Referências Bibliográficas

[1] Arkin, R. C., “Behavior-Based Robotics”, MIT Press, Cambridge Massachusetts, 1998.

[2] Bonari, A., Mateucci, M., Resteli, M., “A Kinematics independent Dead Reckoning Sensor for indoor Mobile Robots”, IEEE Robotics & Automation Magazine, 2003.

[3] Bonari, A., Mateucci, M., Resteli, M., “Automatic error detection and reduction for a odometric sensor based on two optical mice”, IEEE Robotics & Automation Magazine, 2005.

[4] Sorenses, D. K., Smukala, V., Ovinis, M., Lee, S., “On-line optical flow feedback for mobile robot localization/navigation”, IEEE Robotics & Automation Magazine, 2003.

[5] Kasper, M., Fricker, G., Puttkamer, E., “A based behavior architecture for teaching more then reactive behaviors to mobile robots”, IEEE Robotics & Automation Magazine, 1999.

[6] Stenzel, R., “A behavior based architecture”, IEEE Robotics & Automation Magazine, 2000.

[7] Emery, R., Balch, T., “Behavior based control of a non-holonomic robot in pushing tasks” IEEE Robotics & Automation Magazine, 2000.

[8] Huntsberger, T., Aghazarian, H., Baumgartner, E., Schenker, P. S., “Behavior based control systems for planetary autonomous robot outposts” IEEE Robotics & Automation Magazine, 2001.

[9] Li, W., Feng, X., “Behavior fusion for robot navigation in uncertain environments using fuzzy logic”, IEEE Robotics & Automation Magazine, 1994.

[10] Fukayama, A., Ida, M., Katai, O., “Behavior based fuzzy control system for a mobile robot with environment recognition by sensory-motor coordination” IEEE Robotics & Automation Magazine, 1999.

[11] Taliansky, A., Shimki, N., “A behavior based navigation for an indoor mobile robot”, IEEE Robotics & Automation Magazine, 2000.

[12] Suh, H., Lee, S., Kim, B., Yi, B., Oh, S., “Design and implementation of a behavior based control and learning architecture for mobile robots”, IEEE Robotics & Automation Magazine, 2003.

[13] Li, H., Fu, Y., Xu, H., Ma, Y., “Avoiding static and dynamic objects in navigation”, IEEE Robotics & Automation Magazine, 2006.

[14] Schenker, P.S., Huntsberger, T.L., Pirjanian, P., Baumgartner, E., Aghazarian, H., Trebi-Ollenu, A., Leger, P.C., Cheng, Y., Backes, P.G., Tunstel, E.W., “Robotic automation for space planetary surface exploration” MIT Press, 2003.

[15] Xu, L.W., Tso, S.K., “Sensor-based fuzzy reactive navigation of a mobile robot through local target switching”, IEEE Robotics & Automation Magazine, 1999.

[16] Brooks, Rodney A. A robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation RA-2, p. 14-23, 1986.

[17] Pro-Wave Electronic Corp, Disponível em: http://www.prowave.com .tw/english/item/ut.htm Acessado em 04/2009.

[18] MicroChip, Disponível em: http://www.microchip.com/wwwproducts/ Devices.aspx?dDocName=en010280 Acessado em 04/2009.

[19] FairChild Semiconductor, Disponível em: http://www.fairchildsemi. com/pf/QR/QRD1114.html Acessado em 05/2009.

[20] Jones, J., Flynn, A., Seiger, B., Mobile Robots: Inspiration to Implementation,second ed., A.K. Peters, 1998.

[21] Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots." 1985 IEEE International Conference on Robotics and Automation, St. Louis, Missouri, March 25-28, 1990, pp.500-505

[22] Andrews, J. R. and Hogan, N., "Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator." Control of Manufacturing Processes and Robotic Systems, Eds. Hardt,D. E. and Book, W., ASME, Boston, 1983, pp. 243-251.

[23] Krogh, B. (1984). A generalized potential field approach to obstacle avoidance control. In: Proceedings of the SME Conf. Robotics Research, Bethlehem, PA.

[24] Medeiros, A. (1998). Introdução à robótica. In Anais do XVII Encontro Nacional de Automática, volume 1, pág. 56–65, Natal, RN, Brasil.

[25] Heinen, Farlei José. Robótica Autônoma: Integração entre Planificação e Comportamento Reativo; UNISINOS Editora, São Leopoldo, Novembro, 1999.

[26] Murphy, R.R.; "An Introduction to AI Robotics" - MIT Press pp400 1ªEdição, 2000.

[27] Pomerleau, D.; “No Hands Across America! (NHAA)” - Carnegie Mellon University Robotics Institute 1998.

[28] Lemonick, Michel. Dante Tours the Inferno. Time Magazine – Time Domestic/Science. Vol. 144, No. 7. August 15, 1994.

[29] Batavia, Parag; Pomerleau, Dean & Thorpe, Charles. Applying Advanced Learning Algorithms to ALVINN. CMU Technical Report CMU-RI-TR-96-31. Carnegie Mellon University. Pittsburgh. 1996.

[30] Paromtchik, I. E. & Laugier, C. Autonomous Parallel Parking of a Nonholonomic Vehicle. Procedings of the IEEE International Symposium on Intelligent Vehicles. pp. 13-18. September, 1996.

[31] Sheuer, A. & Laugier, C. Planning Sub-Optimal and Continuous-Curvature Paths for Car-Like Robots. IEEE- RSJ International Conference on Intelligent Robots and Systems. Victoria, British-Columbia, Canada. Oct. 1998.

[32] Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, NJ: Prentice Hall.

[33] Mataric, M. J. “Behavior-based control: Examples from navigation, learning and group behavior,” J. Experimental Theoretical Artif. Intell, Special Issue: Software Architecture Phys. Agents, vol. 9, pp. 46–54, 1997.

[34] Nehmzow, U. , “Mobile Robotics: A Practical Introduction”, Springer, London, 2000.

[35] Gat, E., "Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots," in SIGART Bulletin 2, 1991, 70-74.

[36] Arkin, R., 1987, “Motor-Schema Based Navigation for a Mobile Robot: An Approach to Programming by Behavior”. IEEE International Conference on Robotics and Automation, Raleigh, NC, 264-271.

[37] Arbib, M.A., "Schema Theory", in the Encyclopedia of Artificial Intelligence, 2nd Edition, Editor Stuart Shapiro, 2:1427-1443, Wiley, 1992.

[38] Arkin, R. C. 1989b. "Motor Schema-Based Mobile Robot Navigation," International Journal of Robotics Research, Vol. 8, No.4, pp. 92-112.

[39] Arkin, R. C. 1992. "Behavior-Based Robot Navigation for Extended Domains," Adaptive Behavior, Vol. No.2, pp. 201-225.

[40] Cameron, J., MacKenzie, D., Ward, K., Arkin, R., and Book, W. 1993. "Reactive Control for Mobile Manipulation;' Processing of the International Conference on Robotics and Automation, Atlanta, GA, pp. 228-35.

[41] Khatib, O. “Real-time obstacle avoidance for manipulators and mobile robots”. In Proceedings of the IEEE Conference on Robotics and Automation, pages 500{505, 1985).

[42] Krogh, BH. “A Generalized Potential Field Approach to Obstacle Avoidance” Control Robotics International of SME (1984)

[43] Braitenberg, V. 1984. Vehicles: Experiments in Synthetic Psychology, MIT Press, Cambridge, MA.

[44] Benda, M, V. Jagannathan, and R. Dodhiawalla, “On optimal cooperation of knowledge sources”, Technical Report, Boeing Advanced Technology Center, Boeing Computer Services, Seattle, WA, 1986.

[45] Goldberg, Dani and Mataric, Maja J (2001) Design and Evaluation of Robust Behavior-Based Controllers for Distributed Multi-Robot Collection Tasks. Robot Teams: From Diversity to Polymorphism.

