ANDRESSA OSAWA MASSARI
CAMILA SHIROTA
RICARDO SHIROTA FILHO
RODRIGO NAZARIO

Nota final
9,9 (nove e nove)

ROBÔ QUADRÚPEDE - TUGA

Relatório do Trabalho de
Conclusão de Curso

São Paulo
2005

ANDRESSA OSAWA MASSARI
CAMILA SHIROTA
RICARDO SHIROTA FILHO
RODRIGO NAZARIO

ROBÔ QUADRÚPEDE - TUGA

Relatório do Trabalho de Conclusão de Curso

Áreas de concentração:
Engenharias Mecatrônica
e
de Automação e Controle

Orientador:
Prof. Dr. Marcos P. R. Barreto

São Paulo
2005

FICHA CATALOGRÁFICA

Massari, Andressa Osawa
TUGA – robô quadrúpede / A.O. Massari, C. Shirota, R. Shirota Filho, R. Nazario. -- São Paulo, 2005.
p. 38

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Robôs I.Nazario, Rodrigo II.Shirota, Camila III.Shirota Filho, Ricardo IV.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos V.t.

Aos nossos familiares, que nos apoiaram durante essa importante etapa da vida.

AGRADECIMENTOS

Ao nosso orientador, Prof. Dr. Marcos R. P. Barretto, pelas diretrizes, o apoio e o incentivo.

Ao Prof. Dr. Edilson H. Tamai, por gentilmente ceder seu laboratório.

Aos técnicos do laboratório de projeto de máquinas, pela ajuda na fabricação do protótipo.

Aos nossos amigos, pelas diversas dicas, o apoio, o incentivo e a torcida.

Resumo

O projeto visa à construção de um robô quadrúpede com dois graus de liberdade por perna que anda em equilíbrio estático. O robô recebe comandos externos, sendo capaz de girar em torno de seu próprio eixo e andar, sobre superfícies planas. Estudaram-se mecanismos possíveis para a perna, equacionou-se como movimentar o robô em equilíbrio estático, definiu-se a estrutura de comunicação com o robô e projetou-se um mecanismo simples, visando obter um protótipo de baixo custo. O protótipo tem dimensões de aproximadamente 200x300x50 mm^3, sua estrutura é de alumínio e utiliza servomotores de aeromodelo para o acionamento das pernas. Possui um micro-controlador embarcado que se comunica com um computador externo (base), o qual apresenta uma interface para que o usuário envie comandos ao protótipo. O projeto foi elaborado por duas equipes, uma voltada ao projeto mecânico e construção e outra voltada ao projeto de controle e elaboração do software para a movimentação.

Sumário

Lista de Figuras

Introdução

Robôs com rodas são vastamente explorados para as mais diversas finalidades, porém para aumentar a sua autonomia em terrenos diversificados, a utilização de patas é cada vez mais freqüente. Além disso, o projeto e a construção de um robô, abordando os aspectos mecânico e de controle, é uma atividade efetivamente mecatrônica, sendo estes os principais motivos para a escolha deste projeto.

Foi pensando neste desafio que os alunos decidiram criar um grupo interdisciplinar para realizar um verdadeiro projeto de engenharia. Um dos grupos foi responsável pelo projeto mecânico/eletrônico, realizando os dimensionamentos físicos e escolha de motores e drivers; o outro foi responsável pela parte de controle e software, efetuando o projeto e elaboração do sistema de controle, havendo no final do trabalho uma integração dos sistemas.

A meta do projeto era obter um robô quadrúpede que anda sem cair e sem a necessidade de apoiar o corpo no chão.

Capítulo 1

Definição do projeto

São inúmeros os trabalhos com robôs, cada qual com sua configuração particular visando atingir um certo objetivo. Visualmente, as variações quanto ao número de patas são as mais marcantes. Comumente encontram-se robôs com uma, duas, quatro ou seis pernas. Robôs hexápodes (com seis patas) têm o problema do equilíbrio mais simplificado, enquanto sistemas bípedes têm o problema do equilíbrio bastante complexo. Os quadrúpedes estão no meio termo de complexidade.

Existem dois modos de efetuar a locomoção de robôs com patas, por meio de equilíbrio dinâmico ou estático. O primeiro se trata do modo de andar em que a inércia do robô é utilizada para mantê-lo em equilíbrio durante o movimento (como no caso de robôs de uma perna, os quais precisam ficar pulando para não caírem, ou de robôs de duas pernas que ao andar precisam ter "um jogo de corpo"para não caírem). O outro modo não leva em conta esta inércia, resultando em robôs que mantém seu centro de massa sempre apoiado por no mínimo três pernas.

Deve-se notar que existe um compromisso entre as complexidades do sistema de controle e do mecanismo de movimentação do robô. Desta forma, busca-se a solução que seja mais eficiente, sem tornar nenhuma das partes demasiadamente complexa.

1.1 Análise do problema

A análise do problema é necessária para um entendimento adequado do escopo do projeto, por meio da determinação dos seguintes pontos-chave:

- Tipo de equilíbrio do movimento: estático ou dinâmico;

- Número de pernas do robô: um número maior de pernas irá exigir uma quantidade maior de atuadores e, consequentemente, aumento de massa e custos do projeto. Por outro lado, um maior número de pernas tende a facilitar a movimentação em equilíbrio do robô;

- Características das pernas: se elas serão idênticas, ou de tamanhos diferentes etc. Pernas idênticas facilitam a construção, mas podem não ser a melhor escolha do ponto de vista de controle;

- Graus de liberdade (G.L.) por perna: influi na liberdade de movimentação do protótipo. Pode possuir de 1 G.L. (robô de uma perna – que só pula) a N (por exemplo, o robô humanóide da Honda, que apresenta tantas articulações na perna quanto um ser humano);

- Movimentação do robô: é necessário saber quais movimentos ele poderá realizar, delimitando as seqüências de movimentação das patas que o controle deverá comandar;

- Tipos de terreno em que o robô será capaz de andar: isto definirá se serão necessários sensores nos pés, por exemplo, caso ele tenha que subir degraus de alturas diferentes;

- Modo de controle do robô: se ele será autônomo ou se receberá comados externos.

1.2 Requisitos e restrições

Abaixo são definidos os requisitos e as restrições que o protótipo deve atender:

- Minimização da massa: são necessários motores de menor torque;

- Alta rigidez estrutural: suficiente para suportar com mínima vibração o peso do robô, tanto parado quanto em movimento;

1.3 Especificações

Como o projeto de um robô é amplo, percebe-se que os requisitos e restrições apresentam itens básicos a qualquer projeto mecânico. As especificações, apresentadas a seguir, irão efetivamente modelar o projeto.

- Equilíbrio estático para a movimentação;

- Quatro pernas, escolha esta que equilibra a complexidade do controle e não exige um número excessivo de atuadores;

- Pernas idênticas e independentes, decisão para facilitar a elaboração do mecanismo, o dimensionamento da estrutura e o controle;

- Dois graus de liberdade por perna, sendo este o número mínimo para que o robô consiga movimentar-se utilizando equilíbrio estático (menor número de motores a serem controlados, o que implica maior facilidade de controle);

- Controle embarcado, sendo este o método mais comum para controle de motores em projetos como este;

- Movimentação: o protótipo deverá ser capaz de andar para frente, girar e permanecer parado;

- O robô deverá andar em terreno plano, o que elimina a necessidade de sensores nos pés;

- "Controle remoto": o robô não deve ser totalmente autônomo, recebendo externamente os comandos para se movimentar.

Capítulo 2

MECÂNICA

Como o robô deve se mover utilizando equilíbrio estático, o dimensionamento foi realizado para uma estrutura estática.

De acordo com a pesquisa realizada, concluiu-se que não existe um método para definir um mecanismo adequado para as pernas do robô. Foram obtidas informações de outros trabalhos na questão de dimensões e distribuição de forças nas patas.

Poucos mecanismos encontrados eram suficientemente detalhados para sua perfeita compreensão. Além disso, a maioria deles apresentava mecanismos de três graus de liberdade, o que é mais complexo do que o proposto neste projeto. Assim, foi proposta uma estrutura simplificada em relação aos mecanismos encontrados.

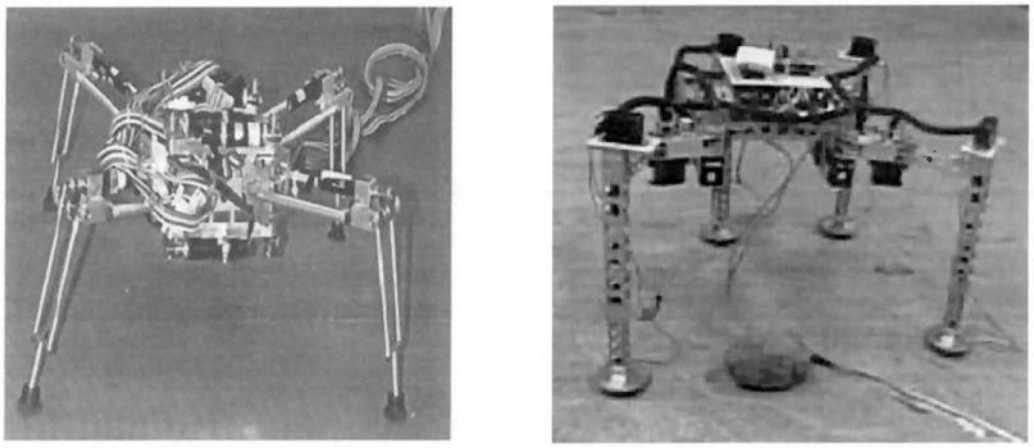


Figura 2.1: Modelos de robôs

2.1 Escolha do mecanismo das pernas

Depois da opção por um robô quadrúpede com patas idênticas e de dois graus de liberdade por perna, a primeira grande decisão com relação à parte mecânica é a definição do sistema de transmissão de movimento às patas. Devido ao número de graus de liberdade por perna determinado, seriam necessários dois atuadores: o primeiro possibilita o movimento vertical da pata, enquanto o segundo, o movimento horizontal. As patas estão dobradas (conforme mostrado nas figuras abaixo) e o "joelho"de cada pata não apresenta rotação, sendo rígido.

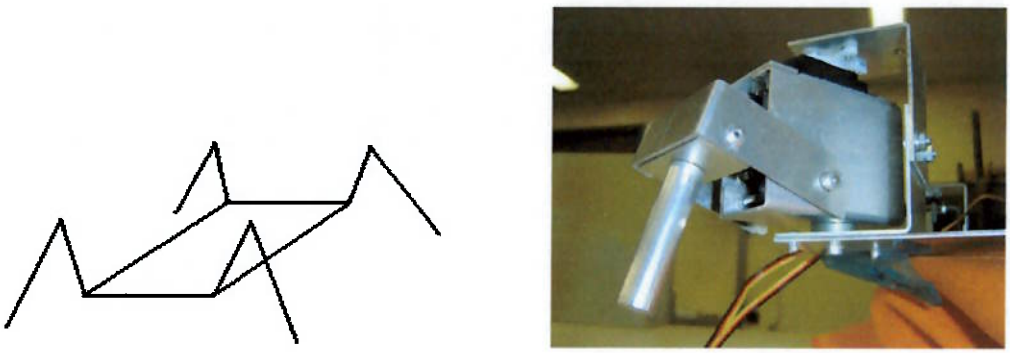


Figura 2.2: Esboço do robô e foto da perna

Grande parte dos artigos encontrados na literatura não mostra explicitamente o mecanismo utilizado para esta transmissão (UMESH, 1998) e (MARTINS FILHO et al., 2000). No caso em que ele é detalhado, trata-se, na maioria dos casos, de mecanismos complexos que não seriam viáveis para nossa aplicação (ZHOU et al., 2000), (DELCOMYN et al., 2000) e (RIDDERSTRÖM, et al., 2001). No entanto, encontraram-se alguns mecanismos interessantes cujas idéias poderiam ser aplicadas ao robô em questão (ZIELINSKA et al., 2002) e (ARIKAWA et al., 1996).

2.1.1 Soluções possíveis

Realizou-se, além das pesquisas bibliográficas, um brainstorming entre os membros do grupo. Obtiveram-se duas idéias gerais para o mecanismo que poderiam ser adotadas:

1. Motores acoplados em série

Nesta solução, os motores estão acoplados em série, de forma que um motor se move junto com a perna durante o movimento causado pelo outro. Assim, um

dos motores controla a rotação do conjunto na horizontal enquanto o outro é responsável pelo movimento vertical da pata.

2. Motores acoplados na base do robô

Nesta configuração, ambos os motores se encontram fixos no corpo do robô, não se movendo com as patas. A grande vantagem desta configuração é justamente o fato de a inércia de um dos motores não ser suportada pelo outro durante o movimento das patas, acarretando um menor consumo de energia e maior estabilidade durante o movimento. Por outro lado, esta construção não possui projeto cinemático simples, pois o fato de a pata ser controlada por dois atuadores fixos implica a utilização de articulações, engrenagens especiais ou escorregamento de peças entre si, o que não ocorreria necessariamente no caso anterior. Tais articulações não devem possuir folga, além de possuírem fabricação razoavelmente complexa. Além disso, em muitos destes casos é necessária a utilização de pistões lineares ao invés de motores rotativos; tais pistões geralmente apresentam custo mais elevado. Um exemplo desta configuração é mostrado abaixo:

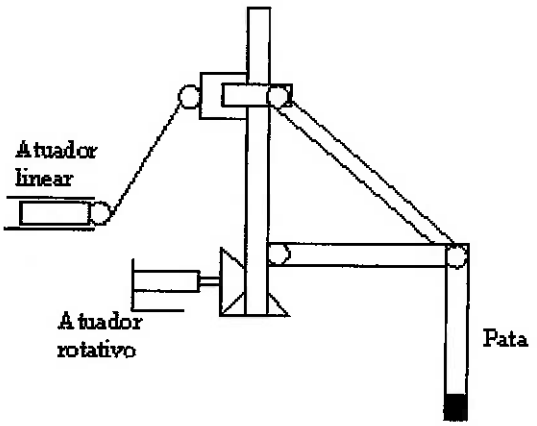


Figura 2.3: Vista frontal de montagem com os motores fixos na base

Após a formalização das idéias, partiu-se para a análise das vantagens e desvantagens de cada uma. Conforme descrito anteriormente, a configuração com motores fixos na base possui projeto cinemático um pouco mais complexo que no outro caso, além de exigir uma precisão suficientemente elevada na montagem das articulações. Na configuração em série, deve-se levar em conta a inércia do motor em movimento. Porém, a priori, esta configuração possui um controle e um projeto mecânico um pouco mais fáceis de serem implementados.

2.1.2 Mecanismo adotado

Analisando-se as vantagens e desvantagens descritas acima, optou-se pela adoção de motores em série dispostos em uma caixa. Neste caso, o eixo de saída do primeiro motor (em posição vertical) está fixo na estrutura sobre a qual a caixa de alumínio gira. Desta forma, ao ser acionado, todo o conjunto gira em torno do eixo vertical, correspondendo ao primeiro grau de liberdade. O eixo de saída do segundo motor (este em posição horizontal) é responsável pelo controle da altura da perna (segundo grau de liberdade). A figura 2.4 é a representação em CAD do mecanismo da perna.

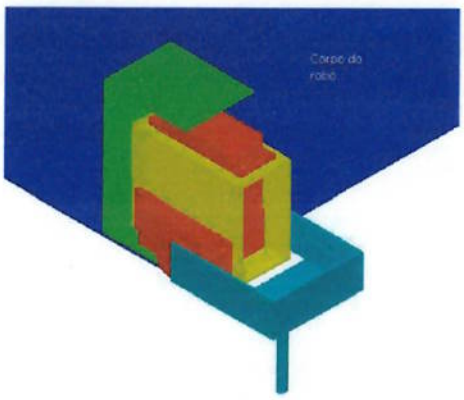


Figura 2.4: Mecanismo das pernas

2.2 Distribuição de forças nas patas

Existem basicamente dois tipos de distribuição de forças para o robô, apresentados a seguir. Ambos dependem dos ângulos em que as patas se encontram.

2.2.1 Três patas apoiadas no chão

A análise desta situação é necessária para o correto dimensionamento dos motores responsáveis por sustentar o robô. Para realizar o equacionamento e determinar qual o valor das reações de apoio em cada pata, considerou-se que existe apenas a componente de força na direção vertical. A seguir é apresentado o esquema utilizado para calcular a distribuição de forças:

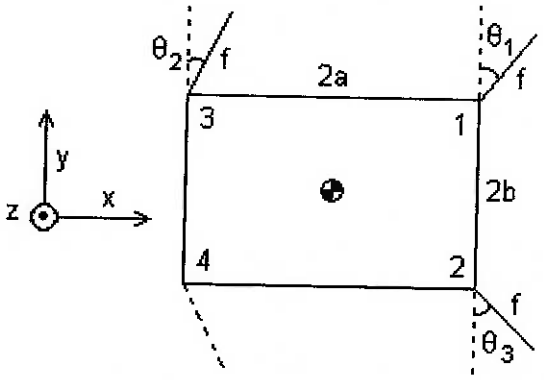


Figura 2.5: Esquema para a distribuição de forças

Em que:

$2a$ é o comprimento do robô;

$2b$ é a largura;

f é o tamanho da perna;
θ_1, θ_2 e θ_3 são os ângulos de rotação das patas.

Cálculo da reação na pata 1 - Z1

$$Z1 = \frac{((a + f sen\theta_2)(2b + f cos\theta_2 + f cos\theta_3) - (b + f cos\theta_2)(2a + f sen\theta_2 - f sen\theta_3))P}{f(sen\theta_2 - sen\theta_1)(2b + f cos\theta_2 + f cos\theta_3) - (2b + f cos\theta_1 + f cos\theta_2)(2a + f sen\theta_2 - f sen\theta_3)}$$

Cálculo da reação na pata 2 - Z2

$$Z2 = \frac{(f(cos\theta_3 - cos\theta_1)(a + f sen\theta_1) + (2a + f sen\theta_1) - f sen\theta_3)(b + f cos\theta_1))P}{(2a + f sen\theta_1 - f sen\theta_3)(2b + f cos\theta_1 + f cos\theta_2) - f^2(sen\theta_2 - sen\theta_1)(cos\theta_3 - cos\theta_1)}$$

Cálculo da reação na pata 3 - Z3

$$Z3 = \frac{((a + f sen\theta_1)(2b + f cos\theta_1 + f cos\theta_2) - f(b + f cos\theta_1)(sen\theta_1 - sen\theta_2))P}{f^2(cos\theta_3 - cos\theta_1)(sen\theta_1 - sen\theta_2) + (2a + f sen\theta_1 - f sen\theta_3)(2b + f cos\theta_1 + f cos\theta_2)}$$

2.2.2 Quatro patas apoiadas no chão

A análise desta situação é útil para avaliar qual o torque necessário para os motores que giram as pernas para frente e para trás (movimento horizontal). Objetiva-se calcular os esforços normais nas patas no momento em que o robô se deslocar para a frente.

Utiliza-se como modelo uma viga bi-apoiada, conforme o esquema a seguir. Calculam-se as forças de reação atuantes no extremo da viga, dividindo-se estas entre as duas pernas. Esta distribuição é adequada para a situação em que o robô

apresenta simetria em relação ao seu eixo longitudinal (antes da 'remada' para andar para a frente, por exemplo). Esta aproximação simplificada foi adotada devido ao caráter hiperestático da estrutura.

