RMS de precisão de velocidade;
- Alimentação de 3,3 a 5 V;
- Potência de consumo de 59/42/33 mA;
- 2 portas seriais NMEA;
- Pino para bateria de *backup*.

O protocolo usado pelo GPS da *San Jose Navigation* é o NMEA, que informa os dados no formato ASCII. Esse é um formato padrão para aplicações com GPS. Mais informações no manual do fabricante.

2.7 *Pololu micro serial 8 servo controller*

Este controlador de servos pode controlar até 8 servos a partir de um computador ou micro-controlador. A velocidade de cada servo é controlada independentemente e a taxa de transmissão de dados da comunicação serial é detectada automaticamente, com valores de *Baud Rate* variando de 1200 a 38400 bps. Este dispositivo suporta até 2 protocolos de comunicação separados, oferecendo assim um controle de velocidade e posição individual para cada servo.

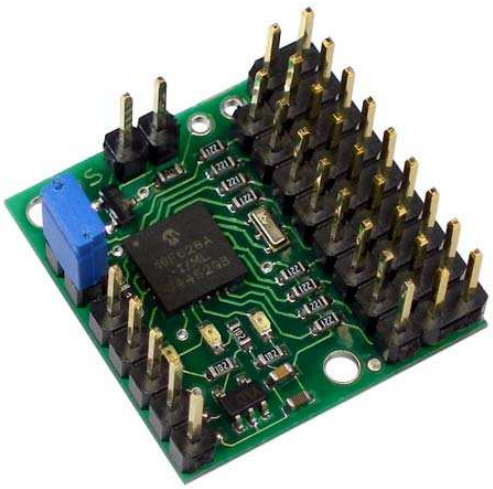


Figura 2.14: *Pololu micro serial 8 servo controller*

Outras características do controlador de servo serial Pololu são:

- Resolução de $0,5\mu s$ (aproximadamente 0,05 graus);
- Alimentação lógica de 5-16 VDC;
- Tensão dos dados 0-5 VDC;
- Taxa do pulso de 50 Hz;
- Corrente média de 5 mA.

A comunicação com o controle de servos Pololu é feita serialmente segundo um protocolo de 5 ou 6 *bytes*. O primeiro *byte* é um valor de sincronização que

deve estar sempre em $0x80(128)$. O segundo é o número do dispositivo Pololu, o qual é $0x01$ para o controlador de 8 servos (isso possibilita a comunicação com dois controladores de servo em uma única rede de comunicação). O *byte* 3 serve para enviar comandos para o controlador dos servos, que são discutidos abaixo. O *byte* 4 é o servo para o qual o comando deve ser aplicado. Os *bytes* 5 e 6 são os valores de dados para o comando enviado. Os comandos para o controlador de servos são:

- Comando 1: *Set Speed*(1 *byte* de dado). Este comando possibilita configurar a qual velocidade os servos irão se mover. Se a velocidade é configurada em 0 (*default*), o pulso de saída instantaneamente mudará para a configuração de posição. Com velocidade 1, o tamanho de pulso muda de 50 microsegundos para 1 segundo; a velocidade máxima de 6,35 ms é alcançada com uma configuração de velocidade 127.

- Comando 2: *Set Position*, 7-*bit* (1 *byte* de dado). Este comando é útil para comunicações velozes pois somente 5 *bytes* são enviados para configurar uma posição. Configurar a posição de um servo irá automaticamente ligar os servos.

- Comando 3: *Set Position, 8-bit(2 bytes de dados).* Este comando é como a versão de 7-bit, exceto que dois *bytes* de dados devem ser enviados para transferir os 8 *bits*. O *bit* 0 do dado 1 contém o *bit* mais significativo, e os *bits* baixos do dado 2 contêm os 7 *bits* menores. (O *bit* 7 nos *bytes do dado devem ser 0*). Configurando uma posição do servo irá automaticamente ligar os servos.

- Comando 4: *Set Position, Absolute* (2 *bytes* de dados). Este comando permite o controle direto da posição interna dos servos. Neutro, faixa e configurações de direções não afetam este comando. Contém 2 dados, o dado 2 contém os 7 *bits* menores e o dado 1 os *bits* maiores. A faixa de valores válida está entre 500 a 5500. Configurando uma posição do servo irá automaticamente ligar os servos.

- Comando 5: *Set Neutral* (2 *bytes* de dados). A configuração do neutro aplica-se somente para os comandos de 7 e 8 *bits*. O valor de neutro configura o meio da faixa, e corresponde ao 7-*bit* de valor de posição 63,5 ou ao 8-*bit* de valor de posição 127,5. O valor de neutro é uma posição absoluta como a do comando 4, e configurando a posição do neutro moverá o servo para aquela posição. O valor padrão é 3000, mas pode ser útil mudá-lo se forem alterados os servos é necessário calibrar o sistema.

2.8 Sensor de pressão

No projeto foram utilizados dois tipos de sensores de pressão: MPXV5004DP (sensor relativo) e MPXA6115A (sensor absoluto).

2.8.1 Sensor MPXV5004DP

A família MPxx5004, Figura 2.15, é uma série de transdutores piezorresistivos, isto é, utiliza uma célula resistiva (*strain gauge*) de alta sensibilidade combinada com técnicas específicas para confecção de sensores de pressão. Esse sensor fabricado pela *Freescale Semiconductor* é ideal para uso com interface A/D uma vez que sua saída é linear (ver Figura 2.16) e varia de 1,0 a 4,9V. A faixa dinâmica de medida é feita de 0 a $3,92kPa$. É construído com tecnologia de silício monolítico, e seu sensor possui compensação de temperatura e já é calibrado. Algumas de suas principais características podem ser citadas:

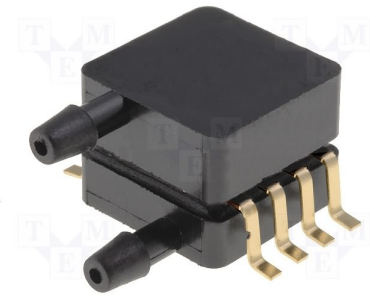


Figura 2.15: Sensor MPXV5004DP (Foto retirada da internet).

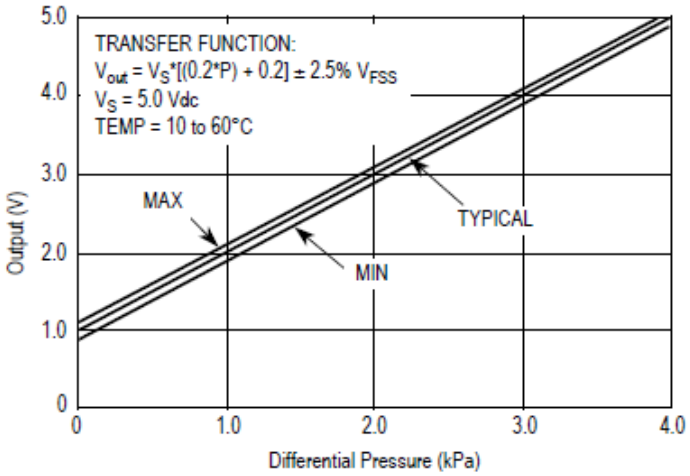


Figura 2.16: Gráfico da resposta do sensor MPXV5004DP (gráfico retirado do *datasheet* do fabricante).

- 1,5% de erro máximo em medidas de 0 a $0,98kPa$ (temperaturas entre 10 e 60 °C);
- 2,5% de erro máximo em medidas de $0,98$ a $3,92kPa$ (temperaturas entre 10 e 60 °C);
- Compensação de temperaturas entre 10 e 60 °C;
- Tensão de alimentação $5,0 \pm 0,25V$ dc;
- Corrente de alimentação de 10mA dc;
- Sensibilidade de $1,0V/kPa$;

Dessa forma, devido a sua simplicidade de conexão, que exige apenas uma alimentação dc e um sinal de saída proporcional à pressão diferencial e linear, além de outros, esse sensor se mostra um bom candidato para integrar o sensoriamento do piloto automático.

2.8.2 Sensor MPXA6115A

A família de sensores de pressão MPXxx6115A, Figura 2.17, fabricados pela *Freescale Semiconductor*, possui características de suportar altas temperaturas

mantendo sua alta precisão. Sua tecnologia é semelhante ao MPxx5004 que é baseado em um transdutor piezoresistivo, porém, este é designado para leitura de pressão absoluta. Seu sensor é integrado (tecnologia de silício monolítico) com circuitos amplificadores operacionais feitos com tecnologia bipolar com alta excursão de sinal de saída.

Pode ser citada algumas de suas principais características:

Figura 2.17: Sensor MPXA6115A (Foto retirada da internet).

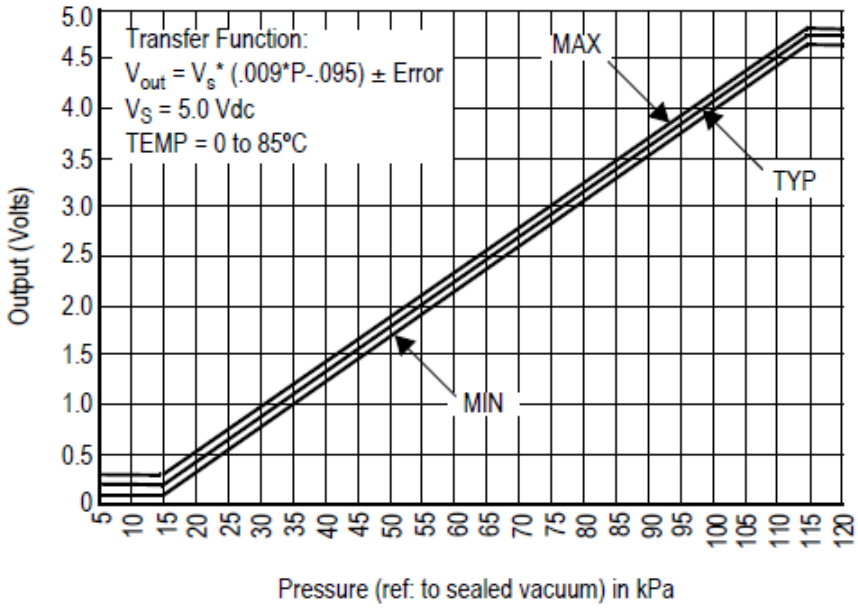


Figura 2.18: Gráfico da resposta do sensor MPXA6115A (gráfico retirado do *datasheet* do fabricante).

- Faixa de leitura entre 15 e 115kPa com saída correspondente a $0,2$ a $4,8V$;
- 1,5% de erro máximo em temperaturas entre 0 e 85 °C;
- Resistência à alta umidade (problema que atinge muitos sensores de pressão);
- Compensação para temperaturas entre -40 e +125 °C;

- Tensão de alimentação $5,0 \pm 0,25V$ dc;

- Corrente de alimentação de 6mA dc;

- Sensibilidade de $45mV/kPa$.

Na folha de dados são citadas algumas aplicações desse sensor, pode-se destacar duas: Altímetro de aviões; Sensor de pressão absoluto em máquinas a serem controladas. Dessa forma, esse sensor também é ideal para a instrumentação do piloto automático pois reúne características ideais para os propósitos do projeto.

2.9 Sensor de temperatura - TMP35

O sensor TMP35 é caracterizado por necessitar de uma baixa tensão para funcionamento, linearidade entre tensão de saída e temperatura (Figura 2.19), grande precisão na medida da temperatura bem como baixa potência.

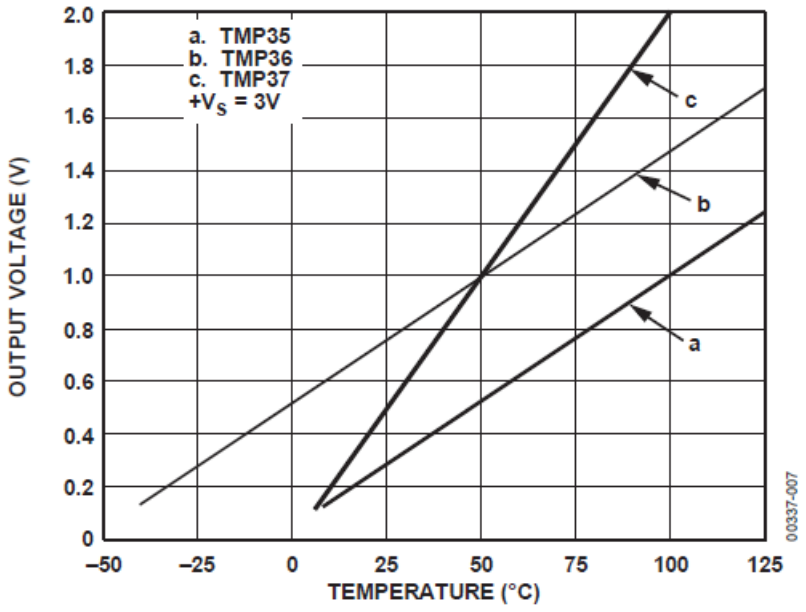


Figura 2.19: Gráfico da resposta do sensor TMP35/36/37 (gráfico retirado do *datasheet* do fabricante).

Algumas de suas características:

- Tensão de alimentação entre 2,7 e 5,5V;

- Sensibilidade de $10mV/°C$;

- Linearidade típica de $\pm 2°C$;

- Faixa de leitura entre -40 e $+125°C$.

O *datasheet* do fabricante recomenda esse CI em aplicações de controle de sistemas integrados conversores A/D e microprocessadores, ou seja, aplicação semelhante a essa abordada neste projeto.

2.10 Conversor analógico digital com saída serial - ADC0838

O *ADC0838* é um conversor analógico digital com resolução de 8 *bits* e interface serial.

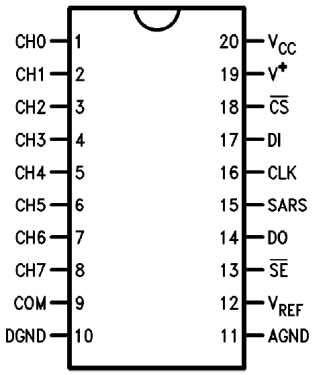


Figura 2.20: Esquema de pinos do *ADC0838*.

As características do *ADC0838* são:

- Fácil interface com microprocessadores

- Opera em módulo "stand-alone";

- Opera ratiometricamente ou com voltagem de referência de 5 V_{dc} ;

- Não requer ajuste de zero ou “full-scale”;
- Multiplexador de entrada configurável com 8 canais;
- Operação remota com link serial;
- Entrada/Saída compatível com TTL/MOS;
- Resolução de 8 bits;
- Fonte de alimentação única de 5 V_{dc};
- Baixa potência: 15mW;
- Tempo de conversão de 32 μs;

O controle do processo de conversão é feito por software. Os passos necessários para a realização de todo o processo são:

1. A conversão é iniciada quando o estado lógico da porta $\bar{CS}$ vai para alto e deve permanecer durante todo o processo;
2. Em cada borda de subida do *clock*, o estado dos dados na porta DI é alterado para o registrador de deslocamento de endereço MUX. O *start bit* é o primeiro “1” lógico que aparece nesta porta (todos os zeros à esquerda são ignorados). Depois do *start bit* o conversor espera os próximos 2 a 4 *bits* para formar a palavra de atribuição da MUX ;
3. Quando o *start bit* foi deslocado para o local de início do registrador da MUX, o canal de entrada foi atribuído e uma conversão está prestes a começar. Um intervalo de 1/2 *clock* (período em que nada acontece) é automaticamente inserido para permitir que o canal MUX selecionado seja ajustado. O estado da porta SAR vai para “alto”, neste momento, é um sinal de que a conversão agora está em curso e a porta DI é desabilitada (O conversor não aceitará mais dados);
4. A porta DO (*Digital output*) sai do *Tri-State* e fornece um zero à esquerda para o período *clock* utilizado para o ajuste da MUX;

5. Quando a conversão começa, a saída do comparador SAR, que indica se a entrada analógica é superior a V+ ou inferior a V- das tensões de cada uma das sucessivas escadas de resistências internas, aparece na porta DO em cada descida do *clock*. Estes dados são o resultado da conversão a ser deslocado para fora (com a primeira vinda MSB) e pode ser lido pelo processador imediatamente;

6. Depois de 8 períodos de *clock* a conversão foi concluída. A porta SAR retorna ao estado "baixo"para indicar o 1/2 *clock* do ciclo posterior;

7. Todos os registradores internos são apagados quando a linha $\bar{CS}$ vai para estado "alto". Se outra conversão é desejada, $\bar{CS}$ deve fazer uma transição do estado lógico de alto para baixo e em seguida todo o ciclo se repete.

Capítulo 3

Resultados

3.1 Reestruturação e instrumentação da plataforma 3D

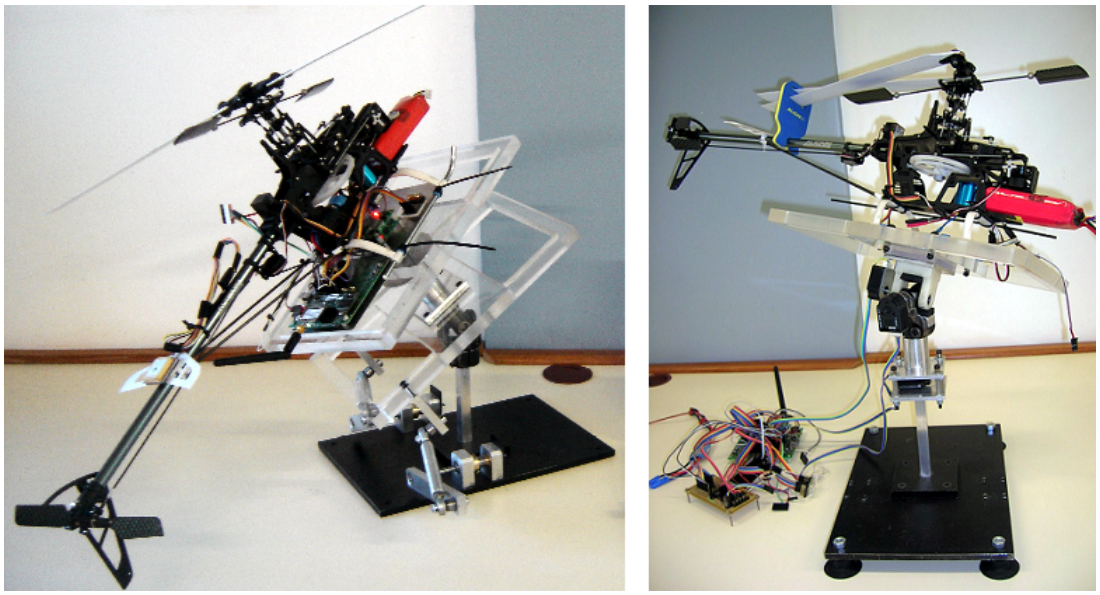


Figura 3.1: Esq: Base de movimento 3D antiga. Dir: Base nova 3D.

