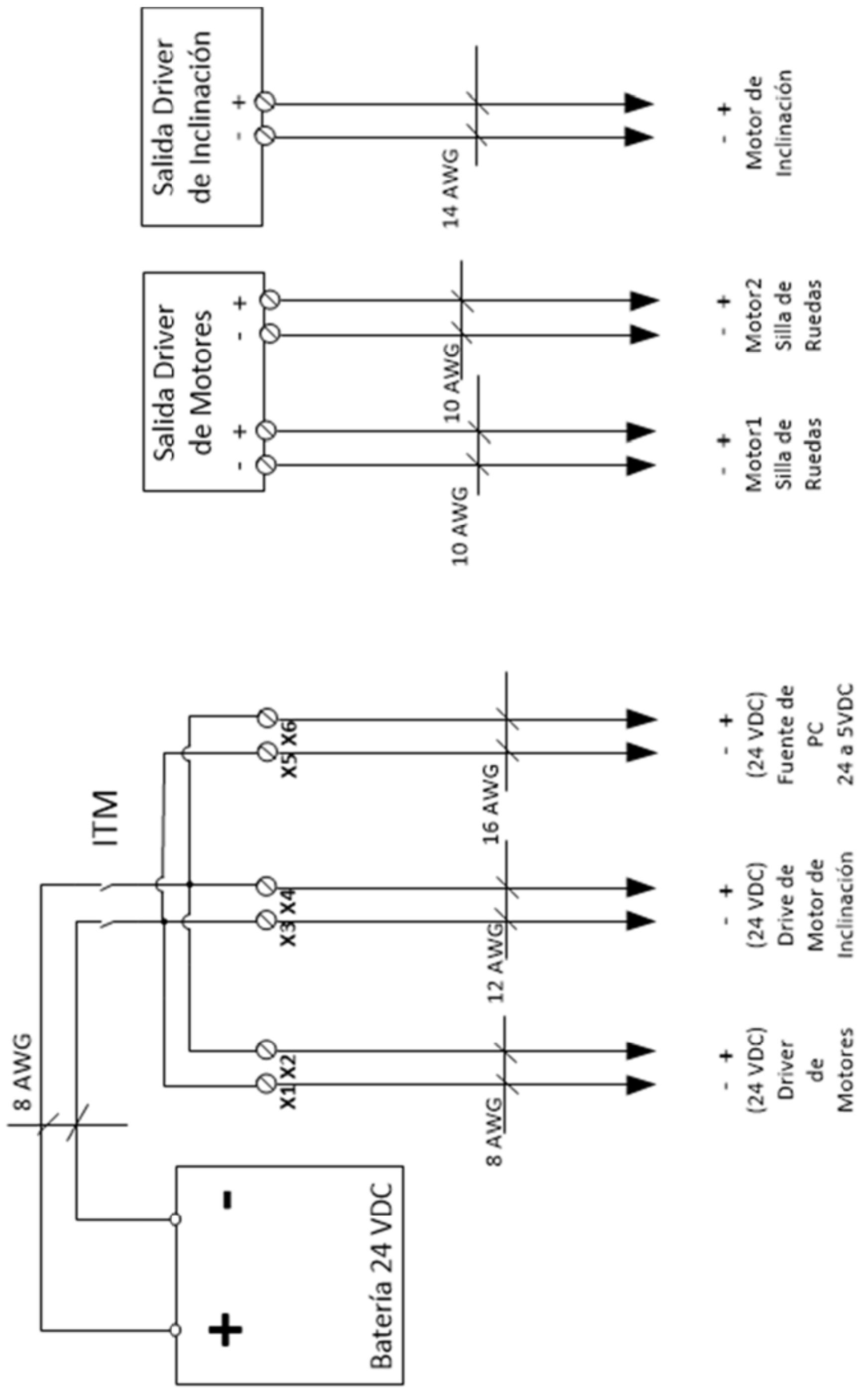
7 Anexos

7.1 Anexo A: Sistemas Electrónicos

7.1.1 Anexo A.1 – Esquema Eléctrico del Sistema Propuesto



Anexo A.1 Plano Eléctrico del Sistema Propuesto
Fuente: Propia

7.2 *Anexo B: Programas*

7.2.1 Anexo B.1 – Programa de la Computadora Principal

```
#ifdef _WIN32
#include <windows.h>
#include <conio.h>
#include <vcclr.h>
#include <iostream>
#endif

#include <GL/gl.h>
#include <GL/glu.h>
#include <glut.h>
#include <iostream>

#ifdef __linux__
#include <unistd.h>
#endif

#include <cmath>

#using <System.dll>

using namespace std;
using namespace System;
using namespace System::IO::Ports;
using namespace System::ComponentModel;
using namespace System::Threading;

#include "edk.h"
#include "edkErrorCode.h"
#include "EmoStateDLL.h"

#define PI 3.1416

bool  oneTime    = true,
      outOfBound = false;
float currX      = 0,
      currY      = 0;
float xmax       = 0,
      ymax       = 0,

      x          = 0;
float preX       = 0,
      preY       = 0;
int   incOrDec   = 10;
int   count      = 0;
char   orden_mov;
char   orden_prev = 's';


static gcroot<String^> in_buffer;
static gcroot<String^> out_buffer;


public ref class PortChat
{
public:
    static SerialPort^ _serialPort;
```

```
public:
    static void Main()
    {
        //String^ name;
        //String^ message;
        //StringComparer^ stringComparer = StringComparer::OrdinalIgnoreCase;
        //Thread^ readThread = gcnew Thread(gcnew
ThreadStart(PortChat::Read));

        // Create a new SerialPort object with default settings.
        _serialPort = gcnew SerialPort();

        // Allow the user to set the appropriate properties.
        _serialPort->PortName = "COM3";
               _serialPort->BaudRate = 9600;
              _serialPort->Parity = Parity::None;
              _serialPort->DataBits = 8;
              _serialPort->StopBits = StopBits::One;
              _serialPort->Handshake = Handshake::None;
              // Set the read/write timeouts
        _serialPort->ReadTimeout = 2000;
        _serialPort->WriteTimeout = 2000;
       }
};


float oldXVal    = 0,
      oldYVal    = 0;
double maxRadius = 10000;
unsigned long pass = 0,
              globalElapsed = 0;

void init(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel (GL_FLAT);
}

void drawCircle( float Radius, int numPoints )
{
  glBegin( GL_LINE_STRIP );
    for( int i=0; i<numPoints; i++ )
    {
              float Angle = i * (2.0*PI/numPoints); // use 360 instead of
2.0*PI if
              float X = cos( Angle )*Radius;  // you use d_cos and d_sin
              float Y = sin( Angle )*Radius;
              glVertex2f( X, Y );
       }
  glEnd();
}

void drawFilledCircle(float radius)
{
       glEnable(GL_POINT_SMOOTH);
       glHint(GL_POINT_SMOOTH_HINT, GL_NICEST);
       glPointSize(radius);
}

/*
Crea el display con los 3 círculos
```

```
*/
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glPushMatrix();

   glColor3f(1.0,1.0,1.0);
   drawCircle(800,100);
   glColor3f(0.0,0.0,1.0);
   drawCircle(maxRadius-4000,800);
   glColor3f(0.0,1.0,1.0);
   drawCircle(maxRadius,1000);


   glColor3f(1.0, 0.0, 0.0);
