DESENVOLVIMENTO DE UM *ROBOT* OMNIDIRECCIONAL PARA FINS DIDÁCTICOS USANDO O *KIT LEGO MINDSTORM*

José Gonçalves * **Paulo Costa** ** **Paulo Moreira** ***

* *goncalves@ipb.pt, Instituto Politécnico de Bragança*
** *paco@fe.up.pt*
*** *amoreira@fe.up.pt*
Faculdade de Engenharia da Universidade do Porto

Resumo:
Este artigo descreve como pode ser desenvolvido rapidamente um *Robot* Omnidireccional para fins didácticos, utilizando o *Kit Lego Mindstorm*, apresentando um papel importantíssimo na prototipagem (Ferrari *et al.* 2002), tendo como vantagens modularidade, conectividade de peças e permitindo reutilização das mesmas.
A diversidade de áreas envolvidas (mecânica, electrónica, controlo...) ilustra bem a interdisciplinaridade da Robótica, tornando-a cada vez mais importante a nível didáctico. De todas as maneiras as pessoas que estão a dar os primeiros passos nesta área podem desmoralizar rapidamente, para que isto não aconteça, são necessárias ferramentas que lhes permitam obter resultados rapidamente, tendo neste aspecto o Kit *Lego Mindstorm* um papel primordial.

1. INTRODUÇÃO

O *Robot* diferencial é provavelmente o *Robot* móvel mais usado, sendo representado na figura 1. O *Robot* diferencial é composto por dois motores, cujos veios passam pelo mesmo eixo, sendo o seu movimento controlado variando independentemente a velocidade de cada um dos motores.

A estrutura do *Robot* diferencial impede que sejam feitos movimentos de translação segundo o eixo que passa pelos veios dos motores (Dudek and Jenkin 2000). Para colmatar esta deficiência do *Robot* diferencial surgiu o *Robot* omnidireccional, permitindo deslocações em todas as direcções (Kalmár-Nagy *et al.* 2002). Para garantir a característica da omnidireccionalidade é necessário que as rodas usadas não tenham atrito no sentido do veio do motor, o que impediria deslocações segundo esse eixo.

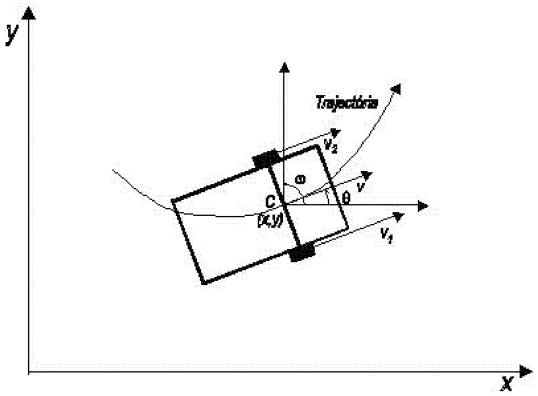

Figura 1. *Robot Diferencial*

Como podemos comprovar da observação da geometria de um *robot* omnidireccional, representada na figura 2, as velocidades $\dot{x}$, $\dot{y}$ e $\dot{\theta}$ variam com a velocidades lineares V_1, V_2 e V_3, baseando-se na equação 1 (Kalmár-Nagy *et al.* 2002).

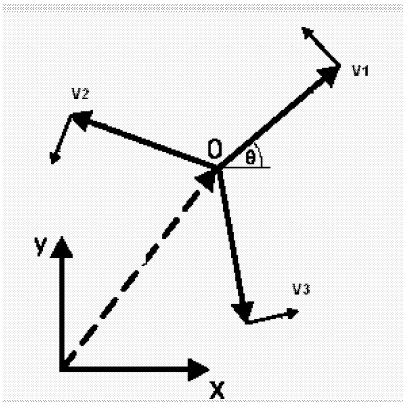

Figura 2. Geometria de um Robot Omnidireccional.

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} -sen(\theta) & cos(\theta) & L \\ -sen(\frac{\pi}{3} - \theta) & cos(\frac{\pi}{3} - \theta) & L \\ -sen(\frac{\pi}{3} + \theta) & cos(\frac{\pi}{3} + \theta) & L \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} \quad (1)$$

Da observação da equação 1 apenas não está explícito na figura 2 o parâmetro L, sendo a distância entre o ponto da roda que assenta no chão e o ponto no qual se intersectam os eixos que passam pelos veios dos motores. O cálculo do controlador e da odometria foram baseados no modelo apresentado na equação 1. Trata-se de um modelo bastante complexo o que implica que o cuidados a ter relativamente à realização do *software* sejam maiores que em ouitras configurações de *robots* pois pode-se facilmente esgotar a memória do dispositivo usado para efectuar o controlo.