A etapa seguinte realizada no projeto foi a reformulação da plataforma de testes do helicóptero, devido aos problemas de estabilização do mini-helicóptero. Esses problemas eram gerados por causa do elevado peso da base do helicóptero, da considerável complexidade mecânica das conexões com os encoders, da baixa

potência do rotor do mini-helicóptero e principalmente da grande distância entre a base onde o helicóptero está fixado e o ponto de giro do mesmo. A Figura 3.1 mostra respectivamente a base antiga e a nova, onde nota-se estes e outros detalhes melhorados, como por exemplo, a simplificação na fixação dos *encoders*. Vale ressaltar, que a nova base não necessita de um *encoder* "vazado", que é um componente ligeiramente diferente dos outros utilizados, Figura 2.6b, para aquisição do ângulo de rotação da plataforma. E com a utilização do *encoder* da Figura 2.6a, o desenvolvimento da base tornou-se mais econômico.

Com o tempo e a tentativa de realizar testes de estabilidade a base ainda se mostrou não funcional. Na verdade o maior problema de se operar com esta base é a distância do centro de massa do sistema móvel junto com a fonte de empuxo (rotor) ao centro de giro (cruzeta). Devido a essa distância, faz-se necessário um esforço maior que o normal do rotor do helicóptero para apenas mantê-lo estável na vertical, pois, diferente da decolagem natural de um helicóptero, que parte da posição horizontal, quando este está preso a base, a decolagem se dá de uma posição "inclinada"ou relaxada como é visto na Figura 3.1.

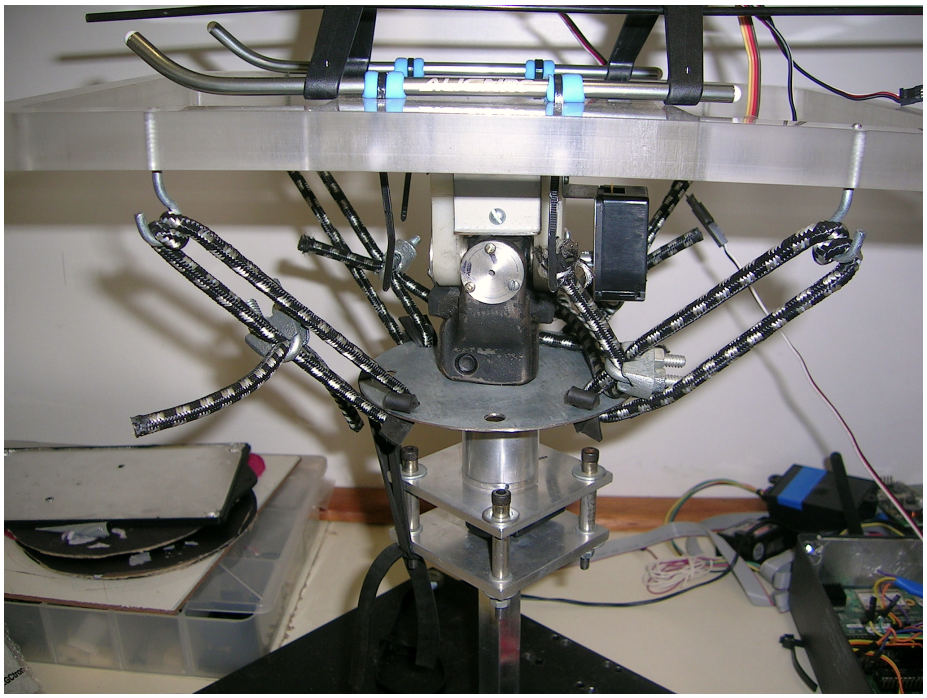


Figura 3.2: Versão final da base para testes em bancada.

Diante desse problema outras soluções foram tomadas para tornar possível

os testes de bancada. A Figura 3.2 mostra a versão final da base. A primeira solução foi limitar o ângulo de inclinação dos 2 graus de liberdade da cruzeta em um ângulo pequeno, de maneira a restringir em poucos graus a liberdade de movimentação do helicóptero (movimentos de arfagem e rolagem). Além disso, um disco metálico foi instalado logo abaixo da cruzeta (girando junto com esta, no sentido da guinada) para servir de suporte para elásticos que são presos à base acrílica a fim de minimizar o efeito do elevado esforço necessário para o helicóptero manter-se estável. Tal solução foi inspirada em sistemas comerciais de base de 3 graus de liberdade que geralmente usam 3 braços compostos por sistema mola/amortecedor junto com *encoder* lineares.

Com esta versão foi possível finalmente realizar o primeiro teste básico de voo com o helicóptero, ou seja, ligar e controlar manualmente este preso a base. Infelizmente após alguns testes uma conexão errada danificou o *speed control*, elemento que aciona o motor, o que impossibilitou realizar outros testes bem como uma possível filmagem para exibir posteriormente.

3.2 Chaveamento entre microprocessador/rádio controle

Um importante passo foi dado nesta etapa que foi resolver o problema do chaveamento entre os comandos do microprocessador e os comandos do receptor de rádio frequência, necessidade esta exposta na seção 2.1 . De posse dos dados do estudo feito na seção 2.2 pôde-se projetar um circuito para fazer o chaveamento entre os comandos recebidos pelo rádio receptor e os comandos enviados pelo *rabbit*.

As diferentes posições (2 no total) da chave B localizada no rádio (vista no canto superior/direito na Figura 2.2) mostraram que a diferença entre as posições da chave estava na largura do pulso contido num sinal periódico, Figura 2.5.

Desta forma, um circuito eletrônico deveria diferenciar quando a largura do

pulso era maior ou menor que um valor de referência, para assim fazer o chaveamento necessário. A solução mais simples do ponto de vista de *hardware* encontrada foi utilizar a interrupção externa de um microcontrolador para desempenhar tal função. O primeiro teste foi realizado em um microcontrolador já disponível *ATMEGA8-16PU* da ATMEL que funcionou perfeitamente. Por este componente ser grande (20 pinos) e possuir muitas funções desnecessárias para o nosso projeto, optou-se pelo microntrolador *ATTINY13-20PU* de 8 pinos também da ATMEL, pequeno e robusto, que atende perfeitamente as nossas necessidades. O programa usado está em anexo ao final deste relatório. A placa que contém este circuito será a de controle dos servos e será descrita na seção 3.6.

3.3 Confecção da placa de circuito impresso da aquisição de dados dos *encoders*

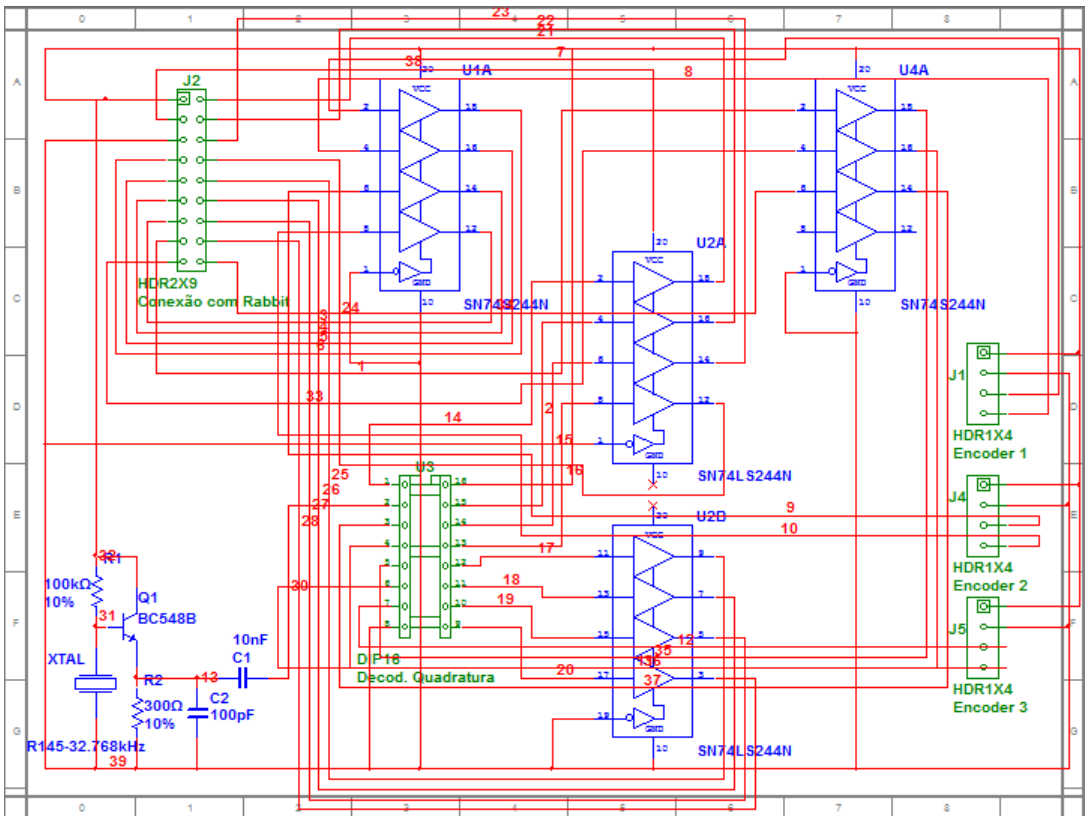


Figura 3.3: Diagrama elétrico do circuito desenvolvido para leitura dos *encoders*.

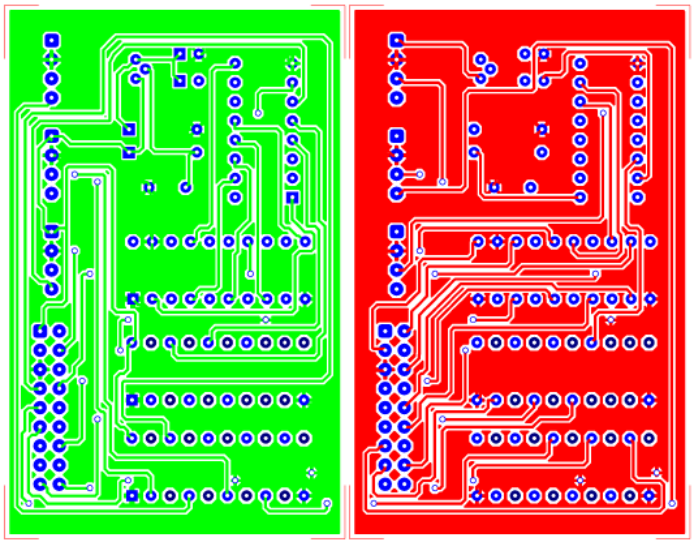


Figura 3.4: Placa de circuito impresso dupla face desenvolvida para leitura dos *encoders*. À esq. a face superior, e à dir. a face inferior. *Nota: Esse padrão será o mesmo para todas as outras placas mostradas aqui.*

Nesta placa estão contidos os *buffers* para nivelar níveis de tensão além de um decodificador de quadratura para o terceiro *encoder* (O kit de desenvolvimento do *Rabbit* possui apenas 2 decodificadores de quadratura internos). O diagrama do circuito elétrico e a placa do circuito impresso desenvolvidos no ambiente *Multisim* da *National Instruments* são mostrados nas figuras 3.3 e 3.4. A placa pronta pode ser vista na Figura 3.5.

No diagrama da Figura 3.3 é visto no canto direito três conectores verdes, que como estão assinalados, são os conectores que serão conectados aos três *encoders* conforme o esquema de ligação visto na Figura 2.7.

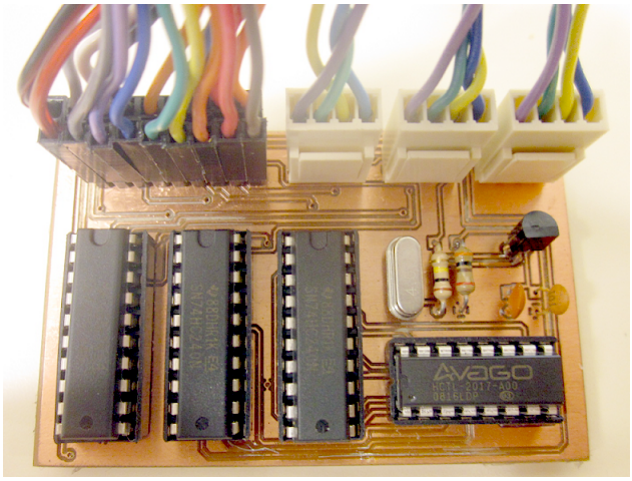


Figura 3.5: Placa montada e funcionando para leitura dos *encoders*.

Note que os *encoders* 1 e 2 ligam-se ao *buffer* U1A, que é o C.I. *SN74S244N* a fim de reduzir a tensão de 5V para 3,3V (compatível com Rabbit), que ligam-se ao conector J2, que por fim irão conectar-se ao Rabbit. Já o *encoder* 3 é ligado ao U3 que é o C.I. decodificador de quadratura. Este C.I. funciona com alimentação de 5V, e como esperado, sua saída antes de ir para o Rabbit deve passar por um *buffer* (U2 - *SN74S244N*) para assim reduzir sua tensão para 3,3V. De maneira análoga, o nível de tensão dos sinais dos comandos enviados pelo Rabbit para o decodificador também precisam ser aumentados de 3,3V para 5V, isso também é feito com um *buffer*, U4 - *SN74S244N*. O pequeno circuito analógico no canto esquerdo do diagrama é um oscilador que funciona na frequência do cristal montado. Este oscilador é necessário para o decodificador de quadratura, pois este irá operar na frequência deste sinal de entrada gerado.

3.4 Circuito do sensor de tensão e corrente

Quando se trata de um veículo aéreo onde sua fonte de energia são baterias, torna-se necessário saber a carga disponível. Dessa forma tornou-se necessária a construção de um circuito que realizasse tal função. Para isso, bastava ler a corrente que estava sendo consumida bem como o valor da tensão instantânea.

A leitura da tensão se dá de maneira direta via conversor A/D, cabendo

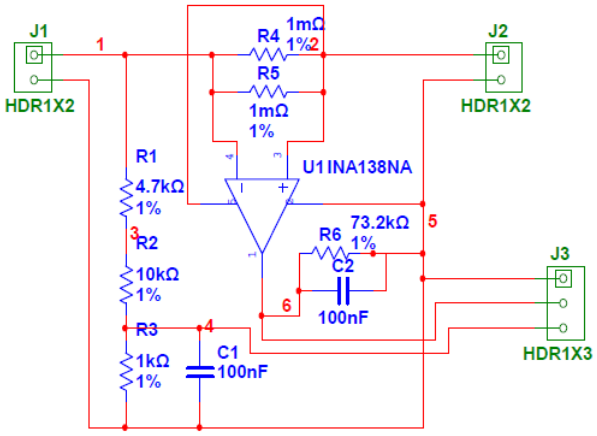


Figura 3.6: Esquemático do sensor de tensão e corrente.

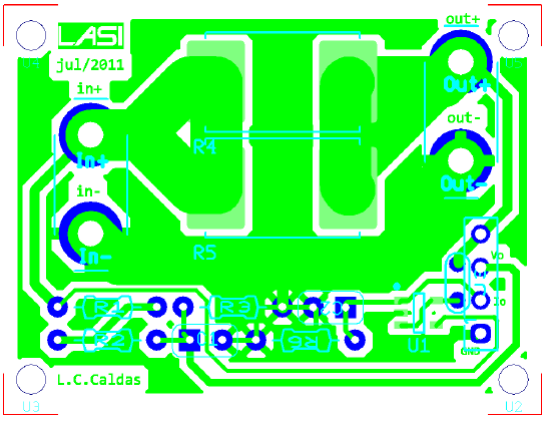


Figura 3.7: Circuito impresso do sensor de tensão e corrente.

apenas respeitar os limites de tensão do conversor, que na prática é feito via divisor resistivo. Para a leitura da corrente usou-se um CI dedicado. Na prática o jeito mais comum de se ler o valor de uma corrente é através de um resistor de *shunt*, que é um resistor de pequeno valor (de maneira que não influencie no funcionamento do circuito) em série com o circuito consumidor, e então é lida a tensão em cima desse resistor. Sabendo o valor da resistência basta aplicar a Lei de Ohm $V = R.I$ para obter a corrente instantânea.

Essas técnicas foram implementadas na instrumentação do helicóptero, porém, foi utilizado um CI dedicado para a leitura da corrente (INA139) que é baseado em um amplificador diferencial. O esquema do circuito é visto na Figura 3.6. O resistor de *shunt* usado foram dois resistores WSR-3.001 de $1m\Omega/3W$ com 1% de tolerância, em paralelo, gerando assim uma resistência de $R_S = 0,5m\Omega$.

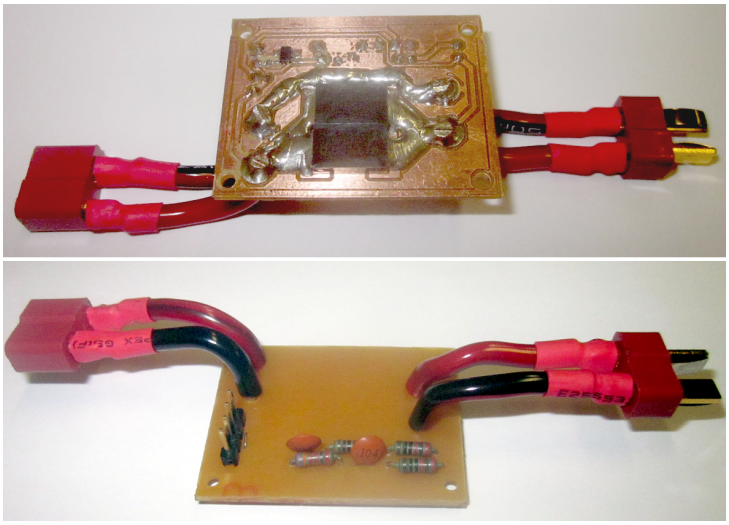


Figura 3.8: Circuito final montado.