   glRectf(currX-400.0, currY-400.0, currX+400.0, currY+400.0);

   glPopMatrix();
   glutSwapBuffers();
}

void changeXY(int x) // x = 0 : idle
{
       if( currX >0 )
       {
              float temp = currY/currX;
              currX -= incOrDec;
              currY = temp*currX;
       }
       else if( currX < 0)
       {
              float temp = currY/currX;
              currX += incOrDec;
              currY = temp*currX;
       }
       else
       {
              if( currY > 0 ) currY -= incOrDec;
              else if( currY <0 ) currY += incOrDec;
       }
       if( x == 0)
       {
        if( (std::abs(currX) <= incOrDec) && (std::abs(currY) <= incOrDec))
              {
                     xmax = 0;
                     ymax = 0;
              }
              else
              {
                     xmax = currX;
                     ymax = currY;
              }
       }
       else
       {
        if( (std::abs(currX) <= incOrDec) && (std::abs(currY) <= incOrDec))
              {
                     xmax = 0;
                     ymax = 0;
              }
       }
}
```

```
void updateDisplay()
{

        int gyroX = 0,gyroY = 0;
   EE_HeadsetGetGyroDelta(0,&gyroX,&gyroY);
   if ( ((orden_mov != 'D') || (gyroX >= 0)) && ((orden_mov != 'A') || (gyroX
<= 0)) )
   {
              xmax += gyroX;       //     integrando el valor del acelerómetro
en X
   }
   ymax -= gyroY;    //     integrando el valor del acelerómetro en Y

   if( outOfBound )
   {
          if( preX != gyroX && preY != gyroY )
          {
                 xmax = currX;
                 ymax = currY;
          }
   }

   double val = sqrt((float)(xmax*xmax + ymax*ymax));

   if  (val <= 2000 )
   {
              orden_mov = 'S';
              if( (orden_prev != 's') && (orden_prev != 'S') )
              {
                     out_buffer = "S";
                     PortChat::_serialPort->Write(out_buffer);
                     PortChat::_serialPort->DiscardOutBuffer();
                     out_buffer = "";
                     orden_prev = orden_mov;
//                   in_buffer = PortChat::_serialPort->ReadLine();
                     PortChat::_serialPort->DiscardInBuffer();
                     Console::WriteLine(in_buffer);
                     //std::cout <<in_buffer << std::endl;
              }
   }
  else
   {
              if ((ymax >= xmax) && (ymax >= -xmax))              //arriba
              {
                     orden_mov = 'a';
                     if(orden_prev != 'a')
                     {
                            out_buffer ="a";
                            PortChat::_serialPort->Write(out_buffer);
                            PortChat::_serialPort->DiscardOutBuffer();
                            out_buffer = "";
                            orden_prev = orden_mov;
//                          in_buffer = PortChat::_serialPort->ReadLine();
                            PortChat::_serialPort->DiscardInBuffer();
                            Console::WriteLine(in_buffer);
                            //std::cout <<in_buffer << std::endl;
                     }
              }