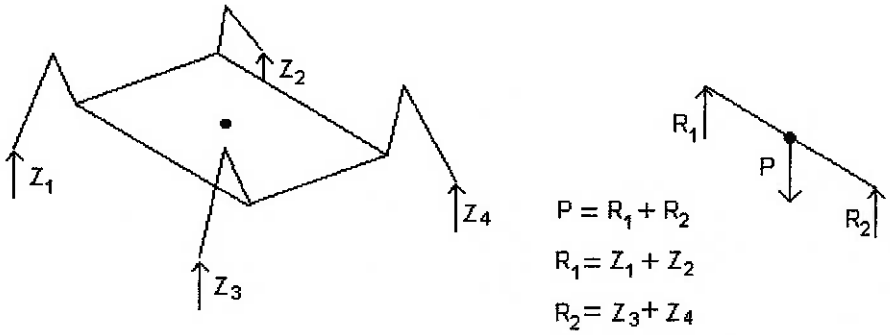


Figura 2.6: Simplificação para o cálculo das reações nas patas

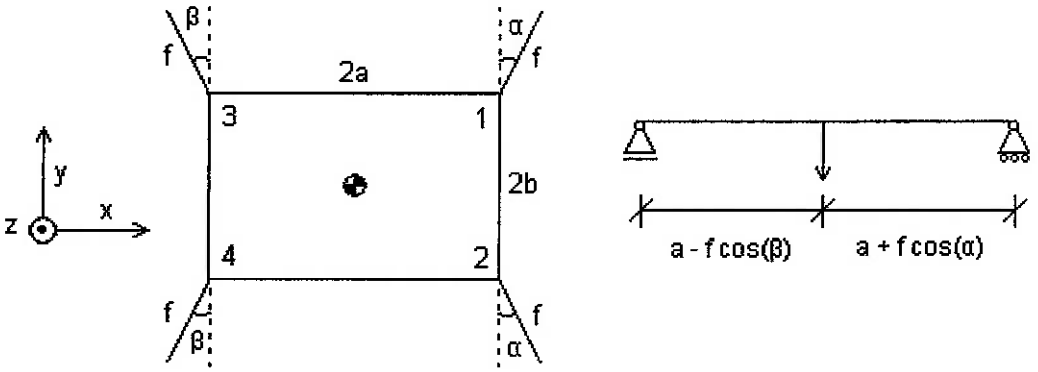


Figura 2.7: Esquema para calcular a distribuição de força nas 4 patas

Os resultados para as reações são:

$$Z1 = Z2 = \frac{(a + f sen\beta)P)}{2(2a + f(sen\alpha + sem\beta))}$$

$$Z3 = Z4 = \frac{(a + f sen\alpha)P}{2(2a + f(sen\alpha + sen\beta))}$$

2.3 Dimensionamento da estrutura

O material escolhido para a estrutura do robô é o alumínio. Seus pontos fortes são a baixa densidade e fácil conformação. Foram utilizadas chapas e barras. Adotou-se para o corpo as dimensões $200 \times 300\,mm^2$, sendo ele formado por uma chapa de alumínio de 1,5 mm de espessura. Dado o valor de densidade específica do alumínio ($\rho_a = 2700\,kg/m^3$), tem-se:

$$\text{Massa do corpo: } m_c = 243,0\,g$$

De acordo com o mecanismo da perna escolhido, obteve-se por CAD o volume de material, o que resultou nos seguintes valores:

$$\text{Massa de uma perna: } m_{1p} = 22,2\,g$$
$$\text{Massa das quatro pernas: } m_{4p} = 88,8\,g$$

Serão necessários oito motores para a movimentação do robô, de massa $45,0\,g$ cada, resultando no total de $360,0\,g$. As massas dos motores, bateria e caixa são estimadas em:

$$\text{Massa dos motores: } m_m = 360,0\,g$$
$$\text{Massa da bateria: } m_b \approx 250,0\,g$$
$$\text{Massa das caixas: } m_{pl} = 263,6\,g$$

Portanto,

$$\text{Massa total: } m_t = 1205,4\,g$$

2.4 Dimensionamento dos motores

De acordo com a pesquisa realizada, resolveu-se utilizar servos de aeromodelo, comumente utilizados para construção de robôs de pequeno porte. Estes servos possuem como vantagem boa relação peso/torque (devido a redutores internos), baixa massa e controle de posição analógico interno.

2.4.1 Motores para levantar a perna

Estes motores requerem maior torque, pois são eles que suportam todo o peso do robô. A estimativa de torque é simples, pois $\tau = Fb$, onde b e é a projeção no plano horizontal da distância entre o ponto de fixação da perna na estrutura e seu ponto de apoio no chão. O problema maior é a estimativa da força F.

A força F foi estimada a partir da distribuição de forças nas patas para o caso em que três delas estão em contato com o chão (visto que esta é a situação em que a perna é mais solicitada). A análise mais simples para este caso é imaginar que duas patas, por exemplo as patas 2 e 3 (uma na frente à direita, outra na parte de trás à esquerda), estão posicionadas de forma que o baricentro do robô esteja sobre a 'linha' que as une. Neste caso, o peso do robô fica distribuído igualmente entre estas duas patas. A partir das equações de distribuição de forças para esta situação, implementou-se uma rotina em Matlab para avaliar qual a influência do ângulo de rotação das pernas do robô.

Conhecendo-se os valores dos ângulos que serão utilizados durante a movimentação do robô de forma a mantê-lo equilibrado, realizaram-se algumas simulações. Concluiu-se que a análise simples descrita acima é uma boa estimativa para avaliar o máximo valor de reação nas patas. Assim, sendo $F = P/2$, tem-se:

$$\tau = \frac{1,2054}{2}.5 = 3,01\, kgf.cm$$

(o torque é apresentado em $kgf.cm$, uma vez que esta é a unidade comumente utilizada em catálogos e especificações de motores)

O servomotor de aeromodelo selecionado é capaz de fornecer este torque.

2.4.2 Motores que rotacionam a perna ao redor do eixo vertical

Como este robô apresenta dois graus de liberdade por perna e, dado o mecanismo de movimentação escolhido, as patas serão arrastadas durante o movimento do dispositivo. Assim, deve-se dimensionar os motores responsáveis por esse movimento para que eles sejam capazes de superar a força de atrito estático, mas sem apresentar um valor excessivamente alto (pois corre-se o risco de o robô ficar patinando no mesmo lugar).

Estes motores serão utilizados para 'remar' o robô, ou seja, serão acionados simultaneamente para empurrar o robô para frente. Neste caso, o torque do motor tem de ser aproximadamente o valor (máximo) da força de atrito estático multiplicado pelo braço da força. Utiliza-se como aproximação para o valor máximo da força de atrito a expressão do atrito de Coulomb, $F_{atrito} = \mu N$.

A partir da análise desenvolvida na seção de distribuição de forças para o caso em que as quatro patas estão em contato com o chão, conclui-se que para as dimensões previstas para este robô, o valor máximo da força normal (em função da variação dos ângulos) é $0,27P$.

A questão mais complicada é estimar o valor de μ. De acordo com pesquisas realizadas, chegou-se aos valores de $(\mu_e = 1)$ para o coeficiente de atrito estático e $(\mu_d = 0,8)$ para o coeficiente de atrito dinâmico, valores estes entre superfícies de concreto e borracha. Desse modo, estima-se que o torque dos motores deve ser aproximadamante $\tau = 0,26\, kgf.cm$

Capítulo 3

CONTROLE

3.1 Posicionamento do centro de massa

A movimentação do robô se dará utilizando o equilíbrio estático. Nesse tipo de movimentação, a projeção do centro de massa encontra-se sempre dentro do polígono formado pelas patas de apoio, garantindo o equilíbrio do robô. Desse modo, para realizar o deslocamento, é necessário planejar antecipadamente a seqüência de movimentação das patas, de forma a garantir o posicionamento do centro de massa. Não é qualquer seqüência das patas, nem uma variação arbitrária dos ângulos delas em relação ao corpo, que evitarão a perda de equilíbrio. Pelo contrário, a sua determinação deve ser bem estudada, pois são muitas variáveis que terminam por garantir uma maior ou menor distância do centro de massa ao limite da região de equilíbrio.

Seja então o modelo do robô como na figura 3.1. O centro de massa (CM) será adotado como sendo o centro do robô, e também como a origem de um sistema cartesiano. O eixo x é transversal ao corpo e o eixo y, longitudinal. Para os ângulos, toma-se como referência uma reta paralela ao eixo das abscissas, como na figura. Valores positivos serão dados para ângulos "acima"da referência, e valores negativos para ângulos "abaixo". Assim, os ângulos estão limitados ao intervalo $(-90°, 90°)$, devido à impossibilidade das patas estarem dentro da região do corpo.

Assume-se que, quando parado, o robô tenha $\varphi_1 = \varphi_2$ e $\theta_3 = \theta_4$. Esta condição de simetria não é absurda, pois, em sistemas reais como em animais, este tipo de situação é observada. Além disso, impõe-se que o robô volte à configuração inicial após a realização do movimento.

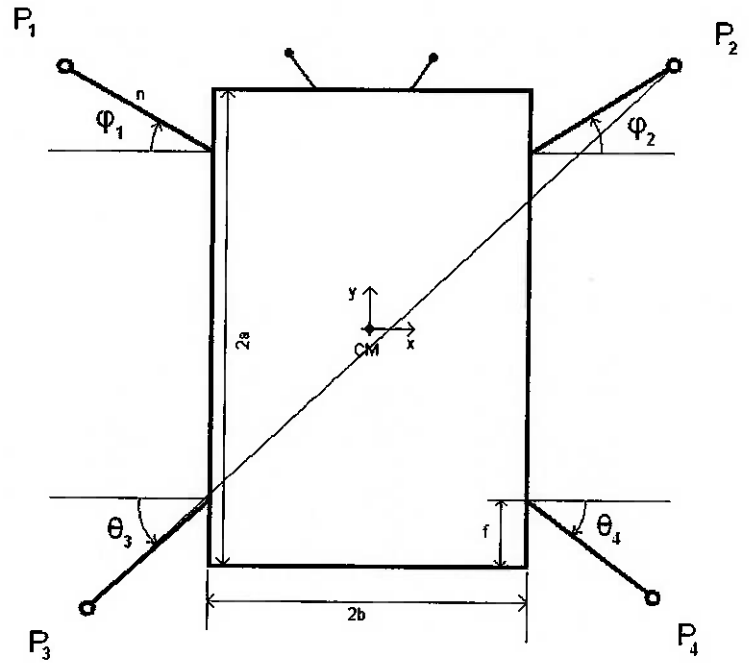


Figura 3.1: Diagrama esquemático do robô

As posições das extremidades das patas são dadas pelas expressões 3.1 a 3.4:

$$P_1 : (-b - n\cos\varphi_1, a - f + n\sin\varphi_1) \tag{3.1}$$

$$P_2 : (b + n\cos\varphi_2, a - f + n\sin\varphi_2) \tag{3.2}$$

$$P_3 : (-b - n\cos\theta_3, -a + f + n\sin\theta_3) \tag{3.3}$$

$$P_4 : (b + n\cos\theta_4, -a + f + n\sin\theta_4) \tag{3.4}$$

Para que uma pata possa ser movimentada, é necessário que o CM esteja dentro do triângulo formado pelas três outras patas que estão apoiadas no chão. O problema do equilíbrio se resume então à posição relativa do CM em relação às patas em diagonal. Dessa forma, seja o problema da localização do CM em relação à diagonal de apoio $\overline{P_2P_3}$. A diagonal é dada pela reta da equação 3.5:

$$y = \frac{y_2 - y_3}{x_2 - x_3}x + cte = \frac{2a - 2f + n(\sin\varphi_2 - \sin\theta_3)}{2b + n(\cos\varphi_2 + \cos\theta_3)}x + cte \tag{3.5}$$

Substituindo as coordenadas de P_2, temos a expressão da constante:

$$cte = \frac{y_2(x_2 - x_3) - x_2(y_2 - y_3)}{x_2 - x_3} \tag{3.6}$$

O posicionamento do CM relativo à diagonal pode ser analisada pelo sinal da constante, pois o CM é a origem do sistema cartesiano e a constante na expressão da reta indica onde ela cruza o eixo das ordenadas. Desenvolvendo a expressão acima, tem-se:

$$\begin{aligned}
(x_2 - x_3) \cdot cte &= (a - f + n\sin\varphi_2)(2b + n\cos\varphi_2 + n\cos\theta_3) - \\
&\quad -(b + n\cos\varphi_2)(2a - 2f + n\sin\varphi_2 - n\sin\theta_3) \\
&= 2ab + an\cos\varphi_2 + an\cos\theta_3 - 2bf - fn\cos\varphi_2 - fn\cos\theta_3 + \\
&\quad +2bn\sin\varphi_2 + n^2\sin\varphi_2\cos\varphi_2 + n^2\sin\varphi_2\cos\theta_3 - 2ab + 2bf - \\
&\quad -bn\sin\varphi_2 + bn\sin\theta_3 - 2an\cos\varphi_2 + 2fn\cos\varphi_2 - \\
&\quad -n^2\cos\varphi_2\sin\varphi_2 + n^2\cos\varphi_2\sin\theta_3 \\
&= an\cos\theta_3 - fn\cos\theta_3 + bn\sin\varphi_2 + n^2\sin\varphi_2\cos\theta_3 - \\
&\quad -an\cos\varphi_2 + fn\cos\varphi_2 + n^2\cos\varphi_2\sin\theta_3 \\
&= n[a(\cos\theta_3 - \cos\varphi_2) + f(\cos\varphi_2 - \cos\theta_3) + \\
&\quad +b(\sin\varphi_2 + \sin\theta_3) + n(\sin\varphi_2\cos\theta_3 + \cos\varphi_2\sin\theta_3)] \\
&= n[b(\sin\varphi_2 + \sin\theta_3) + (a - f)(\cos\theta_3 - \cos\varphi_2) + n\sin(\varphi_2 + \theta_3)]
\end{aligned}$$

Seja a mudança de variáveis dada abaixo:

$$\begin{cases} \beta + \gamma = \theta_3 \\ \beta - \gamma = \varphi_2 \end{cases} \Rightarrow \begin{cases} \beta = \frac{\theta_3 + \varphi_2}{2} \\ \gamma = \frac{\theta_3 - \varphi_2}{2} \end{cases}$$

A equação fica:

$$\begin{aligned}
(x_2 - x_3) \cdot cte &= n[2b\sin\beta\cos\gamma - 2(a - f)\sin\beta\sin\gamma + n\sin(2\beta)] \\
&= n[2b\sin\beta\cos\gamma - 2(a - f)\sin\beta\sin\gamma + 2n\sin\beta\cos\beta] \\
&= 2n\sin\beta[b\cos\gamma - (a - f)\sin\gamma + n\cos\beta]
\end{aligned}$$

E, finalmente,

$$cte = \frac{2n\sin\beta[b\cos\gamma - (a - f)\sin\gamma + n\cos\beta]}{2b + 2n\cos\gamma\cos\beta} \tag{3.7}$$

Se

$$\left\{ \begin{array}{l} \varphi \geq 0 \\ \theta \leq 0 \end{array} \right\} \Rightarrow \begin{cases} -90° \leq \gamma \leq 0 \Rightarrow \sin\gamma \leq 0, \cos\gamma \geq 0 \\ -45° \leq \beta \leq 45° \Rightarrow \cos\beta \geq 0 \end{cases}$$

E $\cos\beta > 0$ se

$$0 < \beta \leq 45° \Rightarrow -\theta_3 < \varphi_2 \leq 90° - \theta_3 \Rightarrow -\theta_3 < \varphi_2 < 90°$$

(porque $\theta_3 \leq 0$ e $\varphi < 90°$). Da expressão 3.7 e das restrições dos ângulos dadas acima, chega-se ao seguinte resultado:

$$cte \left\{ \begin{array}{lll} > 0 & se & -\theta_3 < \varphi_2 < 90° \\ = 0 & se & \varphi_2 = -\theta_3 \\ < 0 & se & 0 < \varphi_2 < -\theta_3 \end{array} \right\} \Rightarrow \text{posição do CM} \left\{ \begin{array}{l} abaixo \\ sobre \\ acima \end{array} \right\} \text{da reta } \overline{P_2P_3}$$

Deve-se notar que, para a diagonal $\overline{P_1P_4}$, o resultado é o mesmo, mudando apenas o sinal da tangente da equação da reta. Assim, com essas condições, agora é possível determinar a seqüência de movimentação das patas.

3.2 Seqüência das patas

Vai-se procurar determinar quais as possíveis ordens de movimentação das patas, de forma a não cair em uma posição que impossibilite a continuação do movimento. Isto acontece quando o levantamento da próxima pata da seqüência implica na perda do equilíbrio do robô, pois o centro de massa não está dentro do triângulo formado pelas outras três patas apoiadas. O esquema a ser utilizado será a movimentação de duas patas seguida de uma remada inicial, a movimentação das duas outras patas e a remada final. Este padrão foi escolhido pois permite uma liberdade maior nos parâmetros se comparado com o esquema de movimentar as quatro patas e depois remar uma única vez. Além disso, a remada significa a ocorrência de escorregamento da pata do robô no chão e, por isso, seria interessante minimizar o número de remadas. Portanto, o esquema apresentado é mais adequado ao robô projetado que a seqüência de remar logo após cada movimento de pata.

1. $P_1 \leftrightarrow P_4$ $(P_2 \leftrightarrow P_3)$

Não é possível movimentar patas em diagonal uma em seguida da outra. Seja, por exemplo, a seqüência $P_1 \rightarrow P_4$. Para movimentar P_1, é necessário que $\overline{P_2P_3}$ esteja acima do CM. Para movimentar P_4, é necessário que $\overline{P_2P_3}$ esteja abaixo do CM, o que é evidentemente impossível logo após a movimentação de P_1.

2. $P_3 \leftrightarrow P_4$

1. Para mexer P_3, é necessário que $\overline{P_1P_4}$ esteja abaixo do CM.
   Mexe P_3.
2. Para mexer P_4, é necessário que $\overline{P_2P_3}$ esteja abaixo do CM.
   Mexe P_4.
3. Mexe o corpo para frente $\Rightarrow$ $\overline{P_1P_4}$ e $\overline{P_2P_3}$ abaixo do CM.
   Não é possível mexer P_1 nem P_2!

3. $P_1 \leftrightarrow P_2$

1. Para mexer P_1, é necessário que $\overline{P_2P_3}$ esteja acima do CM.
   Mexe P_1.
2. Para mexer P_2, é necessário que $\overline{P_1P_4}$ esteja acima do CM.
   Mexe P_2.
3. Mexe o corpo para a frente $\Rightarrow$ CM ? $\overline{P_1P_4}$.
   Forçar $\overline{P_1P_4}$ abaixo do CM.
   Mexe P_3.
4. Forçar $\overline{P_2P_3}$ abaixo do CM.
   Mexe P_4.
5. Remar.
   Impossível voltar à condição inicial $\overline{P_2P_3}$ acima do CM!

4. $P_2 \rightarrow P_4$ $(P_1 \rightarrow P_3)$

1. Para mexer P_2 (P_1), é necessário que $\overline{P_1P_4}$ $(\overline{P_2P_3})$ esteja acima do CM $\rightarrow$ $\overline{P_2P_3}$ $(\overline{P_1P_4})$ acima do CM.
   Mexe $P_2(P_1)$.
2. Para mexer P_4 (P_3), é necessário que $\overline{P_2P_3}$ $(\overline{P_1P_4})$ esteja abaixo do CM.
   Impossível logo após movimentar P_2!

5. $P_4 \rightarrow P_2$ $(P_3 \rightarrow P_1)$

1. Para mexer P_4, é necessário que $\overline{P_2P_3}$ esteja abaixo do CM.
   Mexe P_4.
   Forçar $\overline{P_1P_4}$ acima do CM.
2. Para mexer P_2, é necessário que $\overline{P_1P_4}$ esteja acima do CM.
   Mexe P_2.
3. Remar.
   Forçar $\overline{P_1P_4}$ abaixo do CM.
4. Para mexer P_3, é necessário que $\overline{P_1P_4}$ esteja abaixo do CM.
   Mexe P_3.

Forçar $\overline{P_2P_3}$ acima do CM.

5. Para mexer P_1, é necessário que $\overline{P_2P_3}$ esteja acima do CM.
   Mexe P_1.
6. Remar.
   Forçar CM acima de $\overline{P_2P_3}$ e $\overline{P_1P_4}$.

Portanto, a seqüência de movimentação a ser utilizada é

$$P_4 \rightarrow P_2 \rightarrow \text{rema} \rightarrow P_3 \rightarrow P_1 \rightarrow \text{rema}.$$

É preciso analisar esta seqüência mais detalhadamente, a fim de obter as eventuais restrições dos ângulos para garantir o equilíbrio estático. Os parâmetros livres a determinar são os ângulos da configuração inicial (φ_1, φ_2, θ_3 e θ_4), a variação total (δ) de cada ângulo que será associado ao movimento (soma das duas remadas) e o valor da remada intermediária (α).

Seja a configuração inicial dada por $\varphi_1 = \varphi_2 > 0$ e $\theta_3 = \theta_4 < 0$.