Note que é um valor extremamente baixo de resistência, daí a necessidade de se usar um amplificador operacional dedicado pois a tensão lida será extremamente baixa também. Dessa forma, a tensão de saída do operacional será proporcional a corrente que está sendo lida.

O circuito foi então projetado e montado em uma placa de circuito impresso. O esquema da placa é exibido na Figura 3.7. Vale notar na espessura das trilhas que passam pelos resistores R_4 e R_5 (resistores de *shunt* que devem suportar a alta corrente que por ali irá passar).

A fim de testar o circuito implementado, foi montado um experimento em bancada para realizar o teste de carga. Este constituiu-se de uma carga (no caso uma resistência de chuveiro) ligada através do sensor. Dispondo de um voltímero e amperímetro foi realizada a calibração do sensor. A Figura 3.9 ilustra os pontos aquisitados no ensaio bem como a curva ajustada que relaciona tensão e corrente de entrada com as tensões de saída do sensor.

As equações obtidas para o sensor de corrente e tensão foram respectivamente

$$V_{out_1} = 0,4248 * I_{in} + 0,0587 \quad [V]$$

e

$$V_{out_2} = 0,0476 * V_{in} + 0,0017 \quad [V].$$

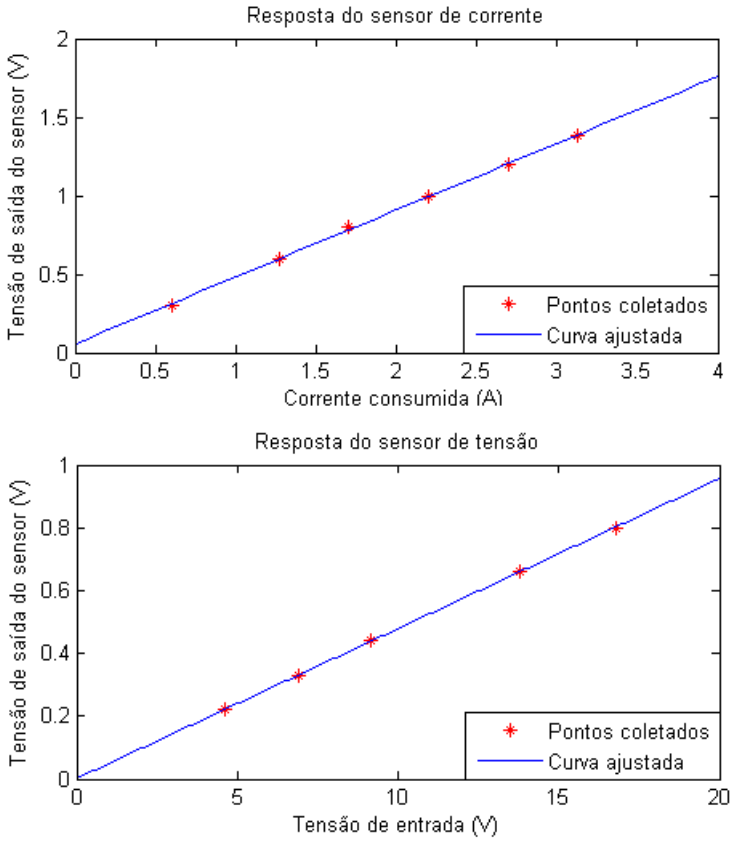


Figura 3.9: Ensaio realizado com o sensor.

3.5 Circuito do sensor de pressão e temperatura

Após a análise dos componentes a serem usados foi possível desenvolver o esquema elétrico da placa que irá conter os sensores de pressão, temperatura, bem como o valor lido da corrente e tensão. Todos estes valores serão lidos pelo conversor A/D ADC0838 e então enviados para o *Rabbit*.

A Figura 3.10 exibe o esquemático montado para esta placa. O conector J1 irá fazer a interface deste circuito com o *Rabbit*, e J2 irá receber os valores de tensão e corrente vindos da outra placa já citada, para então ser enviados os dados para o processador. Vale lembrar que o *foot print*, que é o desenho do componente

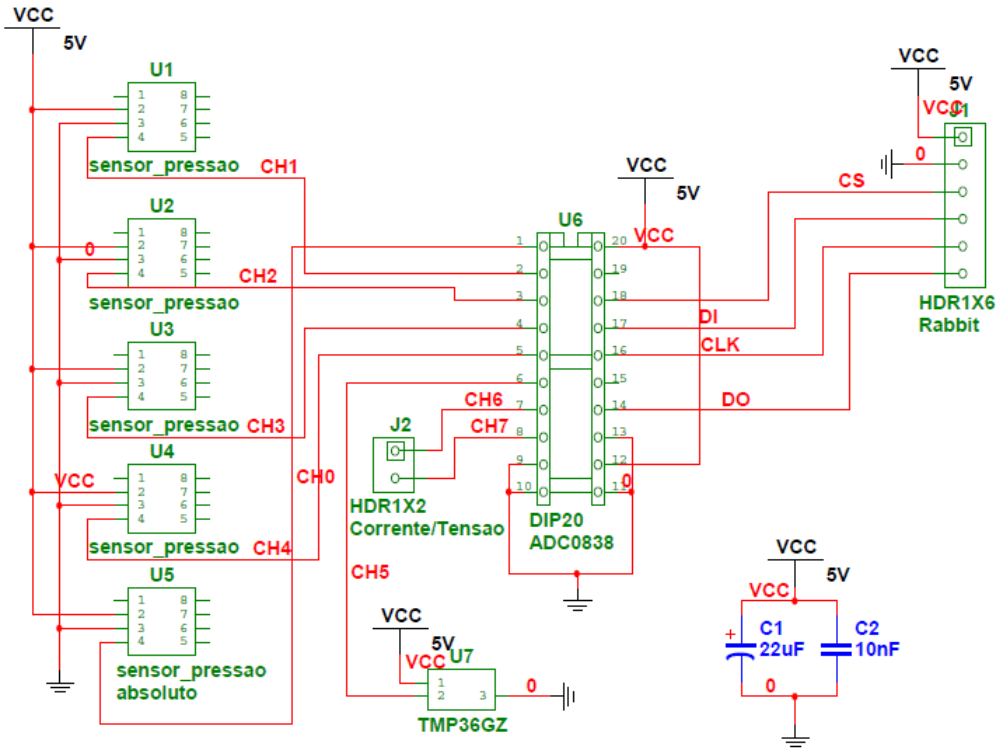


Figura 3.10: Esquema do circuito para aquisição dos dados dos sensores de pressão e temperatura.

na placa de circuito impresso, foi desenvolvido pois a biblioteca do *software* não continha estes sensores.

A placa desenvolvida é vista na Figura 3.11 e ela montada é finalmente vista na Figura 3.12.

Para o teste dessa placa seria necessário criar um protocolo de comunicação, e devido ao pouco tempo hábil, não foi possível realizá-lo. Além do mais, para sua instalação no conjunto de sensores que compõe o piloto automático, deve ser adquirido tubos de Pitot, Figura 3.13, onde a pressão dinâmica gerada está diretamente ligada com a velocidade onde se encontra este dispositivo. Por fim, com a modelagem matemática correta, pode-se estimar a velocidade do helicóptero nos eixos x, y e z.

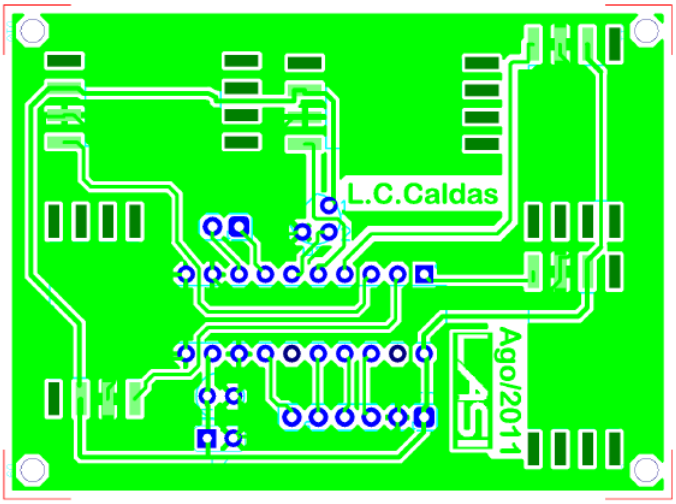


Figura 3.11: Circuito criado a partir do esquema dos sensores.

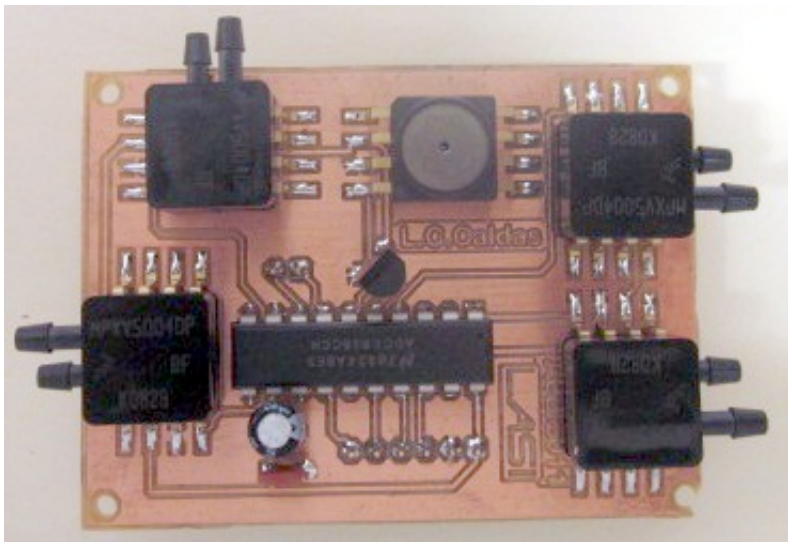


Figura 3.12: Placa montada dos sensores de pressão e temperatura.

3.6 Controlador dos servos - Pololu

A necessidade de se poder escolher entre quem irá pilotar o helicóptero, se é o piloto automático ou o usuário manualmente já foi comentada no item 3.2. Assim, coube desenvolver um esquema que incorporasse a solução já pronta do microcontrolador para acionar uma chave eletrônica, no caso, um multiplexador. Aqui foi escolhido o 74LS157 por trabalhar com os níveis de tensão desejados.

Uma das entradas do multiplexador vem da saída do Futaba, ou seja, do receptor do rádio. A outra vem da placa do Pololu, que é controlada serialmente.

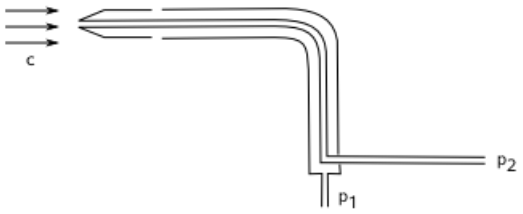


Figura 3.13: Tubo de Pitot usado para medição da pressão dinâmica.

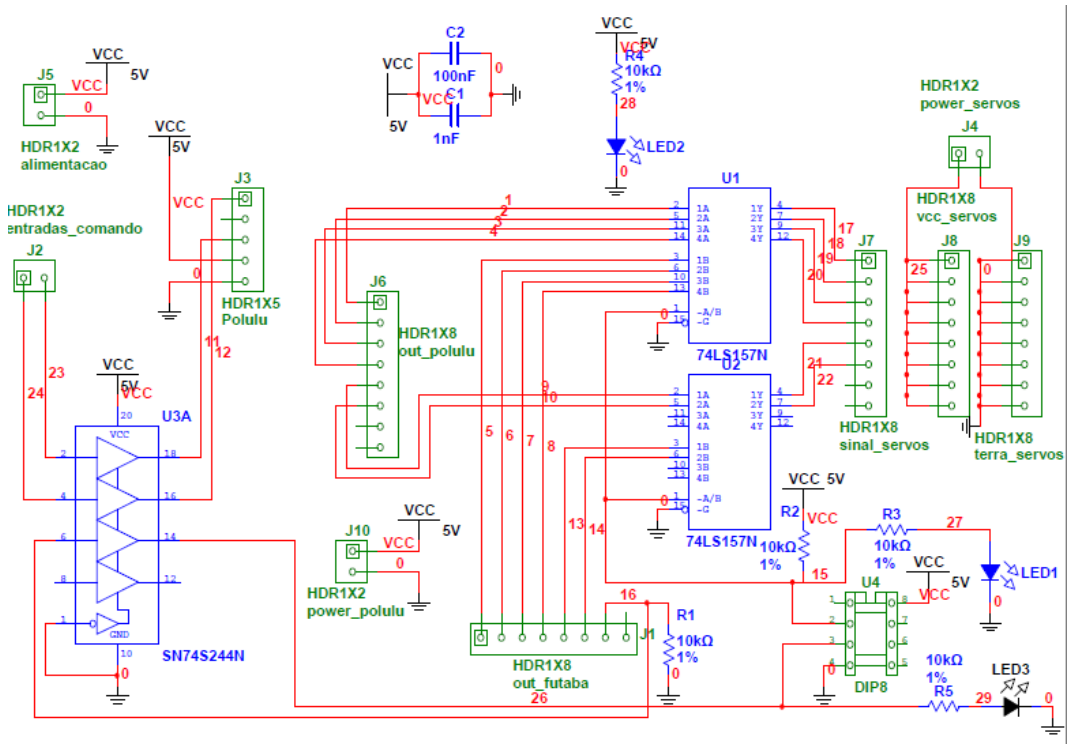


Figura 3.14: Esquema elétrico do multiplexador controlador dos servos.

Assim, basta organizar o circuito de maneira adequada para cumprir com o objetivo. A Figura 3.14 mostra o esquema elétrico proposto. Um *buffer* foi necessário para elevar a tensão de saída do *Rabbit* de 3,3V para 5V devido aos quesitos do Pololu. O conector J6 recebe os comandos dos servos vindas do Pololu. Já o J1 recebe os vindos do receptor de rádio Futaba. Os conectores J7, J8 e J9 são a saída, onde irão se ligar os servos. J5 alimenta o circuito, enquanto J2 recebe o sinal serial de comando do piloto automático vindo do *Rabbit*. O soquete U4 irá conter o microcontrolador *Attiny13*.

A Figura 3.15 mostra a placa de circuito impresso gerada. Na Figura 3.16 tem-se a placa finalmente montada. Note que o Pololu já está conectado a placa,

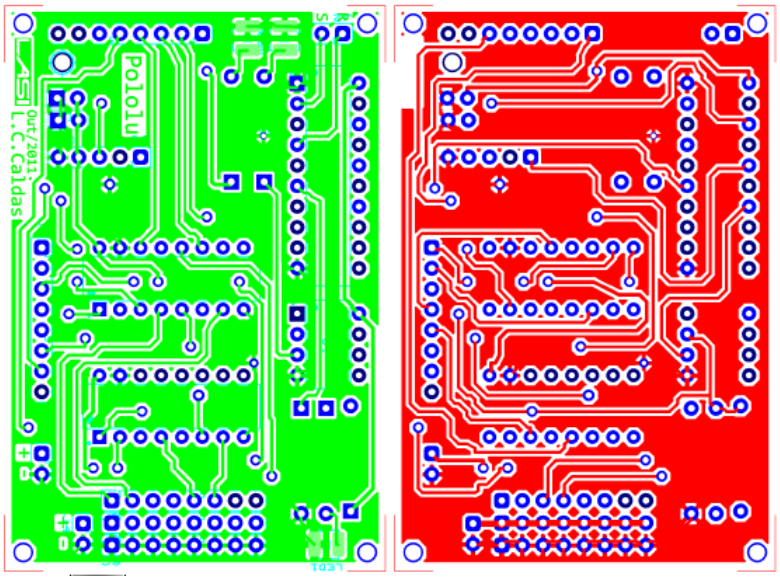


Figura 3.15: Circuito criado para o controle dos servos.

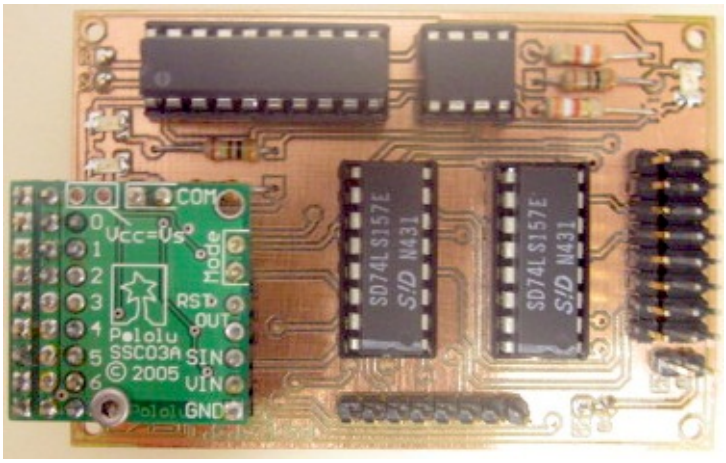


Figura 3.16: Placa montada do controlador com o Pololu.

e como foi montado com soquetes, sua remoção se dá com extrema facilidade.

Foram realizados os testes necessários nesta placa desconectada do piloto automático, e funcionou perfeitamente. O microcontrolador atuou nos multiplexadores e então pode-se escolher entre os comandos do rádio controlador ou os comandos enviados via entrada. O código usado para os testes foi desenvolvido em *MATLAB* pelo Co-Orientador do projeto e encontra-se em anexo ao final deste relatório.

3.7 Circuito do piloto automático

De posse dos componentes periféricos prontos, as extremidades da Figura 2.1, basta integrá-las de maneira a compor este esquema como um todo. É daí que surge o piloto automático. Como já foi mencionado, ele usará o *Rabbit RCM5400W* como processador, e desta forma, cabe de fato conectá-lo aos outros periféricos.

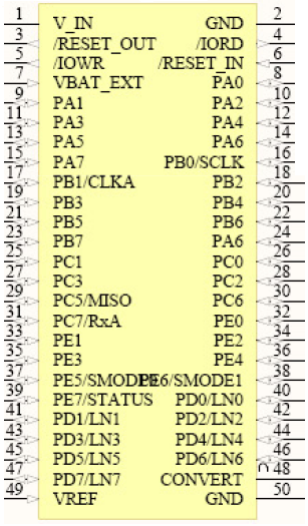


Figura 3.17: Esquema de pinos do *Rabbit*.