              else if ((ymax <= xmax) && (ymax <= -xmax))      //abajo
              {
```

```
                        orden_mov = 'd';
                        if(orden_prev != 'd')
                        {
                                out_buffer ="d";
                                PortChat::_serialPort->Write(out_buffer);
                                PortChat::_serialPort->DiscardOutBuffer();
                                out_buffer = "";
                                orden_prev = orden_mov;
//                              in_buffer = PortChat::_serialPort->ReadLine();
                                PortChat::_serialPort->DiscardInBuffer();
                                Console::WriteLine(in_buffer);
                                //std::cout <<in_buffer << std::endl;
                        }
                }
                else if ((ymax > xmax) && (ymax < -xmax))       //izquierda
                {
                        orden_mov = 'D';
                        if(orden_prev != 'D')
                        {
                                out_buffer = "D";
                                PortChat::_serialPort->Write(out_buffer);
                                PortChat::_serialPort->DiscardOutBuffer();
                                out_buffer = "";
                                orden_prev = orden_mov;
//                              in_buffer = PortChat::_serialPort->ReadLine();
                                PortChat::_serialPort->DiscardInBuffer();
                                Console::WriteLine(in_buffer);
                                //std::cout <<in_buffer << std::endl;
                        }
                }
                else if ((ymax < xmax) && (ymax > -xmax))       //derecha
                {
                        orden_mov = 'A';
                        if(orden_prev != 'A')
                        {
                                out_buffer = "A";
                                PortChat::_serialPort->Write(out_buffer);
                                PortChat::_serialPort->DiscardOutBuffer();
                                out_buffer = "";
                                orden_prev = orden_mov;
//                              in_buffer = PortChat::_serialPort->ReadLine();
                                PortChat::_serialPort->DiscardInBuffer();
                                Console::WriteLine(in_buffer);
                                //std::cout <<in_buffer << std::endl;
                        }
                }
                else
                {
                }
        }



//   std::cout <<"val : " << val << std::endl;

   if( val >= maxRadius )
   {
        changeXY(1);
        outOfBound = true;
        preX = gyroX;
        preY = gyroY;
```

```
    }
    else
    {
            outOfBound = false;
                if(oldXVal == gyroX && oldYVal == gyroY)
                {
                        ++count;
                        if( count > 10 )
                        {

                                changeXY(0);
                        }
                }
                else
                {
                        count = 0;