1. Para mexer P_4, é necessário que o CM esteja acima de $\overline{P_2P_3} \Rightarrow sen\beta_{23} < 0$.
   $sen\beta_{23} < 0 \Rightarrow -90° \leq \theta_3 < -\varphi_2$ e, por simetria, $-90° \leq \theta_4 < -\varphi_1$.
2. Mexer $P_4 \Rightarrow \theta_4' = \theta_4 + \delta$
   Para mexer P_2, é necessário forçar $\overline{P_1P_4}$ acima do CM $\Rightarrow sen\beta_{14}' > 0$
   $$\left.\begin{array}{r} -\varphi < \theta_4' < 90° \Rightarrow -\varphi_1 < \theta_4 + \delta < 90° \\ \text{antes: } -90° < \theta_4 < -\varphi_1 \end{array}\right\} \Rightarrow \theta_4 < -\varphi_1 < \theta_4 + \delta$$
3. Mexer $P_2 \Rightarrow \varphi_2' = \varphi_2 + \delta$
   $\beta_{23}' = \beta_{14}' \Rightarrow \overline{P_2P_3}$ acima do CM $\Rightarrow sen\beta_{23}' > 0$
   $$\left.\begin{array}{r} -\theta_3 < \varphi_2' < 90° \Rightarrow -\theta_3 < \varphi_2 + \delta < 90° \\ \text{antes: } -90° < \varphi_2 < -\theta_3 \end{array}\right\} \Rightarrow \varphi_2 < -\theta_3 < \varphi_2 + \delta$$
4. Remar $\Rightarrow -\alpha$ nas patas ($\alpha < \delta$)
   Para mexer P_3, é necessário forçar $\overline{P_1P_4}$ abaixo do CM $\Rightarrow sen\beta_{14} < 0$
   $-90° < \theta_4'' < -\varphi_1' \Rightarrow -90° < \theta_4 + \delta - \alpha < -\varphi_1 + \alpha \Rightarrow \alpha > \frac{\theta_4 + \delta + \varphi_1}{2} = \frac{\beta + \delta}{2}$
   $\Rightarrow \overline{P_2P_3}$ abaixo do CM (porque $\beta_{23}' = \beta_{14}'$)
5. Mexer $P_3 \Rightarrow \theta_3'' = \theta_3 + \delta - \alpha$
   Para mexer P_1, é necessário forçar $\overline{P_2P_3}$ acima do CM $\Rightarrow sen\beta_{23}'' > 0$
   $$\beta_{23}'' = \frac{\theta_3'' + \varphi_2''}{2} = \frac{\theta_3 + \delta - \alpha + \varphi_2 + \delta - \alpha}{2} = \beta_{23} + \delta - \alpha > 0$$
   $$\left.\begin{array}{r} -\theta_3 < \varphi_2 + 2(\delta - \alpha) \leq 90° + \delta - \alpha \\ \text{antes: } \varphi_2 < -\theta_3 < \varphi_2 + \delta \end{array}\right\} \Rightarrow \varphi_2 < -\theta_3 < \varphi_2 + 2(\delta - \alpha)$$
6. Mexer $P_1 \Rightarrow \varphi_1'' = \varphi_1 + \delta - \alpha$
7. Remar $\Rightarrow -\delta + \alpha$ nas patas

O equilíbrio é garantido pois o robô está apoiado nas quatro patas.

Tem-se agora intervalos em que os ângulos podem estar definidos. Estes intervalos garantem a posição do centro de massa em relação à diagonal de apoio. No

entanto, esta informação não é suficiente para fornecer uma medida quantitativa desta posição. Este tipo de dado é bastante útil no sentido de indicar a margem de segurança no posicionamento do CM, fornecendo a precisão necessária no controle dos motores.

A medida procurada é a distância do CM em relação à diagonal de apoio. Desse modo, o problema é resumido ao cálculo da distância de um ponto a uma reta. Isto é obtido a partir do resultado (ver figura 3.2):

$$d(P,r) = \frac{\|\overrightarrow{AP} \otimes \overrightarrow{v}\|}{\|\overrightarrow{v}\|}$$

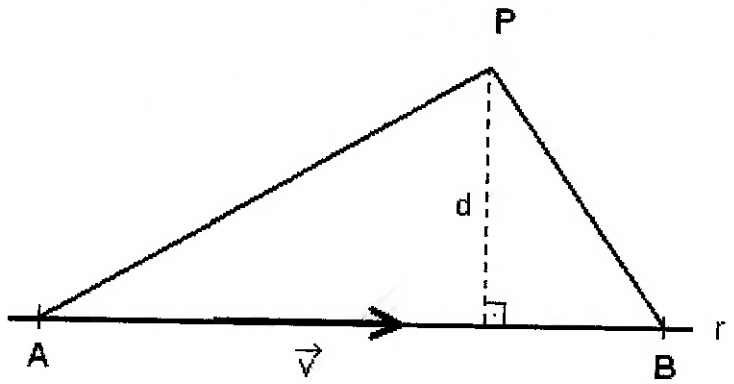


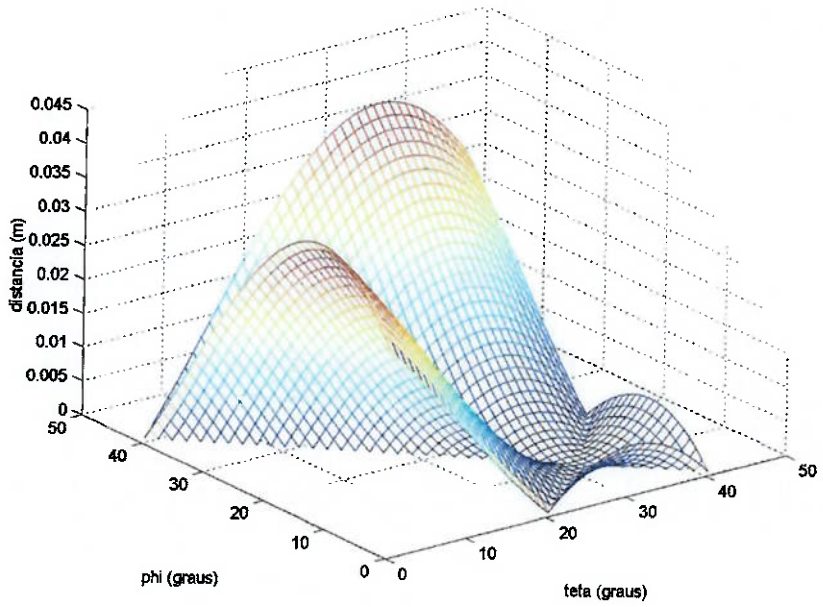
Figura 3.2: Cálculo da distância de um ponto a uma reta

Esta expressão, em função de φ e θ, envolve raiz quadrada e potências à quarta. A sua otimização é, assim, bastante complexa analiticamente. Por isso, no que segue, é utilizada a análise numérica, toda realizada em Matlab, versão 6.0.

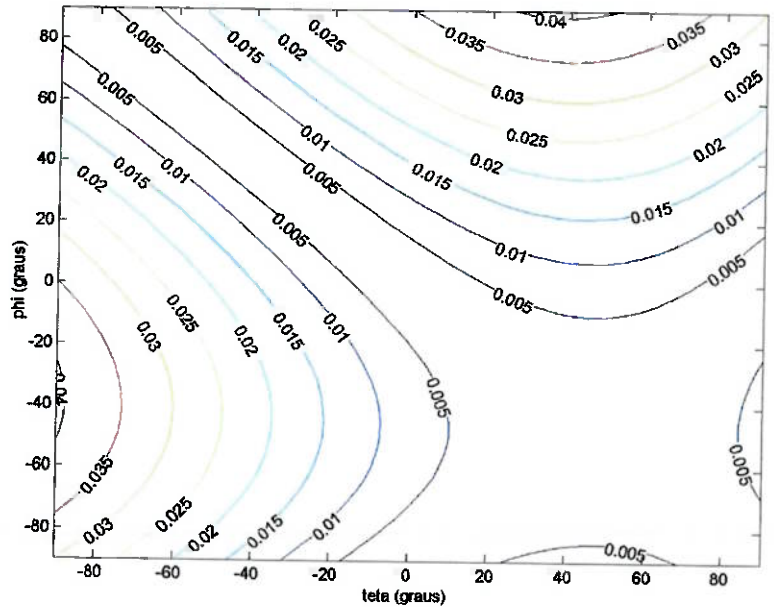
O gráfico da distância, em função de φ e θ, está apresentado na figura 3.3.

Foram testados vários valores de φ, θ, δ e α. O ponto que forneceu melhor margem de segurança foi o $\varphi = 30°$, $\theta = -55°$, $\delta = 50°$ e $\alpha = 25°$. Esta seqüência tem como distância mínima 9,3 mm, e os valores da distância entre o centro de massa e a diagonal de apoio pode ser vista na figura 3.4. Os números próximos aos pontos indicam a ordem em que foram percorridos. A seqüência resultante é apresentada na figura 3.6.

Uma vez determinada a seqüência das patas para a movimentação para frente, foi escolhida outra seqüência para virar à esquerda. Vale ressaltar que a seqüência obtida indicará a ordem para virar à direita, bastando utilizar a simetria imposta no início do problema.



(a) Gráfico da distância



(b) Curvas de nível

Figura 3.3: Distância do CM em relação à diagonal de apoio

A posição inicial deve ser a mesma do movimento para frente, já que se trata da posição de repouso. A seqüência de movimentos é bastante parecida com a do deslocamento para frente. A mudança está nas remadas. Para andar para frente,

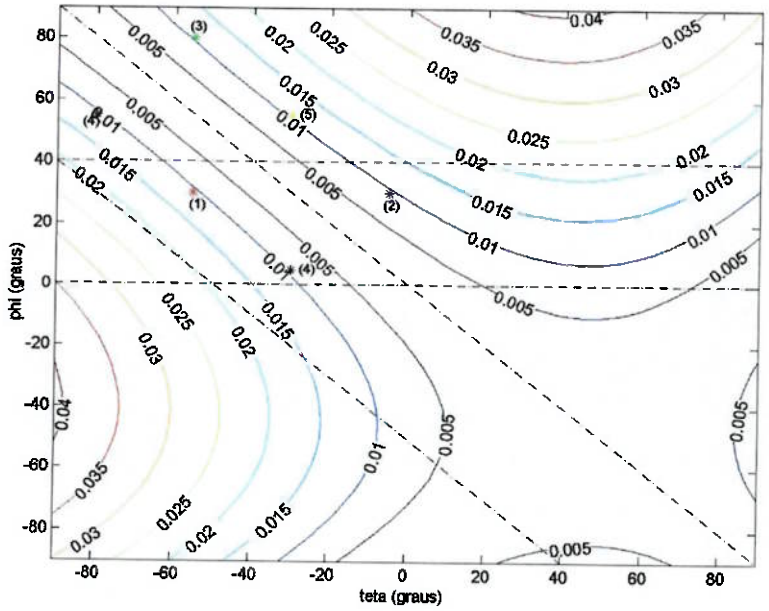


Figura 3.4: Localização da patas ao andar para frente

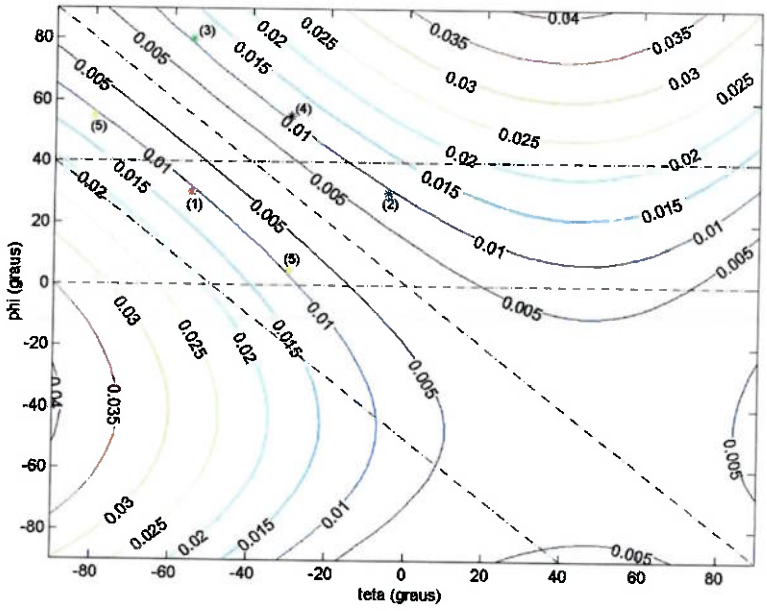
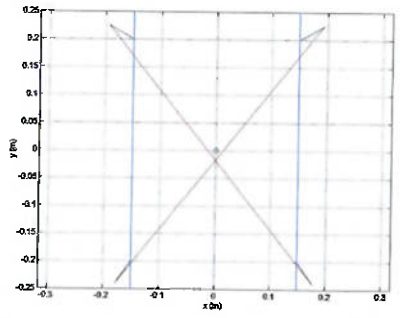
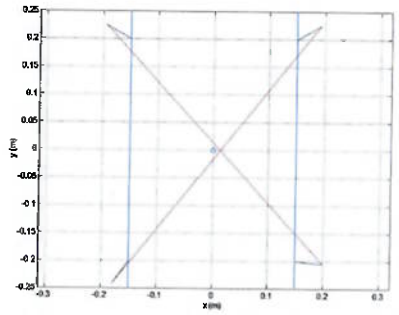


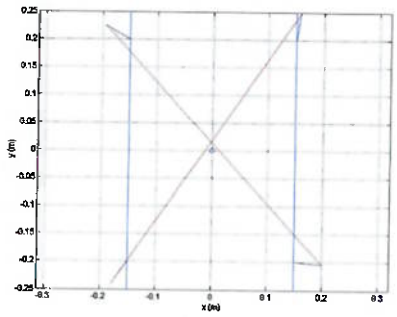
Figura 3.5: Localização da pata ao realizar o giro



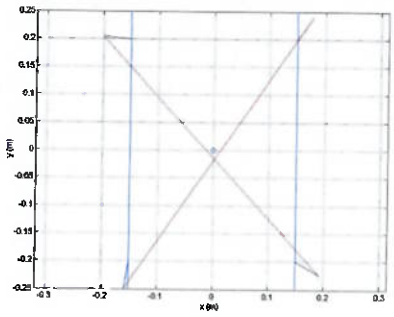
(a) Posição inicial



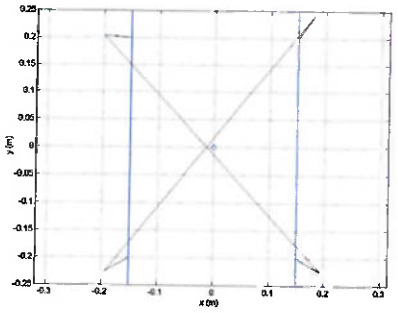
(b) Primeiro passo



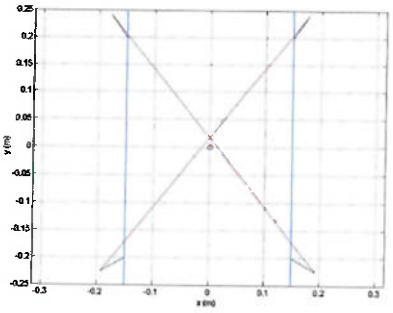
(c) Segundo passo



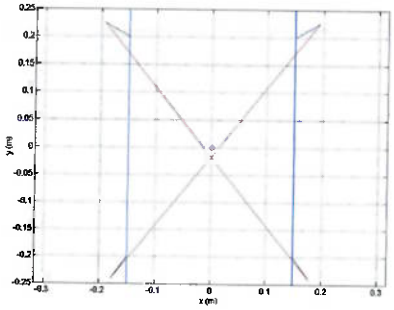
(d) Primeira remada



(e) Terceiro passo



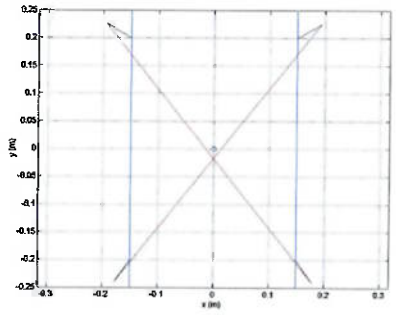
(f) Quarto passo



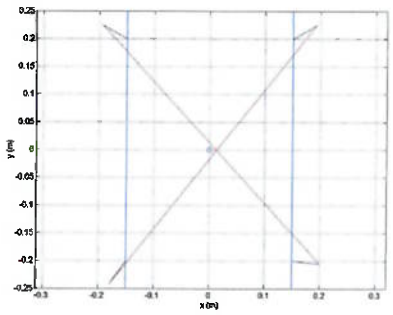
(g) Remada final - Posição inicial

Figura 3.6: Seqüência de movimento das patas no movimento frontal

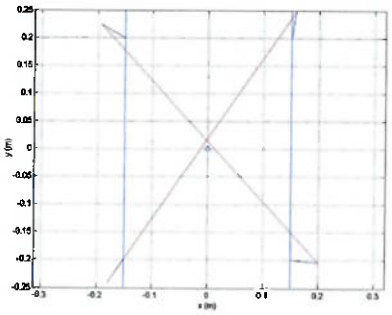
deve-se remar $-\delta$ em todas as patas. Para virar à esquerda, deve-se remar $-\delta$ nas patas de direita, e $+\delta$ nas patas da esquerda. Ora, como a posição do centro de massa em relação à diagonal de apoio depende de $\beta = \frac{\delta+\theta}{2}$, vê-se que o CM continua do mesmo lado antes e depois das remadas intermediária e final. Isso implica em uma mudança na ordem entre P_1 e P_3. O ponto que forneceu melhor margem de segurança foi o $\varphi = 30°$, $\theta = -55°$, $\delta = 50°$ e $\alpha = 25°$. Esta seqüência tem como distância mínima 9,3 mm, e os valores da distância entre o centro de massa e a diagonal de apoio pode ser vista na figura 3.5. Os números próximos aos pontos indicam a ordem em que foram percorridos. A seqüência resultante é apresentada na figura 3.7.



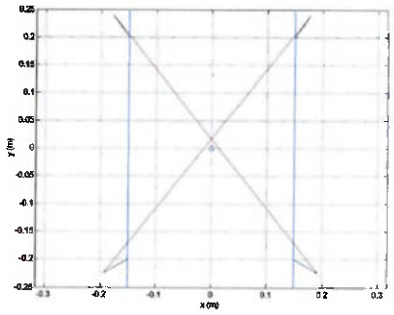
(a) Posição inicial



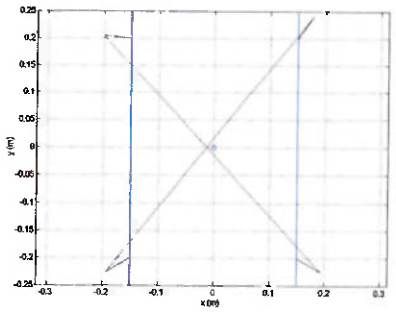
(b) Primeiro passo



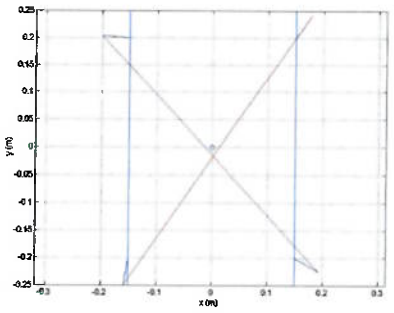
(c) Segundo passo



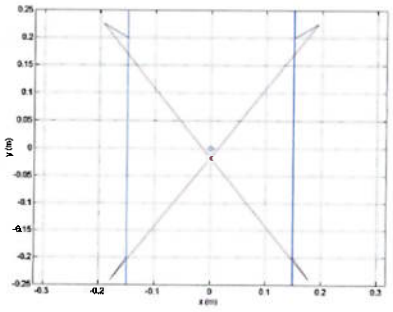
(d) Primeira rotação



(e) Terceiro passo



(f) Quarto passo



(g) Rotação final - Posição inicial

Figura 3.7: Seqüência de movimento das patas ao girar

Capítulo 4

SOFTWARE

O projeto de software pode seguir duas abordagens: a estruturada e a orientada a objetos. Neste trabalho, escolheu-se seguir a abordagem orientada a objetos, direcionada para a programação em linguagens como C++ e Java.

4.1 Declaração de Objetivos

Deseja-se desenvolver um sistema de controle para o robô quadrúpede. Este sistema deve receber informações sobre o movimento que se deseja realizar. O comando de movimento deverá ser fornecido pelo operador em um terminal base. Esta informação será enviada à estação embarcada no robô. A seguir, a estação remota deverá executar os movimentos.

O sistema deve sinalizar o final do movimento ao operador. Além disso, deve permitir o monitoramento de suas atividades. Quando o sistema é ligado, deve ser executada uma rotina de inicialização. Quando se deseja desligar o robô, o sistema deve colocá-lo em uma posição em que o desacionamento dos motores não provoque nenhuma queda ou dano ao robô.

4.2 Projeto do software

A abordagem adotada é interessante do ponto de vista de projeto de um sistema direcionado a linguagens orientadas a objetos. Ela fornece, portanto, uma boa visão das classes que compõem o sistema, e sua interligação. No entanto a visão do sistema como um todo se dá apenas em níveis mais avançados de projeto. Para facilitar a visualização do funcionamento do sistema, foram utilizadas duas ferramentas da abordagem estruturada: o DFD (Diagrama de Fluxo de Dados) e o Diagrama de Hierarquia.