[46] Stone, Peter (2007) “Intelligent Autonomous Robotics: A Robot Soccer Case Study” Synthesis Lectures on Artificial Intelligence andMachine Learning. ISBN: 1598291262

[47] Mataric, M. J., 1997, “Behavior Based Robotics as a Tool for Synthesis of Articial Behavior and Analysis of Natural Behavior” Trends in Cognitive Science, Vol.2 N.3 March 1998, 82-87

[48] Brooks, Rodney A. “Cambrian Intelligence: The Early History of the New AI”. MIT Press, Cambridge,Massachusetts, 1999

[49] Maes, P. & Brooks, R. A. (1990), Learning to Coordinate Behaviours, in “Proceedings, AAAI-91”, Boston, MA, pp. 796-802.

[50] Connell, J. H. (1990), Minimalist Mobile Robotics: A Colony Architecture for an Artificial Creature, Academic Press.

[51] Rosenblatt, J., DAMN: a distributed architecture for mobile navigation. J. Exp. Theor. Artif. Intell. 9(2-3): 339-360 (1997)

[52] ROSENSCHEIN, S.J. & L.P. KAELBLING, The synthesis of digital machines with provable epistemic properties. In J.Y. Halpern (ed.), Theoretical aspects of reasoning about knowledge: Proc. Fifth Conference, p. 83-97. San Francisco (Cal.) Morgan Kaufmann, 1986.

[53] Zeltzer, D. and Johnson, M. B., Motor Planning: Am architecture for specifying and controlling the behaviour of virtual actors. Journal of Visualization and Computer Animation, 2:74-80, 1991.

[54] Firby, R. J. and Earl, C. Combined Execution and Monitoring for Control of Autonomous Agents. 1988-1995

[55] iRobot Robotics, Disponível em: http://store.irobot.com/home/ index.jsp Acessado em 06/2009.

[56] NASA Rovers, Disponível em: http://marsrover.nasa.gov/home /index.html Acessado em 06/2009.

[57] Boston Dynamics, Disponível em: http://www.bostondynamics.com /robot_bigdog.html Acessado em 12/2009.

[58] Wolf, Denis F., Disponível em: www.icmc.usp.br/~denis/files/ Player_man_v20.pdf Acessado em 12/2009.

Apêndice I

Arquivos com código fonte de cada simulação e experimento que acompanham este trabalho são especificados a seguir:

motorschema.cpp > Código fonte da simulação do robô principal com controle baseado em comportamento.

inimigo.cpp > Código fonte da simulação do robô inimigo/predador com controle baseado em comportamento.

motorcomp-bbr.cpp > Código fonte da simulação do robô principal com controle baseado em comportamento programado com comportamentos simplificados para comparação.

inimigocomp-bbr.cpp > Código fonte da simulação do robô inimigo/predador com controle baseado em comportamento programado com comportamentos simplificados para comparação.

motorcomp-classic.cpp > Código fonte da simulação do robô principal com programação de controle clássica para comparação.

inimigocomp-bbr.cpp > Código fonte da simulação do robô inimigo/predador com programação de controle clássica para comparação.

predador.cpp > Código fonte da simulação do robô predador.

presa.cpp > Código fonte da simulação do robô presa.

tese1-laser.cfg > Definições da simulação principal executada no software player/stage.

tese1-laser.world > Definições do ambiente e dos robôs na simulação principal executada no software player/stage.

tese1-comp.cfg > Definições da simulação simplificada executada no software player/stage para comparação entre diferentes tipos de programação.

tese1-comp.world > Definições do ambiente e dos robôs na simulação simplificada executada no software player/stage para comparação entre diferentes tipos de programação.

tese-sumo.cfg > Definições da simulação dos robôs predador e presa executada no software player/stage.

tese-sumo.world > Definições do ambiente e dos robôs na simulação dos robôs predador e presa executada no software player/stage.

santerio.c > Código fonte do experimento do robô predador.

santerio-pres.c > Código fonte do experimento do robô presa.

Apêndice II

Primeiramente código fonte resumido explicativo para entendimento do conceito e logo em seguida código completo da simulação completa.

```
//  Código Fonte Resumido Programação Baseada em Comportamento
Utilizando Arquitetura de Esquemas Motores
//  Fabiano Santério - 2009

// Inicializando o robo e sensores

////////// Inicialização das Flags Dos Comportamentos
////////// Funcionam para a tomada de decisão do Arbitrador

int aquisitar_flag = 0;
int pegar_flag = 0;
int entregar_flag = 0;
int soltar_flag = 0;

///////////////// Definindo Estrutura Básica Do Vetor Resposta

struct resultante {

      float velocidade;       // Magnitude Vetor Velocidade
      float angulo;           // Angulo
      float peso;             // Pesos dos Comportamentos
};

/////  Definindo COMPORTAMENTOS PRIMITIVOS (Esquemas Perceptivos)

// ANDA E GIRA -> Anda em linha reta durante um determinado tempo
e gira em seu próprio eixo

// BUSCA -> Busca Objeto AZUL id = 1
// Campo Potencial Atrativo

// ENTREGA -> Procura Local de Entrega (Objeto amarelo) id = 4
// Campo Potencial Atrativo

// FOGE - > Foge de objeto vermelho   id = 0
// Campo Potencial Repulsivo

//  DESVIA -> Desvia de Obstáculos Estáticos Utilizando Sonar
//  Campo Potencial Repulsivo

//  COLISAO -> Evita Colisão
//  Campo Potencial Repulsivo

// RUIDO -> Gera Ruído Aleatório para evitar situações singulares

// FECHAR_garra -> Aciona a Garra para pegar objeto.
```

```
// ABRIR_garra -> Aciona a garra para soltar objeto

// BATIDA -> Executa manobra após detectar colisão
//  Campo Potencial Repulsivo

////////////////////  FUNÇÃO PRINCIPAL     ////////////////

int main(void)
{
      robot.Read();     //Faz a primeira varredura nos sensores

   // Chamando os esquemas perceptivos

// Anda durante um tempo e gira em torno do próprio eixo
// Quando detecta objetivo por cor , vai em sua direção
// Quando detecta local de entrega por cor, vai em sua direção
// Quando detecta inimigo por cor, desvia
// Desvia de obstáculo usando Sonar
// Evita colisões
// Ruído Aleatório
// Fechar Garra
// Abrir Garra
// Batida

/////////////// Definindo Comportamentos  ///////////////////////

//// Arbitrador

      if (soltar_flag == 1)
      std::cout << "Comportamento Ativo ->  !! SOLTAR !! \n"
      else if (entregar_flag == 1)
      std::cout << "Comportamento Ativo ->  !! ENTREGAR !!\n"
      else if (pegar_flag == 1)
      std::cout << "Comportamento Ativo ->  !! PEGAR !!\n"
      else if (aquisitar_flag == 1)
      std::cout << "Comportamento Ativo ->  !! AQUISITAR !!\n"
      else
      std::cout << "Comportamento Ativo ->  !! EXPLORAR !!\n"

// Definindo Limites de Velocidade e Taxa de Rotação
// Atua nos motores do robô
   pp.SetSpeed(somatorio-> velocidade, somatorio -> angulo)  ;

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

A seguir é apresentado o código fonte completo da simulação completa.

```
//  Código Fonte Programação Baseada em Comportamento Utilizando
Arquitetura de Esquemas Motores
//  Fabiano Santério   2009


#include <includes/playerc++.h>
#include <iostream>
#include <time.h>             // Biblioteca da variável Tempo
#include <pthread.h>          // Biblioteca de Multi-tasking C++
#include "includes/args.h"
```

```
using namespace PlayerCc;

PlayerClient robot (gHostname, gPort);    // Inicializando o robo
Position2dProxy pp (&robot, gIndex);      // Inicializando função
de posição 2D
SonarProxy sp (&robot, gIndex);           // Inicializando
sensores de ultra som
BlobfinderProxy bp (&robot, gIndex);      // Inicializando sensor
de cores
GripperProxy gp (&robot, gIndex);         // Inicializando garra
BumperProxy bump (&robot, gIndex);        // Inicializando Bumper
LaserProxy lp(&robot, gIndex);            // Inicializando
Sensores a Laser



////////// Inicialização das Flags Dos Comportamentos //////////

int aquisitar_flag = 0;
int pegar_flag = 0;
int entregar_flag = 0;
int soltar_flag = 0;