2. PORQUÊ USAR *LEGO MINDSTORM*

Entende-se genericamente por *Lego* um conjunto de peças de plástico para construção de modelos mecânicos, sendo a palavra *Lego* uma marca registada (*The free dictionary* 2004). Quase todos nós fomos criados a fazer *Legos*, o que os torna uma ferramenta com a qual estamos muito familiarizados. Com peças *Lego* é possível efectuar rapidamente diferentes configurações de robots, sendo uma motivação extra para as pessoas que estão a dar os primeiros passos no mundo da Robótica, em especial no mundo da Robótica móvel. As peças *Lego* permitem conectividade, suprimindo a necessidade de usar parafusos ou cola, tornando-se bastante limpo efectuar construções. Nesta situação particular esta vantagem não foi totalmente aproveitada, dado não ser fácil efectuar conectividade entre peças *Lego* com ângulos diferentes de múltiplos de 90º, sendo esta condição necessária para a construção de um *Robot* omnidireccional. Também se torna uma ferramenta ecológica pois apesar de o plástico não ser um material fácil de reciclar, as peças *Lego* nunca são inutilizadas, consequentemente nunca constituem um desperdício, podendo ser usadas ano após ano. Em suma os *Lego Mindstorm* são uma boa ferramenta educacional para o estudo de alguns princípios da Robótica. É uma ferramenta reutilizável, modular, podendo a mesma peça ser diferentes coisas consoante a aplicação que lhe queiramos dar, motivando muito as pessoas que a usam e sendo apresentada a um custo relativamente acessível.

3. DESENVOLVIMENTO DE HARDWARE

O protótipo realizado consiste num *Robot* omnidireccional realizado com o *Kit Lego MindStorm*, estando representado na figura 3, os sensores e actuadores usados são descritos nas subsecções seguintes.

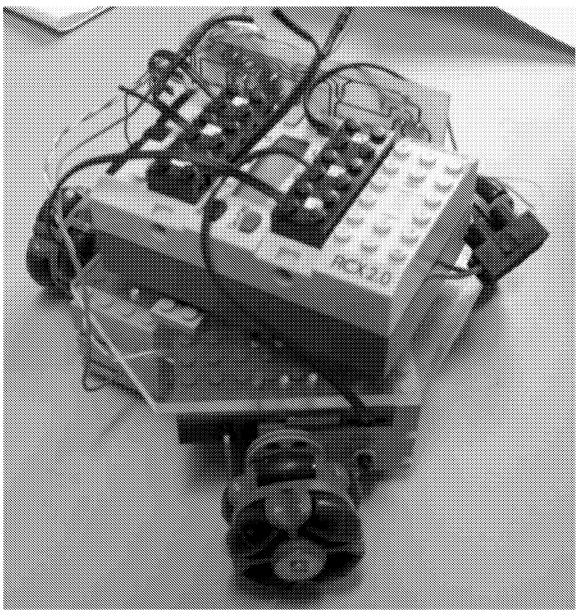

Figura 3. Protótipo realizado usando o *Kit Lego Mindstorm*

3.1 Sensores

O sensor usado para efectuar a actualização da odometria são *encoders* incrementais baseados no esquema eléctrico da figura 4, sendo realizados por nós pois não são disponibilizados com o *Kit Lego Mindstorm*.

Sempre que interrompido o sinal de infravermelhos, o foto transístor deixa de estar polarizado, suprimindo-se a corrente na resistência Rt, consequentemente passando a ter a alimentação (E) na saída(Pallas-Areny and Webster 1991). O sensor apresentado funciona como um interruptor electrónico. O sinal de saída no colector do foto

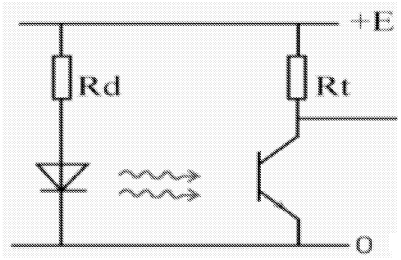

Figura 4. Circuito usado para o encoder

transístor é apresentado na figura 5, passando posteriormente por um comparador com histerese, convertendo-o numa onda quadrada.