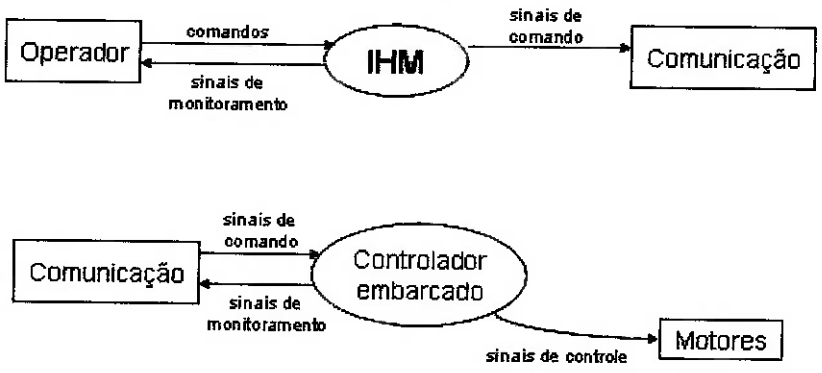


Figura 4.1: DFD nível 0

Foram esquematizados dois níveis do DFD, apresentados nas figuras 4.1, 4.2 e 4.3.

Os documentos referentes à abordagem orientada a objetos encontram-se nos **anexos A e B**, a saber, a especificação de casos de uso e o diagrama de classes (e as classes de implementação).

Terminado o projeto do software, a programação propriamente dita foi dividida em duas partes. Uma delas foi responsável pela realização do programa do controlador 'remoto', com a interface homem-máquina. Por facilidade de construção e pela não necessidade de velocidades extremamente altas, essa primeira parte foi realizada em Java. A outra buscou programar o controlador embarcado, realizada em Basic.

4.3 Software do controle 'remoto'

Inicialmente, planejava-se programar o software da estação remota em C++. No entanto, a dificuldade em encontrar referências e ferramentas para a construção de elementos gráficos e de comunicação pela porta serial do computador nessa linguagem, fizeram com que essa escolha fosse reconsiderada. Dessa forma, outra linguagem orientada a objetos e largamente utilizada atualmente foi escolhida, a linguagem Java.

Programas escritos em Java apresentam algumas desvantagens com relação aos escritos em C++. Entre outros, pode-se citar a impossibilidade de gerar arquivos executáveis (.exe) e o menor controle sobre o uso da memória. No caso deste projeto, esses não são fatores preocupantes, pois os computadores disponíveis têm as ferramentas necessárias para a execução de programas em Java e dispõem

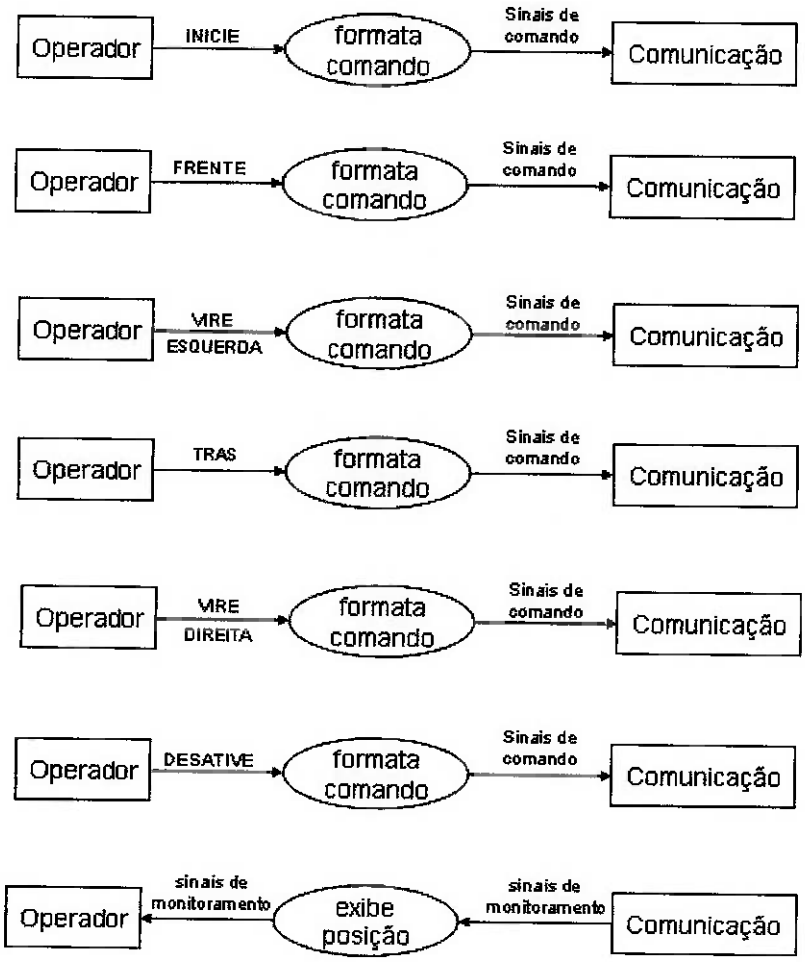


Figura 4.2: DFD nível 1 do primeiro controlador - computador base

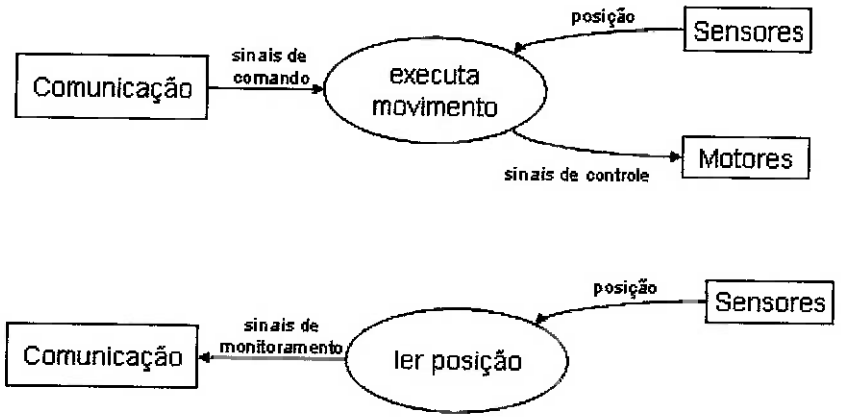


Figura 4.3: DFD nível 1 do segundo controlador - computador embarcado

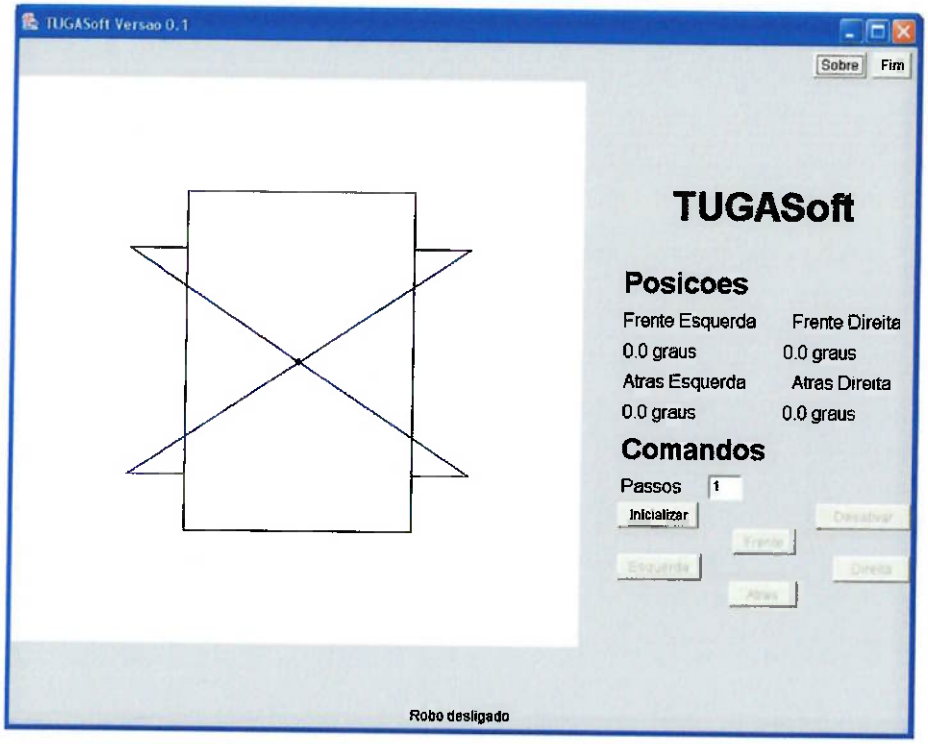


Figura 4.4: Apresentação do TUGASoft, versão 0.1

de memória mais que suficiente para a aplicação.

A seguir, descreve-se o funcionamento do software da estação remota pela exploração da interface gráfica com o usuário.

4.3.1 A interface gráfica do TUGASoft

A interface gráfica, na abertura, está ilustrada na figura 4.4.

A janela é dividida em três áreas. A primeira delas, mais à esquerda, é a de visualização do robô. É nela que o usuário vai acompanhar visualmente a movimentação do robô. É possível analisar o posicionamento da diagonal de apoio das patas que determina o equilíbrio do TUGA, e a localização do centro de massa em relação a ela. A representação gráfica do robô é atualizada a cada passada.

A segunda área da interface gráfica é a área de monitoramento das posições das patas do robô. Ela se localiza na parte superior da coluna da direita. É através dela que o usuário poderá saber o posicionamento das patas ao longo dos movimentos. Os ângulos são dados em graus, e são atualizados a cada passada.

A última área é a área de comandos. Lá estão os recursos que permitirão ao usuário enviar instruções ao sistema embarcado. Ela se localiza na parte inferior direita da janela.

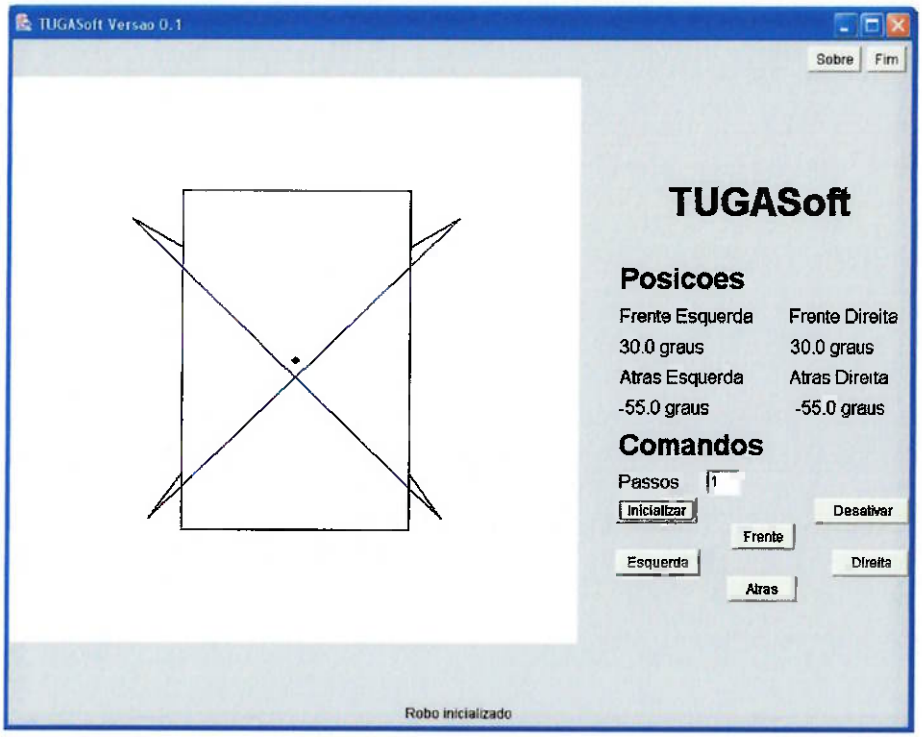


Figura 4.5: TUGASoft, robô inicializado

Além disso, há dois botões na região superior direita da janela, que fornecem informações sobre o programa ("Sobre") e permitem a finalização do programa ("Sair"). Na parte inferior, há também uma linha que indica o movimento realizado pelo robô.

Como pode ser visto na figura 4.4, ao iniciar o software, o único comando que o usuário pode enviar ao robô é o de inicialização, colocando-o em uma posição conhecida (pré-determinada). Esse também é o caso após o envio do comando de desativação. À medida que o robô se movimenta, a área de visualização vai sendo atualizada, assim como as posições das patas na área de monitoramento. A figura 4.5 mostra o robô inicializado.

4.3.2 Determinação dos comandos

O software permite a execução de 6 comandos diferentes:

1. Inicializar: põe o robô em uma posição conhecida (pré-determinada).
2. Desativar: desativa o robô.
3. Frente: o robô anda para frente.
4. Atrás: o robô anda para trás.

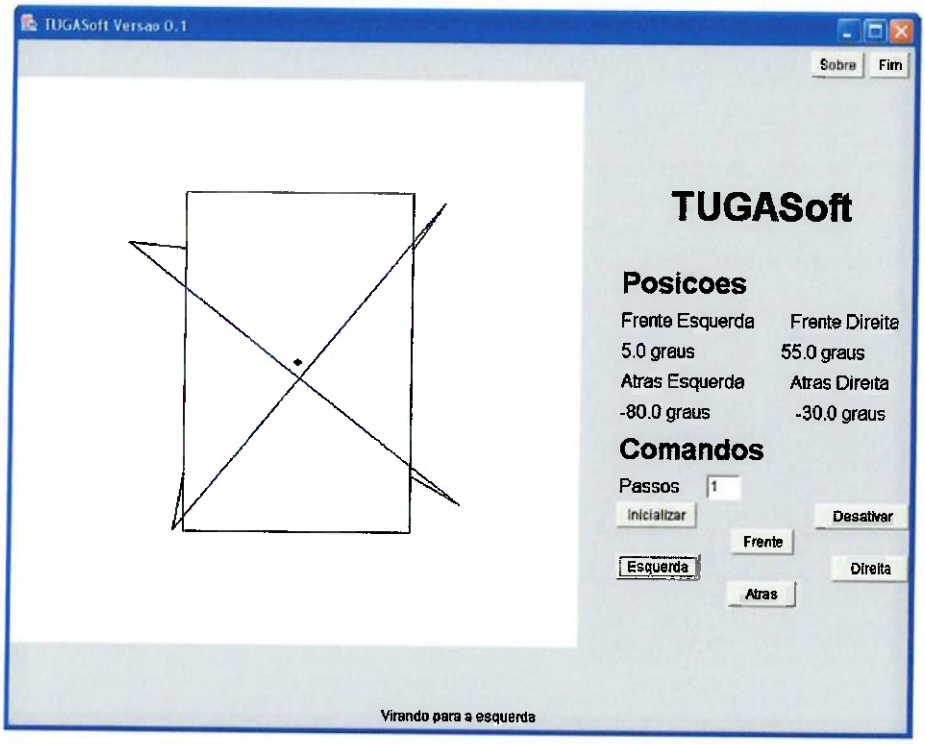


Figura 4.6: TUGASoft, movimento para a esquerda

5. Esquerda: o robô anda para a esquerda.

6. Direita: o robô anda para a direita.

Além disso, pode-se escolher o número de passos a serem dados. Um passo corresponde a um ciclo 2 movimentos-1 remada-2 movimentos-1 remada. O número de passos foi (arbitrariamente) limitado a 30 passos no máximo. A cada botão há uma função associada que determina a ordem de movimentação das patas, além das variações angulares de cada uma. As patas se mexem uma a uma, exceto quando o robô deve "remar"e movimentar as quatro patas simultaneamente. Uma pata não deve mexer antes de se ter certeza de que a anterior já finalizou o movimento, pois isso poderia desequilibrar o robô e provocar sua queda. Além disso, após a confirmação de cada movimento executado, as áreas de visualização e monitoramento das posições das patas são atualizadas. A figura 4.6 ilustra o software durante um passo para a esquerda.

O acionamento de um botão durante a execução de um movimento não afeta sua continuação, sendo o comando escolhido o próximo a ser efetuado. Futuramente, espera-se poder alterar o campo de comando de número de passos para outro em que o usuário poderá escolher a distância total percorrida pelo robô. Nesse caso, o número de passos será determinado automaticamente pelo software.

O módulo de comunicação está descrita na seção 'Comunicação entre controladores'. Os códigos fonte (.java) estão no **anexo C**.

4.4 Programação do controlador embarcado

A tarefa do controlador embarcado é receber os comandos enviados pela estação base e gerar os sinais de controle para os motores que movimentam as pernas. Apesar de ser uma tarefa aparentemente simples, foi necessário projetar muito bem as funções que o processador embarcado desempenharia. Isso porque o controle do robô é feita em tempo real, sendo então fundamental a execução imediata do movimento desejado. Nesse sentido, devido ao limitado processamento disponível no microcontrolador (necessário para redução de custo, menor consumo e redução de tamanho), optou-se pela transferência das tarefas que necessitassem de maior poder de processamento para dentro da rotina executada na estação base, deixando para a estação embarcada basicamente o trabalho de gerar os pulsos de PWM para os motores.

Em termos de hardware, optou-se pela aquisição de uma placa comercial em vez do desenvolvimento de um circuito dedicado devido a restrições de tempo. Além disso, é possível encontrar diversas soluções disponíveis no mercado que se encaixam perfeitamente nos requisitos do projeto, fazendo com que a segunda opção se torne mais custosa e trabalhosa que a primeira. A placa microcontroladora adotada neste projeto foi o modelo Fast Basic 8k da empresa Symphony, placa esta especialmente projetada para este tipo de aplicação (controle de robôs). Esta placa é baseada em um processador AVR da Atmel, o Atmel Mega8 de 16 MHz de freqüência máxima. Ele possui 8k de memória interna para programas, além de vários pinos de saída digital, necessários para o envio de sinais PWM para os motores. As dimensões da placa são bastante reduzidas (65 mm x 35 mm) ideal para a aplicação em questão. A seguir serão apresentados e discutidos outras características desta placa.

4.4.1 Detalhamento da placa

A placa Fast Basic 8k é um microcontrolador compacto projetado para sistemas de baixo custo e alta performance. Possui um processador Atmel Mega8 com freqüência 16 MHz, memória interna com 8kb para programas, 16 portas I/O digitais configuráveis individualmente como entrada/saída e 6 portas I/O analógicas de entrada.

A tensão de alimentação da placa é de 12 a 18 volts DC. Tal tensão pode ser facilmente obtida por meio de baterias comuns, não necessitando de fontes especiais ou conversores de tensão. Com as 16 portas digitais é possível controlar até 16 servo-motores, pois cada servo-motor necessita de apenas um sinal de controle. Esse número é superior ao necessário para este projeto, uma vez que o robô terá apenas 8 atuadores.

A freqüência de processamento (16 MHz) é suficiente para este projeto. Como não existe necessidade de altas velocidades de deslocamento, uma taxa de amostragem em torno de 1 kHz é o suficiente para garantir a estabilidade do sistema. Considerando-se o controle de 8 motores, poderia-se acessar cada motor com freqüência de 2 Mhz caso o processador realizasse apenas esta função. No entanto, existem outras funções a serem desempenhadas (como por exemplo a comunicação com o computador base) o que requer um pouco do processamento. Mesmo assim, pode-se perceber que a taxa de acesso será muito superior ao 1 kHz citado, mostrando que o processador trabalhará com grande folga.

O processador possui controle de interrupções, que serão utilizados para a geração dos pulsos de controle dos motores e da transmissão de dados via porta serial. No caso do controle dos motores, utilizar-se-á ainda os timers internos, que controlarão o tempo que o pulso fica em nível alto e baixo. Como mostraremos a seguir, a maioria dessas funcionalidades serão implementadas por software através de funções internas do compilador, facilitando muito a sua utilização.

4.4.2 Programação

O microcontrolador Fast Basic pode ser programado em diversas linguagens de programação. No entanto, existem duas que são mais utilizadas: Visual Basic (VB) e C. A linguagem VB é mais simples de utilizar, sendo muito popular entre programadores amadores e iniciantes em robótica. A linguagem C é mais complicada porém mais poderosa, utilizada por programadores mais experientes. A opção pela utilização do compilador VB se deu pela maior simplicidade da linguagem, sem prejudicar o desempenho do sistema. O compilador adotado foi o BASCOM-AVR (versão gratuita disponível no site http://www.mcselec.com/).

Como foi citado anteriormente, a função básica do controlador é a geração dos pulsos para controle dos motores da perna. Cada motor terá seu próprio sinal, ficando reservado 8 pinos de saída digital para essa função. Para entender como os pulsos foram gerados, vamos explicar como funciona o PWM.

A idéia do PWM é bem simples. Tendo-se uma onda quadrada, com período constante, altera-se a proporção do tempo em que o sinal fica em alto e baixo.

Numa onda quadrada normal, essa proporção é de 50%. No PWM, essa proporção é variável, podendo ser alterado de 0 até 100 %. Cada uma dessas proporções corresponde a uma posição angular do motor, sendo então necessário fazer a correspondência da posição do motor e a proporção do pulso em nível lógico alto.