                        currX = xmax;
                        currY = ymax;
                        oldXVal = gyroX;
                        oldYVal = gyroY;
                }
    }
#ifdef _WIN32
      Sleep(15);
#endif
#ifdef __linux__
      sleep(1);
#endif
   glutPostRedisplay();
}

void reshape(int w, int h)
{
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-50000.0, 50000.0, -50000.0, 50000.0, -1.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void mouse(int button, int state, int x, int y)
{
       switch (button) {
      case GLUT_LEFT_BUTTON:
         if (state == GLUT_DOWN)
            glutIdleFunc(updateDisplay);
         break;
      case GLUT_MIDDLE_BUTTON:
         if (state == GLUT_DOWN)
            glutIdleFunc(NULL);
         break;
      default:
         break;
   }
}

#ifdef __linux__
double GetTickCount()
{
    struct timespec now;
    if (clock_gettime(CLOCK_MONOTONIC, &now))
```

```
      return 0;
    return now.tv_sec * 1000.0 + now.tv_nsec / 1000000.0;
}
#endif

/*
 *  Request double buffer display mode.
 *  Register mouse input callback functions
 */

int main(int argc, char** argv)
{

       EmoEngineEventHandle hEvent = EE_EmoEngineEventCreate();
   EmoStateHandle eState = EE_EmoStateCreate();
   unsigned int userID = -1;
   EE_EngineConnect();

   if(oneTime)
   {
      std::cout << "Start after 8 seconds\n";
#ifdef _WIN32
         Sleep(8000);
#endif
#ifdef __linux__
      sleep(8);
#endif


#ifdef _WIN32
   globalElapsed = GetTickCount();
#endif
#ifdef __linux__
   globalElapsed = ( unsigned long ) GetTickCount();
#endif
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize (650, 650);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
       PortChat::Main();
       PortChat::_serialPort->Open();


   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutIdleFunc(updateDisplay);
   glutMainLoop();

       PortChat::_serialPort->Close();
   EE_EngineDisconnect();
   EE_EmoStateFree(eState);
   EE_EmoEngineEventFree(hEvent);



   return 0;
   }
}
```

7.2.2 Anexo B.2 – Programa del Sistema Embebido

```
#define valorT 31249U
#define F_CPU 16000000UL                    // CPU clock setup
#define BAUD 9600                                     // transmission
speed Probe sensor
#define BAUD2 9600                                // transmission speed
Inertial sensor
#define BAUDRATE ((F_CPU)/(BAUD*8UL)-1)
#define BAUDRATE2 ((F_CPU)/(BAUD2*8UL)-1)
#define RETARDO 10
#define num_canal 3

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

//char RX0_RCV[100];
unsigned char RX2_DATA[100];
//char TX0_RCV[100];
unsigned char TX2_DATA[100];
unsigned char buffer0;
unsigned char buffer2;
uint8_t data_length_COM2 = 0;
uint8_t data_length_COM0 = 0;
uint8_t temp = 0;
uint8_t complete_RX_COM2 = 0;
uint8_t complete_RX_COM0 = 0;

char carril = 0x0A;
char salto = 0x0D;

int    temp_ADC;
int sensores_ADC[num_canal];
int TX_ADC[num_canal*5];
int    no_avance;


void TIMER1_setup(void)
{
       TCCR1A |= (0<<COM1A1 | 0<<COM1A0 | 0<<WGM11 | 0<<WGM10);
       TCCR1B |= (0<<WGM13 | 1<<WGM12 | 1<<CS12 | 0<<CS11 | 1<<CS10);
//Prescaler 1024
       OCR1A = valorT; // inicializa variable comparador