////////// Definindo Estruturas Básicas ////////

struct resultante {

      float velocidade;       // Magnitude Vetor Velocidade
      float angulo;                 // Angulo
      float peso;             // Pesos dos Comportamentos
};

/////////////////  DEFININDO FUNÇÕES ////////////////////////

//  Função "lr_rot" retorna -1 ou 1  para virar à esquerda ou à
direita

int lr_rot(int lr_detect)     // lr_detect é o argumento da funcao
{

if (lr_detect == 1)           // se for esquerda
      return -1;        //  vira p/ direita
else
      return 1;         // se nao, vira p/ esq.
}

//  Função "bump_detect" detecta colisao na esqueda ,direita e
traseira
//  Retorna 1 esq,  2 dir,  3  traseira , 4 frontal  e  0 sem
colisao

double bump_detect_min_dist = 1.5;    //  distancia minima para
detecção de colisao

int bump_detect()
{

if      (sp[1] < bump_detect_min_dist || sp[2] <
bump_detect_min_dist)         // esquerda frontal
```

```
      return 1;
else if (sp[5] < bump_detect_min_dist || sp[6] <
bump_detect_min_dist)         // direita frontal
      return 2;
else if (sp[10] < bump_detect_min_dist || sp[11] <
bump_detect_min_dist || sp[12] < bump_detect_min_dist || sp[13] <
bump_detect_min_dist)   // traseira
      return 3;
else if (sp[3] < bump_detect_min_dist || sp[4] <
bump_detect_min_dist)         // frontal
      return 4;
else if (sp[0] < bump_detect_min_dist || sp[7] <
bump_detect_min_dist || sp[8] < bump_detect_min_dist || sp[15] <
bump_detect_min_dist)   // laterais
      return 5;
else
return 0;
}


///Definindo COMPORTAMENTOS PRIMITIVOS (Esquemas Perceptivos) ////

// ANDA E GIRA -> Anda em linha reta durante um determinado tempo
e gira em seu próprio eixo

double andagira_tempo_cru =  7;           // Tempo de Cruzeiro
double andagira_vel_cru =  0.6;           // Velocidade de
Cruzeiro
double andagira_tempo_rot =  5;           // Tempo de Rotação
double andagira_vel_rot =  1;       // Velocidade de Rotação
time_t andagira_tempo_inicio = 0;   // Contador de Tempo
Regressivo
int andagira_acao_atual = 1;        // Ultima Ação Executada   0 =
Andar  1 = Rodar

resultante* andagira ()
{
      time_t tempo_atual = time(NULL);
      double tempo_comparacao = (andagira_acao_atual == 0)?
andagira_tempo_cru : andagira_tempo_rot;

      //std::cout << difftime (tempo_atual,andagira_tempo_inicio)
<< std::endl;

      if (difftime (tempo_atual,andagira_tempo_inicio) >
tempo_comparacao)       // Decisão para troca de ação
      {
            andagira_acao_atual = (andagira_acao_atual == 0)? 1:0;
            andagira_tempo_inicio = time(NULL);

            andagira_vel_rot = (andagira_acao_atual == 0)?
andagira_vel_rot * -1 : andagira_vel_rot ;  // Inverte lado
rotação
      }

      resultante* andagira_vet = new resultante ();   // Criando o
Vetor de saída
      andagira_vet-> peso = 1;                  // Peso AndaGira

      if (andagira_acao_atual == 0)
      {
```

```
            andagira_vet-> velocidade = andagira_vel_cru;
            andagira_vet-> angulo = 0;
      }
      else
      {
            andagira_vet-> velocidade = andagira_vel_cru/2;
            andagira_vet-> angulo = andagira_vel_rot;
      }

            for (int i=0; i < bp.GetCount(); i++)     //  Desvia
do local de depósito amarelo
            {
                  if ( bp.GetBlob(i).id == 4 && bp.GetBlob(i).top
< 20) // Se objeto for amarelo e perto
                  {
                        andagira_vet-> velocidade =
andagira_vel_cru / 4;
                        andagira_vet-> angulo = 2*
andagira_vel_rot;
                  }
            }
      return andagira_vet;
}

// BUSCA -> Busca Objeto AZUL id = 1
// Campo Potencial Atrativo

double busca_vel_rot = 3.3;         // Velocidade de Rotação
double busca_pos_mediana = 40;      // Posição na qual o objeto se
encontra em frente ao robo
double busca_vel_cru = 0.7;         // Velocidade de Cruzeiro

resultante* busca()
{
      resultante* busca_vet = new resultante();
      busca_vet-> peso = 1;
      busca_vet-> velocidade = 0;
      busca_vet-> angulo = 0;
      aquisitar_flag = 0;

      for (int i=0; i < bp.GetCount(); i++)     //  Procura em
todos os objetos que encontrar
      {
            if ( bp.GetBlob(i).id == 1 )  // Se objeto for azul
            {
                  aquisitar_flag = 1;     // Ativa comportamento
AQUISITAR

                  if ( bp.GetBlob(i).top < 3 )  // Se estiver
muito próximo do objeto diminui a vel. até parar
                  {
                        busca_vet-> velocidade = (busca_vel_cru *
bp.GetBlob(i).top) / 3 ;
                        busca_vet-> angulo = busca_vel_rot *
((busca_pos_mediana- bp.GetBlob(i).x)/busca_pos_mediana);   //
(40-x)/40
                  }
                  else
                  {
                        busca_vet-> velocidade = busca_vel_cru ;
```

```
                        busca_vet-> angulo = busca_vel_rot *
((busca_pos_mediana- bp.GetBlob(i).x)/busca_pos_mediana);   //
(40-x)/40
                  }
            }
      }

      return busca_vet;
}

// ENTREGA -> Procura Local de Entrega (Objeto amarelo) id = 4
// Campo Potencial Atrativo

double entrega_vel_rot = 3;         // Velocidade de Rotação
double entrega_pos_mediana = 39;    // Posição na qual o objeto se
encontra em frente ao robo
double entrega_vel_cru = 0.8;       // Velocidade de Cruzeiro

resultante* entrega()
{
int garra_aux = gp.GetBeams();

      resultante* entrega_vet = new resultante();
      entrega_vet-> peso = 1;
      entrega_vet-> velocidade = entrega_vel_cru;

      for (int i=0; i < bp.GetCount(); i++)     //  Procura em
todos os objetos que encontrar
      {
            if ( bp.GetBlob(i).id == 4 && gp.GetState() == 2)
      // Se objeto for amarelo e Garra Fechada
            {
                  entregar_flag = 1;      // Ativa comportamento
ENTREGAR

                  if ( bp.GetBlob(i).top |= 0)  // Vai em direção
ao deposito amarelo estar sobre o mesmo
                  {
                        soltar_flag = 0;
                        entrega_vet-> velocidade =entrega_vel_cru;
                        entrega_vet-> angulo = entrega_vel_rot *
((entrega_pos_mediana- bp.GetBlob(i).x)/entrega_pos_mediana);   //
(40-x)/40
                              if ( (garra_aux |= 1) && (garra_aux
|= 2) && (garra_aux |= 3))    // Se objeto nao estiver dentro da
garra libera entregar e pegar
                                    {
                                    entregar_flag = 0;
                                    pegar_flag = 0;
                                    aquisitar_flag = 1;
                                    }
                  }
                  else        // Aciona Soltar e libera Entregar
                  {
                        soltar_flag = 1;
                        entregar_flag = 0;
                  }
            }
}
      return entrega_vet;
}
```

```
// FOGE - > Foge de objeto vermelho   id = 0
// Campo Potencial Repulsivo

double foge_vel_rot = 1;            // Velocidade de Rotação
double foge_min_dist = 3.9;         // Dist Min para detecção
double foge_vel_cru = 0.8;          // Velocidade de Cruzeiro

resultante* foge()
{
      resultante* foge_vet = new resultante();
      foge_vet-> peso = 1;                // Peso Foge
      foge_vet-> velocidade = 0;
      foge_vet-> angulo = 0;

      for (int i=0; i <= 360; i++)  //  Procura em toda a faixa de
leitura do sensor laser
      {
            if ( i <= 180 && lp[i] <= foge_min_dist)  // Se
inimigo no lado direito
            {
                  foge_vet-> velocidade = 2 * foge_vel_cru /
lp[i];
                  foge_vet-> angulo = (foge_vel_rot / ( (lp[i])  )
);
            }
            else if ( i >= 181 && i <= 360 && lp[i] <=
foge_min_dist) // Se inimigo no lado esquerdo
            {
                  foge_vet-> velocidade = 2 * foge_vel_cru /
lp[i];
                  foge_vet-> angulo = -(foge_vel_rot / ( (lp[i])
) ) ;
            }
      }
      return foge_vet;
}