Esses pulsos são facilmente obtidos através de uma função integrada do compilador BASCOM. Tal função chama-se **Config Servos**. Ela recebe dois parâmetros, primeiro o pino em que cada servo será ligado. Por exemplo, pode-se indicar o primeiro servo no pino B0, o segundo no D4, o terceiro no B3, etc (não é necessário manter uma ordem lógica). O segundo parâmetro é o que o compilador chama de "**Reload**", que corresponde ao tempo em ms que o controlador espera até atualizar o valor do pulso. O período do pulso depende do valor do **Reload** setado. O tempo que o pulso fica em alto é configurado através do vetor **Servo()**. Dentro do parêntese insere-se o número do servo que se deseja controlar, e o valor atribuído é o tempo em ms que o pulso deverá ficar em nível lógico alto.

Outra tarefa que o controlador tem que executar é a comunicação serial com a estação base. Tal comunicação é feita através dos pinos Tx e Rx presentes no controlador, especialmente dedicados para esta função. A comunicação serial é feita através da leitura e envio de dados para estes pinos, sendo funções básicas da linguagem. Foi necessário habilitar buffers de envio e recebimento de dados para evitar perdas de bytes, problema enfrentado ao testar o protótipo.

O programa então realiza os seguintes passos:

1. Lê o comando enviado pela estação base ao microcontrolador através da comunicação serial. Tal comando já está fracionado, não sendo necessário grande processamento por parte do embarcado. Um sinal de confirmação de recebimento dos dados é enviado de volta para a base.

2. Encaminha para o respectivo motor a nova posição desejada. Isso é feito através da modulação do pulso que é enviado para o servo-motor. O controle da posição é feito internamente ao servo, por isso independe do microcontrolador.

3. Envia um sinal de término do movimento a base.

4. Aguarda nova instrução para execução, e reinicia o ciclo.

4.5 Comunicação entre os controladores

Na inicialização do software 'remoto' (TUGASoft), detectam-se as portas de comunicação do computador, e escolhe-se uma porta do tipo serial (RS-232). O

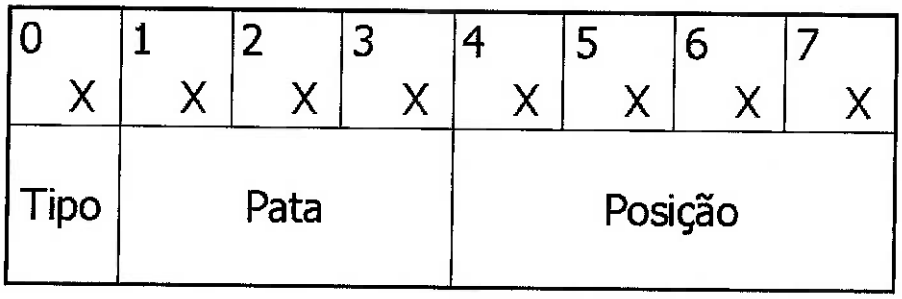


Figura 4.7: Formato dos bytes enviados ao controlador embarcado

programa torna-se proprietário dessa porta, só a liberando quando o software é finalizado.

O envio de comandos é liberado apenas se a estação embarcada tiver sinalizado estar pronta para receber novas instruções. As informações estão em pacotes de bytes, e a memória do microcontrolador foi um fator limitante que determinou o formato dos comandos enviados. Ao invés do envio de instruções como 'ande um passo para a frente' ou 'execute um ciclo para virar para a esquerda', os bytes contêm a identificação da pata a ser movimentada e o seu deslocamento angular. A figura 4.7 ilustra o formato adotado.

Os bytes contêm três informações:

1. Tipo de movimento a ser realizado: o controlador embarcado precisa saber se deve movimentar uma pata de cada vez (0) ou aguardar o recebimento de 4 comandos para mexer as quatro patas de uma só vez ('remar', 1).

2. A pata a ser movimentada: as patas foram numeradas, de 0 a 3, seguindo a mesma ordem da figura 3.1. Foram ocupados 3 bits para caso se possibilitar o envio de comandos separados para cada motor.

3. A posição para a qual a pata deve ser movimentada: para o controlador gerar o pulso PWM correspondente.

Capítulo 5

Conclusão

Como grande parte dos projetos, o TUGA foi se adaptando às condições que foram aparecendo pelo caminho. Inicialmente, pretendia-se fazê-lo com motores de passo, o que resultou em uma solução plausível, porém de alto custo. A sustentação do robô com a massa do motor exigia um alto torque o que resultava na necessidade de um motor de alto torque, consequentemente de maior dimensão e de maior massa. Assim, buscou-se utilizar outro tipo de motor, que possuísse melhor relação peso/potência e conseqüentemente viabilizasse a sua utilização e escolheu-se o servomotor de aeromodelo para acionamento do protótipo.

O servomotor de aeromodelo trouxe consigo a diminuição nos componentes inicialmente necessários para o controle, pois dispensa o uso de sensores externos devido à presença de um controle analógico de posição interno. Dessa forma, o sistema de controle embarcado passou a não mais necessitar da capacidade de processamento antes imaginada. As limitações quanto ao peso do robô também implicaram a utilização de um novo sistema de controle. Viu-se que, nesse caso, seria muito mais vantajoso utilizar um microcontrolador que um computador. Além de ser menor, mais leve e de mais fácil instalação, sua capacidade de processamento se mostrou mais adequada ao problema em questão.

No projeto de software optou-se por carregar mais o sistema remoto, que passará a interpretar os comandos do operador. Dessa forma, o sistema embarcado apenas receberá os comandos individuais para cada pata e porá os respectivos motores em movimento, cabendo ao sistema remoto a tradução do comando do operador em movimentos das patas. Além disso, a flexibilidade antes proporcionada pelo computador embarcado quanto à comunicação foi perdida, necessitando-se escolher dentre formas mais simples de comunicação, como, por exemplo, em rádio-freqüência. Finalmente, o microcontrolador, devido à utilização de servomotores, não foi mais programado em C++, mas sim em Basic. A vantagem de

se utilizar esta linguagem é que a geração de pulsos para os servomotores torna-se muito mais facilitada do que utilizando C++, com comandos já próprios para isso.

O objetivo principal do projeto foi atingido. O robô consegue se movimentar sem cair, não há problemas no mecanismo das patas que causem travamentos, ele executa quatro movimentos básicos e o operador pode acompanhar a movimentação do robô pelo monitor em tempo real. No entanto, há vários pontos que poderiam ser melhorados, muitos dos quais não foram realizados pelo grupo ter idealizado um projeto completo muito grande. Dentre inúmeros pontos, destacam-se:

1. Alimentação embarcada por baterias: são necessários dois níveis de tensão diferentes, 6V para os motores e 12V (até 18V) para o controlador. A fonte de 6V deve suportar uma corrente relativamente alta, pois os motores verticais precisam de torque para sustentar o robô.

2. Comunicação por rádio-freqüência (RF): existem módulos de RF que transmitem os sinais no padrão RS-232, que poderiam ser conectados nas saídas dos controladores embarcado e remoto (computador). É preciso que esses módulos trabalhem em duas freqüências diferentes para haver duas vias de comunicação entre os controladores (RX e TX). Juntamente com a alimentação embarcada, a comunicação por RF eliminaria todos os fios que atrapalham e limitam a movimentação do robô.

3. Calibração das patas: os comandos de posição enviados para o controlador embarcado foram pré-determinados experimentalmente, e não correspondem necessariamente ao movimento calculado na parte de controle. Isso porque os motores estão posicionados de forma diferente e o robô não é exatamente simétrico e, assim, o modelo utilizado não corresponde exatamente ao protótipo construído. Um estudo melhor do protótipo melhoraria os movimentos executados eliminando, por exemplo, o pequeno deslocamento lateral que ocorre quando o robô se movimenta para a frente.

4. Estudo do atrito entre as patas e o chão: o escorregamento durante os movimentos poderia ser diminuído, proporcionando movimentos mais suaves e eficientes.

O grupo espera que outros alunos se interessem pela continuação do projeto, que foi muito gratificante e divertido fazer, e ainda pode ser bastante explorado.

Bibliografia

[1] ARAKAWA,A. et al. Control of a Quadruped Robot using Double Crank-Slider Mechanism

[2] ZHOU, D. et al. An efficient foot-force distribution algorithm for quadruped walking robots. Robotica, v.18, p. 403-413, 2000.

[3] ZIELINSKA, T. et al. Multifuncional walking quadruped. Robotica, v.20, p. 585-593, 2002.

[4] RIDDERSTRÖM, C. et al. Quadruped posture control based on simple force distribution - a notion and a trial. Proceedings of the 2001 IEEE/RSJ, p.2326-2331, 2001.

[5] MA, S. et al. Omnidirectional Static Walking of a Quadruped Robot. IEEE Transactions on Robotics, 2005

[6] PAN, J. Measurment and Control of Attitudes of Quadruped Robots.

[7] PACK, D. J. et al. A Simplified Forward Gait Control for a Quadruped Walking Robot.

[8] MARTINS FILHO, L. S. et al. Locomotion control of a four legged robot embedding real-time reasoning in the force distrbution. Robotics and Autonomous Systems 32, p. 219-235, 2000.

[9] UMESH, K. N. Dexterous mechanisms for robot locomotion. Mech. Mach. Theory, v.33, n. 8, p.1153-1165, 1998.

[10] Notas de aula PMR2490 - Sistemas de informação, Prof.Marcos R. P. Barreto, 2^o semestre 2004.

[11] Notas de aula PCS2408 - Fundamentos de Engenharia de Software, Prof.Kechi Hirama, 1^o semestre 2005.

Apêndice A

Especificação de Casos de Uso

ESPECIFICAÇÃO DE CASOS DE USO PARA *CONTROLE DE UM ROBÔ QUADRÚPEDE*

Sistema TUGASoft

USP::Poli::PMR2550/PTC2531

Versão *1.0*

Histórico das Revisões

Data	Versão	Descrição	Autor
27/06/05	0.0		
19/09/05	0.1		
09/12/05	1.0		

Índice

1 Apresentação

1.1 Objetivo

O objetivo deste documento é a especificação de requisitos para o conjunto de funcionalidades referido como controle de um robô quadrúpede.

2 Conceitos Gerais

2.1 Dicionário de conceitos

Não aplicável

3 *Inicializar – US001*

3.1 Breve Descrição

Coloca o robô na configuração incial pré-definida.

3.2 Atores

Este caso de uso é dirigido ao operador do robô quadrúpede.

3.3 Pré-Condições

- O robô deve estar energizado.

3.4 Fluxo de Eventos

3.4.1 Fluxo Básico

1. O ator solicita inicialização.
2. O sistema limpa as variáveis da memória.
3. O sistema mexe os motores.
4. O sistema finaliza o fluxo.

3.4.2 Fluxos Alternativos

Não aplicável.

3.4.3 Requerimentos Especiais

Não aplicável.

3.4.4 Pós-Condições

- O robô encontra-se na configuração pré-definida.

3.4.5 Pontos de Extensão

Nenhum.

4 Frente - US002

4.1 Breve Descrição

Destina-se à execução de movimento para frente.

4.2 Atores

Este caso de uso é dirigido ao operador do robô quadrúpede.

4.3 Pré-Condições

- O robô deve estar na configuração inicial pré-definida.

4.4 Fluxo de Eventos

4.4.1 Fluxo Básico

1. O ator solicita execução de movimento para frente e informa o número de ciclos a serem executados.
2. O sistema executa o movimento.
3. O sistema emite sinal de encerramento do movimento.
4. O sistema encerra o fluxo.

4.4.2 Fluxos Alternativos

Não aplicável.

4.4.3 Requerimentos Especiais

Não aplicável.

4.4.4 Pós-Condições

- O robô realizou um movimento para frente.

4.4.5 Pontos de Extensão

Nenhum.

5 Trás - US003

5.1 Breve Descrição

Destina-se à execução de movimento para trás.

5.2 Atores

Este caso de uso é dirigido ao operador do robô quadrúpede.

5.3 Pré-Condições

- O robô deve estar na configuração inicial pré-definida.

5.4 Fluxo de Eventos

5.4.1 Fluxo Básico

1. O ator solicita execução de movimento para trás e informa o número de ciclos a serem executados.
2. O sistema executa o movimento.
3. O sistema emite sinal de encerramento do movimento.
4. O sistema encerra o fluxo.

5.4.2 Fluxos Alternativos

Não aplicável.

5.4.3 Requerimentos Especiais

Não aplicável.

5.4.4 Pós-Condições

- O robô realizou um movimento para trás.

5.4.5 Pontos de Extensão

Nenhum.

6 *Vire esquerda - US004*

6.1 Breve Descrição

Destina-se à execução de movimento para virar para a esquerda.

6.2 Atores

Este caso de uso é dirigido ao operador do robô quadrúpede.

6.3 Pré-Condições

- O robô deve estar na configuração inicial pré-definida.

6.4 Fluxo de Eventos

6.4.1 Fluxo Básico

1. O ator solicita execução de movimento para virar para a esquerda e informa o número de ciclos a serem executados.
2. O sistema executa o movimento.
3. O sistema emite sinal de encerramento do movimento.
4. O sistema encerra o fluxo.

6.4.2 Fluxos Alternativos

Não aplicável.

6.4.3 Requerimentos Especiais

Não aplicável.

6.4.4 Pós-Condições

- O robô virou para a esquerda.

6.4.5 Pontos de Extensão

Nenhum.

7 *Vire direita - US005*

7.1 Breve Descrição

Destina-se à execução de movimento para virar para a direita.

7.2 Atores

Este caso de uso é dirigido ao operador do robô quadrúpede.

7.3 Pré-Condições

- O robô deve estar na configuração inicial pré-definida.

7.4 Fluxo de Eventos

7.4.1 Fluxo Básico

1. O ator solicita execução de movimento para virar para a direita e informa o número de ciclos a serem executados.
2. O sistema executa o movimento.
3. O sistema emite sinal de encerramento do movimento.
4. O sistema encerra o fluxo.

7.4.2 Fluxos Alternativos

Não aplicável.

7.4.3 Requerimentos Especiais

Não aplicável.

7.4.4 Pós-Condições

- O robô virou para a direita.

7.4.5 Pontos de Extensão

Nenhum.

8 Desativar - US006

8.1 Breve Descrição

Destina-se ao posicionamento do robô em configuração final pré-definida para desligamento do mesmo.

8.2 Atores

Este caso de uso é dirigido ao operador do robô quadrúpede.

8.3 Pré-Condições

- O robô deve estar na configuração inicial pré-definida.

8.4 Fluxo de Eventos

8.4.1 Fluxo Básico

1. O ator solicita o desligamento.
2. O sistema executa o movimento.
3. O sistema emite sinal de encerramento do movimento.
4. O sistema encerra o fluxo.

8.4.2 Fluxos Alternativos

Não aplicável.

8.4.3 Requerimentos Especiais

Não aplicável.

8.4.4 Pós-Condições

- O robô foi desativado.

8.4.5 Pontos de Extensão

Nenhum.

Apêndice B

Diagrama de classes

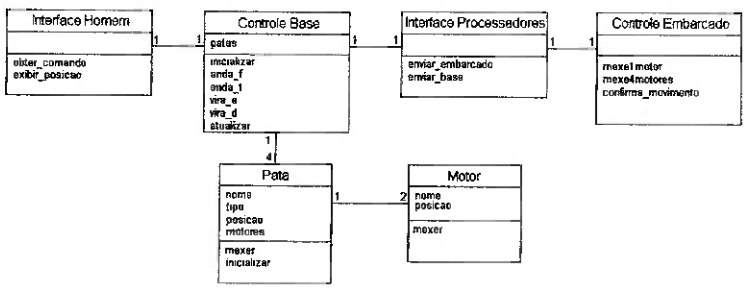


Figura B.1: Diagrama de classes do software

Apêndice C

Códigos fonte: controlador 'remoto'

Arquivo: Motor.java

```
import java.awt.*;

class Motor{
	double pos;		//posicao do motor, em graus, (-90,90)

	Motor(){
		pos = 0;
	}

	Motor(double p){
		pos = p;
	}

	void mexer(double delta){
		pos = pos + delta;
	}

	public String toString(){
		String s = "Motor posicao: "+pos;
		return s;
	}

	/*public static void main(String[] args){
		Motor m1 = new Motor();
		Motor m2 = new Motor(9.5);
		System.out.println(m1);
		System.out.println(m2);
	}*/

}


class Pata{
	int tipo;		//qual pata? 1=fe, 2=fd, 3=te, 4=td
	double pos;		//posicao da pata, em graus, (-90,90)
	Motor vert, hor;

	Pata(int t){
		tipo = t;
		pos = 0;
		vert = new Motor();
		hor = new Motor();
	}

	Pata(int t, double p){
		tipo = t;
		pos = p;
		vert = new Motor();
		hor = new Motor(p);
	}

	void mexer(double delta){
		pos = pos + delta;
		hor.pos = hor.pos + delta;
	}

	void inicializar(double a){
		pos = a;
		hor.pos = a;
	}

	public String toString(){
		String s = "*Pata " + tipo + " posicao: " + pos + "\n";
		s = s + hor.toString() + "\n";
```

```
            s = s + vert.toString();
            return s;
        }

        /*public static void main(String[] args){
            Pata p1 = new Pata();
            Pata p2 = new Pata(9.5);
            System.out.println(p1);//p1.imprimir();
            System.out.println(p2);//p2.imprimir();
        }*/

}


class CBase{
        private double a = 15;      //metade do comprimento, em cm
        private double b = 10;      //metade da largura, em cm
        private double f = 5;       //insercao da pata
        private double n = 5;       //comprimento da projecao horizontal da pata
        private double phi=30;      //angulo das patas da frente
        private double teta=-55;    //angulo das patas de tras
        private double delta=50;    //variacao angular para frente
        private double alfa=25;     //variacao angular da remada

        Pata fe, fd, te, td;        //olhando robo de cima
        Comunica com;               //comunicacao por porta serial
        Espera e;                   //insere pausas temporais

        //Construtor
        CBase(){
            fe = new Pata(1);
            fd = new Pata(2);
            te = new Pata(3);
            td = new Pata(4);
            e = new Espera();
            com = new Comunica();
        }

        //inicializacao
        void inicializar(){
            fe.inicializar(phi);
            fd.inicializar(phi);
            te.inicializar(teta);
            td.inicializar(teta);
            com.enviar(fe,fd,te,td);
        }

        //andar para a frente
        void anda_f(int npassos, MonCanvas m){
            for(int i=1; i<=npassos; i=i+1){
                td.mexer(delta); com.enviar(td);
                m.update(m.getGraphics());//e.espera();
                fd.mexer(delta); com.enviar(fd);
                m.update(m.getGraphics());//e.espera();
                td.mexer(-alfa); fd.mexer(-alfa);
                te.mexer(-alfa); fe.mexer(-alfa);
                com.enviar(td,fd,te,fe);
                m.update(m.getGraphics());//e.espera();
                te.mexer(delta); com.enviar(te);
                m.update(m.getGraphics());//e.espera();
                fe.mexer(delta); com.enviar(fe);
                m.update(m.getGraphics());//e.espera();
                td.mexer(alfa-delta); fd.mexer(alfa-delta);
                te.mexer(alfa-delta); fe.mexer(alfa-delta);
                com.enviar(td,fd,te,fe);
                m.update(m.getGraphics());
                //System.out.println(i);
            }
```

```
}

//andar para tras
void anda_t(int npassos, MonCanvas m){
       for(int i=1; i<=npassos; i=i+1){
              td.mexer(-alfa+delta); fd.mexer(-alfa+delta);
              te.mexer(-alfa+delta); fe.mexer(-alfa+delta);
              com.enviar(td,fd,te,fe);
              m.update(m.getGraphics());//e.espera();
              fe.mexer(-delta); com.enviar(fe);
              m.update(m.getGraphics());//e.espera();
              te.mexer(-delta); com.enviar(te);
              m.update(m.getGraphics());//e.espera();
              td.mexer(alfa); fd.mexer(alfa);
              te.mexer(alfa); fe.mexer(alfa);
              com.enviar(td,fd,te,fe);
              m.update(m.getGraphics());//e.espera();
              fd.mexer(-delta);com.enviar(fd);
              m.update(m.getGraphics());//e.espera();
              td.mexer(-delta);com.enviar(td);
              m.update(m.getGraphics());
              //System.out.println(i);
       }
}

//virar para a esquerda
void vira_e(int npassos, MonCanvas m){
       for(int i=1; i<=npassos; i=i+1){
              td.mexer(delta); com.enviar(td);
              m.update(m.getGraphics());//e.espera();
              fd.mexer(delta); com.enviar(fd);
              m.update(m.getGraphics());//e.espera();
              td.mexer(-alfa); fd.mexer(-alfa);
              te.mexer(alfa); fe.mexer(alfa);
              com.enviar(td,fd,te,fe);
              m.update(m.getGraphics());//e.espera();
              fe.mexer(-delta); com.enviar(fe);
              m.update(m.getGraphics());//e.espera();
              te.mexer(-delta); com.enviar(te);
              m.update(m.getGraphics());//e.espera();
              td.mexer(alfa-delta); fd.mexer(alfa-delta);
              te.mexer(delta-alfa); fe.mexer(delta-alfa);
              com.enviar(td,fd,te,fe);
              m.update(m.getGraphics());
              //System.out.println(i);
       }
}

//virar para a direita
void vira_d(int npassos, MonCanvas m){
       for(int i=1; i<=npassos; i=i+1){
              te.mexer(delta); com.enviar(te);
              m.update(m.getGraphics());//e.espera();
              fe.mexer(delta); com.enviar(fe);
              m.update(m.getGraphics());//e.espera();
              te.mexer(-alfa); fe.mexer(-alfa);
              td.mexer(alfa); fd.mexer(alfa);
              com.enviar(td,fd,te,fe);
              m.update(m.getGraphics());//e.espera();
              fd.mexer(-delta); com.enviar(fd);
              m.update(m.getGraphics());//e.espera();
              td.mexer(-delta); com.enviar(td);
              m.update(m.getGraphics());//e.espera();
              te.mexer(alfa-delta); fe.mexer(alfa-delta);
              td.mexer(delta-alfa); fd.mexer(delta-alfa);
              com.enviar(td,fd,te,fe);
              m.update(m.getGraphics());
              //System.out.println(i);
```

```
            }
        }

    /*
        void desativar();
    */

        public String toString(){
            String s = "*CBase*" + "\n";
            s = s + fe.toString() + "\n";
            s = s + fd.toString() + "\n";
            s = s + te.toString() + "\n";
            s = s + td.toString();
            return s;
        }

        //calculo de sin e cos em graus
        static double sind(double angled){
            return(Math.sin(Math.toRadians(angled)));
        }

        static double cosd(double angled){
            return(Math.cos(Math.toRadians(angled)));
        }

        //atualizacao da interface grafica
        void desenhar(Graphics gr,int larg,int alt,int esc){
            int dx=larg-(int)(esc*b);
            int dy=alt-(int)(esc*a);
            gr.setColor(Color.white);
            //gr.setColor(Color.black);                    //para fundo branco
            //Centro de Massa
            gr.fillOval(larg-3,alt-3,6,6);
            //corpo
            gr.drawRect(dx,dy,(int)(b*esc*2),(int)(a*esc*2));
            double[] Xpes={dx-n*esc*cosd(fe.pos), dx+(2*b+n*cosd(fd.pos))*esc,
                dx-n*cosd(te.pos)*esc, dx+(2*b+n*cosd(td.pos))*esc};
            double[] Ypes={dy+(f-n*sind(fe.pos))*esc, dy+(f-n*sind(fd.pos))*esc,
                dy+(2*a-f-n*sind(te.pos))*esc, dy+(2*a-f-n*sind(td.pos))*esc};
            //pata da frente, esquerda
            gr.drawLine(dx,dy+(int)(f*esc),(int)Xpes[0],(int)Ypes[0]);
            //pata da frente, direita
            gr.drawLine(dx+(int)(2*b*esc),dy+(int)(f*esc),(int)Xpes[1],
                                                        (int)Ypes[1]);
            //pata de tras, esquerda
            gr.drawLine(dx,dy+(int)((2*a-f)*esc),(int)Xpes[2],
                                                        (int)Ypes[2]);
            //pata de tras, direita
            gr.drawLine(dx+(int)(2*b*esc),dy+(int)((2*a-f)*esc),
                                                (int)Xpes[3],(int)Ypes[3]);
            //centro de massa
            gr.setColor(Color.yellow);
            //gr.setColor(Color.blue);        //para fundo branco
            //diagonais de apoio
            gr.drawLine((int)Xpes[0],(int)Ypes[0],(int)Xpes[3],
                                                        (int)Ypes[3]);
            gr.drawLine((int)Xpes[1],(int)Ypes[1],(int)Xpes[2],
                                                        (int)Ypes[2]);
        }

    /*  public static void main(String[] args){
            CBase c = new CBase();
            c.imprimir();
            c.inicializar();
            c.imprimir();
            c.anda_f(2);
        }
```

```
*/
}


//Classe de pausas temporais
class Espera extends Thread{
	int d;

	Espera(){
		d=1000;
	}

	void espera(){
		try{
			sleep(d);
			}catch (InterruptedException k){}
	}

	void espera2(){
		try{
			sleep(100);
			}catch (InterruptedException k){}
	}
}
```