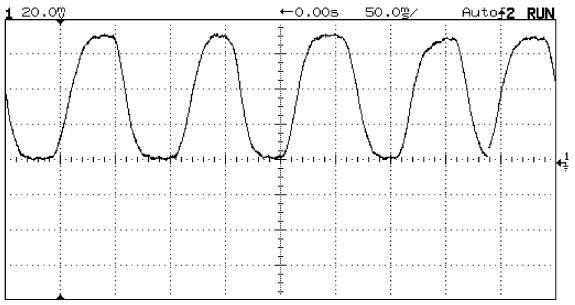

Figura 5. Sinal no colector do foto transístor

Associado ao veio do motor encontra-se uma folha de acetato representada na figura 6, a qual permite interromper o sinal de infravermelhos, a cada impulso gerado corresponde um deslocamento angular determinístico.

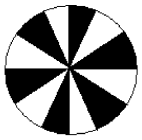

Figura 6. Folha de acetato associada ao veio do motor

Os impulsos registados, são convertidos em velocidade linear do ponto associado às rodas, usando-se uma aproximação de primeira ordem (assume-se que a velocidade é constante durante um *sampling time*). Para o efeito multiplica-se os impulsos por um factor que permite saber a velocidade, estando representada na equação 2

$$\frac{Distancia\_percorrida}{Tempo} = factor\_roda \times \frac{}{impulsos\_num\_sampling\_time} \quad (2)$$

O "factor roda"converte impulsos em velocidade linear, seguindo várias etapas. Em primeiro lugar os impulsos são convertidos em deslocamento linear, correspondendo 12 a um deslocamento de *Pi*Diâmetro da Roda*. Após a obtenção do deslocamento este é divido pelo tempo durante o qual foram lidos impulsos, obtendo-se velocidade linear, tal como é explicitado na equação 3, a qual representa o factor roda.

$$\frac{\pi \times diametro}{(total\_de\_impulsos\_por\_volta \times sampling\_Time)} \quad (3)$$

3.2 Actuadores

Os actuadores usados para o Protótipo são motores DC de 9 Volt, tendo caixa redutora interna. A caixa redutora permite que a velocidade angular da roda que lhe está associada seja mais baixa, aumentando o binário disponível (Bagnall 2002). Aos motores DC estão acopladas umas rodas especiais para omnidireccionais, representadas na figura 7, cuja particularidade é terem pouco atrito no sentido do veio do Motor, permitindo deslizamento(L.Williams *et al.* 2002).

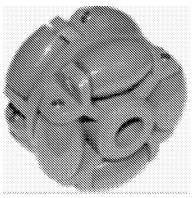

Figura 7. Roda especial para *Robots* Omnidireccionais

4. SOFTWARE DE CONTROLO EM JAVA

"It is amazing that they managed to squeeze a multithreading environment into what is effectively 14 kilobytes of memory. And that still leaves you with 16 kilobytes for your programs."

- Simon Ritter, Java Technology Evangelist, Sun Microsystems

4.1 Porquê Java

A escolha da linguagem de programação recaiu sobre a linguagem *Java* pelo facto de ter uma *API* bastante mais rápida que a alternativa *NQC*, tendo-se durante o projecto de desenvolvimento do *Software* de Controlo do *Robot* melhorado para sensivelmente 10 vezes mais rápido as funções *sin*(), *cos*() e *sqrt*(), sendo o código feito pelo Professor Paulo Costa (Costa 2003). Também se tornou uma motivação extra usar *Java* para efectuar *Software* de Controlo, usando *Threads* para emular os tradicionais *Timers*.

O código deste programa consiste em duas *Threads* não sincronizadas (*Thread Omni e Thread Tmot*), sendo os eventos periódicos gerados colocando as *Threads* em modo *sleep* durante um certo tempo determinístico, após este tempo as *Threads* acordam e efectuam todos os cálculos e operações voltando ao modo de *sleep*, emulando assim o funcionamento de um timer.

4.2 Thread Omni

A *thread* principal tem como objectivo o cálculo do Controlador e da Odometria baseando-se na equação 1. Esta *Thread* tem como nome *Omni*, à qual lhe está associado um *Sampling Time*, sendo escolhido de maneira a que as actualizações da odometria e do controlador sejam o mais rápidas possível. A escolha foi 200 ms pelo facto de após vários testes se verificar que processador demora entre 140 a 160 ms a efectuar estas duas funções. Para se ter alguma margem de manobra optou-se por um valor ligeiramente superior, majorando um pouco o tempo necessário para os cálculos.