//  DESVIA -> Desvia de Obstáculos Estáticos Utilizando Sonar
//  Campo Potencial Repulsivo

double desvia_limite_max_cru =  0.5;      // Limite Máximo de
Velocidade de Cruzeiro
double desvia_limite_max_rot =  0.6;      // Velocidade de Rotação
double desvia_dist_max = 4;               // Distância Máxima para
deteção de obstáculso
double desvia_dist_azul = 0.5;            // Distãncia na qual
desvia é desligado

float dist[16] ;                    // Vetor de variáveis de
contribuições para rotação

resultante* desvia()
{

      resultante* desvia_vet = new resultante();

      desvia_vet-> velocidade = 0;
      desvia_vet-> angulo = 0 ;
      desvia_vet-> peso = 1 ;
```

```
      for (int i = 0; i<16; i++)    // Roda em todos os sensores e
no vetor de contribuições de rotação
      {
            if (! (( (i == 3 || i == 4) < desvia_dist_azul ) &&
aquisitar_flag == 1))
            {
                  if (sp[i] < desvia_dist_max)
                  {
                        desvia_vet-> velocidade += 1/
((sp[i]*sp[i])*8);      // Desvia com quadrado da distãcia
                        desvia_vet-> angulo += dist[i]/
(sp[i]*sp[i]*0.1);      // Desvia com quadrado da distãcia
                  }
            }
            else
            {
            desvia_vet-> peso = 0 ;
            }
}
      if (desvia_vet-> velocidade > desvia_limite_max_cru)  //
Define limite máximo de velocidade e angulo
      {
            desvia_vet-> velocidade = desvia_limite_max_cru;
      }
      if (desvia_vet-> angulo > desvia_limite_max_rot ||
desvia_vet-> angulo < -desvia_limite_max_rot )
      {
            desvia_vet-> angulo = (( desvia_vet-> angulo > 0)? 1 :
-1) * desvia_limite_max_rot;
      }
      return desvia_vet;
}

//  COLISAO -> Evita Colisão
//  Campo Potencial Repulsivo

double colisao_vel_rot = 0.8;       // Velocidade de Rotação
double colisao_vel_cru = -0.3;            // Velocidade de Ré
double colisao_dist_azul = 2;

resultante* colisao()
{

      resultante* colisao_vet = new resultante();
      colisao_vet-> peso = 1;             // Peso Colisao

if (! (( (sp[3] || sp[4]) < colisao_dist_azul ) && aquisitar_flag
== 1))      // Desliga sensores 3 e 4 qd obj azul na frente
{
            int es_hit =  0;              // Variável Local para
detecção de objetos muito próximos */
            es_hit =  bump_detect();//  Chama Função Bumper Detect

      if (es_hit == 3)                    //  Colisão Traseira
      {
            colisao_vet-> velocidade = -2 * colisao_vel_cru;
      //  Anda um pouco pra frente
            colisao_vet-> angulo = 0;
      //  Nao gira
      }
```

```
      else if  (es_hit  ==  1  ||  es_hit == 2 || es_hit == 4)
      //  If Colisão Esquerda Direita ou Frontal
      {
            colisao_vet-> velocidade = colisao_vel_cru;
      //  Dá Ré
            colisao_vet-> angulo = lr_rot(es_hit) *
colisao_vel_rot;  //  Gira Sentido Contrário ao Obstáculo
      }
      else if  (es_hit  ==  5)
      //  If Colisão Lateral
      {
            colisao_vet-> velocidade = -5 * colisao_vel_cru;
      //  Acelera ao máximo
            colisao_vet-> angulo = 0;
      //  Não Gira
      }
}
else
{
colisao_vet-> peso = 0;
}
      return colisao_vet;
}

// RUIDO -> Gera Ruído Aleatório para evitar situações singulares

resultante* ruido ()
{
      resultante* ruido_vet = new resultante ();      // Criando o
Vetor de saída
      ruido_vet-> peso = 1;               // Peso Ruido

      float rd = rand() % 20 ;    // Numero aleatório entre 0 e 20

      ruido_vet-> velocidade = ((rd/(100)) - 0.05) ;  // Gerando
aleatório entre -0.05 e 0.15
      ruido_vet-> angulo = ((rd/(100)) - 0.1) ; // Gerando
aleatório entre -0.1 e 0.1

      return ruido_vet;
}

// FECHAR_garra -> Aciona a Garra para pegar objeto.

resultante* fechar_garra ()
{
      resultante* fechar_garra_vet = new resultante ();     //
Criando o Vetor de saída
      fechar_garra_vet-> peso = 1;                    // Peso
Fechar garra

// gp.GetBeams -> Sensor Frontal Atuado = 1 - Sensor Traseiro
Atuado = 2 - Ambos Atuados = 3
// gp.GetState -> Estado Aberto = 1  - Estado Fechado = 2 - Estado
Movendo = 3

    if ( gp.GetBeams() == 2 || gp.GetBeams() == 3)    // Se objeto
estiver dentro da garra
      {
      pegar_flag = 1;
        gp.Close();
```

```
            if ( gp.GetState() == 2 )     // Se Garra Fechada
Libera Flag Pegar e Aciona Flag Entregar
            {
                  pegar_flag = 0;
                  entregar_flag = 1;
            }
      }
      else
      {
        gp.Open();
      entregar_flag = 0;
      pegar_flag = 0;
      }

      fechar_garra_vet-> velocidade = 0.08;
      fechar_garra_vet-> angulo = 0;
      return fechar_garra_vet;
}

// ABRIR_garra -> Aciona a garra para soltar objeto

time_t abrir_garra_tempo_inicio = 0;      // Contador Tempo
double abrir_garra_tempo_cru = 0.8; // Tempo de entrada na zona de
depósito amarela
double abrir_garra_tempo_parada = 0.2;    // Tempo de parada na
zona de depósito amarela
double abrir_garra_tempo_rot = 1;   // Tempo de rotacao na zona de
depósito amarela
double abrir_garra_vel_rot = 1.8;   // Velocidade de rotacao na
zona de depósito amarela
double abrir_garra_vel_cru = 0.5;   // Velocidade de cruzeiro na
zona de depósito amarela
int abrir_garra_acao_atual = 2;     // Ultima Ação Executada   0 =
Andar  1 = Parar  2= Girar

resultante* abrir_garra ()
{
      resultante* abrir_garra_vet = new resultante ();      //
Criando o Vetor de saída
      abrir_garra_vet-> peso = 1;         // Peso Abrir garra

      if ( soltar_flag == 1 ) // Se flag acionado abre garra
      {
            entregar_flag = 0;
            aquisitar_flag = 0;
            pegar_flag = 0;

            time_t abrir_garra_tempo_atual = time(NULL);

            double abrir_garra_tempo_comparacao = 0;

            if ( abrir_garra_acao_atual == 0 )
            {
            abrir_garra_tempo_comparacao = abrir_garra_tempo_cru;
            }
            else if ( abrir_garra_acao_atual == 1 )
            {
            abrir_garra_tempo_comparacao =
abrir_garra_tempo_parada;
            }
```

```
            else
            {
            abrir_garra_tempo_comparacao = abrir_garra_tempo_rot;
            }

            if (difftime
(abrir_garra_tempo_atual,abrir_garra_tempo_inicio) >
abrir_garra_tempo_comparacao)       // Decisão para troca de ação
            {

                        if ( abrir_garra_acao_atual == 0 )
                        {
                        abrir_garra_acao_atual = 1;
                        }
                        else if ( abrir_garra_acao_atual == 1 )
                        {
                        abrir_garra_acao_atual = 2;
                        }
                        else
                        {
                        abrir_garra_acao_atual = 0;
                        }

                  abrir_garra_tempo_inicio = time(NULL);