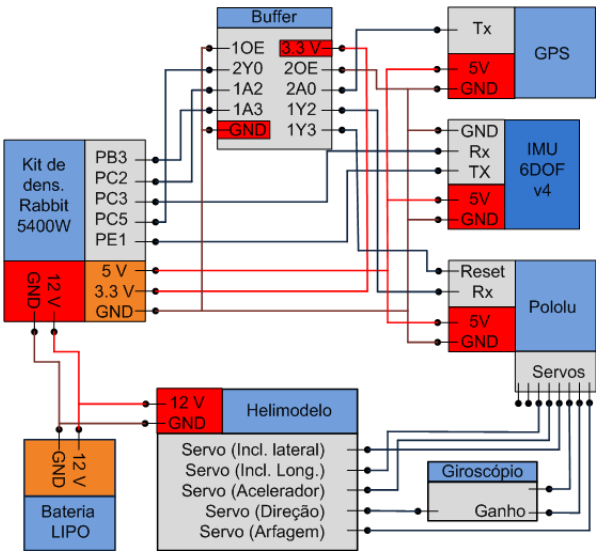


Figura 3.18: Esquemático do piloto automático.

O esquema dos pinos do *Rabbit* é visto na Figura 3.17. Diante dessa in-

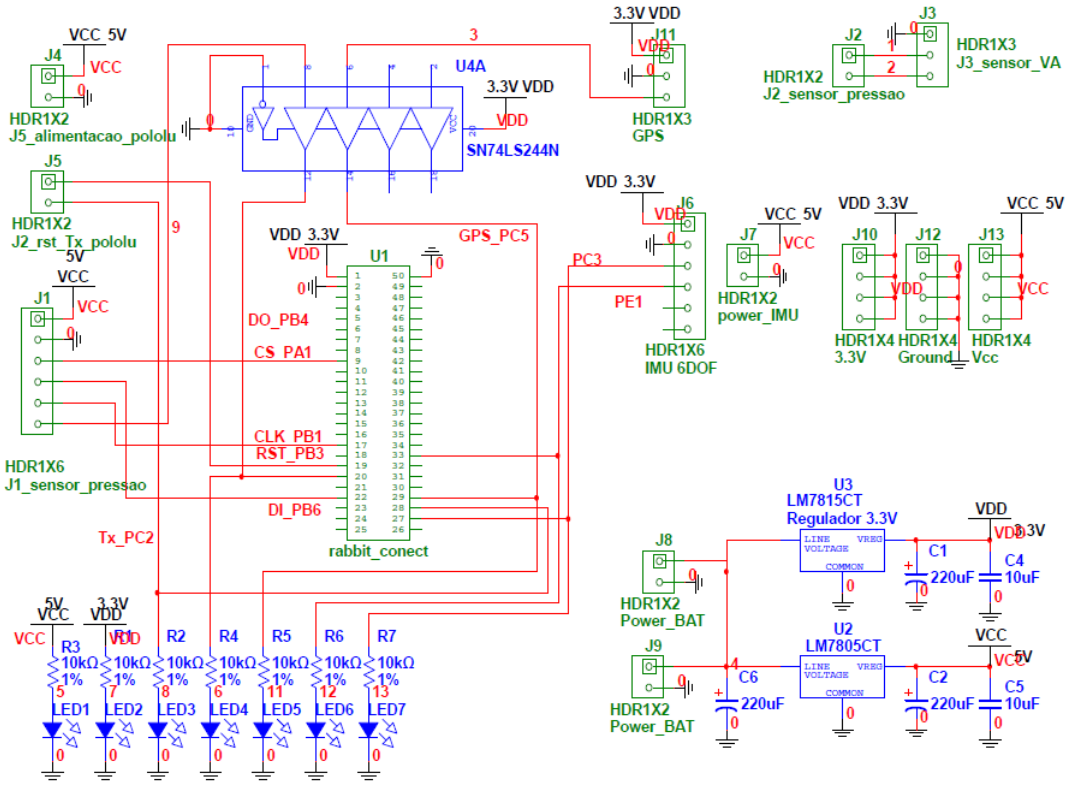


Figura 3.19: Esquema elétrico do piloto automático.

formação, e com o diagrama de blocos proposto na Figura 3.18 foi possível elaborar o esquema elétrico da *main board* como é visto na Figura 3.19.

Note a grande simplicidade do esquema elétrico, trata-se das conexões das placas prontas com o módulo do *Rabbit*. Alguns detalhes como estabilizadores de tensão, LEDs indicativos de comunicação foram adicionados à placa. As duas grandes dificuldades da confecção desta placa foram: desenvolver o *foot print* do conector de 50 pinos do *Rabbit*, que são extremamente pequenas as distâncias entre os pinos, e, a localização geométrica na placa física dos terminais e furos, de tal maneira que se pudesse parafusar e encaixar as outras placas feitas na chamada placa principal.

O resultado é exibido na Figura 3.20 onde pode-se observar os comentários levantados acima, que também são vistos na placa já montada nas figuras 3.21 3.22. Note na Figura 3.22 que as placas puderam ser fixadas na placa principal, bem como o IMU.

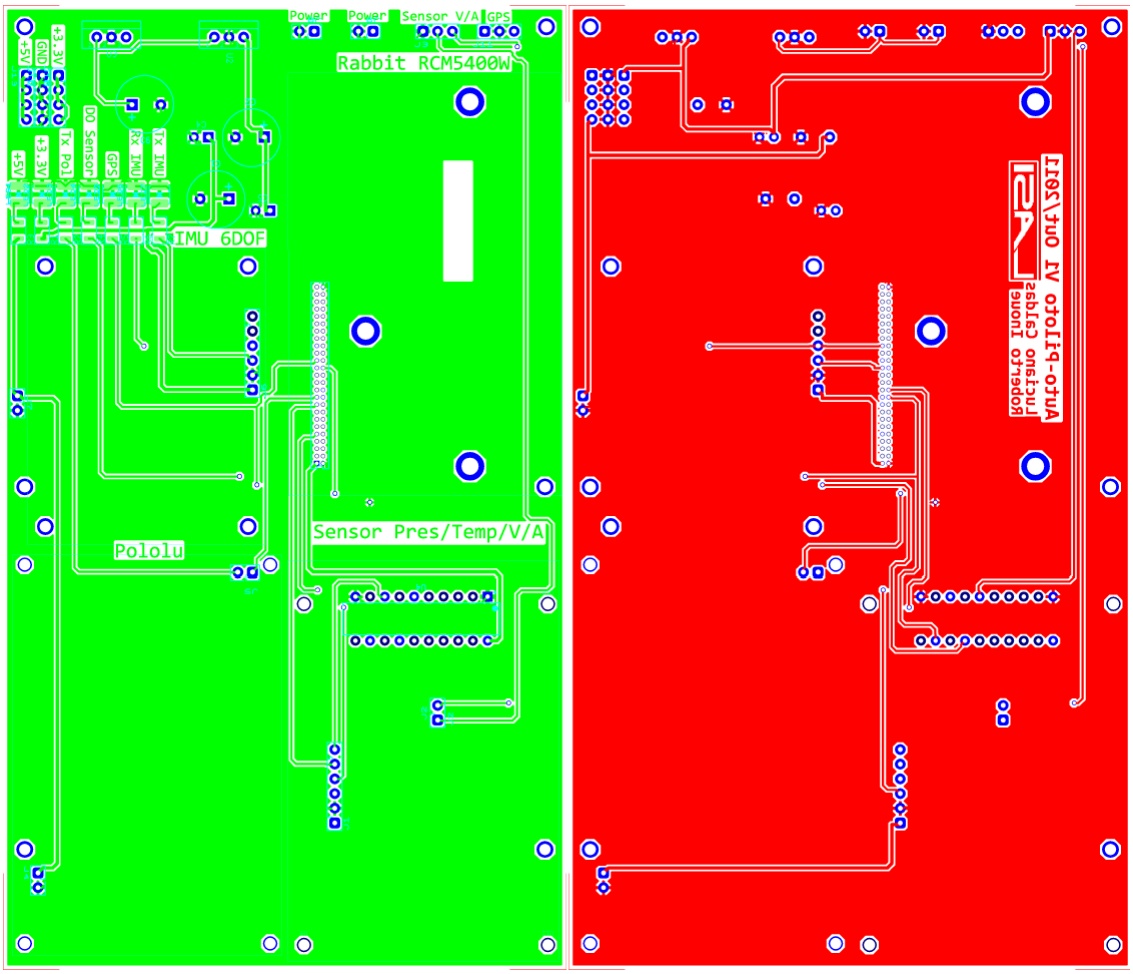


Figura 3.20: Esquema do piloto automático gerado a partir do esquema elétrico.

A Figura 3.23 mostra um bom resultado do teste de alimentação da placa.

Com alguns testes já realizados nesta placa, já foram levantados alguns problemas que levam à necessidade da confecção da segunda versão. Tais problemas já detectados podem ser citados:

- O uso do CI estabilizador de tensão $LM7833$ de 3,3V não suporta o consumo necessário de corrente, pois, devido à relativa alta tensão na alimentação (por volta de 11V), cerca de 70% da potência consumida é dissipada nele, na forma de calor, o que inviabiliza seu uso e se faz necessário o uso de um conversor DC-DC para alimentação do piloto automático.

- A posição do IMU na placa não está correta, pois, esta deve se localizar no ponto central da placa, de tal forma que fique centralizada com o eixo do

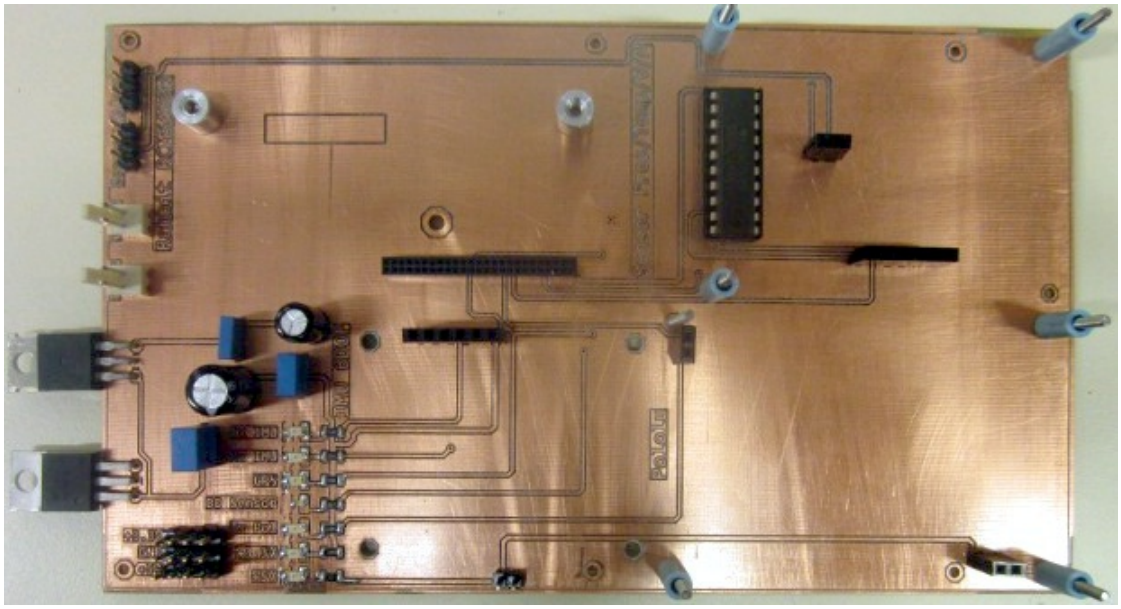


Figura 3.21: Vista de cima do piloto automático sem as placas externas.

rotor do helicóptero.

- Os LEDs indicativos do *Rabbit* foram ligados diretamente aos pinos dele, o que gerou problema de sobrecarga, levando assim à necessidade do uso de *Buffers* em todas as suas portas de comunicação.

- O dimensionamento da placa tem que ser revisto devido ao pouco espaço disponível na base do mini-helicóptero.

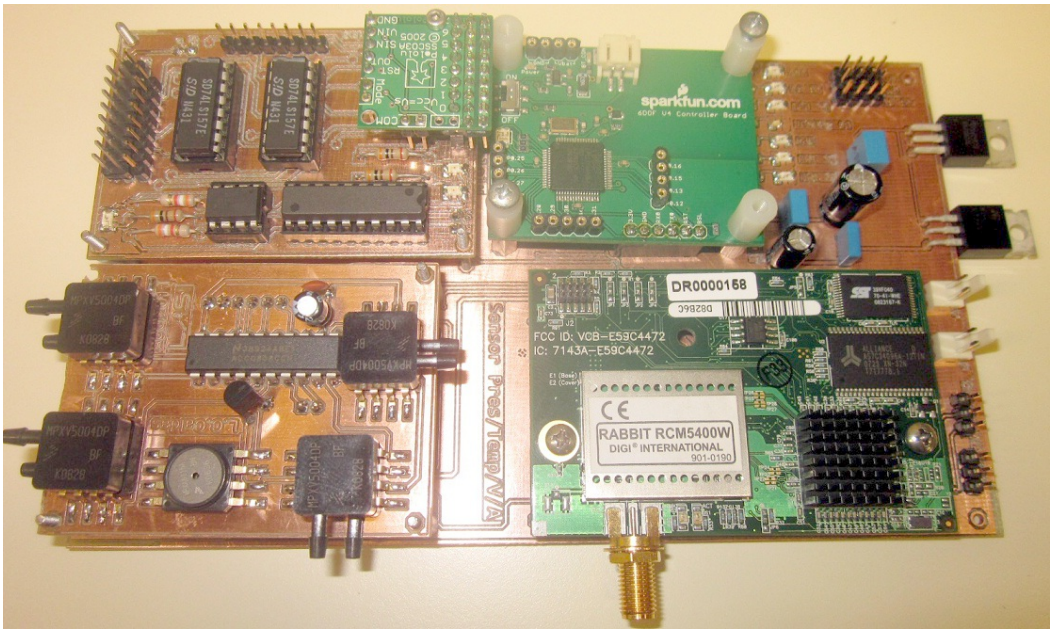


Figura 3.22: Piloto automático finalizado.

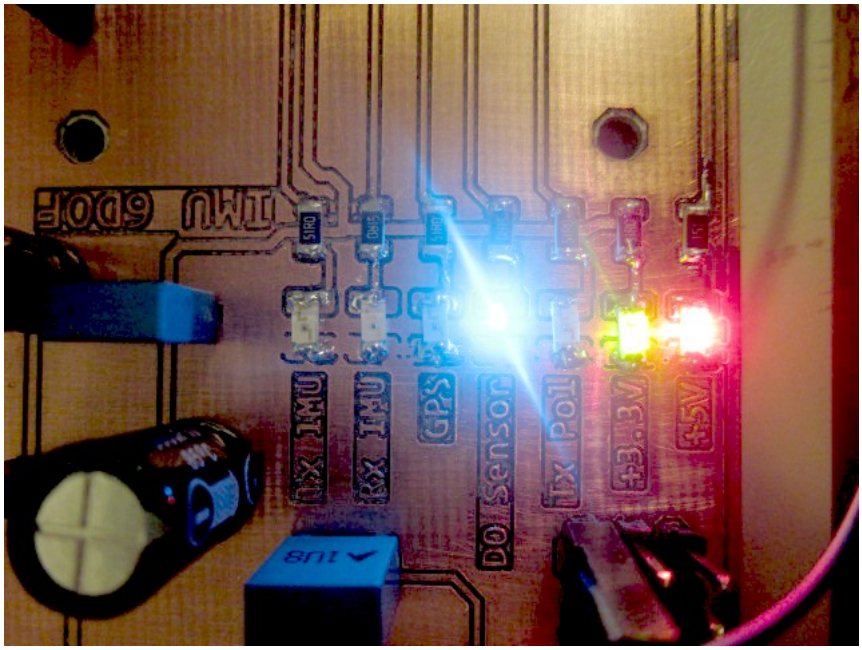


Figura 3.23: Teste da alimentaçao do circuito e dos LEDs.

Capítulo 4

Testes de bancada

Com o helicóptero montado (figuras 4.1 e 4.2) sobre a base de testes junto com toda instrumentação necessária foi possível, com um auxílio do Co-orientador do projeto, realizar alguns testes. Devido aos problemas apresentados com a placa principal do piloto automático, esta não pode ser usada nos testes.

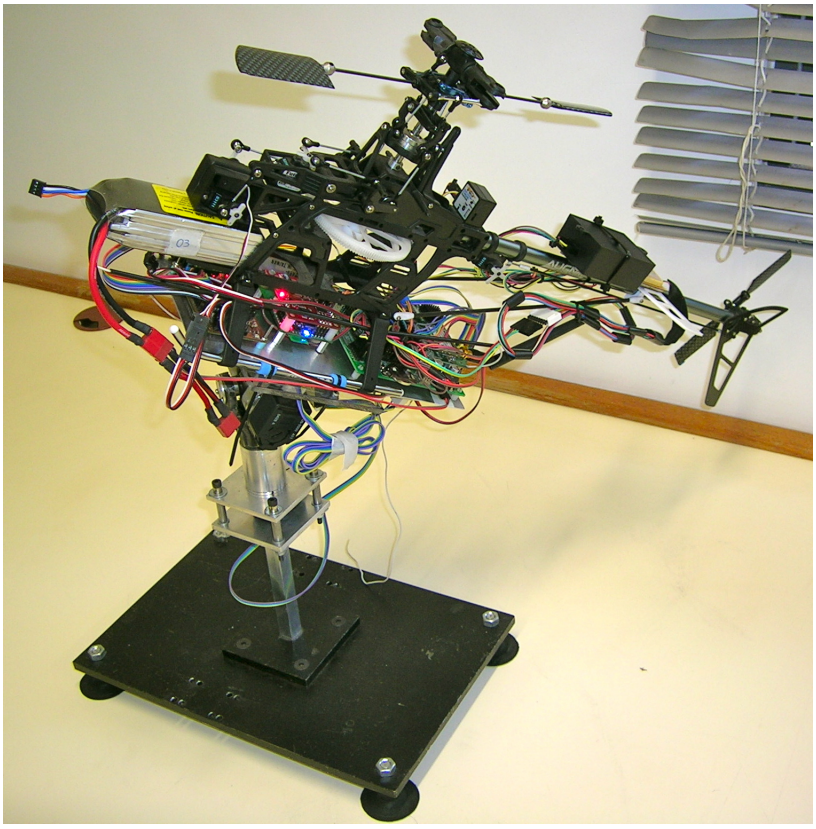


Figura 4.1: Montagem do sistema para testes.