Arquivo: Interface.java

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

/* O canvas (onde se desenha) */
class MonCanvas extends Canvas implements MouseListener{
	final static int larg = 250;	//largura do fundo do desenho
	final static int alt = 250;	//altura do fundo do desenho
	final static int escala = 10;	//escala do desenho

	CBase g = null;	//gerencia os graficos
	Desenho1 des;

	MonCanvas(CBase gg,Desenho1 l){
		g = gg;
		des = l;
		addMouseListener(this);
	}

	public void paint(Graphics gr){
		fundo(gr);
		g.desenhar(gr,larg,alt,escala);
		des.posicoes();
	}

	public void update(Graphics gr){
		gr.clearRect(0,0,708,734);
		paint(gr);
	}

	//fundo da tela
	void fundo(Graphics gr){
		gr.setColor(Color.black);		//cor
		//gr.setColor(Color.white);	//imprimir com fundo branco
		gr.fillRect(0,0,2*larg,2*alt);	//formato e tamanho
	}
```

```
        public void mousePressed(MouseEvent e){}
        public void mouseReleased(MouseEvent e){}

        // Focaliza no Canvas quando o mouse esta sobre o desenho
        public void mouseEntered(MouseEvent e){}
        public void mouseExited(MouseEvent e){}

        //detecta o elemento mais proximo
        public void mouseClicked(MouseEvent e){}

}


/* A janela GUI */
class Desenho1 extends JFrame implements WindowListener{
        MonCanvas fenetreDessin;                //canvas para desenhar
        private JPanel p, o;                    //painel principal
        //Label que da as informacoes
        Label info=new Label("Robo desligado",Label.CENTER);
        private Label[] k=new Label[13];        //vetor de Labels fixos
        Label[] pos=new Label[2];
        TextField Jr[]=new TextField[1];
        String p1, p2, p3, p4;
        int passos = 0;
        int cont = 1;
        private Button ini, des, paraf, parat, parae, parad;        //botoes


        //Construtor
        Desenho1(){
                super("TUGASoft Versao 0.1");                   //nome do Frame
                //escolha do Border Layout
                getContentPane().setLayout(new BorderLayout());
                setSize(800,640);                               //tamanho da janela
                //initializacao do Canvas
                fenetreDessin=new MonCanvas(new CBase(),this);
                fenetreDessin.setVisible(true);                 //torna-se visivel


                //JPanel de entradas
                p=new JPanel();
                GridBagLayout gridbag = new GridBagLayout();  //escolha de Layout
                p.setLayout(gridbag);


                //Restricao para as linhas
                //para breaklines centralizados
                GridBagConstraints gbc = new GridBagConstraints();
                gbc.gridwidth = GridBagConstraints.REMAINDER;
                //para breakline no comeco da linha
                GridBagConstraints gbc2 = new GridBagConstraints();
                gbc2.gridwidth = GridBagConstraints.REMAINDER;
                gbc2.anchor=GridBagConstraints.LINE_START;
                //para penultimo elemento em comeco de linha
                GridBagConstraints gbce = new GridBagConstraints();
                gbce.gridwidth = GridBagConstraints.RELATIVE;
                gbce.anchor=GridBagConstraints.LINE_START;
                //para elemento no final da linha
                GridBagConstraints gbcd = new GridBagConstraints();
                gbcd.gridwidth = GridBagConstraints.REMAINDER;
                gbcd.anchor=GridBagConstraints.LINE_END;

                //Titulo do Panel p
                k[0] = new Label("TUGASoft");                   //titulo
                k[0].setFont(new Font("Monotype Corsiva", Font.BOLD , 34));  //fonte
                gridbag.setConstraints(k[0], gbc);              //linha centralizada
                p.add(k[0]);                                    //coloca no Panel
```

```
Label enter=new Label("");                          //pula uma linha
gridbag.setConstraints(enter, gbc);
p.add(enter);

//Secao Posicoes
k[1] = new Label("Posicoes");
//escolha de fonte
k[1].setFont(new Font("Century Gothic", Font.BOLD, 25));
gridbag.setConstraints(k[1], gbc2);
p.add(k[1]);

//Patas dianteiras
k[2] = new Label("Frente Esquerda        Frente Direita");
k[2].setFont(new Font("Times New Roman", Font.ROMAN_BASELINE, 16));
gridbag.setConstraints(k[2],gbc);
p.add(k[2]);
p1 = Double.toString(fenetreDessin.g.fe.pos);
p2 = Double.toString(fenetreDessin.g.fd.pos);
pos[0] = new Label(p1+" graus                          "+p2+" graus ");
pos[0].setFont(new Font("Times New Roman", Font.ROMAN_BASELINE, 16));
gridbag.setConstraints(pos[0],gbc2);
p.add(pos[0]);

//Patas traseiras
k[3] = new Label("Atras Esquerda          Atras Direita");
k[3].setFont(new Font("Times New Roman", Font.ROMAN_BASELINE, 16));
gridbag.setConstraints(k[3],gbc2);
p.add(k[3]);
p3 = Double.toString(fenetreDessin.g.te.pos);
p4 = Double.toString(fenetreDessin.g.td.pos);
pos[1]=new Label(p3+" graus                          "+p4+" graus ");
pos[1].setFont(new Font("Times New Roman", Font.ROMAN_BASELINE, 16));
gridbag.setConstraints(pos[1],gbc2);
p.add(pos[1]);


//Secao Comandos
k[4]=new Label("Comandos");
k[4].setFont(new Font("Century Gothic", Font.BOLD , 25));
gridbag.setConstraints(k[4], gbc2);
p.add(k[4]);

//Linha de entrada do no. de passos
o=new JPanel();
o.setLayout(gridbag);            //reinicializa `o`
k[5] =  new Label("Passos   ");  //Label de passos
k[5].setFont(new Font("Palatino", Font.PLAIN , 16));
o.add(k[5]);                     //adiciona k[5] ao Panel 'o'
Jr[0] = new TextField("1",1);
o.add(Jr[0]);
gridbag.setConstraints(o,gbc2);  //adiciona o Panel `o` ao Panel `p`
p.add(o);


//////////////////////////////////////////////////////
//  BOTOES  //
//////////////

//Inicializar
ini=new Button(" Inicializar ");
ini.setSize(ini.getMinimumSize());
ActionListener mni=new ActionListener(){
      public void actionPerformed(ActionEvent e){
            info.setText("Robo inicializado");
            fenetreDessin.g.inicializar();  //inicializacao e desenha
            fenetreDessin.update(fenetreDessin.getGraphics());
            iniciou();
```

```
        }
};
ini.addActionListener(mni);
gridbag.setConstraints(ini,gbce);
p.add(ini);

//Desativar
des=new Button("   Desativar   ");
des.setSize(des.getMinimumSize());
ActionListener mnds=new ActionListener(){
        public void actionPerformed(ActionEvent e){
                info.setText("Robo desativado");
                //fenetreDessin.g.desativar();   //desativacao e desenha
                //fenetreDessin.update(fenetreDessin.getGraphics());
                desabilitou();
        }
};
des.addActionListener(mnds);
gridbag.setConstraints(des,gbcd);
des.setEnabled(false);
p.add(des);


//Para frente
paraf=new Button(" Frente ");
paraf.setSize(paraf.getMinimumSize());
ActionListener mnf=new ActionListener(){
        public void actionPerformed(ActionEvent e){
                info.setText("Um passo para frente");
                lireDate()                  //le o numero de passos
                //anda e desenha
                fenetreDessin.g.anda_f(passos,fenetreDessin);
        }
};
paraf.addActionListener(mnf);
gridbag.setConstraints(paraf,gbc);
paraf.setEnabled(false);
p.add(paraf);


//Para a esquerda
parae=new Button(" Esquerda ");
parae.setSize(parae.getMinimumSize());
ActionListener mne=new ActionListener(){
        public void actionPerformed(ActionEvent e){
                info.setText("Virando para a esquerda");
                lireDate();                         //le o numero de passos
                //vira e desenha
                fenetreDessin.g.vira_e(passos,fenetreDessin);
        }
};
parae.addActionListener(mne);
gridbag.setConstraints(parae,gbce);
parae.setEnabled(false);
p.add(parae);


//Para a direita
parad=new Button("   Direita   ");
parad.setSize(parad.getMinimumSize());
ActionListener mnd=new ActionListener(){
        public void actionPerformed(ActionEvent e){
                info.setText("Virando para a direita");
                lireDate();                         //le o numero de passos
                //vira e desenha
                fenetreDessin.g.vira_d(passos,fenetreDessin);
        }
};
```

```
        parad.addActionListener(mnd);
        gridbag.setConstraints(parad,gbcd);
        parad.setEnabled(false);
        p.add(parad);


        //Para tras
        parat=new Button("   Atras   ");
        parat.setSize(paraf.getMinimumSize());
        ActionListener mnt=new ActionListener(){
              public void actionPerformed(ActionEvent e){
                    info.setText("Um passo para tras");
                    lireDate();                        //le o numero de passos
                    //anda e desenha
                    fenetreDessin.g.anda_t(passos,fenetreDessin);
              }
        };
        parat.addActionListener(mnt);
        gridbag.setConstraints(parat,gbc);
        parat.setEnabled(false);
        p.add(parat);

        getContentPane().add(p,"East");  //localizacao do Panel p no Frame
        getContentPane().add(info,"South");
        getContentPane().add(fenetreDessin);    //Canvas no meio do Frame


        final Button bQuit=new Button("Fim");          //botao para sair
        final Button bSur=new Button("Sobre");         //botao sobre
        ActionListener menu=new ActionListener(){
              public void actionPerformed(ActionEvent e){          //para sair
                    click(0);
              }
        };
        ActionListener menu1=new ActionListener(){                 //sobre
              public void actionPerformed(ActionEvent e){
                    click(1);
              }
        };
        bQuit.addActionListener(menu);
        bSur.addActionListener(menu1);
        JPanel o=new JPanel(new FlowLayout(FlowLayout.RIGHT));
        o.add(bSur);
        o.add(bQuit);
        getContentPane().add(o,"North");        //adiciona no alto a direita

        addWindowListener(this);                //adiciona listener
        setVisible(true);
}//fim do construtor


//Gestao de janelas
public void windowOpened(WindowEvent e) {}
public void windowClosed(WindowEvent e) {}
public void windowIconified(WindowEvent e) {}
public void windowDeiconified(WindowEvent e) {}
public void windowActivated(WindowEvent e) {}
public void windowDeactivated(WindowEvent e) {}


//Icone de fechamento de janela
public void windowClosing(WindowEvent e){
      dispose();
      System.exit(0);
}

////////////////////////////
//Para habilitar os botoes //
```

```
////////////////////////////
public void iniciou() {
	des.setEnabled(true);
	paraf.setEnabled(true);
	parat.setEnabled(true);
	parae.setEnabled(true);
	parad.setEnabled(true);
}

public void desabilitou() {
	des.setEnabled(false);
	paraf.setEnabled(false);
	parat.setEnabled(false);
	parae.setEnabled(false);
	parad.setEnabled(false);
}
////////////////////

public void posicoes(){
	p1 = Double.toString(fenetreDessin.g.fe.pos);
	p2 = Double.toString(fenetreDessin.g.fd.pos);
	pos[0].setText(p1+" graus                    "+p2+" graus ");
	p3 = Double.toString(fenetreDessin.g.te.pos);
	p4 = Double.toString(fenetreDessin.g.td.pos);
	pos[1].setText(p3+" graus                    "+p4+" graus ");
}


//cria Popup 'Sobre' e saida do aplicativo
public void click(int i){
	PopupFactory fac=new PopupFactory();
	final Popup p;
	JPanel o=new JPanel(new GridLayout(5,1));
	Label [] sur=new Label[4];
	sur[0]=new Label("TUGASoft, versao 0.1",Label.CENTER);
	sur[0].setFont(new Font("Times New Roman", Font.PLAIN , 18));
	sur[1]=new Label ("Equipe:",Label.CENTER);
	sur[2]=new Label("Camila Shirota e Ricardo Shirota Filho");
	sur[2].setFont(new Font("Sans Serif", Font.BOLD , 15));
	sur[3]=new Label("All rights reserved");
	Button sortir=new Button("Sair");
	final Desenhol h=this;

	for(int cont=0;cont<4;cont++)o.add(sur[cont]);
	o.add(sortir);
	o.setSize(o.getMinimumSize());
	o.setBackground(Color.white);
	p=fac.getPopup(null,o,(int)(getX()+((getWidth()-o.getWidth())/2)),
	   (int)(getY()+((getHeight()-o.getHeight())/2)));   //local du Popup

	ActionListener exit=new ActionListener(){  //botao para fechar pop-up
		public void actionPerformed(ActionEvent e){
			p.hide();
			h.setEnabled(true);
			fenetreDessin.g.com.fecharPorta();
		}
	};
	sortir.addActionListener(exit);

	switch (i){
		case 0: System.exit(0);
			fenetreDessin.g.com.fecharPorta();   //fecha porta serial
			break;
		case 1: h.setEnabled(false);
			p.show();
			break;
	}
}
```

```
        //le o numero de passos
        private boolean lireDate(){
                try{
                        passos=Integer.parseInt(Jr[0].getText());
                        if(passos<1 || passos>30){      //limita o numero de passos a 30
                                info.setText("O numero de passos deve ser entre 1 e 30");
                                throw new Exception();
                        }
                }catch(java.lang.NumberFormatException ee){
                        info.setText("O  numero  de  passos  devem  ser  numeros  inteiros
                                                                positivos");     //erros
                        return(true);
                }catch (Exception gg){ return(true);}
                return(false);                  //nao ha erros
        }


        //programa principal
        public static void main(String[] args){
                Desenho1 f=new Desenho1();
                f.fenetreDessin.g = new CBase();
        }


}
```

Arquivo: Comunica.java

```
import java.io.*;
import java.util.*;
import javax.comm.*;

class Comunica implements SerialPortEventListener{
        static Enumeration portList;
        static CommPortIdentifier portId;
        static SerialPort serialPort;
        static OutputStream outputStream;       //onde se escreve a msg a enviar
        InputStream inputStream;                //onde se recebe a msg do embarcado
        static boolean temSerial = false;
        static boolean remotoOK = true;         //se pode enviar mais msgs
        private int msg;                        //o byte a enviar
        Espera e = new Espera();                //para inserir pausas

        //Construtor
        Comunica(){
                portList = CommPortIdentifier.getPortIdentifiers();
                portId = (CommPortIdentifier) portList.nextElement();

                if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                        temSerial = true;
                        if (portId.getName().equals("COM1")) {
                                try {
                                        serialPort  =  (SerialPort)  portId.open("TUGASoft",
                                                                                2000);
                                } catch (PortInUseException e) {}
                                try {
                                        System.out.println(portId.getName());
                                        //para escrever na porta
                                        outputStream = serialPort.getOutputStream();
                                        //para ler da porta
                                        inputStream = serialPort.getInputStream();
```

```
                        } catch (IOException e) {}
                        try {
                                serialPort.addEventListener(this);
                        } catch (TooManyListenersException e) {}
                        serialPort.notifyOnDataAvailable(true);
                        try {
                                serialPort.setSerialPortParams(9600,
                                        SerialPort.DATABITS_8,
                                        SerialPort.STOPBITS_1,
                                        SerialPort.PARITY_NONE);
                        } catch (UnsupportedCommOperationException e) {}
                }
        }
        if(!temSerial)
                System.out.println("Nao tem porta serial no sistema!");
}

//envia String para controlador embarcado
void enviar(String s){
        remotoOK = false;
        while(!remotoOK){
                try {
                        outputStream.write(s.getBytes());
                } catch (IOException e){
                        System.out.println("Problemas na transmissao!");
                }
                e.espera();
        }
}

//envia char para controlador embarcado
void enviar(char c){
        remotoOK=false;
        msg=(byte)c;
        manda();
}

//envia o movimento de uma pata
void enviar(Pata p){
        enviar(p, true);
}

//envia o movimento de 4 patas
void enviar(Pata p1, Pata p2, Pata p3, Pata p4){
        remotoOK = false;
        enviar(p1, false);
        enviar(p2, false);
        enviar(p3, false);
        enviar(p4, false);
        //System.out.println("Acabei um remo!");
}