            }
            if (abrir_garra_acao_atual == 0)
            {
                  abrir_garra_vet-> velocidade =
abrir_garra_vel_cru;
                  abrir_garra_vet-> angulo = 0;

                  if ( gp.GetState() == 1 )     // Se Garra Aberta
Libera Flag Soltar
                  {
                  soltar_flag = 0;
                  abrir_garra_acao_atual = 2;
                  }
                  else
                  {
                  soltar_flag = 1;
                  }
            }
            else if (abrir_garra_acao_atual == 1)
            {
                  abrir_garra_vet-> velocidade = 0;
                  abrir_garra_vet-> angulo = 0;
                  gp.Open();
            }
            else
            {
                  abrir_garra_vet-> velocidade = 0;
                  abrir_garra_vet-> angulo = abrir_garra_vel_rot;
            }
      }
return abrir_garra_vet;
}

// batida -> Executa manobra após detectar colisão
// 0 Dir     1 Esq     2 Tras
```

```
double batida_vel_rot = 5;          // Velocidade de Rotação
double batida_vel_cru = 4;          // Velocidade de Ré

resultante* batida ()
{
      resultante* batida_vet = new resultante ();     // Criando o
Vetor de saída
      batida_vet-> peso = 1;                    // Peso batida
      batida_vet-> velocidade = 0;
      batida_vet-> angulo = 0;

      if (bump.IsAnyBumped())
      {
            batida_vet-> velocidade = batida_vel_cru;

            if (bump.IsBumped(0))                     // Se bater
no tras. dir. anda pra frente e gira pra esq.
            {
                  batida_vet-> angulo = batida_vel_rot;
            }
            else if ((bump.IsBumped(1)))              // Se bater
no tras. esq. anda pra frente e gira pra dir.
            {
                  batida_vet-> angulo = -batida_vel_rot;
            }
            else if ((bump.IsBumped(2)))              // Se bater
no traseiro anda pra frente apenas
            {
                  batida_vet-> angulo = 0;
            }
            else if ((bump.IsBumped(3)))              // Se bater
no lateral dir. recua e gira pra esq.
            {
                  batida_vet-> angulo = batida_vel_rot;
                  batida_vet-> velocidade = -batida_vel_cru;
            }
            else                                      // Se bater no
lateral esq. recua e gira pra dir.
            {
                  batida_vet-> angulo = -batida_vel_rot;
                  batida_vet-> velocidade = -batida_vel_cru;
            }
      }
      return batida_vet;
}


//////////////////////////////////////////////////////////////////
////////////////////// FUNÇÃO PRINCIPAL //////////////////////////

int main(void)
{
      robot.Read();     //Faz a primeira varredura nos sensores

//    Definindo Parâmetros
      dist[0] = -0.1; dist[1] = -0.2;     dist[2] = -0.3;
      dist[3] = -0.4;   dist[4] = 0.4;    dist[5] = 0.3;
      dist[6] = 0.2;    dist[7] = 0.1;    dist[8] = 0.1;
      dist[9] = 0.2;    dist[10] = 0.3;   dist[11] = 0.4;
```

```
      dist[12] = -0.4;dist[13] = -0.3;    dist[14] = -0.2;
      dist[15] = -0.1;

      for(;;)
      {

   robot.Read();  //Faz leitura de todos os sensores

   // Chamando os esquemas motores

   resultante* andagira_vet = andagira ();
// Anda durante um tempo e gira em torno do proprio eixo

   resultante* busca_vet = busca ();
// QUando detecta objetivo por cor , vai em sua direção

   resultante* entrega_vet = entrega ();
// Quando detecta local de entrega por cor, vai em sua direção

   resultante* foge_vet = foge ();
// Quando detecta inimigo por cor, desvia

   resultante* desvia_vet = desvia ();
// Desvia de obstáculo usando Sonar

   resultante* colisao_vet = colisao ();
// Evita colisões

   resultante* ruido_vet = ruido ();
// Ruído Aleatório

   resultante* fechar_garra_vet = fechar_garra ();
// Fechar Garra

   resultante* abrir_garra_vet = abrir_garra ();
// Abrir Garra

   resultante* batida_vet = batida ();
// Batida


/////////////// Definindo Comportamentos  ////////////////////////

   // Comportamento

   resultante* somatorio = new resultante();

   somatorio-> velocidade = 0;
   somatorio-> angulo =  0 ;

////////// Arbitrador   //////////

      if (soltar_flag == 1)                    //  FOGE, RUIDO,
ABRIR GARRA
      {
      std::cout << "Comportamento Ativo ->  !! SOLTAR !! \n"  <<
std::endl;

      andagira_vet->peso = 0;
      busca_vet->peso = 0;
      entrega_vet->peso = 0;
```

```
      desvia_vet->peso = 0;
      colisao_vet->peso = 0;
      fechar_garra_vet->peso = 0;
      batida_vet->peso = 0;
      }
      else if (entregar_flag == 1)              // ENTREGA , FOGE
,DESVIA,COLISAO,RUIDO, BATIDA
      {
      std::cout << "Comportamento Ativo ->  !! ENTREGAR !!\n"  <<
std::endl;

      andagira_vet->peso = 0;
      busca_vet->peso = 0;
      fechar_garra_vet->peso = 0;
      abrir_garra_vet->peso = 0;
      }
      else if (pegar_flag == 1)
      {
      std::cout << "Comportamento Ativo ->  !! PEGAR !!\n"  <<
std::endl;        // FOGE,RUIDO,FECHAR GARRA , BATIDA

      andagira_vet->peso = 0;
      busca_vet->peso = 0;
      entrega_vet->peso = 0;
      desvia_vet->peso = 0;
      colisao_vet->peso = 0;
      abrir_garra_vet->peso = 0;
      }
      else if (aquisitar_flag == 1)
      {
      std::cout << "Comportamento Ativo ->  !! AQUISITAR !!\n"  <<
std::endl;  // BUSCA , FOGE , DESVIA , COLISAO , RUIDO , BATIDA

      andagira_vet->peso = 0;
      entrega_vet->peso = 0;
      fechar_garra_vet->peso = 0;
      abrir_garra_vet->peso = 0;
      }
      else
      {
      std::cout << "Comportamento Ativo ->  !! EXPLORAR !!\n"  <<
std::endl;  // ANDAGIRA , FOGE , DESVIA , COLISAO , RUIDO , BATIDA

      busca_vet->peso = 0;
      entrega_vet->peso = 0;
      fechar_garra_vet->peso = 0;
      abrir_garra_vet->peso = 0;
      }

  somatorio-> velocidade = 0
                  + andagira_vet->velocidade * andagira_vet->peso
                  + busca_vet->velocidade * busca_vet->peso
                  + entrega_vet->velocidade * entrega_vet->peso
                  + foge_vet->velocidade * foge_vet->peso
                  + desvia_vet->velocidade * desvia_vet->peso
                  + colisao_vet->velocidade * colisao_vet->peso
                  + ruido_vet->velocidade * ruido_vet->peso
                  + fechar_garra_vet->velocidade *
fechar_garra_vet->peso
                  + abrir_garra_vet->velocidade *
abrir_garra_vet->peso
```

```
                    + batida_vet->velocidade * batida_vet->peso
                                                          ;

  somatorio-> angulo = 0
                    + andagira_vet->angulo * andagira_vet->peso
                    + busca_vet->angulo * busca_vet->peso
                    + entrega_vet->angulo * entrega_vet->peso
                    + foge_vet->angulo * foge_vet->peso
                    + desvia_vet->angulo * desvia_vet->peso
                    + colisao_vet->angulo * colisao_vet->peso
                    + ruido_vet->angulo * ruido_vet->peso
                    + fechar_garra_vet->angulo * fechar_garra_vet-
>peso
                    + abrir_garra_vet->angulo * abrir_garra_vet-
>peso
                    + batida_vet->angulo * batida_vet->peso
                                                          ;