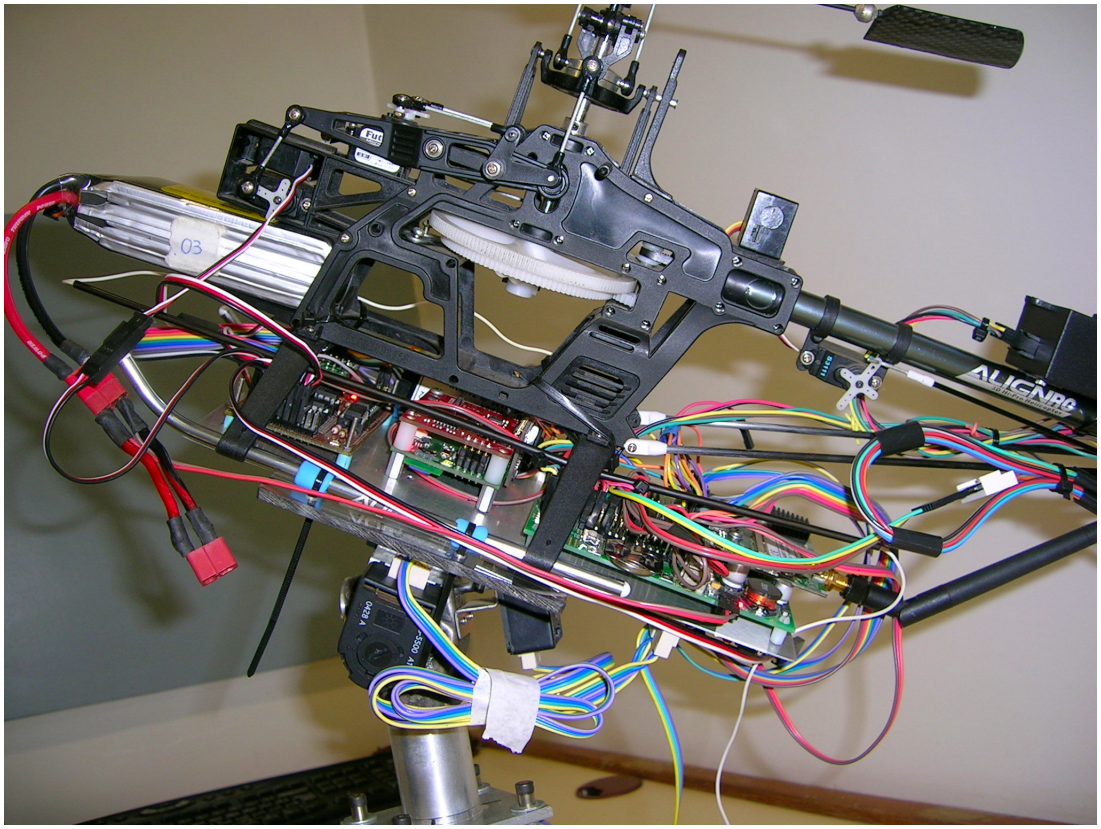


Figura 4.2: Vista mais detalhada da instrumentação do helicóptero.

Dessa forma, o circuito foi montado com auxílio mecânico de uma base para fixação de cada parte. Na Figura 4.1 a instrumentação presente é composta por: Rabbit RCM5400W para o processamento dos sinais dos sensores, criação do protocólo de comunicação e envio das informações via *Wi-Fi*, bem como o recebimento dos comandos; O circuito de controle dos servos composto pelo Pololu e o circuito de chaveamento entre rádio controle ou piloto automático; Circuito para leitura dos *encoders*; IMU 6DOF; GPS e Magnetômetro (ambos na cauda do helicóptero).

Infelizmente não foi possível realizar testes com o rotor funcionando devido à problemas apresentados com o *speed control*, que é o controlador e acionador do motor do rotor.

Foram testados o acionamento dos servos através de comandos fornecidos pelo computador (via MATLAB), junto com a leitura dos sensores dinâmicos que o compõe. As medidas realizadas com os acelerômetros, giroscópios e mag-

netômetros foram comparadas com a referência que é gerada pelos *encoders*.

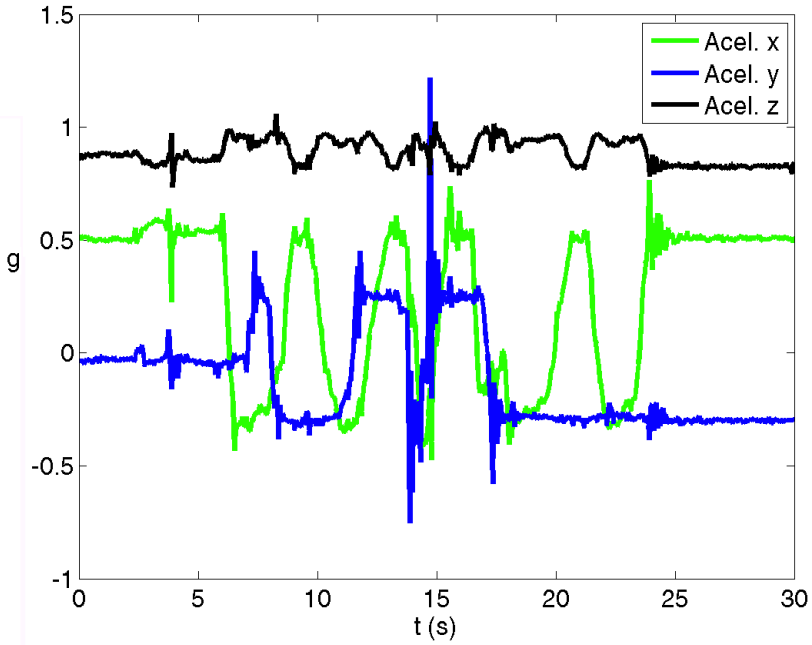


Figura 4.3: Gráfico dos valores lidos diretamente dos acelerômetros.

Os valores lidos diretamente nos sensores são exibidos nas figuras 4.3, 4.4 e 4.5.

Os valores então recebidos foram processados via MATLAB. Para estimar os ângulos através do sinal dos acelerômetros é necessário o uso das seguintes relações trigonométricas.

$$\phi = atan2\left(\frac{-a_y}{a_z}\right),$$

sendo a_y e a_z são as medidas dos acelerômetros nos eixos y e z respectivamente,

$$\theta = atan2\left(\frac{-a_x}{\sqrt{a_y^2 + a_z^2}}\right),$$

e a_x é o valor do acelerômetro no eixo x.

O trecho do código que realiza essa operação é descrito abaixo:

```
eangles.phi = atan2(-a_a(2),-a_a(3));
```

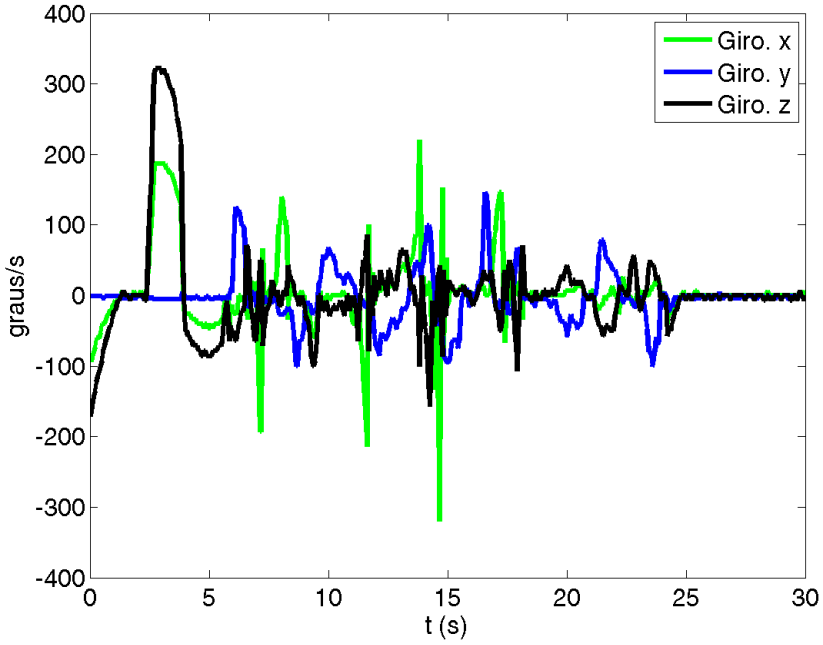


Figura 4.4: Gráfico dos valores lidos diretamente dos giroscópios.

```
eangles.theta = -atan2(-a_a(1),sqrt(a_a(2)^2 + a_a(3)^2));
```

sendo phi (ϕ) e theta (θ) os ângulos de arfagem obtidos a partir dos dados recebidos no vetor 'a_a'. O ângulo de guinada psi (ψ) é obtido através do valor lido do magnetômetro (densidade de fluxo magnético B) onde o processo é semelhante ao anterior, sua interpretação é feita seguindo o pseudo-código:

$$\psi = 0$$

$$\text{se } B_y > 0 \text{ , entao } \psi = \frac{\pi}{2} - atan\left(\frac{B_x}{B_y}\right)$$

$$\text{se } B_y < 0, \text{ entao } \psi = \pi + \frac{\pi}{2} - atan\left(\frac{B_x}{B_y}\right)$$

$$\text{se } B_y = 0 \text{ and } B_x < 0, \text{ entao } \psi = \pi$$

$$\text{se } B_y = 0 \text{ and } B_x > 0, \text{ entao } \psi = 0$$

e o código em MATLAB correspondente é dado por:

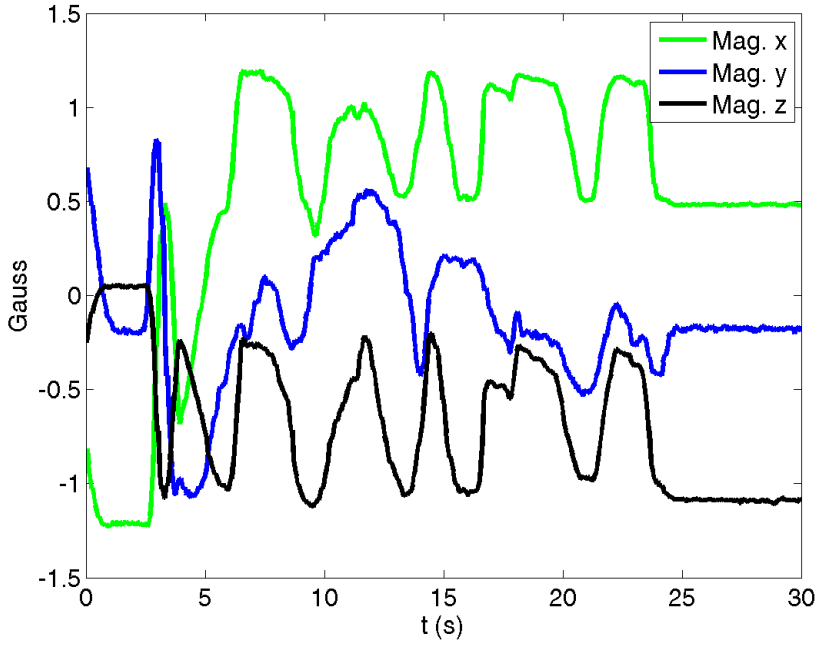


Figura 4.5: Gráfico dos valores lidos diretamente do magnetômetro.

```
angle = 0;
if m(2) > 0
      angle = pi/2 - atan(m(1)/m(2));
elseif m(2)<0
     angle = pi + pi/2 - atan(m(1)/m(2));
elseif m(2)==0 & m(1)<0
     angle = pi;
elseif m(2)==0 & m(1)>0
     angle = 0;
end
```

no qual o vetor 'm' contém os valores lidos do sensor. O código completo da leitura destes sensores é visto ao final do relatório nos Anexos.

Diante disso foi possível então colher dados de um pequeno teste realizado a fim de verificar o sistema. O teste baseou-se em colher e armazenar os dados num instante de tempo de cerca de 30 segundos. Nesse tempo alguns movimentos

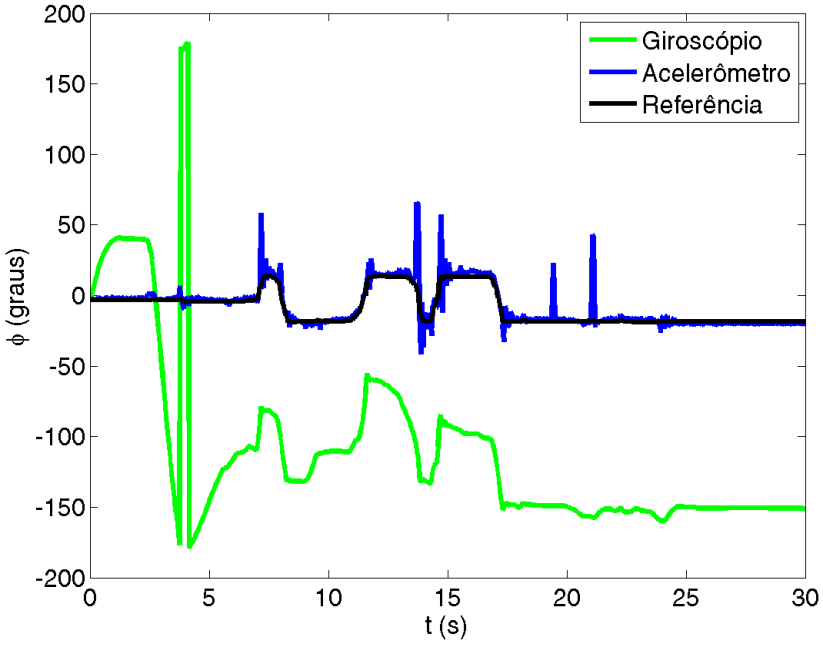


Figura 4.6: Gráfico do ângulo ϕ obtido no teste.

foram feitos manualmente sobre o helicóptero simulando movimentos reais do mesmo. As figuras 4.6, 4.7 e 4.8 exibem os resultados obtidos dos sinais lidos.

Como pode ser visto, o valor obtido através do giroscópio sofre sensivelmente a influência do erro de integração, fato acentuado na Figura 4.7. Os valores obtidos através dos acelerômetros possuem relativa fidelidade em relação à referência (valor lido dos *encoders*) porém apresenta bastante ruído na leitura. Percebe-se que estes dados compõem apenas um teste do sistema e não uma aquisição de dados científicos relevantes.

Por fim, através de uma interface criada no MATLAB foi possível além de conectar com o piloto automático, enviar comandos para os servos, ler os ângulos do sistema, e com esses valores, uma animação é vista exibindo a posição real do helicóptero na base. A Figura 4.9 exibe essa interface.

No lado direito da Figura 4.9 são vistas as opções de conectar, onde é selecionado IP do destino e logo abaixo a porta em que é feita a conexão. Logo acima, as barras de rolagem têm a função de mandar comandos para os servos, sendo assim possível controlá-los manualmente. Isto é útil pricipalmente para a

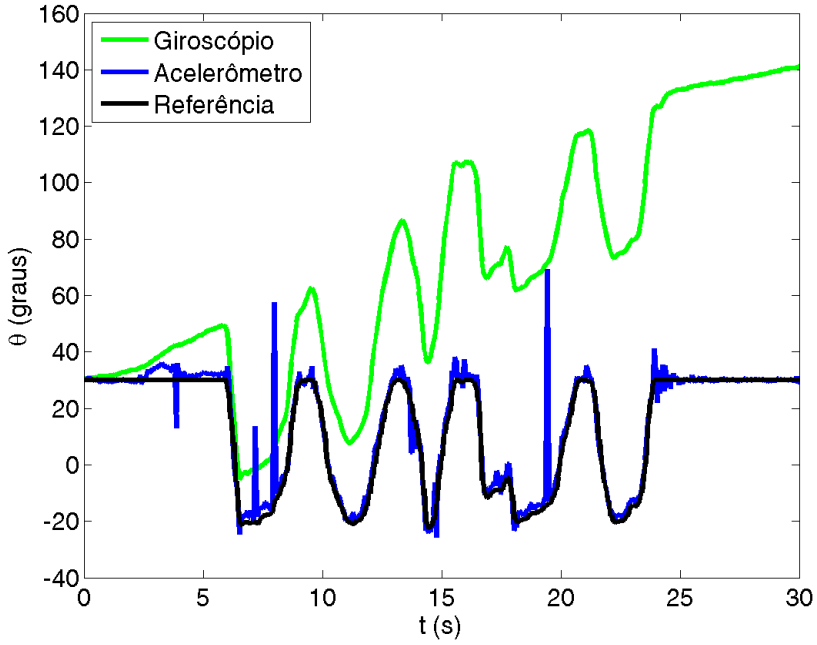


Figura 4.7: Gráfico do ângulo θ obtido no teste.

realização de testes. No lado esquerdo/acima é vista a animação do helicóptero em 3D, que retrata a posição em tempo real em que está o helicóptero na base. Abaixo é visto os ângulos de rolagem, arfagem e guinada.