//codifica um byte para envio
void enviar(Pata p, boolean flag){
        remotoOK = false;
        //se flag, msg=(10000000)=4 patas, se nao, msg=0=1 pata
        msg = (flag ? 128:0);

        switch(p.tipo){
                case 1:
                case 2:
                        switch((int)p.pos){
                                case(30-50): msg = msg + 1; break;
                                case(30-25): msg = msg + 2; break;
                                case(30): msg = msg + 3; break;
                                case(30+25): msg = msg + 4; break;
                                case(30+50): msg = msg + 5; break;
                        }
```

```
                        msg = ((p.tipo==2) ? msg+16 : msg);
                        break;
                case 3:
                case 4:
                        switch((int)p.pos){
                                case(-55-50): msg = msg + 1; break;
                                case(-55-25): msg = msg + 2; break;
                                case(-55): msg = msg + 3; break;
                                case(-55+25): msg = msg + 4; break;
                                case(-55+50): msg = msg + 5; break;
                        }
                        msg = ((p.tipo==3) ? msg+32 : msg+48);
                        break;
        }
        manda();
}

//envia o byte para o controlador embarcado
void manda(){
        System.out.print("envio "+(byte)msg+" ");
        while(!remotoOK){
                try {
                        outputStream.write((byte)msg);
                } catch (IOException e){
                        System.out.println("Problemas na transmissao!");
                }
        }
}

//recebe bytes enviados pelo controlador embarcado
public void serialEvent(SerialPortEvent event) {
        if(event.getEventType()== SerialPortEvent.DATA_AVAILABLE){
                byte[] readBuffer = new byte[100];
                try {
                        while(inputStream.available()>0) {
                                int numBytes = inputStream.read(readBuffer);
                        }
                        if(readBuffer[0]==(byte)'8'){
                                remotoOK = true;
                                System.out.println("recebi "+readBuffer[0]);
                        }
                        System.out.print(readBuffer[0]+" ");
                } catch (IOException e) {}
        }
}

//fecha a porta serial
public void fecharPorta() {
        if (serialPort != null) {
                try {               // close the i/o streams
                        outputStream.close();
                        inputStream.close();
                } catch (IOException e) {
                        System.err.println(e);
                }
        // Close the port
        serialPort.close();
        }
        System.out.println("Porta serial fechada!");
}


//para teste da classe
public static void main (String[] args){
        Comunica c = new Comunica();
        Pata p1 = new Pata(1,30);Pata p2 = new Pata(2,30);
        Pata p3 = new Pata(3,-55);Pata p4 = new Pata(4,-55);
        c.enviar(p1,p2,p3,p4);
```

```
            c.e.espera(); c.e.espera();
            p1.mexer(-25);
            c.enviar(p1,p2,p3,p4);
            c.e.espera(); c.e.espera();
            p1.mexer(25);
            c.enviar(p1,p2,p3,p4);
            c.e.espera(); c.e.espera();
            p1.mexer(-25);
            c.enviar(p1,p2,p3,p4);
            c.e.espera(); c.e.espera();
            p1.mexer(25);
            c.enviar(p1,p2,p3,p4);
            //c.e.espera();
            //Pata p1 = new Pata(3,-55); c.enviar(p1);
            //p1.mexer(25); c.enviar(p1);
            //p1.mexer(25); c.enviar(p1);
            /*c.enviar('1');
            c.enviar('2');
            c.enviar('3');
            c.enviar('4');*/
            c.fecharPorta();
        }

}
```

Apêndice D

Códigos fonte: análise de estabilidade no Matlab

Arquivo: CM.m

```
%%Para visualizacao do centro de massa (CM)
%%em relacao ao posicionamento das patas
%%durante movimento

%%Unidades: m, rad

%%CM na origem do sistema cartesiano (0,0)
%%comprimento longitudinal: 2*a
%%            transversal:  2*b
%%distancia ateh 'ombro':   f
%%projecao da pata no chao: n ('braco')

%%Patas: vista superior
%%     1 - frente, esquerda
%%     2 - frente, direita
%%     3 - atras, esquerda
%%     4 - atras, direita

close all
clear all

a = 0.25; b = 0.15;
f = 0.05;
n = 0.05;
phi1 = 30*pi/180; phi2 = phi1;
teta3 = -55*pi/180; teta4 = teta3;

desenha([a,b,f,n], [phi1,phi2,teta3,teta4]);

delta = 50*pi/180;          %%passo angular, em %%(k*1,8)%% rad
alfa = 25*pi/180;           %%primeira remada

%distancia(delta, alfa, [a,b,f,n]);
grafico([phi1,phi2,teta3,teta4], delta, alfa, [a,b,f,n])

%%Para frente
%anda_f([4 2 3 1], [phi1,phi2,teta3,teta4], delta, alfa, [a,b,f,n])
%%[phi,teta,delta,alfa]
%%[10, -60, 70, 20] - 6,1mm
%%[30, -60, 50, 20] - 7,6mm
%%[30, -55, 50, 25] - 9,3mm

%%Virar esquerda
vira_e([4 2 1 3], [phi1,phi2,teta3,teta4], delta, alfa, [a,b,f,n])
%%[phi,teta,delta,alfa]
%%[30, -60, 45, 20] - 5,8mm
%%[30, -60, 50, 30 ou 20] - 7,6mm
%%[30, -55, 50, 25] - 9,3mm
```

Arquivo: anda_f.m

```
function [] = anda_f(ordem, angulos, delta, alfa, dimensoes)

%%Simula robo andando para frente: anda 2 patas, rema, anda outras duas patas, rema
%%      ordem:       vetor de tamanho 4, ordem de movimento das patas
%%     angulos:      vetor de angulos em rad, [phi1, phi2, teta3, teta4]
%%      delta:       passo angular para frente em rad
%%       alfa:       passo angular da primeira remada em rad

for i=1:4
      angulos(ordem(i)) = angulos(ordem(i)) + delta;
      desenha(dimensoes, angulos);
```

```
        if i==2
              angulos = angulos - alfa;
              desenha(dimensoes, angulos);
        end
end

angulos = angulos - (delta - alfa);
desenha(dimensoes, angulos);
```

Arquivo: vira_e.m

```
function [] = vira_e(ordem, angulos, delta, alfa, dimensoes)

%%Simula robo andando para frente: anda 2 patas, rema, anda outras duas patas, rema
%%      ordem:       vetor de tamanho 4, ordem de movimento das patas
%% angulos:  vetor de angulos em rad, [phi1, phi2, teta3, teta4]
%%      delta:       passo angular para frente em rad
%%       alfa:       passo angular da primeira remada em rad

for i=1:4
       switch(ordem(i))
       case{1,3}
              angulos(ordem(i)) = angulos(ordem(i)) - delta;
              desenha(dimensoes, angulos);
       case{2,4}
              angulos(ordem(i)) = angulos(ordem(i)) + delta;
              desenha(dimensoes, angulos);
       end

       if i==2
              angulos = angulos + [alfa -alfa alfa -alfa];
              desenha(dimensoes, angulos);
       end
end

angulos = angulos + [delta-alfa -delta+alfa delta-alfa -delta+alfa];
desenha(dimensoes, angulos);
```

Arquivo: desenha.m

```
function [] = desenha(dimensoes, angulos)

%%Desenha o robozinho e as patas
%%Mostra posicao relativa do CM e diagonais de apoio

a = dimensoes(1); b = dimensoes(2); f = dimensoes(3); n = dimensoes(4);
phi1 = angulos(1); phi2 = angulos(2);
teta3 = angulos(3); teta4 = angulos(4);

Xcorpo = [-b b; b b; b -b; -b -b];      %uma reta por linha da matriz
Ycorpo = [a a; a -a; -a -a; -a a];
% Xombro = [-b; b; -b; b];
% Yombro = [a-f; a-f; -a+f; -a+f];
Xpes = [-b-n*cos(phi1); b+n*cos(phi2); -b-n*cos(teta3); b+n*cos(teta4)];
Ypes = [a-f+n*sin(phi1); a-f+n*sin(phi2); -a+f+n*sin(teta3); -a+f+n*sin(teta4)];
Xpatas = [-b -b-n*cos(phi1); b b+n*cos(phi2); b b+n*cos(teta4);...
            -b -b-n*cos(teta3)];
Ypatas = [a-f a-f+n*sin(phi1); a-f a-f+n*sin(phi2); -a+f -a+f+n*sin(teta4);...
            -a+f -a+f+n*sin(teta3)];
%sqrt((Ypatas(:,2)-Ypatas(:,1)).^2+(Xpatas(:,2)-Xpatas(:,1)).^2)
figure;
plot(Xcorpo, Ycorpo, '-k', 0,0,'ok');axis off;
%axis([-0.2 0.2 -0.3 0.3]);                    %CUIDADO!!!! ESCALAS DIFERENTES!!!
```

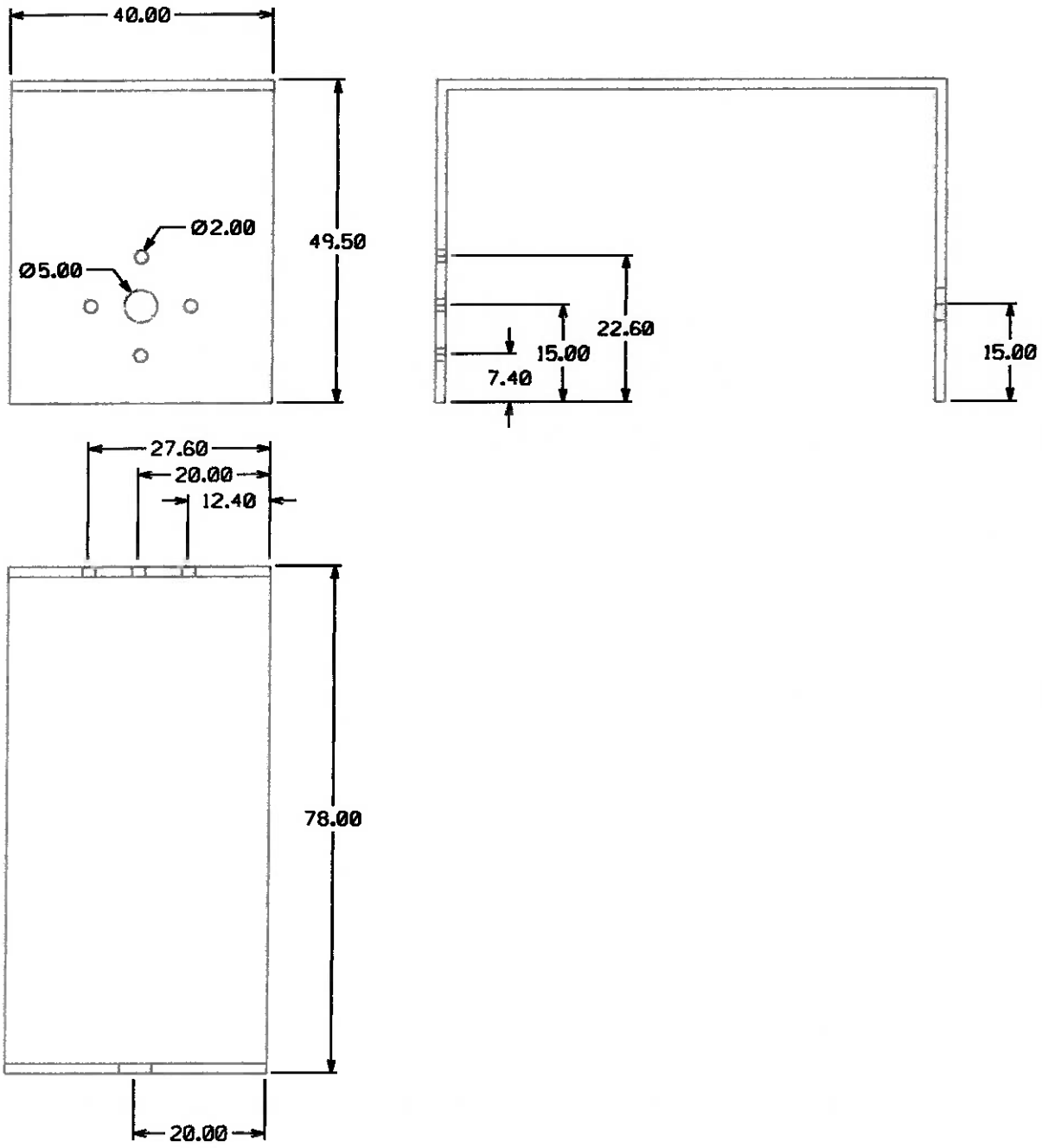
```
axis equal;
%grid;
hold; plot(Xpatas(1,:), Ypatas(1,:), '-k');axis off;
plot(Xpatas(2,:), Ypatas(2,:), '-k');axis off;
plot(Xpatas(3,:), Ypatas(3,:), '-k');axis off;
plot(Xpatas(4,:), Ypatas(4,:), '-k');axis off;

plot([Xpes(1) Xpes(4)], [Ypes(1) Ypes(4)], '-b');axis off;
plot([Xpes(2) Xpes(3)], [Ypes(2) Ypes(3)], '-b');axis off;
hold off;
xlabel('x (m)'); ylabel('y (m)');

prova14 = (Xpes(1)*(Ypes(1)-Ypes(4))-Ypes(1)*(Xpes(1)-Xpes(4)))^2;
prova14 = sqrt(prova14/((Xpes(1)-Xpes(4))^2+(Ypes(1)-Ypes(4))^2))
prova23 = (Xpes(2)*(Ypes(2)-Ypes(3))-Ypes(2)*(Xpes(2)-Xpes(3)))^2;
prova23 = sqrt(prova23/((Xpes(2)-Xpes(3))^2+(Ypes(2)-Ypes(3))^2))
```

Arquivo: gráfico.m

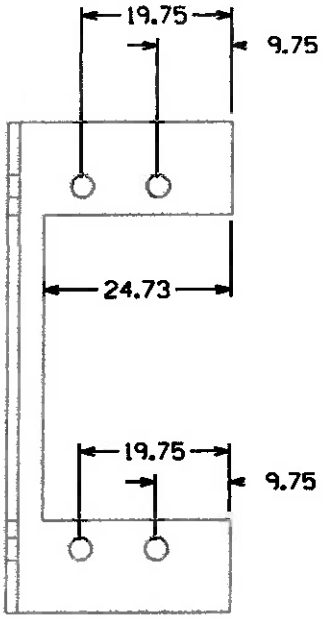
```
function [] = grafico(angulos, delta, alfa, dimensoes)

a = dimensoes(1); b = dimensoes(2); f = dimensoes(3); n = dimensoes(4);
angulos = angulos*180/pi;
phi1 = angulos(1); phi2 = angulos(2);
teta3 = angulos(3); teta4 = angulos(4);
delta = delta*180/pi;
alfa = alfa*180/pi;

% a = 0.25; b = 0.15;
% f = 0.05;
% n = 0.05;
% phi1 = 30; phi2 = phi1;
% teta3 = -55; teta4 = teta3;
%
% delta = 50;       %%passo angular, em %%(k*1,8)%% rad
% alfa = 25;        %%primeira remada

distancia(delta*pi/180, alfa*pi/180, [a,b,f,n]);
hold;
%%para frente
% plot(teta3, phi2,'*r');
% plot(teta3+delta, phi2,'*b');
% plot(teta3, phi2+delta,'*g');
% plot(teta3+delta-alfa, phi2-alfa,'*k');
% plot(teta3-alfa, phi2+delta-alfa,'*k');
% plot(teta3-alfa+delta, phi2+delta-alfa,'*y');