O código base usado para esta Thread foi o seguinte:

```
   {
//inicialização de variáveis
while(1==1)
    {
// efectuar actualização da odometria
odoUpdate();
//executar Controlador
Controlador();
    }
//deltaT=sampling time
LastTime=LastTime+deltaT;
SleepTime=LastTime-
   (int)System.currentTimeMillis();
if (SleepTime>0)
      {
     try {
Thread.currentThread().sleep(SleepTime);
         }
   catch (InterruptedException ie) {}
      }
    }
```

4.2.1. Controlador O controlador tem como função definir qual a velocidade a que os motores devem rodar, de maneira a que o *Robot* rode com uma determinada velocidade angular $\dot{\theta}$ e se desloque com uma velocidade linear $\dot{x}$ e $\dot{y}$

O controlador foi optimizado desenvolvendo as funções $sin(\theta+\frac{\pi}{3})$, $cos(\theta+\frac{\pi}{3})$, $cos(\theta-\frac{\pi}{3})$ e $sin(\theta-\frac{\pi}{3})$, usando as formulas:

$$\sin(a+b) = \sin(a)\times\cos(b)+\cos(a)\times\sin(b) \quad (4)$$

$$\sin(a-b) = \sin(a)\times\cos(b)-\cos(a)\times\sin(b) \quad (5)$$

$$\cos(a+b) = \cos(a)\times\cos(b)-\sin(a)\times\sin(b) \quad (6)$$

$$\cos(a-b) = \cos(a)\times\cos(b)+\sin(a)\times\sin(b) \quad (7)$$

O objectivo de desenvolver estas funções trigonométricas foi o de o controlador se tornar computacionalmente menos pesado, pois é muito menos dispendioso a nível de tempo de cálculo calcular apenas $sin(\theta)$ e $cos(\theta)$, que efectuar o cálculo de vários senos e cosenos, tal como é apresentado na equação 1. Consequentemente para o cálculo das velocidades V_2 e V_3 torna-se vantajoso efectuar as alterações acima referidas.

Assim para o cálculo de V_2, torna-se computacionalmente mais eficiente usar a equação 9 em vez da equação 8 .

$$v2 = -sin(\frac{\pi}{3}-\theta)\times\dot{x}-cos(\frac{\pi}{3}-\theta)\times\dot{y}+L\times\dot{\theta} \quad (8)$$

$$v2 = -(sin(\frac{\pi}{3})\times cos(\theta)-cos(\frac{\pi}{3})\times sin(\theta))\times\dot{x}- (cos(\frac{\pi}{3})\times cos(\theta)-sin(\frac{\pi}{3})\times sin(\theta))\times\dot{y}+L\times\dot{\theta} \quad (9)$$

O mesmo se verifica para o calculo de V_3 tornando-se mais eficiente usar a equação 11 que a 10

$$v3 = -sin(\frac{\pi}{3}+\theta)\times\dot{x}-cos(\frac{\pi}{3}+\theta)\times\dot{y}+L\times\dot{\theta} \quad (10)$$

$$v3 = (sin(\frac{\pi}{3})\times cos(\theta)+cos(\frac{\pi}{3})\times sin(\theta))\times\dot{x}- (cos(\frac{\pi}{3})\times cos(\theta)-sin(\frac{\pi}{3})\times sin(\theta))\times\dot{y}+L\times\dot{\theta} \quad (11)$$

Deste modo é usado o seguinte código para o controlador, pré calculando-se apenas um coseno e um seno, sendo tudo o resto constantes.

Código usado:

```
//v_teta - velocidade angular do Robot
//v_x - velocidade linear do Robot
 segundo o eixo dos X
//v_y - velocidade linear do Robot
segundo o eixo dos Y
//cos_pi3 - constante
//Pré calcula-se o cos e sin de teta
cos_teta=Math.cos(teta);
sin_teta=Math.sin(teta);
// estabelece-se o objectivo
v_x=0.08;
v_y=0.08;
v_teta=0.0;
v1=-sin_teta*v_x+cos_teta*v_y+L*v_teta;
v2=-(sin_pi3*cos_teta-cos_pi3*sin_teta)
*v_x-(cos_pi3*cos_teta-sin_pi3*sin_teta)
*v_y+L*v_teta;
v3=(sin_pi3*cos_teta+cos_pi3*sin_teta)*v_x
```

```
-(cos_pi3*cos_teta-sin_pi3*sin_teta)*v_y+
L*v_teta;
```

4.2.2. Odometria O posicionamento do *Robot* é calculado baseando-se na odometria. A odometria consiste em usar a medida da velocidade de cada roda para estimar a posição do *Robot* (Borestein *et al.* 1996), apresentando como vantagem o facto de poder tornar o *Robot* completamente autónomo, mas apresentando como desvantagem o crescente acumular de erros, os quais não podem ser corrigidos a não ser que existam outros sensores que deiam informação relativamente ao posicionamento do *Robot*.