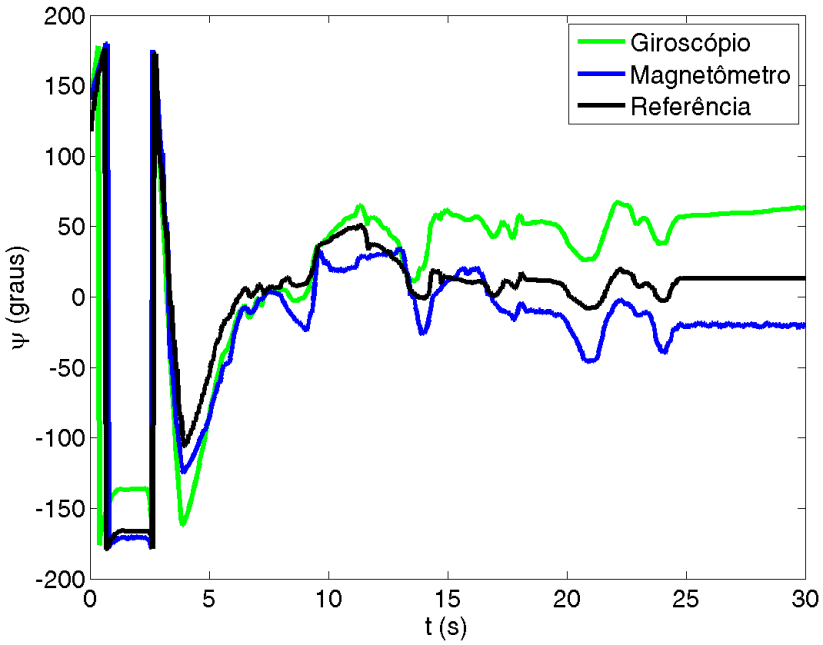


Figura 4.8: Gráfico do ângulo ψ obtido no teste.

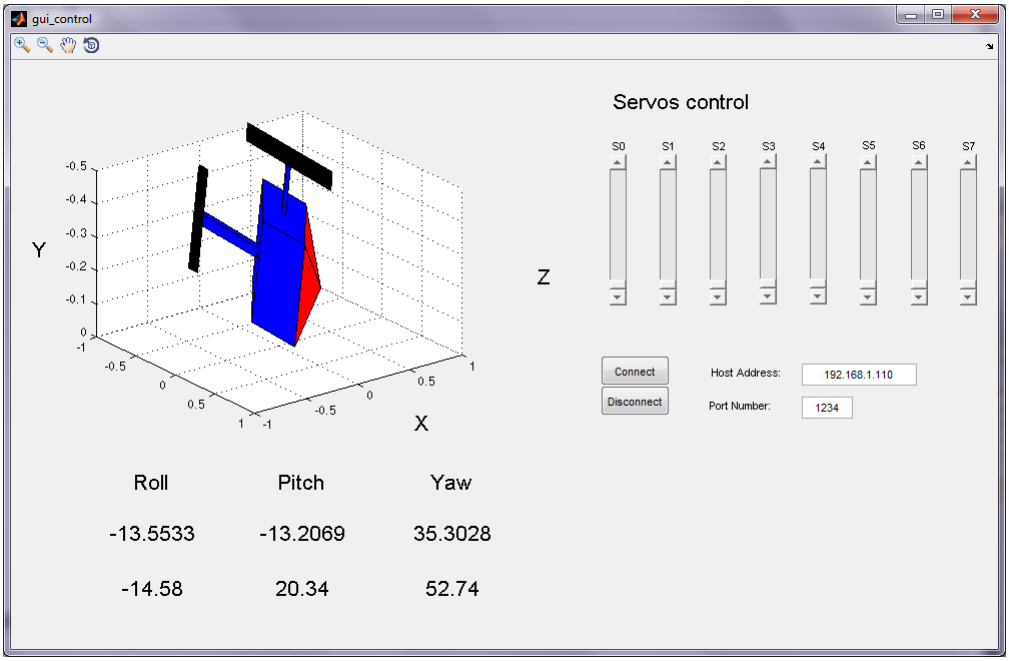


Figura 4.9: Interface gráfica criada no MATLAB para comunicação com o piloto automático.

Capítulo 5

Conclusão

No decorrer do projeto foram enfatizadas a familiarização com o ambiente de VANTs e sobretudo a confecção das placas que compõem o piloto automático. Na primeira parte foi dedicado tempo na familiarização com o ambiente de simulação bem como o kit de desenvolvimento *Rabbit*. Também foi dedicado tempo no estudo dos componentes usados no piloto automático. Em termos de resultados, foi desenvolvida a placa de leitura dos *encoders*, solucionado o problema de chaveamento dos comandos dos servos e a reformulação da plataforma 3D.

Na segunda parte do projeto o foco foi sobretudo na confecção das placas do piloto automático. Com o *software Multisim* da *National Instruments* foram desenvolvidas todas as placas que fazem parte do piloto automático: Controle dos servos; Sensores de pressão e temperatura com interface serial; Sensor de tensão e corrente; placa principal.

Por fim, os testes realizados serviram para mostrar o funcionamento do sistema, sobretudo com relação às medidas essenciais, ou seja, valores inerciais da IMU.

Capítulo 6

Trabalhos futuros

De fato há alguns trabalhos que devem ser finalizados para que possam ser realizados testes de voo, como por exemplo:

- Confecção da segunda versão da placa principal do piloto automático corrigindo os problemas relatados;
- Criar o protocolo para comunicação com a placa dos sensores de pressão para leitura serial dos valores dos sensores de tensão, corrente, temperatura e os 5 sensores de pressão;
- Estudo da viabilidade do uso dos tubos de *Pitot* no helicóptero. Caso viável, instalá-los e então conectá-los aos sensores de pressão relativa;
- Testes de estabilidade em bancada;
- Levantar o modelo do helicóptero em voo para uso no controle feito pelo autopiloto;

Estes são os principais tópicos que devem ser abordados nos trabalhos futuros. Com estes problemas resolvido deve-se então relizar testes exaustivos na base, para um posterior ensaio em voo. Um primeiro teste de controle que pode ser feito é o controle de cauda, que é o mais simples e baseia-se apenas em o piloto automático tentar controlar a posição da cauda em um determinado ângulo.

Com este resultado positivo, a próxima etapa possivelmente é uma tentativa de estabilização em todos os ângulos na base. Futuramente podem ser feitos ensaios em voo.

Referências Bibliográficas

BOGDANOV et al. (2004) BOGDANOV, A.; WAN, E.; HARVEY, G. (2004). *SDRE* Flight Control For X-Cell and *R*-Max Autonomous Helicopters. In *Proceedings of the 43rd IEEE Conference on Decision and Control, Atlantis, Paradise Island, Bahamas.*

INOUE et al. (2007) INOUE, R. S.; SIQUEIRA, A. A. G.; TERRA, M. H. (2007). Experimental Results on the Nonlinear $\mathcal{H}_\infty$ Control via Quasi-LPV Representation and Game Theory for Wheeled Mobile Robots. In *To appear in the proceedings of the IEEE Multi-conference on Systems and Control, Singapore.*

ISIDORI et al. (2003) ISIDORI, A.; MARCONI, L.; SERRAN, A. (2003). Robust Nonlinear Motion Control of a Helicopter. *IEEE Transactions on Automatic Control*, 48:413–426.

JANG & LICCARDO (2006) JANG, J. S.; LICCARDO, D. (2006). Automation of small UAVs using a low cost MEMS sensor and embedded computing platform. Technical report, Crossbow Technology, Inc., San Jose, California.

JUNG & TSIOTRAS (2007) JUNG, D.; TSIOTRAS, P. (2007). Inertial Attitude and Position Reference System Development for a Small UAV. In *American Institute of Aeronautics and Astronautics*, Rohnert Park, California.

SARIPALLI & SUKHATME (2007) SARIPALLI, S.; SUKHATME, G. S. (2007). Landing a Helicopter on a Moving Target. In *Proceedings of the IEEE International Conference on Robotics and Automation, Roma, Italy.*