%%virando a esquerda
% plot(teta3, phi2, '*r');
% plot(teta3+delta, phi2, '*b');
% plot(teta3, phi2+delta, '*g');
% plot(teta3+delta-alfa, phi2+alfa, '*k');teta3+delta-alfa, phi2+alfa,
% plot(teta3+alfa, phi2+delta-alfa, '*k');teta3+alfa, phi2+delta-alfa,
% plot(teta3+delta-alfa, phi2+alfa-delta, '*y');
% plot(teta3+alfa-delta, phi2+delta-alfa, '*y');
```

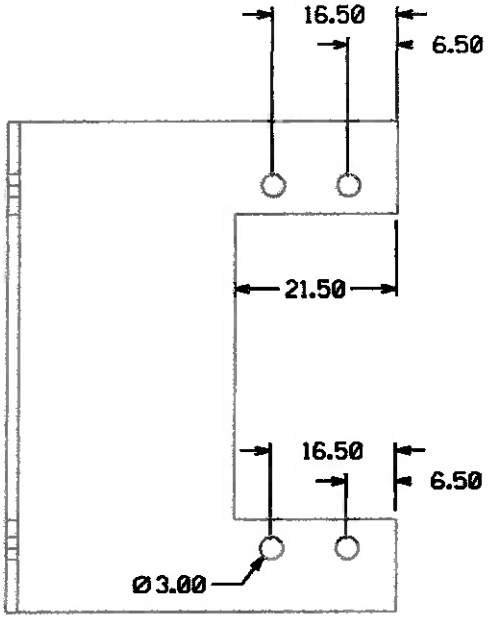
Apêndice E

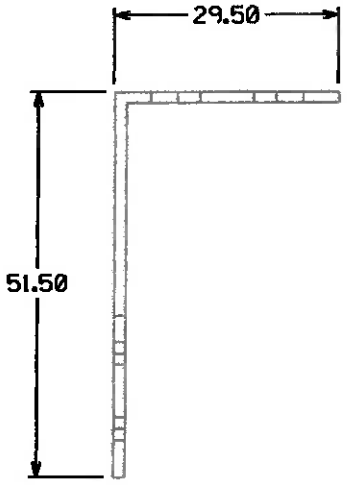
Desenhos técnicos



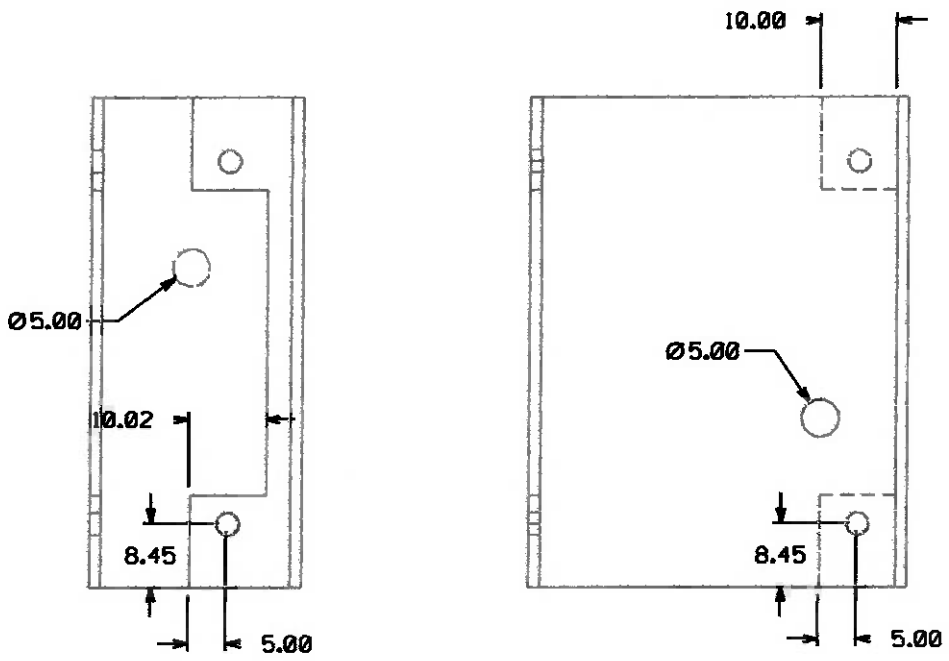
Des.: 01	SUPORTE DA CAIXINHA		2005
PMR 2501	Trabalho de Conclusão de Curso		Escala 1:1
EPUSP	No. de peças: 4	Material: aluminio	Unidade: mm

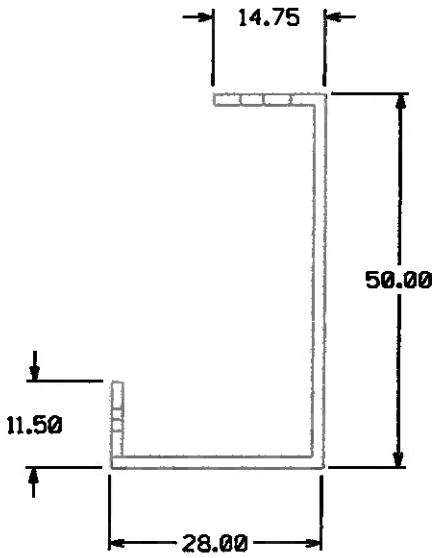




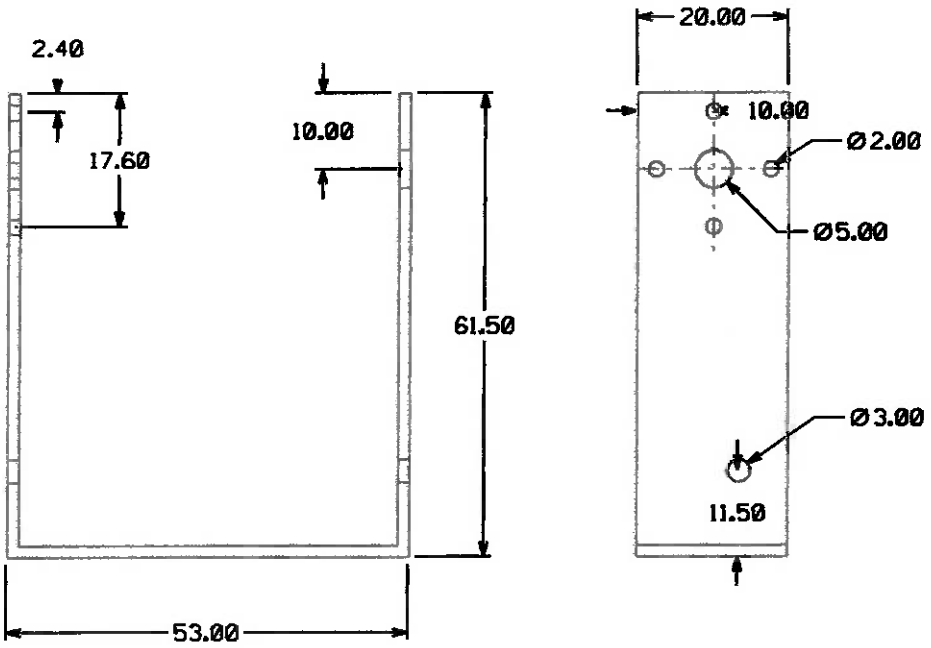


Des.: 02	CAIXINHA - Suporte dos motores		2005
PMR 2501	Trabalho de Conclusão de Curso		Escala 1:1
EPUSP	No. de peças: 4	Material: aluminio	Unidade: mm

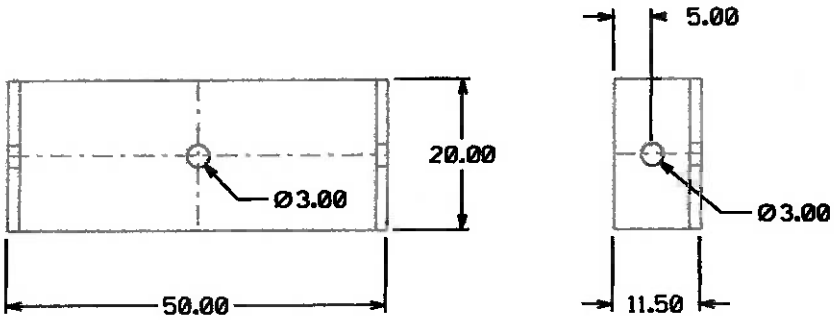




Des.: 03	CAIXINHA - Peça para eixos de suporte		2005
PMR 2501	Trabalho de Conclusão de Curso		Escala 1:1
EPUSP	No. de peças: 4	Material: alumínio	Unidade: mm

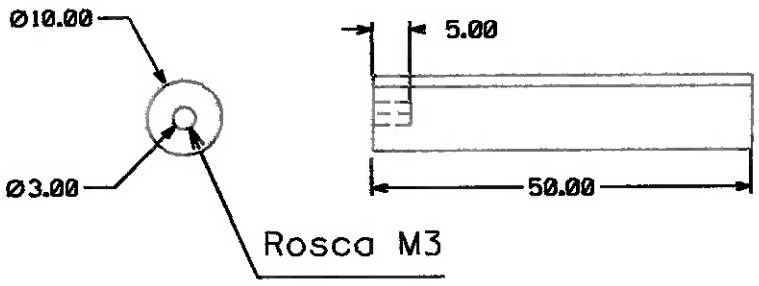


Des.: 04	PERNA - Peça para fixar no motor		2005
PMR 2501	Trabalho de Conclusão de Curso		Escala 1:1
EPUSP	No. de peças: 4	Material: alumínio	Unidade: mm

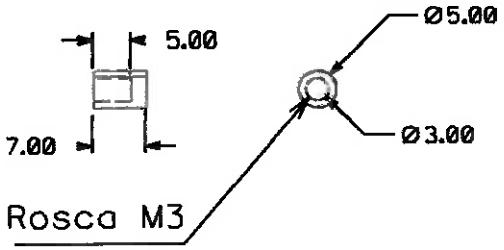


Des.: 05	PERNA - Peça de união		2005
PMR 2501	Trabalho de Conclusão de Curso		Escala 1:1
EPUSP	No. de peças: 4	Material: alumínio	Unidade: mm

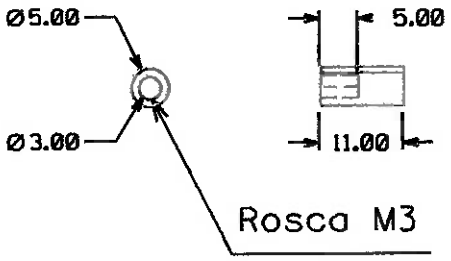
PERNA - Peça que apóia no chão



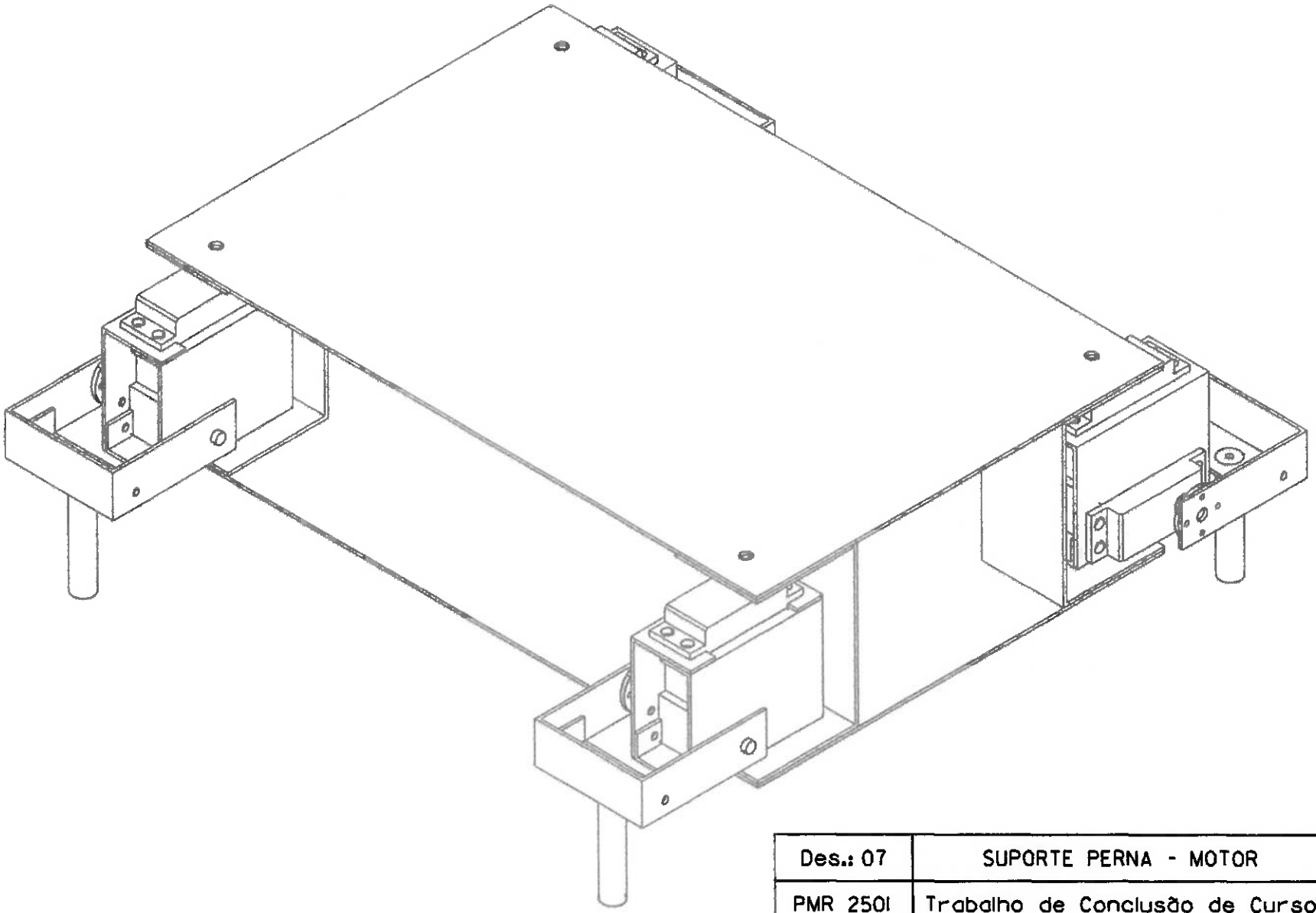
Eixo de suporte para a perna



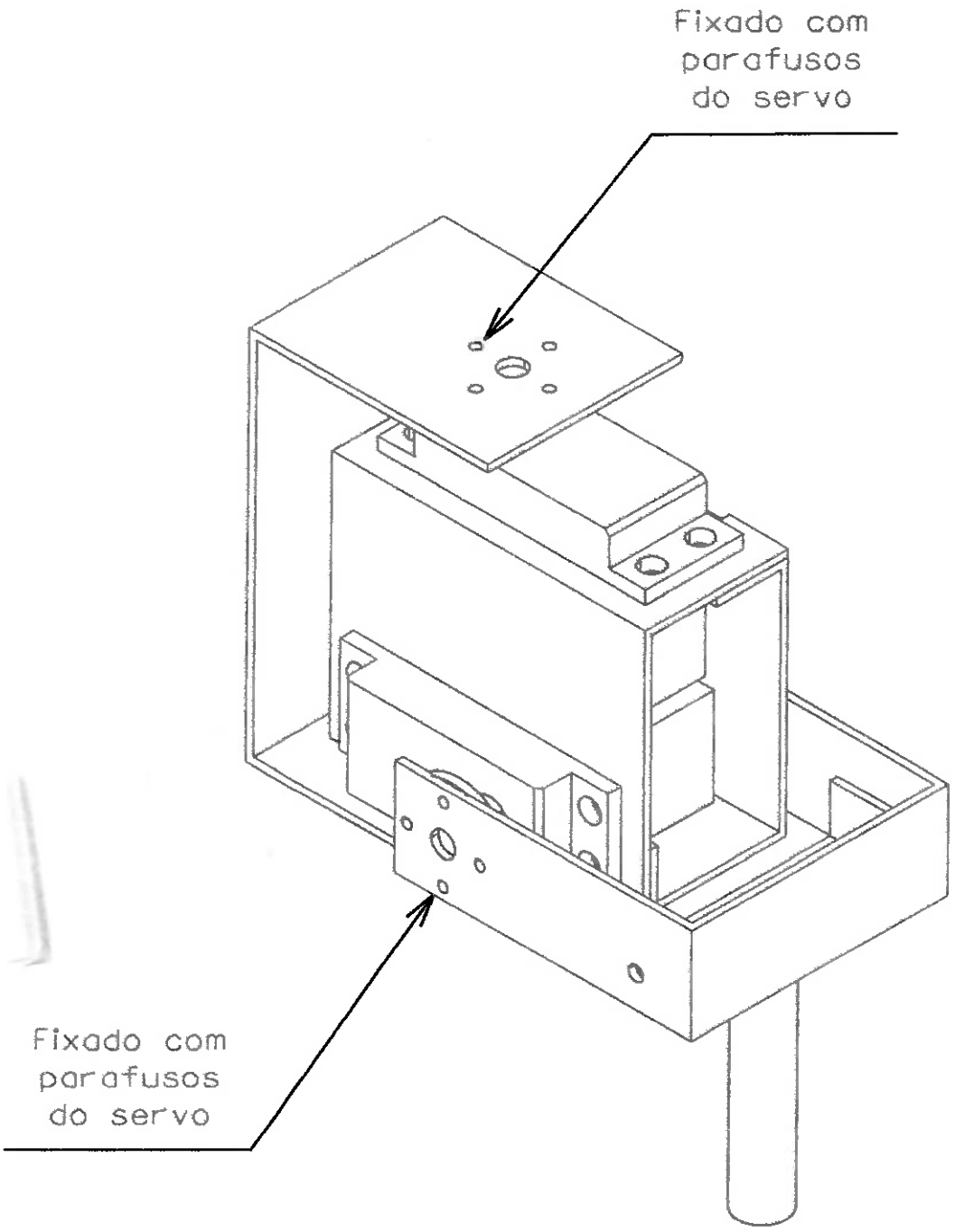
Eixo de suporte para a caixinha



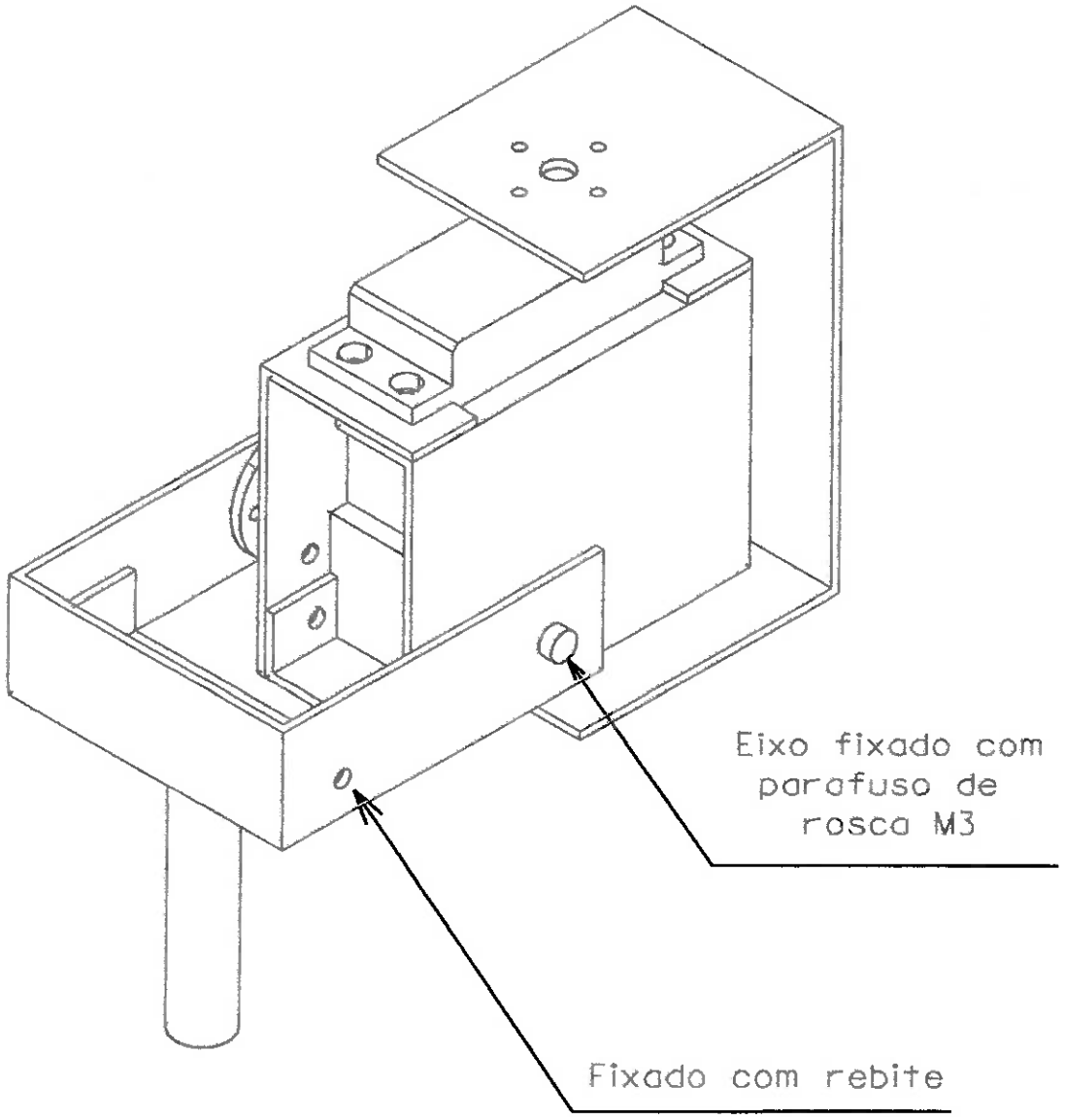
Des.: 06	Peças cilíndricas		2005
PMR 2501	Trabalho de Conclusão de Curso		Escala 1:1
EPUSP	No. de peças: 4	Material: alumínio	Unidade: mm



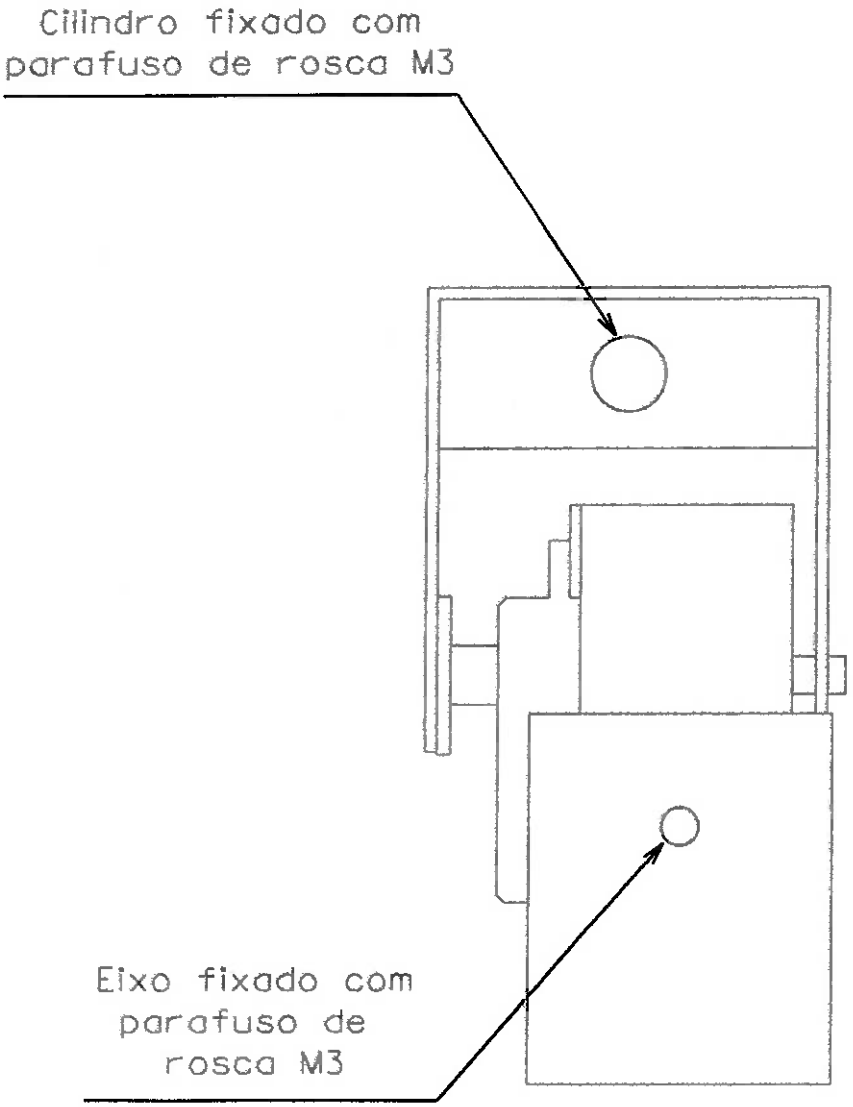
Des.: 07	SUPORTE PERNA - MOTOR	2005
PMR 2501	Trabalho de Conclusão de Curso	Escala 1:2
EPUSP	Número de peças: 4	Unidade: mm



Des.: 08	DESENHO DE MONTAGEM - Vista isométrica I	2005
PMR 2501	Trabalho de Conclusão de Curso	Escala 1:1
EPUSP	TUGA - Robô quadrúpede	Unidade: mm



Des.: 09	DESENHO DE MONTAGEM - Vista isométrica 2	2005
PMR 2501	Trabalho de Conclusão de Curso	Escala 1:1
EPUSP	TUGA - Robo quadrúpede	Unidade: mm



Des.: 10	DESENHO DE MONTAGEM - Vista inferior	2005
PMR 2501	Trabalho de Conclusão de Curso	Escala 1:1
EPUSP	TUGA - Robô quadrúpede	Unidade: mm