//// Definindo Limites de Velocidade e Taxa de Rotação
somatorio-> velocidade = (somatorio-> velocidade > 1)? 1 :
somatorio-> velocidade;
somatorio-> velocidade = (somatorio-> velocidade < -0.7)? -0.7 :
somatorio-> velocidade;
somatorio-> angulo = (somatorio-> angulo > 5)? 5 : somatorio->
angulo;
somatorio-> angulo = (somatorio-> angulo < -5)? -5 : somatorio->
angulo;

// Atuando no motores do robô

   pp.SetSpeed(somatorio-> velocidade, somatorio -> angulo)  ;


  std::cout << "Velocidade =" << somatorio-> velocidade << " -
Taxa Rotação = " << somatorio-> angulo << "\n\n" << std::endl;

  }
}
```

---------- // ----------

.

Apêndice III

Datasheets dos componentes utilizados nas simulações e experimentos:

- Motor *Integy Matrix Pro Lathe Motor 75T Single SCM7501*

***Matrix Pro Lathe Motor 75T Single SCM7501**

Product Description

Product Description

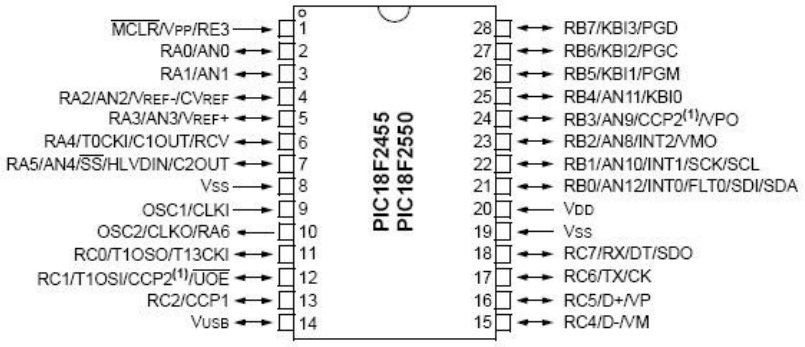
This is the 75T Pro Lathe Motor by Team Matrix. / / FEATURES: Low speed motor for lathe application / Ball bearings / Rebuildable motor design, endbell can be removed with removal of / two screws / / INCLUDES: One 75T Pro Lathe motor by Team Matrix / / REQUIRES: INTC1394 (motor lathe) / / SPECS: Length: 2-5/8"" (64mm) / Diameter: 1-1/2"" (36mm) / Shaft Diameter: 1/8"" (3mm) / / ajw 10/8/06 / ir/jxs

- Micro controlador PIC2550.

PIC18F2455/2550

Pin Diagrams

28-Pin PDIP, SOIC



PIC18F2550 Status: In Production

PIC18F2455/2550/4455/4550 Data sheet (10/27/2009)

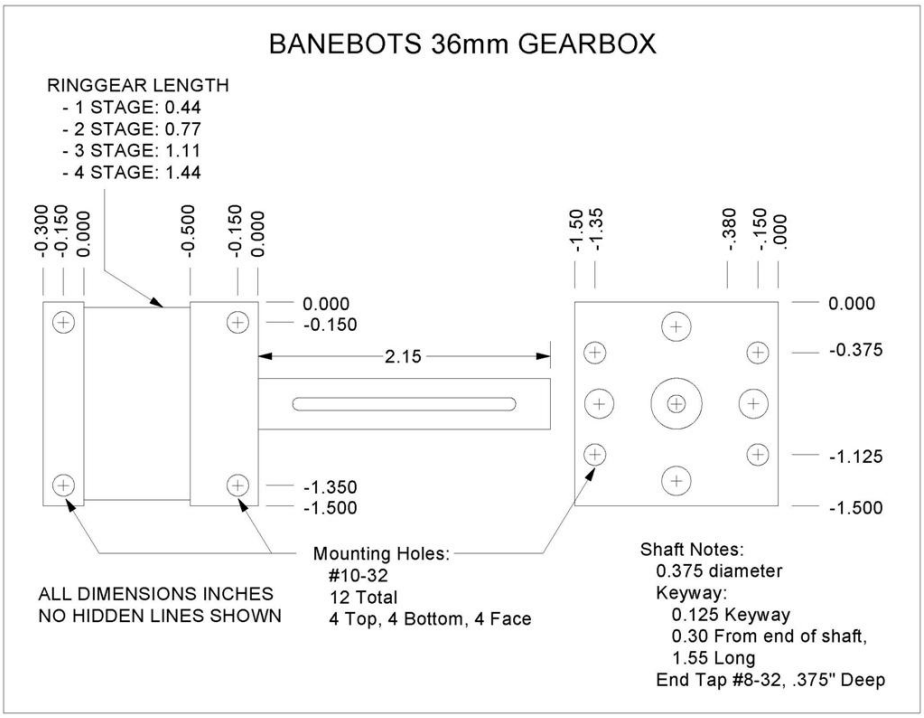
Ideal for low power (nanoWatt) and connectivity applications that benefit from the availability of three serial ports: FS-USB (12 Mbit/s), I²C™ and SPI™ (up to 10Mbit/s) and an asynchronous (LIN capable) serial port (EUSART). Large amounts of RAM memory for buffering and Enhanced FLASH program memory make it ideal for embedded control and monitoring applications that require periodic connection with a (legacy free) Personal Computer via USB for data upload/download and/or firmware updates. While operating up to 48 MHz, the PIC18F2550 is also mostly software and hardware compatible with the PIC16C745 Low-Speed USB OTP devices. THE PICSTART® Plus does NOT currently support this device but may support it in the future.
USB Application Design Center

Parameter Name	Value
Program Memory Type	Flash
Program Memory (KB)	32
CPU Speed (MIPS)	12
RAM Bytes	2,048
Data EEPROM (bytes)	256
Digital Communication Peripherals	1-A/E/USART, 1-MSSP(SPI/I2C)
Capture/Compare/PWM Peripherals	2 CCP
Timers	1 x 8-bit, 3 x 16-bit
ADC	10 ch, 10-bit
Comparators	2
USB (ch, speed, compliance)	1, Full Speed, USB 2.0
Temperature Range (C)	-40 to 85
Operating Voltage Range (V)	2 to 5.5
Pin Count	28

Features

Full Speed USB 2.0 (12Mbit/s) interface
- 1K byte Dual Port RAM + 1K byte GP RAM
- Full Speed Transceiver
- 16 Endpoints (IN/OUT)
- Internal Pull Up resistors (D+/D-)
- 48 MHz performance (12 MIPS)
- Pin-to-pin compatible with PIC16C7X5

- Caixa de redução *BaneBots* 36mm 16:1

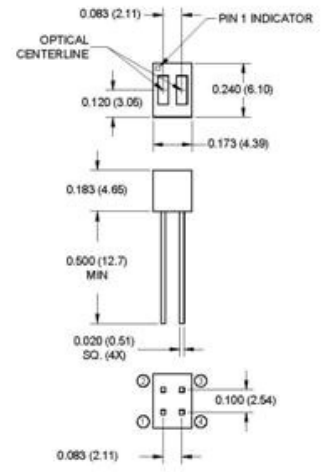


- Sensor infravermelho *FairChild* QRD1114.

FAIRCHILD
SEMICONDUCTOR

QRD1113/1114
REFLECTIVE OBJECT SENSOR

PACKAGE DIMENSIONS



PIN 1 COLLECTOR PIN 3 ANODE
PIN 2 EMITTER PIN 4 CATHODE

NOTES:

1. Dimensions for all drawings are in inches (millimeters).
2. Tolerance of ± .010 (.25) on all non-nominal dimensions unless otherwise specified.
3. Pins 2 and 4 typically .050" shorter than pins 1 and 3.
4. Dimensions controlled at housing surface.

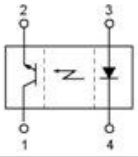
FEATURES

- Phototransistor Output
- No contact surface sensing
- Unfocused for sensing diffused surfaces
- Compact Package
- Daylight filter on sensor

NOTES (Applies to Max Ratings and Characteristics Tables.)

1. Derate power dissipation linearly 1.33 mW/°C above 25°C.
2. RMA flux is recommended.
3. Methanol or isopropyl alcohols are recommended as cleaning agents.
4. Soldering iron 1/16" (1.6mm) from housing.
5. As long as leads are not under any spring tension.
6. D is the distance from the sensor face to the reflective surface.
7. Cross talk (I_{CX}) is the collector current measured with the indicator current on the input diode and with no reflective surface.
8. Measured using an Eastman Kodak neutral white test card with 90% diffused reflecting as a reflective surface.

SCHEMATIC



ABSOLUTE MAXIMUM RATINGS (T_A = 25°C unless otherwise specified)

Parameter	Symbol	Rating	Units
Operating Temperature	T_{OPR}	-40 to +85	°C
Storage Temperature	T_{STG}	-40 to +85	°C
Lead Temperature (Solder Iron)[2,3]	T_{SOL-I}	240 for 5 sec	°C
Lead Temperature (Solder Flow)[2,3]	T_{SOL-F}	260 for 10 sec	°C
EMITTER			
Continuous Forward Current	I_F	50	mA
Reverse Voltage	V_R	5	V
Power Dissipation[1]	P_D	100	mW
SENSOR			
Collector-Emitter Voltage	V_{CEO}	30	V
Emitter-Collector Voltage	V_{ECO}		V
Power Dissipation[1]	P_D	100	mW

FAIRCHILD
SEMICONDUCTOR

QRD1113/1114
REFLECTIVE OBJECT SENSOR

ELECTRICAL / OPTICAL CHARACTERISTICS (T_A = 25°C)

PARAMETER	TEST CONDITIONS	SYMBOL	MIN	TYP	MAX	UNITS
EMITTER Forward Voltage	I_F = 20 mA	V_F	—	—	1.7	V
Reverse Current	V_R = 5 V	I_R	—	—	100	µA
Peak Emission Wavelength	I_F = 20 mA	λ_{PE}	—	940	—	nm
SENSOR Collector-Emitter Breakdown	I_C = 1 mA	BV_{CEO}	30	—	—	V
Emitter-Collector Breakdown	I_E = 0.1 mA	BV_{ECO}	5	—	—	V
Dark Current	V_{CE} = 10 V, I_F = 0 mA	I_D	—	—	100	nA
COUPLED QRD1113 Collector Current	I_F = 20 mA, V_{CE} = 5 V D = .050" (6,8)	$I_{C(ON)}$	0.300	—	—	mA
QRD1114 Collector Current	I_F = 20 mA, V_{CE} = 5 V D = .050" (6,8)	$I_{C(ON)}$	1	—	—	mA
Collector Emitter Saturation Voltage	I_F = 40 mA, I_C = 100 µA D = .050" (6,8)	$V_{CE(SAT)}$	—	—	0.4	V
Cross Talk	I_F = 20 mA, V_{CE} = 5 V, E_E = 0 (7)	I_{CX}	—	.200	10	µA
Rise Time	V_{CE} = 5 V, R_L = 100 Ω	t_r	—	10	—	µs
Fall Time	$I_{C(ON)}$ = 5 mA	t_f	—	50	—	µs

FAIRCHILD
SEMICONDUCTOR

QRD1113/1114
REFLECTIVE OBJECT SENSOR

TYPICAL PERFORMANCE CURVES

Fig. 1 Forward Voltage vs. Forward Current

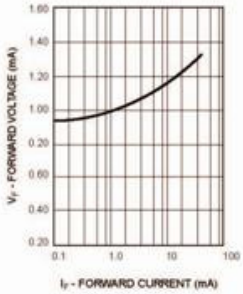


Fig. 2 Normalized Collector Current vs. Forward Current

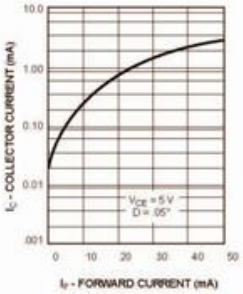