Capítulo 7

Anexos

Código referente ao primeiro teste com apenas um *encoder*.

```
////////////////////////////////////////////////////////////////////////
%%%% CÓDIGO USADO LER 1 ENCODER %%%%

#define QD1_USEPORTD    /* Habilita o uso do decod. de quadratura do Rabbit */

void main()  {
  long reading1, aux1;
  qd_init(1);    /* Inicializa a leitura */
  qd_zero(2);
  reading1 = 0;
  while(1)  {
    costate  {
    aux1 = qd_read(1);  /* Faz a leitura do valor e armazena em aux1 */
    if(aux1 != reading1) reading1 = aux1;
        printf("qd1: %10ld\n", reading1);
        } }
} }

////////////////////////////////////////////////////////////////////////
%%%% CÓDIGO USADO PARA LEITURA DOS 3 ENCODERS %%%%

#define QD1_USEPORTD   /* Habilita o uso do decod. de quadratura 1 */
#define QD2_USEPORTD   /* Habilita o uso do decod. de quadratura 2 */

void DispStr(int x, int y, char *s)  // Função que escreve na tela
{
   x += 0x20;
   y += 0x20;
   printf ("\x1B=%c%c%s", x, y, s);
```

```
}

main()
{
   char display_01[100];
   char display_02[100];
   char display_03[100];
   int aux_enc_1, aux_enc_2, aux_enc_3, encoder[2];

   qd_init(1);  // Inicializa o decodificador 1
   qd_init(2);  // Inicializa o decodificador 2
   qd_zero(1);  // Zera o buffer do decodificador 1
   qd_zero(2);  // Zera o buffer do decodificador 2

   WrPortI(PEDDR,&PDDDRShadow,0xFF);     // Porta D definida como entrada e saída
   WrPortI(PEAHR,&PEAHRShadow,0x80);     // Porta E definida como entrada e saída
   BitWrPortI(PEDR, &PEDRShadow,0, 5);   // Seleciona a parte mais baixa (1) e mais alta (0) dos bytes
   BitWrPortI(PEDR, &PEDRShadow,0, 6);   // OE 0 leitura
   BitWrPortI(PEDR, &PEDRShadow,0, 6);   //
   BitWrPortI(PEDR, &PEDRShadow,1, 7);   // Reseta 0
   BitWrPortI(PEDR, &PEDRShadow,0, 7);   //

  // BoardInit();  // Config Rabbit
for(;;)
{
   costate{
      // Leitura dos encoders 1 e 2
      aux_enc_1 = qd_read(1);
      aux_enc_2 = qd_read(2);
      // Rotina para leitura do encoder 3 (Avago)
      BitWrPortI(PEDR, &PEDRShadow,1, 6);  // Inicializa a leitura
      BitWrPortI(PEDR, &PEDRShadow,0, 5);  // Seleciona a parte menos significativa dos bytes
      waitfor(DelayMs(1));                 // aguarda estabilizar o valor
      encoder[0] = RdPortI(PADR);          // armazena o valor
      BitWrPortI(PEDR, &PEDRShadow,1, 5);  // Seleciona a parte mais significativa dos bytes
      waitfor(DelayMs(1));                 // aguarda estabilizar o valor
      encoder[1] = RdPortI(PADR);          // armazena o valor
      encoder[1] = encoder[1] << 8;        // Desloca de 8 a parte mais alta para somar com a baixa dos bytes
      BitWrPortI(PEDR, &PEDRShadow,0, 6);  // Término da leitra
      aux_enc_3 = ~(encoder[1] ^ encoder[0]); // Junta a parte menos com a mais signif. para gerar a leitura
      sprintf(display_01, "Encoder 1: %d", aux_enc_1);
      sprintf(display_02, "Encoder 2: %d", aux_enc_2);
      sprintf(display_03, "Encoder 3: %d", aux_enc_3);
      DispStr(8, 6, "                                                  ");  // limpa tela
      DispStr(8, 7, "                                                  ");
      DispStr(8, 8, "                                                  ");
      DispStr(8, 6, display_01);  // mostra o valor lido no display do encoder 1
```

```
      DispStr(8, 7, display_02);  // mostra o valor lido no display do encoder 2
      DispStr(8, 8, display_03);  // mostra o valor lido no display do encoder 3
   }
   }   }

////////////////////////////////////////////////////////////////////////
%%%% CÓDIGO USADO PARA PROGRAMAR O MICROCONTROLADOR %%%%

 /* ------------------------------------------------------------------
 * Title: Switch Servos Control
 * Hardware: ATtiny13
 * Software: WinAVR 20060421
 ------------------------------------------------------------------*/
 #define F_CPU 1200000 // clock 1.2MHz, internal oscillator
 #include <avr/io.h>
 #include <util/delay.h>
 #include <avr/interrupt.h>

 #define OUT PB3 //PB3 = pin2 saida que ira controlar o multiplexador
 #define IN  PB4 //PB4 = pin3 sinal do receptor futaba
 #define _BV(bit) (1 << (bit))
 #define bit_is_set(sfr, bit) (_SFR_BYTE(sfr) & _BV(bit))
 #define bit_is_clear(sfr, bit) !(_SFR_BYTE(sfr) & _BV(bit))

 int main(void) {
DDRB |= (1 << OUT); // Set direction register output
DDRB &= ~(1<< IN); // Set direction register input


PORTB |= (1 << OUT); // Set 1 on LED pin
_delay_ms (1000);
PORTB &= ~(1 << OUT); // Set 0 on LED pin

while(1) {
if (bit_is_set(PINB,4)) { // testa o bit 4 da PORTA B
    _delay_us (1400); //aguarda 1,4ms para fazer um teste de bit
if (bit_is_clear(PINB,4)) {
    _delay_us (200); //se detectado nivel baixo, aguarda mais 0,2ms para evitar disparo por ruído
if (bit_is_clear(PINB,4)) {
PORTB |= (1 << OUT); // Set 1 on LED pin
  }
else;   }
else PORTB &= ~(1 << OUT); // Set 0 on LED pin
    _delay_us (1500);
else; }
  return 0; }
```

```
//////////////////////////////////////////////////////////////////////
%%%% CÓDIGO USADO PARA ENVIAR COMANDOS PARA O POLOLU (EM MATLAB) %%%%
%% FUNCAO CONECTA
if ~exist('pserial')
disp('Making serial.')
    pserial = serial(get(gui.polulo.edit_serial_porta,'String'),'baudrate',str2num(get(gui.polulo.edit_baudrate,'String')));

end

serial_status = pserial.Status;

if ~strcmp(serial_status, 'open')
    disp('Serial is open.')
    fopen(pserial)
else
    disp('Serial is already open.')
end

%% FUNCAO PARA DESCONECTAR
fclose(pserial)
disp('Serial is close.')
if strcmp(serial_status, 'close')
   clear pserial
   disp('Serial is clear.')
end

%% FUNCAO QUE CRIA O PROTOCOLO DA MENSAGEM
function set_position8(s,motor_number,position)

msg(1) = 128;
msg(2)=  1;
msg(3) = 3;
msg(4) = motor_number;
msg(5)= bitand(128,position)-1;
msg(6)= bitand(127,position);
fwrite(s,msg)

%% MAIN
gui.polulo= guidata(gui_polulo);

set(gui.polulo.pushbutton_serial_open,'Callback','pb_connect_serial');
set(gui.polulo.pushbutton_serial_close,'Callback','pb_disconnect_serial');
set(gui.polulo.slider0,'Callback','aciona_servo(0,pserial,gui)');
set(gui.polulo.slider1,'Callback','aciona_servo(1,pserial,gui)');
set(gui.polulo.slider2,'Callback','aciona_servo(2,pserial,gui)');
set(gui.polulo.slider3,'Callback','aciona_servo(3,pserial,gui)');
set(gui.polulo.slider4,'Callback','aciona_servo(4,pserial,gui)');
```

```
set(gui.polulo.slider5,'Callback','aciona_servo(5,pserial,gui)');
set(gui.polulo.slider6,'Callback','aciona_servo(6,pserial,gui)');
set(gui.polulo.slider7,'Callback','aciona_servo(7,pserial,gui)');

//////////////////////////////////////////////////////////////////////
%%%% CÓDIGO USADO PARA ENVIAR PROCESSAR OS DADOS DA IMU (EM MATLAB) %%%%

clear all
pathdef;
% load dados\data_IMU_9dofrazor_jacto.mat

% load jacto_rolagem.mat
% load jacto_arfagem.mat
% load jacto_guinada.mat
% load jacto_guinada_02.mat
% load jacto_guinada_03.mat
load data_luciano_20120207_2;
data = data_sensor;
%%%%%%%%%%%%%%%%%%%%%%%%%% Condições iniciais %%%%%%%%%%%%%%%%%%%%%%%%%%
% f = 15.53;                              % Hz
f = 17.8;                               % Hz
T = 1/f;                               % taxa de amostragem [s].
g = -9.81;                             % gravidade [m/s^2].
me = 1.455;
% tempo de correlação [s].
% tau_r = [2500 2066.11 661.157];
tau_r = [626.8115 6468.0515 602.5784]; % é equivalente ao tauB
% tau_r = [774.7493 88.6402 845.3541];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Cal. IMU 6DOF Red
adj.acel.x =  -485.7428;
adj.acel.y =  -522.2699;
adj.acel.z =  -511.7547;

adj.rate.x = -405.0094;
adj.rate.y = -420.1132;
adj.rate.z = -412;

adj.acel.kx = 1/189.8723;
adj.acel.ky = 1/190.4218;
adj.acel.kz = 1/189.4132;

adj.rate.kx = 1/0.15;
adj.rate.ky = 1/0.19;
adj.rate.kz = 1/0.14;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% Cal. MAG. PNI
adj.mag.x = -13;
adj.mag.y = 7;
adj.mag.z = -21.5;

adj.mag.kx = 1/163;
adj.mag.ky = 1/183;
adj.mag.kz = 1/154.5;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
% Ponderações
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% tempo de correlação [s].
tau_r = [626.8115 6468.0515 602.5784]; % é equivalente ao tauB
tau_r = [2500 2066.11 661.157];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
adj = adj_IMU_9dofrazor_jacto;
%%%%%%%%%%%%%%%%%%
% Ponderações
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sigma1 = sqrt([0.014058; 0.01535; 0.016099; 0.00028462; 0.00023844; 0.00025606])*pi/180;
sigma1 = sqrt([0.34644; 0.2384; 0.36668; 100; 50; 0.87167])*pi/180;
% Ponderações do sistema de referência de atitute e proa
R1 = zeros(6,6);
R1(4,4) = 0.00059539;    %variancia do acelerometro
R1(5,5) = 0.00062077;     %variancia do acelerometro
R1(6,6) = 0.001261;    %variancia do acelerometro
R1(1,1) = 0.0000069877;   %variancia do magnetometro
R1(2,2) = 0.000010179;  %variancia do magnetometro
R1(3,3) = 0.0000098957;  %variancia do magnetometro
%%%%%%%%%%%%%%%%%%%
R1(4,4) = 0.0075685;
R1(5,5) = 0.0058868;
R1(6,6) = 0.0045295;
R1(1,1) = 0.000098;
R1(2,2) = 0.000098;
R1(3,3) = 0.000098;

% Condições iniciais
contador  = 0;   % Contagem das vezes que o corpo acelera
%P1 = 10^-3*eye(7);
load P1_Kalman_9dofrazor
%load P1_robust
v1 = zeros(9,1); % Ruído estimado
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bg = zeros(3,1);  % Bias do giroscópio
ba = zeros(3,1);  % Bias do acelerômetro
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sys_a.alfa = 1;
range = 500;  % Range do plot
opt = 0;
k = 0;
tic
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n = size(data.scaled.acel.x,2);
% n=150
t_v = (1:n)*T;
r2d = 180/pi;
d2r = pi/180;
for k =1:n
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     k = k+1;
%     msg = fread(t)';
%     data =  read_IMU_9dofrazor(char(msg),adj);
%     enc_pos = strfind(msg,'UU');
%     data.encoders = read_encoders_v4(msg(enc_pos(1):enc_pos(1)+15));

    %% Aceleração normalizada
    a_a_norm(1,1) = data.scaled.acel.x(k);
    a_a_norm(2,1) = -data.scaled.acel.y(k);
    a_a_norm(3,1) = -data.scaled.acel.z(k);
    %% Aceleração em m/s^2
    a_a = a_a_norm*9.81;
    %% Aceleração corrigida pelo bias
    a = a_a;% - ba;       % Correção da aceleração pelo bias
    %% Velocidade angular
    if abs(data.scaled.rate.x(k)) > 1000
        data.scaled.rate.x(k) = data.scaled.rate.x(k-1)
    end
    if abs(data.scaled.rate.y(k)) > 1000
        data.scaled.rate.y(k) = data.scaled.rate.y(k-1)
    end
    if abs(data.scaled.rate.z(k)) > 1000
        data.scaled.rate.z(k) = data.scaled.rate.z(k-1)
    end
    omega_g = [ -data.scaled.rate.x(k); -data.scaled.rate.y(k); -data.scaled.rate.z(k)]*d2r;
    %% Campo magnético
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % z y
    % Mudança do sistema de coordenada de x _|/ para  x _
                                                      % |\
                                                      % z y
    m = [data.scaled.mag.x(k); -data.scaled.mag.y(k); -data.scaled.mag.z(k)];
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%     m_v(:,k) = m;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Verificação da aceleração do corpo
    mu = norm(a_a) + g;
    if k > 20
        mu_v(1:19) = mu_v(2:20);
        mu_v(20) = mu;
    else
        mu_v(k) = mu;
    end

    u = norm(omega_g);
    u_v(k) = u;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Cálculo dos ângulos de euler baseado nos sensores
    %% Ângulo de guinada obtido pelo magnetômetro
    eangles.psi     = -headingangle(m);              % yaw angle
    %% Ângulo de arfagem obtido pelo acelerômetro
    eangles.phi = atan2(-a_a(2),-a_a(3));
    eangles.theta = -atan2(-a_a(1),sqrt(a_a(2)^2 + a_a(3)^2));

    psi_mag_v(k) = normalize_angle(eangles.psi,-pi);
    phi_acel_v(k) = normalize_angle(eangles.phi,-pi);
    theta_acel_v(k) = normalize_angle(eangles.theta,-pi);

   %% Primeira iteração
    if k == 1
        % Obtenção da primeira estimativa dos ângulos de euler do corpo
        % através dos sensores
        qhat = euler2quat(eangles);
        eangleshat = eangles;
        % Transformação dos ângulos de euler de radianos para graus
        psi_hat =  eangles.psi;
        theta_hat = eangles.theta;
        phi_hat = eangles.phi;
%         [psi_hat, theta_hat, phi_hat] = r2d_f(eangleshat);
        % Definição do estado do sistema de atitude
        x1 = [qhat; bg];%; ba];
        if opt == 2
            sys_a.H = Hquat_m(qhat,g,me,1);
            P1 = (P1^(-1) + sys_a.H'*R1^(-1)*sys_a.H)^(-1);
        end
        %                         if opt == 1
        %                             H = Hquat_m(qhat,g,me,1);
        %                             L = eye(7);
        %                             S = eye(7);
        %                             S_bar = L'*S*L;
```

```
    %                               %des = P1^-1 - sys_a.gamma^(-2)*L'*L;
    %                               des = P1^-1 - sys_a.theta*S_bar + sys_a.H'*inv(sys_a.R)*sys_a.H;
    %                               eig_des = eig(des);
    %                               for cont_eig = 1:size(eig_des,1)
    %                                   if ~(eig_des(cont_eig) > 0)
    %                                       disp('Condição não satisfeita do filtro H Infinito inicio')
    %                                       pause
    %                                   end
    %                               end
    %                           end


    % Define a condição inicial da integração numérica do giroscópio
    phi_gyro = phi_hat;
    psi_gyro = psi_hat;
    theta_gyro = theta_hat;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    phi_gyro_v(k) = normalize_angle(phi_gyro,-pi);
    theta_gyro_v(k) = normalize_angle(theta_gyro,-pi);
    psi_gyro_v(k) = normalize_angle(psi_gyro,-pi);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    phi_enc_v(k) = normalize_angle(data.angle_rd_01(k),-pi);
    theta_enc_v(k) = normalize_angle(-data.angle_rd_02(k) ,-pi);
    psi_enc_v(k) = normalize_angle(data.angle_rd_03(k) ,-pi);

    % Define a condição inicial do estado para a integração numérica a
    % partir da predição
    xp1_2 = x1; % Para fazer a integração
    % Define a condição inicial da integração numérica a partir da
    % predição
    phi_pred(k) = phi_hat(k);
    psi_pred(k) = psi_hat(k);
    theta_pred(k) = theta_hat(k);
else
    % mu_f = filtro_media(mu, 20, 1);
    mu_f = mean(mu_v);
    mu_f_v(k) = mu_f;
    %% Parâmetros de ajuste da verificação da aceleração do corpo
    beta_mu =  2;   %4;     %1;    %2;   %1;
    beta_muf = 1.2; %2.4;   %0.6;  %1.2; %0.6;
    beta_g = 4;     %6      %2;    %4;   %2;
    %% Verifica se o corpo está acelerando
    cond(1,k) = mu < beta_mu;
    cond(2,k) = u < beta_g ;
    cond(3,k) = mu_f < beta_muf;
    cond(4,k) = norm(a_a_norm(1)) <= 1;
```

```
cond(5,k) = norm(a_a_norm(2)) <= 1;
state_sensor =  (cond(1,k) && cond(2,k)  && cond(3,k));   % && cond(4,k)&& cond(5,k))
state_sensor_v(k) = state_sensor;
  state_sensor = 1;   % Sem verificação de aceleração do corpo
if state_sensor == 1
    % Ajusta a ponderação R se o corpo estiver acelerando
    R1_2 = R1;
    R1_2(4:6,4:6) = R1(4:6,4:6) + eye(3,3)*(0.1*mu)^2;
else
    R1_2 = R1;
end
%%%%%%%%%%%%%%%%%% Sistema de referência de proa %%%%%%%%%%%%%%%%%%%%%%
% Correção da velocidade angular do corpo
omegahat =  omega_g - bg;
Rothat =  rot(qhat);
[sys_a] = sys_ati(sys_a,omegahat, qhat, sigma1, R1_2, g,me, tau_r, T, state_sensor); % Matrizes do sistema
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Observação %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if opt == 0
    [xp1, Pp1] = ekf_p (x1,P1,sys_a);           % Predição
end
if opt == 1
    [xp1] = ekfHInfty_p (x1,sys_a);             % Predição
end
if opt == 2
    [xp1, Pp1] = erkfsayed_p(x1, P1, sys_a);    % Predição Robusto
end
if opt == 3
    xp1 = sys_a.Phi*x1 + sys_a.Gd*v1(1:6);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xp1_2 = sys_a.Phi*xp1_2;
eangles_pred   = quat2euler(xp1_2);
% Transformação dos ângulos de euler da predição de radianos para graus
[psi_pred(k), theta_pred(k), phi_pred(k)] = r2d_f(eangles_pred);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xp1(1:4) = xp1(1:4)/norm(xp1(1:4)); % Normalização do estado
qhatp = xp1(1:4);
Rothat = rot(qhatp);
switch state_sensor
    case 1
        z1  = [m;a];
        hhat1 = [me*Rothat(1,1); me*Rothat(2,1); me*Rothat(3,1); g*Rothat(1,3); g*Rothat(2,3); g*Rothat(3,3)];
    case 0
        contador = contador + 1;
        z1  = m;
        hhat1 = [me*Rothat(1,1); me*Rothat(2,1); me*Rothat(3,1)];
end
```

```
        sys_a.H = Hquat_m(qhatp,g,me,state_sensor);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Erro de observação da atitude e de proa
        deltaz1 = z1 - hhat1;
        deltaz1
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if opt == 0
            [x1,P1] = ekf_u(xp1, Pp1, deltaz1, sys_a,1);       % Filtragem
        end
        if opt == 1
            [x1,P1] = ekfHInfty_u2(x1, P1, deltaz1, sys_a);    % Filtragem
            % [x1,P1] = ekfHInfty_u_hassibi(x1, P1, deltaz1, sys_a);    % Filtragem
        end
        if opt == 2
            [x1,P1] = erkfsayed_u(xp1, Pp1, deltaz1, sys_a);   % Filtragem Robusta
        end
        if opt == 3
            [x1,v1,P1] = erkfbianco_u3(xp1, P1, sys_a, deltaz1);
        end
%         if state_sensor ==0
%             x1=xp1;
%         end
        x1(1:4) = x1(1:4)/norm(x1(1:4));
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Bias do giroscópio estimada
        bg   = x1(5:7);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        qhat = x1(1:4);
        eangleshat   = quat2euler(qhat);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        qhat = x1(1:4);
        eangleshat   = quat2euler(qhat);
        psi_hat = normalize_angle(eangleshat.psi, -pi);
        theta_hat = normalize_angle(eangleshat.theta, -pi);
        phi_hat = normalize_angle(eangleshat.phi, -pi);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        psi_hat_v(k)   = psi_hat;
        theta_hat_v(k) = theta_hat;
        phi_hat_v(k)   = phi_hat;
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if k >1
            phi_gyro = (omega_g(1) )*T + phi_gyro;
            theta_gyro = (omega_g(2))*T + theta_gyro;
            psi_gyro = (omega_g(3))*T + psi_gyro;
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        phi_gyro_v(k) = normalize_angle(phi_gyro,-pi);
```

```
        theta_gyro_v(k) = normalize_angle(theta_gyro,-pi);
        psi_gyro_v(k) = normalize_angle(psi_gyro,-pi);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        phi_enc_v(k) = normalize_angle(data.angle_rd_01(k),-pi);
        theta_enc_v(k) = normalize_angle(-data.angle_rd_02(k) ,-pi);
        psi_enc_v(k) = normalize_angle(data.angle_rd_03(k) ,-pi);


    end
end
t = toc;
r2d = 180/pi;
d2r = pi/180;
% encoders_v = normalize_angle(encoders_v,-pi)




figure(1)
axes1 = axes('Parent',1,'FontSize',16);
erro = normalize_angle(phi_enc_v(10) - phi_acel_v(10),-pi)
plot_01 = plot(axes1,t_v,phi_gyro_v*r2d,'g',t_v,phi_acel_v*r2d,'b',t_v,normalize_angle((phi_enc_v - erro),-pi)*r2d,'k','LineWidth
ylabel('\phi (graus)')
xlabel('t (s)')
legend('Giroscópio','Acelerômetro','Referência')
% hold on
% plot(t_v,state_sensor_v*50,'b')
% plot(t_v,cond(1,:)*10,'g')
% plot(t_v,cond(2,:)*20,'r')
% plot(t_v,cond(3,:)*30,'k')
% hold off

figure(2)
axes2 = axes('Parent',2,'FontSize',16);
erro = normalize_angle(theta_enc_v(10) - theta_acel_v(10),-pi)
% plot(t_v,theta_gyro_v*r2d,'g',t_v,theta_acel_v*r2d,'b',t_v,theta_hat_v*r2d,'r',t_v,normalize_angle((theta_enc_v - erro),-pi) *
plot_02 =plot(axes2,t_v,theta_gyro_v*r2d,'g',t_v,theta_acel_v*r2d,'b',t_v,normalize_angle((theta_enc_v - erro),-pi) *r2d,'k','Lin
ylabel('\theta (graus)')
xlabel('t (s)')
legend('Giroscópio','Acelerômetro','Referência','Location','NorthWest')
% hold on
% plot(t_v,state_sensor_v*50,'b')
% plot(t_v,cond(1,:)*10,'g')
% plot(t_v,cond(2,:)*20,'r')
% plot(t_v,cond(3,:)*30,'k')
% hold off
```

```
figure(3)
axes3 = axes('Parent',3,'FontSize',16);
erro = normalize_angle(psi_enc_v(10) - psi_mag_v(10),-pi)
% plot(t_v,psi_gyro_v*r2d,'g',t_v,psi_mag_v*r2d,'b',t_v,psi_hat_v*r2d,'r',t_v,normalize_angle((psi_enc_v - erro),-