Para o cálculo da odometria foi usado o seguinte código:

```
  public static void odoUpdate()
  {
//ler os impulsos que cada roda andou
Odo1=FloydMotors.OdoCount1;
Odo2=FloydMotors.OdoCount2;
Odo3=FloydMotors.OdoCount3;
//saber quanto impulsos andou durante
um sampling time de 0.2 segundos
do1=Odo1-lastOdo1;
do2=Odo2-lastOdo2;
do3=Odo3-lastOdo3;
//actualizar a leitura
lastOdo1=Odo1;
lastOdo2=Odo2;
lastOdo3=Odo3;
//converter impulsos em deslocamento
//  em va,vb,vc
va=factor_roda*do1;
vb=factor_roda*do2;
vc=factor_roda*do3;
//calcular as velocidade
// no sentido do eixo dos x,y e teta
v_x = F2 (va,vb,vc);
v_y = F1(va,vb,vc) ;
v_teta = F3(va,vb,vc);
// com a aproximação de primeira
// ordem efectuar o cálculo
// da nova posição do Robot
x=x+v_x*0.2;
y=y+v_y*0.2;
teta=teta+v_teta*0.2;
  }
```

4.3 Thread Tmot

A *Thread Tmot* tem como objectivo actualizar a contagem de impulsos recebidos dos encoders e actualizar a velocidade dos motores. O seu *Sampling time* foi escolhido com base na frequência máxima a que podem ser recebidos os impulsos gerados pelos *encoders*. Por outro lado existe também uma restrição relativa ao tempo que é necessário para processar todos os cálculos, tendo este que ser majorado. Os cálculos da *Thread* Tmot demoram sensivelmente 15 ms, logo o intervalo de tempo entre acordar esta *Thread* tem que ser superior a este valor. O tempo escolhido foi 20 ms, pois permite que sejam feitos os cálculos, não sendo perdidas transições. Para que não sejam perdidos impulsos o tempo deve ser inferior ao tempo mínimo entre transições, sendo este da ordem dos 40 ms, como pode ser observado na figura 5, na qual a roda se encontra à velocidade máxima o que implica um tempo de intervalo entre impulsos mínimo.

A contagem dos impulsos funciona como uma máquina de estados, estando representada na figura 8. Periodicamente verifica-se se houve transição (dif). Caso haja transição e a referência de velocidade do motor seja maior que zero incrementamos o contador ou então decrementamos caso seja menor que zero. Na situação de o motor não estar travado a sua velocidade pode ser maior ou menor que zero, podemos saber em que sentido o motor roda baseando-nos na velocidade das outras duas rodas, sendo o código apresentado apenas válido para esfasamento entre os veios dos motores de 120º, tal como apresentado na figura 9, o que implica simetria na geometria do *Robot*.

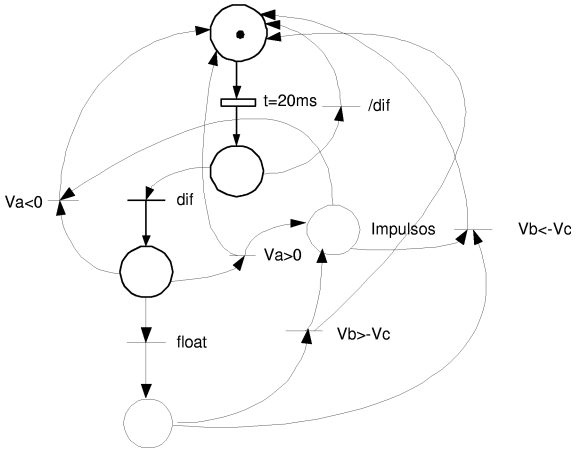

Figura 8. Diagrama de estados relativo à actualização dos impulsos dos encoders

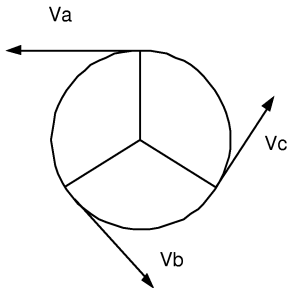

Figura 9. Geometria de um *robot* omnidireccional com esfasamento entre veios de 120º