Fig. 3 Normalized Collector Current vs. Temperature

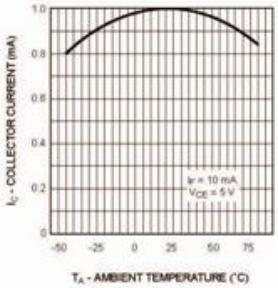


Fig. 4 Normalized Collector Dark Current vs. Temperature

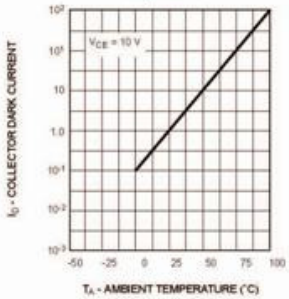
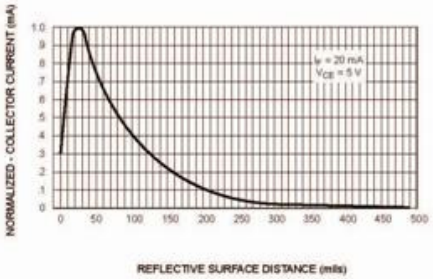


Fig. 5 Normalized Collector Current vs. Distance



FAIRCHILD
SEMICONDUCTOR

QRD1113/1114
REFLECTIVE OBJECT SENSOR

DISCLAIMER

FAIRCHILD SEMICONDUCTOR RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN TO IMPROVE RELIABILITY, FUNCTION OR DESIGN. FAIRCHILD DOES NOT ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT DESCRIBED HEREIN; NEITHER DOES IT CONVEY ANY LICENSE UNDER ITS PATENT RIGHTS, NOR THE RIGHTS OF OTHERS.

LIFE SUPPORT POLICY

FAIRCHILD'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF FAIRCHILD SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and (c) whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury of the user.
2. A critical component in any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

- Sensor Ultra som SRF10.

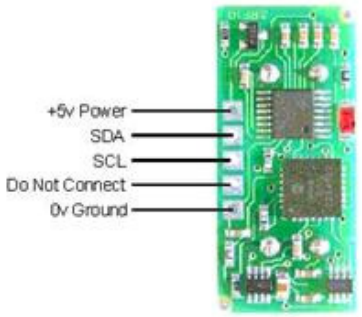
SRF10 Ultrasonic range finder

Technical Specification

Communication with the SRF10 ultrasonic rangefinder is via the I2C bus. This is available on popular controllers such as the OOPic and Stamp BS2p, as well as a wide variety of micro-controllers. To the programmer the SRF10 behaves in the same way as the ubiquitous 24xx series eeprom's, except that the I2C address is different. The default shipped address of the SRF10 is 0xE0. It can be changed by the user to any of 16 addresses E0, E2, E4, E6, E8, EA, EC, EE, F0, F2, F4, F6, F8, FA, FC or FE, therefore up to 16 sonar's can be used. We have examples of using the SRF10 module with a wide range of popular controllers.

Connections

The connections to the SRF10 are identical to the SRF08. The "Do Not Connect" pin should be left unconnected. It is actually the CPU MCLR line and is used once only in our workshop to program the PIC16F87 on-board after assembly, and has an internal pull-up resistor. The SCL and SDA lines should each have a pull-up resistor to +5v somewhere on the I2C bus. You only need one pair of resistors, not a pair for every module. They are normally located with the bus master rather than the slaves. The SRF10 is always a slave - never a bus master. If you need them, I recommend 1.8k resistors. Some modules such as the OOPic already have pull-up resistors and you do not need to add any more.



Registers

The SRF10 appears as a set of 4 registers.

Location	Read	Write
0	Software Revision	Command Register
1	Unused (reads 0x80)	Max Gain Register (default 16)
2	Range High Byte	Range Register (default 255)
3	Range Low Byte	N/A

Only locations 0, 1 and 2 can be written to. Location 0 is the command register and is used to start a ranging session. It cannot be read. Reading from location 0 returns the SRF10 software revision. By default, the ranging lasts for 65mS, but can be changed by writing to the range register at location 2. The SRF10 will not respond to commands on the I2C bus whilst it is ranging. See the **Changing Range** and

Analogue Gain sections below.

Locations, 2 and 3, are the 16bit unsigned result from the latest ranging - high byte first. The meaning of this value depends on the command used, and is either the range in inches, or the range in cm or the flight time in uS. A value of 0 indicates that no objects were detected.

Commands

The are three commands to initiate a ranging (80 to 82), to return the result in inches, centimeters or microseconds. There is also a set of commands to change the I2C address.

Command		Action
Decimal	Hex	
80	0x50	Ranging Mode - Result in inches
81	0x51	Ranging Mode - Result in centimeters
82	0x52	Ranging Mode - Result in micro-seconds
160	0xA0	1st in sequence to change I2C address
165	0xA5	3rd in sequence to change I2C address
170	0xAA	2nd in sequence to change I2C address

Ranging Mode

To initiate a ranging, write one of the above commands to the command register and wait the required amount of time for completion and read the result. The echo buffer is cleared at the start of each ranging. The default and recommended time for completion of ranging is 65mS, however you can shorten this by writing to the range register before issuing a ranging command.

Checking for Completion of Ranging

You do not have to use a timer on your own controller to wait for ranging to finish. You can take advantage of the fact that the SRF10 will not respond to any I2C activity whilst ranging. Therefore, if you try to read from the SRF10 (we use the software revision number a location 0) then you will get 255 (0xFF) whilst ranging. This is because the I2C data line (SDA) is pulled high if nothing is driving it. As soon as the ranging is complete the SRF10 will again respond to the I2C bus, so just keep reading the register until its not 255 (0xFF) anymore. You can then read the sonar data. Your controller can take advantage of this to perform other tasks while the SRF10 is ranging.

Changing the Range

The maximum range of the SRF10 is set by an internal timer. By default, this is 65mS or the equivalent of 11 metres of range. This is much further than the 6 metres the SRF10 is actually capable of. It is possible to reduce the time the SRF10 listens for an echo, and hence the range, by writing to the range register at location 2. The range can be set in steps of about 43mm (0.043m or 1.68 inches) up to 11 metres.
The range is ((Range Register x 43mm) + 43mm) so setting the Range Register to 0 (0x00) gives a maximum range of 43mm. Setting the Range Register to 1 (0x01) gives a maximum range of 86mm. More usefully, 24 (0x18) gives a range of 1 metre and 93 (0x5D) is 4 metres. Setting 255 (0xFF) gives the original 11 metres (255 x 43 + 43 is 11008mm). There are two reasons you may wish to reduce the range.
1. To get at the range information quicker
2. To be able to fire the SRF10 at a faster rate.
If you only wish to get at the range information a bit sooner and will continue to fire the SRF10 at 65ms of slower, then all will be well. However if you wish to fire the SRF10 at a faster rate than 65mS, you will definitely need to reduce the gain - see next section.