plot_03 =plot(axes3,t_v,psi_gyro_v*r2d,'g',t_v,psi_mag_v*r2d,'b',t_v,normalize_angle((psi_enc_v - erro),-pi)*r2d,'
ylabel('\psi (graus)')
xlabel('t (s)')
title('\psi')
legend('Giroscópio','Magnetômetro','Referência')
% hold on
% plot(t_v,state_sensor_v*50,'b')
% plot(t_v,cond(1,:)*10,'g')
% plot(t_v,cond(2,:)*20,'r')
% plot(t_v,cond(3,:)*30,'k')
% hold off
cd figuraslu
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filename = 'phi_lu'
tipo = 1;
saveas(tipo,filename, 'psc2')
saveas(tipo,filename, 'fig')
saveas(tipo,filename, 'pdf')
saveas(tipo,filename, 'png')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filename = 'theta_lu'
tipo = 2;
saveas(tipo,filename, 'psc2')
saveas(tipo,filename, 'fig')
saveas(tipo,filename, 'pdf')
saveas(tipo,filename, 'png')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filename = 'psi_lu'
tipo = 3;
saveas(tipo,filename, 'psc2')
saveas(tipo,filename, 'fig')
saveas(tipo,filename, 'pdf')
saveas(tipo,filename, 'png')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% if isvalid(s)
```

```
%     fclose(s1);
% end



% t = toc;
% r2d = 180/pi;
% d2r = pi/180;
% encoders_v = normalize_angle(encoders_v,-pi)
%
%
% figure(1)
% erro = normalize_angle(-encoders_v(200) - psi_hat_v(200),-pi)
% plot(t_v,psi_hat_v*r2d,'r',t_v,normalize_angle((-encoders_v-erro),-pi)*r2d,'k')
% ylabel('\psi (graus)')
% xlabel('t (s)')
% title('\psi')
%
% figure(2)
% erro = normalize_angle(encoders_v(2) - phi_hat_v(2),-pi)
% plot(t_v,phi_hat_v*r2d,'r',t_v, normalize_angle(encoders_v-erro,-pi)*r2d,'k')
% ylabel('\phi (graus)')
% xlabel('t (s)')
%
% figure(3)
% erro = normalize_angle(encoders_v(200) - theta_hat_v(200),-pi)
% plot(t_v,theta_hat_v*r2d,'r',t_v, normalize_angle(encoders_v-erro,-pi)*r2d,'k')
% ylabel('\theta (graus)')
% xlabel('t (s)')
%
% % if isvalid(s)
% %     fclose(s1);
% % end

////////////////////////////////////////////////////////////////////////
// Código usado para programacao do piloto automático (rabbit RCM5400W)

#define TCPCONFIG               5

#define _PRIMARY_STATIC_IP      "192.168.1.110"
#define _PRIMARY_NETMASK        "255.255.255.0"
#define MY_GATEWAY              "10.10.6.1"
#define MY_NAMESERVER           "10.10.6.1"
#define IFC_WIFI_SSID           "LASI"
#define IFC_WIFI_ROAM_ENABLE   1
#define IFC_WIFI_ROAM_BEACON_MISS  20
#define IFC_WIFI_MODE           IFPARAM_WIFI_INFRASTRUCTURE
```

```
#define IFC_WIFI_REGION          IFPARAM_WIFI_REGION_AMERICAS
//#define IFC_WIFI_ENCRYPTION IFPARAM_WIFI_ENCR_TKIP
//#define IFC_WIFI_WPA_PSK_HEXSTR \ "1234567890"
#define IFC_WIFI_ENCRYPTION      IFPARAM_WIFI_ENCR_WEP
#define IFC_WIFI_WEP_KEYNUM      0
#define IFC_WIFI_WEP_KEY0_HEXSTR "1234567890"

#memmap xmem
#use "dcrtcp.lib"

#use "UAV_CONFIG_9DOFRAZOR_3seriais.lib"
#use "servos_pololu.lib"
#use "IMU.LIB"
#use "gps_mod.LIB"

#define BUFFER_LEN 32
#define MSG_LEN 1024 //in Hz
#define Min_NMEA 40


#use "kalman_filter.lib"
#use "libmat.lib"
#use "matrices_a.lib"
#use "matrices_p.lib"
// Encoder Rabbit
#define QD1_USEPORTD
#define QD2_USEPORTD

tcp_Socket sock;
GPSPosition pos, pos_ant, pos_IMU;
NEDFrame    vel, vel_ant, vel_IMU;
ENUFrame vel_enu;
GPSDop dop;
struct tm ntime;
char sentence[MSG_LEN], *p_sen;
char buffer[BUFFER_LEN];
char *motor_number, *position, *position_end;
unsigned char msg_IMU[128];
unsigned char msg_send[90];
unsigned long  time, time0;


int bytes;
int cont;
int nb, nb_mag;
int checksum;
```

```
int IMU_OK, MAG_OK;
int GPS_OK, VEL_OK, POS_OK, DOP_OK, TM_OK;
int initIMU_OK;
int initGPS_OK;
int restart_initGPS;
int restart_initIMU;
int mem;

float dlambda, dphi;

char buffer_imu[128],buffer_mag[128], *pos_01, *pos_02,*pos_imu, *pos_mag;
unsigned char msg_send_imu_gps[128];
// ****************************************************************************
char display_01[100];
char display_02[100];
char display_03[100];
char display_04[100];
int aux_enc_1, aux_enc_2, aux_enc_3, encoder[2];
// ****************************************************************************
void DispStr(int x, int y, char *s)
{
   x += 0x20;
   y += 0x20;
   printf ("\x1B=%c%c%s", x, y, s);
}
// ****************************************************************************
void reset_IMU() {
   BitWrPortI(PBDR, &PBDRShadow, 0, 4);
   msDelay(100);
BitWrPortI(PBDR, &PBDRShadow, 1, 4);
}

main()
{
// ****************************************************************************
   qd_init(1);
   qd_init(2);
qd_zero(1);
   qd_zero(2);

   WrPortI(PEDDR,&PDDDRShadow,0xFF);     // Port D Input and output Pins
   WrPortI(PEAHR,&PEAHRShadow,0x80);     // Port E Input and output Pins
   BitWrPortI(PEDR, &PEDRShadow,0, 5);   // SEL LOW (1) and HIGH (0) bytes
   BitWrPortI(PEDR, &PEDRShadow,0, 6);   // OE 0 to read
   BitWrPortI(PEDR, &PEDRShadow,0, 6);   //
   BitWrPortI(PEDR, &PEDRShadow,1, 7);   // Reset 0
   BitWrPortI(PEDR, &PEDRShadow,0, 7);   //
```

```
// ****************************************************************************
pos_imu = &buffer_imu[0];
   pos_mag = &buffer_mag[0];
// ****************************************************************************
   initIMU_OK = 0;
   initGPS_OK = 0;
   IMU_OK = 0;
   MAG_OK = 0;
   GPS_OK = 0;
   POS_OK = 0;
   VEL_OK = 0;
   DOP_OK = 0;
   TM_OK = 0;
   restart_initGPS = 0;
   restart_initIMU = 0;
   mem = 1;
   nb_mag = 0;
// ****************************************************************************
   GPS_init(&pos, &pos_ant, &vel, &vel_ant, &dop, &ntime);
// ****************************************************************************
  BoardInit();  // Config Rabbit
   adj_IMU(&IMU);
   reset_servos();
   reset_IMU();
// ****************************************************************************
sock_init_or_exit(1);
tcp_listen(&sock, 1234, 0, 0, NULL, 0);
// ****************************************************************************
   //while (!sock_established(&sock)) {
  // tcp_tick(NULL);
   //}
// ****************************************************************************
  time  = MS_TIMER;

// Setting PNI MAG

// ****************************************************************************
mem = 1;
while (mem)
{
      serCwrite("go\n", sizeof("go\n"));
    if ((nb = serCread(sentence,127,20)) > 0)
    {
       sentence[nb] = '\0';
        //DispStr(8, 5, "                                                  ");
        //DispStr(8, 5, sentence);
```

```
         pos_01 = _n_strstr( sentence, "$raw" );
         if(strncmp(pos_01, "$raw", 4) == 0)
         {
          mem = 0;
          //DispStr(8, 8, "                                                  ");
       //DispStr(8, 8, "$raw");
         }
      }
}
// ****************************************************************************
  while(1)
  {
// ****************************************************************************
   time  = MS_TIMER;
      costate
      {
  if  (!sock_established(&sock))
        {
tcp_listen(&sock, 1234, 0, 0, NULL, 0);
          while(!sock_established(&sock))
            {
tcp_tick(NULL);
             waitfor(DelayMs(20));
          }
       }
      }
time  = MS_TIMER;
// ****************************************************************************
costate{

      // Reading encoder 1 and 2
      aux_enc_1 = qd_read(1);
aux_enc_2 = qd_read(2);
      // Reading encoder 3 (Avago)
      BitWrPortI(PEDR, &PEDRShadow,1, 6);  // Start reading
      BitWrPortI(PEDR, &PEDRShadow,0, 5);  // Sel Low byte
      waitfor(DelayMs(1));
    encoder[0] = RdPortI(PADR);
      BitWrPortI(PEDR, &PEDRShadow,1, 5); // Sel High byte
      waitfor(DelayMs(1));
      encoder[1] = RdPortI(PADR);
encoder[1] = encoder[1] << 8;
      BitWrPortI(PEDR, &PEDRShadow,0, 6);  // End reading
      aux_enc_3 = ~(encoder[1] ^ encoder[0]);

// ****************************************************************************
/*      msg_send[0] = 'U';
```

```
      msg_send[1] = 'U';
      msg_send[5] =  (0x000000FF & ((long int) (aux_enc_1) ) );         // Encoder 1
      msg_send[4] =  (0x0000FF00 & ((long int) (aux_enc_1) ) ) >> 8;
      msg_send[3] =  (0x00FF0000 & ((long int) (aux_enc_1) ) ) >> 16;
      msg_send[2] =  (0xFF000000 & ((long int) (aux_enc_1) ) ) >> 24;
      msg_send[9] =  (0x000000FF & ((long int) (aux_enc_2) ) );         // Encoder 2
      msg_send[8] =  (0x0000FF00 & ((long int) (aux_enc_2) ) ) >> 8;
      msg_send[7] =  (0x00FF0000 & ((long int) (aux_enc_2) ) ) >> 16;
      msg_send[6] =  (0xFF000000 & ((long int) (aux_enc_2) ) ) >> 24;
      msg_send[13] =  (0x000000FF & ((long int) (aux_enc_3) ) );         // Encoder 3
      msg_send[12] =  (0x0000FF00 & ((long int) (aux_enc_3) ) ) >> 8;
      msg_send[11] =  (0x00FF0000 & ((long int) (aux_enc_3) ) ) >> 16;
      msg_send[10] =  (0xFF000000 & ((long int) (aux_enc_3) ) ) >> 24;

      // checksum
      checksum = msg_send[2];
      for (cont = 3; cont<14; cont++) {
        checksum += msg_send[cont];
      }
      msg_send[15] = (0x00FF & checksum) ;
      msg_send[14] = (0xFF00 & checksum) >> 8;

      //serDwrite(msg_send, sizeof(msg_send));
*/
      // Display
      sprintf(display_01, "Encoder 1: %d", aux_enc_1);
      sprintf(display_02, "Encoder 2: %d", aux_enc_2);
      sprintf(display_03, "Encoder 3: %d", aux_enc_3);
      DispStr(8, 6, "                                                    ");
   DispStr(8, 7, "                                                  ");
      DispStr(8, 8, "                                                    ");
      DispStr(8, 6, display_01);
      DispStr(8, 7, display_02);
      DispStr(8, 8, display_03);
      waitfor(DelayMs(1));
   }
// ****************************************************************************
      costate
      {  IMU_OK = 0;
       if ((nb = serEread(buffer_imu,127,20)) > 0)
       {
          pos_01 = _n_strchr(buffer_imu,'A');
            pos_02 = _n_strchr(pos_01,'Z');

          if (!pos_01 == NULL || !pos_02 == NULL)
          {
            pos_imu = pos_01;
```

```
            //DispStr(8, 9, "                                                  ");
       //DispStr(8, 9, pos_imu);
            IMU_OK = 1;
          }
         }
         waitfor(DelayMs(1));
      }
// ****************************************************************************
costate
   {
    MAG_OK = 0;
      if ((nb = serCread(buffer_mag,127,20)) > 0)
      {
       buffer_mag[nb] = '\0';
         pos_01 = _n_strstr( buffer_mag, "$raw" );
         pos_02 = _n_strchr(pos_01, '*');
         if (!pos_01 == NULL || !pos_02 == NULL)
         {

            pos_mag = pos_01;
            //DispStr(8, 10, "                                                  ");
       //DispStr(8, 10, pos_mag);
            MAG_OK = 1;
         }
         waitfor(DelayMs(1));
      }
   }
 // ****************************************************************************
      costate
      {
     if ((nb = serDread(sentence,1023,20)) > 0)
{
          //Saving previous data
         if (GPS_OK) {
         pos_ant = pos;
         }
            GPS_OK = 0;
            POS_OK =  0;
            VEL_OK = 0;
          DOP_OK = 0;
         TM_OK = 0;
// ----------------------------------------------------------------------------
         sentence[nb] = '\0';
          p_sen = sentence;
while ((&sentence[nb] - p_sen) > Min_NMEA) {
          p_sen = _n_strchr(p_sen, '$');
             if(p_sen == NULL)
```

```
            break;
       //Getting new data
if (gps_get_pos(&pos,p_sen) == 0) {
               POS_OK = 1;
                 //break;
               }
               /*
               if (gps_get_vel(&vel_enu,p_sen) == 0) {
                  vel.n = vel_enu.n;
                  vel.e = vel_enu.e;
                  vel.d = -vel_enu.u;
                  VEL_OK = 1;
                 // break;
               }
               if (POS_OK == 1 && VEL_OK)
                   GPS_OK = 1;
               */
    if (gps_get_dop(&dop, p_sen) == 0) {
               DOP_OK = 1;
               }
               if (gps_get_utc(&ntime, p_sen)==0) {
               TM_OK = 1;
               }
            p_sen++;
           }
// ----------------------------------------------------------------------------
         if (GPS_OK) {
          llh_2_ned(&vel, &pos, &pos_ant);
         }
         else {
          pos = pos_ant;
         }
// ----------------------------------------------------------------------------
         }
         waitfor(DelayMs(1));
      }
// ****************************************************************************
      costate
      {
       if (IMU_OK)
         {
          msg_send_imu_gps[0] = 'U';
          msg_send_imu_gps[1] = 'U';
          //*****************************************************************
           msg_send_imu_gps[5] =  (0x000000FF & ((long int) (pos.lat_degrees*1e7) ) );         // Latitude
          msg_send_imu_gps[4] =  (0x0000FF00 & ((long int) (pos.lat_degrees*1e7) ) ) >> 8;
          msg_send_imu_gps[3] =  (0x00FF0000 & ((long int) (pos.lat_degrees*1e7) ) ) >> 16;
```

```
        msg_send_imu_gps[2] =  (0xFF000000 & ((long int) (pos.lat_degrees*1e7) ) ) >> 24;

        msg_send_imu_gps[9] =  (0x000000FF & ((long int) (pos.lon_degrees*1e7) ) );            // Longitude
        msg_send_imu_gps[8] =  (0x0000FF00 & ((long int) (pos.lon_degrees*1e7) ) ) >> 8;
        msg_send_imu_gps[7] =  (0x00FF0000 & ((long int) (pos.lon_degrees*1e7) ) ) >> 16;
        msg_send_imu_gps[6] =  (0xFF000000 & ((long int) (pos.lon_degrees*1e7) ) ) >> 24;

        msg_send_imu_gps[13] =  (0x000000FF & ((long int) (pos.altitude*1e5) ) ) ;             // Altitude
        msg_send_imu_gps[12] =  (0x0000FF00 & ((long int) (pos.altitude*1e5) ) ) >> 8;
        msg_send_imu_gps[11] =  (0x00FF0000 & ((long int) (pos.altitude*1e5) ) ) >> 16;
        msg_send_imu_gps[10] =  (0xFF000000 & ((long int) (pos.altitude*1e5) ) ) >> 24;

        msg_send_imu_gps[17] =  (0x000000FF & ((long int) (vel.n*1e7) ) );
        msg_send_imu_gps[16] =  (0x0000FF00 & ((long int) (vel.n*1e7) ) ) >> 8;
        msg_send_imu_gps[15] =  (0x00FF0000 & ((long int) (vel.n*1e7) ) ) >> 16;
        msg_send_imu_gps[14] =  (0xFF000000 & ((long int) (vel.n*1e7) ) ) >> 24;

        msg_send_imu_gps[21] =  (0x000000FF & ((long int) (vel.e*1e7) ) );
        msg_send_imu_gps[20] =  (0x0000FF00 & ((long int) (vel.e*1e7) ) ) >> 8;
        msg_send_imu_gps[39] =  (0x00FF0000 & ((long int) (vel.e*1e7) ) ) >> 16;
        msg_send_imu_gps[38] =  (0xFF000000 & ((long int) (vel.e*1e7) ) ) >> 24;

        msg_send_imu_gps[25] =  (0x000000FF & ((long int) (vel.d*1e7) ) );
        msg_send_imu_gps[24] =  (0x0000FF00 & ((long int) (vel.d*1e7) ) ) >> 8;
        msg_send_imu_gps[23] =  (0x00FF0000 & ((long int) (vel.d*1e7) ) ) >> 16;
        msg_send_imu_gps[22] =  (0xFF000000 & ((long int) (vel.d*1e7) ) ) >> 24;

        msg_send_imu_gps[29] =  (0x000000FF & ((long int) (pos.ts*1e5) ) ) ;          // Tempo
        msg_send_imu_gps[28] =  (0x0000FF00 & ((long int) (pos.ts*1e5) ) ) >> 8;
        msg_send_imu_gps[27] =  (0x00FF0000 & ((long int) (pos.ts*1e5) ) ) >> 16;
        msg_send_imu_gps[26] =  (0xFF000000 & ((long int) (pos.ts*1e5) ) ) >> 24;

        msg_send_imu_gps[31] =  (0x000000FF & ((pos.num_SVs) ) ) ;           // N?mero de SatÈlites
        msg_send_imu_gps[30] =  (0x0000FF00 & ((pos.num_SVs) ) ) >> 8;

        msg_send_imu_gps[32] = GPS_OK;

          // checksum
        checksum = msg_send_imu_gps[2];
        for (cont = 3; cont<33; cont++) {
        checksum += msg_send_imu_gps[cont];
        }
        msg_send_imu_gps[34] = (0x00FF & checksum) ;
        msg_send_imu_gps[33] = (0xFF00 & checksum) >> 8;

//-------------------------------------------------------------------------
msg_send_imu_gps[35] = 'E';
```

```
      msg_send_imu_gps[36] = 'E';

           msg_send_imu_gps[40] =  (0x000000FF & ((long int) (aux_enc_1) ) );         // Encoder 1
      msg_send_imu_gps[39] =  (0x0000FF00 & ((long int) (aux_enc_1) ) ) >> 8;
      msg_send_imu_gps[38] =  (0x00FF0000 & ((long int) (aux_enc_1) ) ) >> 16;
      msg_send_imu_gps[37] =  (0xFF000000 & ((long int) (aux_enc_1) ) ) >> 24;

           msg_send_imu_gps[44] =  (0x000000FF & ((long int) (aux_enc_2) ) );         // Encoder 2
      msg_send_imu_gps[43] =  (0x0000FF00 & ((long int) (aux_enc_2) ) ) >> 8;
      msg_send_imu_gps[42] =  (0x00FF0000 & ((long int) (aux_enc_2) ) ) >> 16;
      msg_send_imu_gps[41] =  (0xFF000000 & ((long int) (aux_enc_2) ) ) >> 24;

           msg_send_imu_gps[48] =  (0x000000FF & ((long int) (aux_enc_3) ) );         // Encoder 3
      msg_send_imu_gps[47] =  (0x0000FF00 & ((long int) (aux_enc_3) ) ) >> 8;
      msg_send_imu_gps[46] =  (0x00FF0000 & ((long int) (aux_enc_3) ) ) >> 16;
      msg_send_imu_gps[45] =  (0xFF000000 & ((long int) (aux_enc_3) ) ) >> 24;

      // checksum
      checksum = msg_send_imu_gps[2];
      for (cont = 37; cont<49; cont++) {
       checksum += msg_send_imu_gps[cont];
      }
      msg_send_imu_gps[50] = (0x00FF & checksum) ;
      msg_send_imu_gps[49] = (0xFF00 & checksum) >> 8;
//-------------------------------------------------------------------------------
           for (cont = 0;cont<23;cont++)
            msg_send_imu_gps[cont+51] = pos_imu[cont];

           msg_send_imu_gps[73] = IMU_OK;


          // mem =1;
          //cont = 0;
          //if (MAG_OK) {
          // while(mem) {
          for (cont = 0; cont<27; cont++)
           msg_send_imu_gps[cont+74] = pos_mag[cont];
          msg_send_imu_gps[101] = MAG_OK;

          msg_send_imu_gps[102] = '\0';
          //    cont++;
          //    if (pos_mag[cont] == '*')
          // mem = 0;
          // }
          // msg_send_imu_gps[cont + 57] = '\0';
          //}
          // else
```

```
          // {
          // msg_send_imu_gps[55] = '\0';
          // }
        //   serDwrite(msg_send_imu_gps, sizeof(msg_send_imu_gps));
        DispStr(8, 16, "                                                    ");
        DispStr(8, 16, msg_send_imu_gps);

        if (sock_established(&sock)) {
        tcp_tick(NULL);
         sock_fastwrite( &sock, msg_send_imu_gps, sizeof(msg_send_imu_gps));
  }
        }
        IMU_OK = 0;
        waitfor(DelayMs(1));
     }
// ****************************************************************************
      costate {
        if  (sock_established(&sock)) {
       tcp_tick(NULL);
          bytes = sock_fastread(&sock, buffer, BUFFER_LEN-1);
       if (bytes > 0) {
          buffer[bytes] = '\0';
             if(!_n_strncmp(buffer,"set_position8",13)) {
                   motor_number = _n_strchr(buffer,'(');
                  position = _n_strchr(buffer,',');
                  position_end = _n_strchr(buffer,')');
                  if ((motor_number != NULL) & (position != NULL) & (position_end != NULL)) {
                   *position = '\0';
                   *position_end = '\0';
                   motor_number++;
                   position ++;
                   set_position8(atoi(motor_number),atoi(position));
                  }
             }
               if(!_n_strncmp(buffer,"restart_initGPS",15))
                restart_initGPS = 1;
               if(!_n_strncmp(buffer,"restart_initIMU",15))
                restart_initIMU = 1;
               if(!_n_strncmp(buffer,"reset_servos",12))
                reset_servos();
               if(!_n_strncmp(buffer,"reset_IMU",12))
                reset_IMU();
               if(!_n_strncmp(buffer,"reset_encoder(1)",16))
        qd_zero(1);
               if(!_n_strncmp(buffer,"reset_encoder(2)",16))
             qd_zero(2);
                if(!_n_strncmp(buffer,"reset_encoder(3)",16))
```

```
                {
          BitWrPortI(PEDR, &PEDRShadow,1, 7);   // Reset 0
    BitWrPortI(PEDR, &PEDRShadow,0, 7);   //
           }
       }
         }
         //reset_servos();
         //set_position8(0,100);
         //waitfor(DelayMs(100));
         //set_position8(0,0);
         //waitfor(DelayMs(100));


         waitfor(DelayMs(1));
      }


     }
}
// ***************************************************************************
```