O código que faz esta operação é o seguinte:

```
public void run()
  {
  while (Active)
    {
//actualiza o impulsos
recebidos de um dos encoders
if(Sensor.S1.
readBooleanValue()!= Odo1)
   {

Odo1=!Odo1;
if(SpeedMotorA>0) OdoCount1++;
if(SpeedMotorA<0) OdoCount1++;

if(SpeedMotorA==0 &
   SpeedMotorB>-SpeedMotorC)
{OdoCount1++;}

if(SpeedMotorA==0 &
   SpeedMotorB<-SpeedMotorC)
{OdoCount1--;}

   }

LastTime=LastTime+20;
SleepTime=LastTime-
   (int)System.currentTimeMillis();

if (SleepTime>0)
try {
sleep(SleepTime);
    }
catch (InterruptedException ie) {}
   }
```

5. CONCLUSÕES

Existem situações em que efectuar o cálculo da velocidade de cada roda não é directo pois quando um motor não está travado não existe informação relativa ao sentido de rotação, simplesmente são recebidos impulsos, os quais são convertidos em velocidade. O sentido de rotação é conhecido nessa situação a partir do conhecimento da velocidade dos outros dois motores, existindo a necessidade de o esfasamento entre os veios dos motores ser de 120º, caso contrário o algoritmo usado não seria válido.

O cálculo do posicionamento do *Robot* a partir da odometria foi objectivo final deste trabalho, sendo um método muito usado sempre que um veículo é ou se tem que tornar temporariamente autónomo, não existindo a possibilidade de a sua posição ser actualizada por outro método. Não deve ser o único método a usar para medir a localização de um *Robot*, pois os erros de odometria são cumulativos, tornando-se cada vez maiores. Torna-se um método muito útil quando existe fusão de dados de várias fontes, podendo-se usar um filtro de kalman, não ficando o sistema dependente de apenas um sensor externo ou interno.

Apesar de o hardware ser muito limitado, principalmente a nível de robustez, torna-se muito aliciante fazer *Robots* com os *kits Lego Mindstorm*, existindo um inúmero conjunto de conceitos que podem ser explorados usando este tipo de tecnologia.

Não foi possível apresentar gráficos para exemplificar os resultados obtidos experimentalmente, dado a memória ser bastante limitada, sendo o resultado da odometria vizualizado no próprio *display do Brick da Lego*, sem que haja comunicação com o PC.

REFERÊNCIAS

Bagnall, Brian (2002). *Core LEGO MINDSTORMS Programming: Unleash the Power of the Java Platform*. Syngress Publishing, Inc.

Borestein, Everett and Feng (1996). *where am I, Sensores and Methods for Mobile Robot Positioning*. Prepared by the University of Michigan.

Costa, Paulo (2003). Api docs. http://lejos.sourceforge.net/apidocs/.

Dudek, Gregory and Michael Jenkin (2000). *Computational Principles of Mobile Robotics*. Cambridge University Press.

Ferrari, Mario, Giulio Ferrari and Ralph Hempel (2002). *Building Robots with lego Mindstorms The ultimate tool for mindstorm maniacs*. Syngress Publishing, Inc.

Kalmár-Nagy, Tamás, Raffaello D'Andrea and Pritam Ganguly (2002). Near-optimal dynamic trajectory generation and control of an omnidirectional vehicle.. Sibley School of Mechanical and Aerospace Engineering Cornell University Ithaca, NY 14853, USA.

Laverde, Dario, Giulio Ferrari and Jurgen Stuber (2002). *Programming Lego Mindstorms with Java*. Syngress Publishing, Inc.

L.Williams, Robert, Brian E. Carter, Paolo Gallina and Giulio Rosati (2002). Dinamic model with slip for wheeled omnidirectional robots. In: *IEEE TRANSACTIONS on robotics and automation Vol 18*. IEEE Press.

Pallas-Areny, Ramon and John G. Webster (1991). *Sensors and Signal Conditioning*. John Wiley and Sons Inc.

The free dictionary (2004). http://www.thefreedictionary.com/Lego.