The range is set to maximum every time the SRF10 is powered-up. If you need a different range, change it once as part of your system initialization code.

Analogue Gain

The analogue gain register sets the *Maximum* gain of the analogue stages. To set the maximum gain, just write one of these values to the gain register at location 1. During a ranging, the analogue gain starts off at its minimum value of 40. This is increased at approx. 96uS intervals up to the maximum gain setting, set by register 1. Maximum possible gain is reached after about 100mm (4inches) of range. The purpose of providing a limit to the maximum gain is to allow you to fire the sonar more rapidly than 65mS. Since the ranging can be very short, a new ranging can be initiated as soon as the previous range data has been read. A potential hazard with this is that the second ranging may pick up a distant echo returning from the previous "ping", give a false result of a close by object when there is none. To reduce this possibility, the maximum gain can be reduced to limit the modules sensitivity to the weaker distant echo, whilst still able to detect close by objects. The maximum gain setting is stored only in the CPU's RAM and is initialized to maximum on power-up, so if you only want do a ranging every 65mS, or longer, you can ignore the Range and Gain Registers. The Gain Register is set to 16 (a gain of 700) at power-up. This can be decreased as required.

Gain Register		Maximum Analogue Gain
Decimal	Hex	
0	0x00	Set Maximum Analogue Gain to 40
1	0x00	As above - Analogue Gain to 40
2	0x01	Set Maximum Analogue Gain to 50
3	0x02	Set Maximum Analogue Gain to 60
4	0x03	Set Maximum Analogue Gain to 70
5	0x04	Set Maximum Analogue Gain to 80
6	0x05	Set Maximum Analogue Gain to 100
7	0x06	Set Maximum Analogue Gain to 120
8	0x07	Set Maximum Analogue Gain to 140
9	0x08	Set Maximum Analogue Gain to 200
10	0x09	Set Maximum Analogue Gain to 250
11	0x0A	Set Maximum Analogue Gain to 300
12	0x0B	Set Maximum Analogue Gain to 350
13	0x0C	Set Maximum Analogue Gain to 400
14	0x0D	Set Maximum Analogue Gain to 500
15	0x0E	Set Maximum Analogue Gain to 600
16	0x0F	Set Maximum Analogue Gain to 700

Note that the relationship between the Gain Register setting and the actual gain is not a linear one. Also there is no magic formula to say "use this gain setting with that range setting". It depends on the size, shape and material of the object and what else is around in the room. Try playing with different settings until you get the result you want. If you appear to get false readings, it may be echo's from previous "pings", try going back to firing the SRF10 every 65mS or longer (slower).
If you are in any doubt about the Range and Gain Registers, remember they are automatically set by the SRF10 to their default values when it is powered-up. You can ignore and forget about them and the SRF10 will work fine, detecting objects up to 6 metres away every 65mS or slower.

LED

The red LED is used to flash out a code for the I2C address on power-up (see below). It also gives a brief flash during the "ping" whilst ranging.

Changing the I2C Bus Address

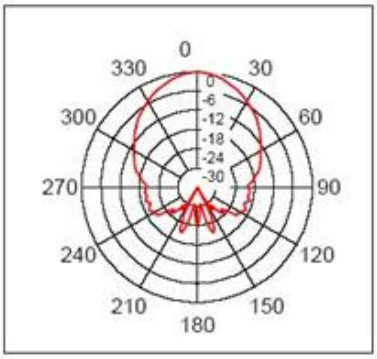
To change the I2C address of the SRF10 you must have only one sonar on the bus. Write the 3 sequence commands in the correct order followed by the address. Example; to change the address of a sonar currently at 0xE0 (the default shipped address) to 0xF2, write the following to address 0xE0; (0xA0, 0xAA, 0xA5, 0xF2). These commands must be sent in the correct sequence to change the I2C address, additionally, No other command may be issued in the middle of the sequence. The sequence must be sent to the command register at location 0, which means 4 separate write transactions on the I2C bus. When done, you should label the sonar with its address, however if you do forget, just power it up without sending any commands. The SRF10 will flash its address out on the LED. One long flash followed by a number of shorter flashes indicating its address. The flashing is terminated immediately on sending a command the SRF10.

Address		Long Flash	Short flashes
Decimal	Hex		
224	E0	1	0
226	E2	1	1
228	E4	1	2
230	E6	1	3
232	E8	1	4
234	EA	1	5
236	EC	1	6
238	EE	1	7
240	F0	1	8
242	F2	1	9
244	F4	1	10
246	F6	1	11
248	F8	1	12
250	FA	1	13
252	FC	1	14
254	FE	1	15

Take care not to set more than one sonar to the same address, there will be a bus collision and very unpredictable results.

Changing beam pattern and beam width

You can't! This is a question which crops up regularly, however there is no easy way to reduce or change the beam width that I'm aware of. The beam pattern of the SRF10 is conical with the width of the beam being a function of the surface area of the transducers and is fixed. It is possible to make the sonar less sensitive to objects off to the side by reducing the maximum gain register from 16 to a lower level. This is a the expense of shorter range, however most small robots don't need 6m of range. A value of 8 (max. gain 140) will reduce the practicable range to about 2m, but it will be much less sensitive to objects off the center line. The beam pattern of the transducers used on the SRF10, taken from the manufacturers data sheet, is shown below.



There is more information in the sonar faq.

Mounting the SRF10

You may have notice that there are no mounting holes on the SRF10 module! That was deliberate to keep the module as small as possible. So how do you mount it? Here are three suggestions:

1. A straight or right angle 0.1 inch connector soldered to your PCB.

2. Using two 9.5mm rubber grommets. Two holes should be drilled into the panel you're mounting the SRF10 to. The hole centers should be 0.7inches (17.78mm) apart and the holes drilled 0.5 inches (12.7mm) in diameter. The two grommets should then be fitted to the panel and the SRF10 gently pushed into them.

3. Using our SRF10 Mounting Kit, shown below.