       TIMSK1 |= (1<<OCIE1A); // Comparator Interrupt enable
}

void USART0_setup(void)
{
       UBRR0H = (BAUDRATE>>8);
       UBRR0L = BAUDRATE;
       UCSR0A |= (0<<RXC0 | 0<<TXC0 | 1<<U2X0 | 0<<MPCM0);
       // Comunicación asíncrona, sin paridad, 1 bit de parada, 8 bits de
datos
       UCSR0C |= (0<<UMSEL01 | 0<<UMSEL00 | 0<<UPM01 | 0<<UPM00 | 0<<USBS0 |
1<<UCSZ01 | 1<<UCSZ00);
       // Tx y Rx habilitadas, interrupción Rx habilitada
       UCSR0B |= (1<<RXCIE0 | 0<<TXCIE0 | 0<<UDRIE0 | 1<<RXEN0| 1<<TXEN0 |
0<<UCSZ02 | 0<<TXB80);
}
unsigned char USART0_receive()
{
```

```
        // Se crea un lazo mientras el buffer de transmisión no este vacío
        while(!(UCSR0A & (1<<UDRE0)));
        // Cuando el buffer este vacio, when the buffer is empty write data to
the transmitted
        return UDR0;
}
void USART0_send(unsigned char data)
{
        // Se crea un lazo mientras el buffer de transmisión no este vacío
        while(!(UCSR0A & (1<<UDRE0)));
        // Cuando el buffer este vacio, when the buffer is empty write data to
the transmitted
        UDR0 = data;
}

void USART2_setup(void)
{
        UBRR2H = (BAUDRATE2>>8);
        UBRR2L = BAUDRATE2;
        UCSR2A = 0x02;
        // Comunicación asíncrona, paridad par, 1 bit de parada, 7 bits de
datos
        UCSR2C = 0x24;
        // Tx y Rx habilitadas, interrupción Rx habilitada
        UCSR2B = 0x98;
}
unsigned char USART2_receive()
{
        // Se crea un lazo mientras el buffer de transmisión no este vacío
        while(!(UCSR2A & (1<<UDRE2)));
        // Cuando el buffer este vacio, when the buffer is empty write data to
the transmitted
        return UDR2;
}
void USART2_send(unsigned char data)
{
        // Se crea un lazo mientras el buffer de transmisión no este vacío
        while(!(UCSR2A & (1<<UDRE2)));
        // Cuando el buffer este vacio, when the buffer is empty write data to
the transmitted
        UDR2 = data;
}

ISR (USART2_RX_vect)
{
        cli();
        buffer2 = USART2_receive();

        if(buffer2 == 0x0D)
        {
                RX2_DATA[data_length_COM2] = buffer2;
                complete_RX_COM2 = 1;
        }
        else
        {
                RX2_DATA[data_length_COM2] = buffer2;
                data_length_COM2++;
        }
        sei();
}

ISR (USART0_RX_vect)
```

```
{
        cli();
        buffer0 = USART0_receive();

        sei();
}

void ADCInit()
{
        ADMUX = _BV(REFS0) | _BV(ADLAR)  | _BV(MUX0) ;
        ADCSRA = _BV(ADEN) | _BV(ADPS0) | _BV(ADPS1) | _BV(ADPS2);
}
int ADCRead(unsigned int canal)
{
        ADMUX = (ADMUX & 0xF8) | (canal & 0x07);
        // se inicia la conversión.
        ADCSRA |= _BV(ADSC);
        // espera a que termina la conversión.
        while(ADCSRA & _BV(ADSC));
        // regresamos el valor de la conversión.
        temp_ADC = ADCH;
        return temp_ADC;
}
void Lectura_ADC(void)
{
        int i=0;
        for (i=0; i<=(num_canal-1) ; i++)
        {
                sensores_ADC[i] = 4*ADCRead(i);
        }
        no_avance = 0;
        for (i=0; i<=(num_canal-1) ; i++)
        {
                if (sensores_ADC[i] < 100)
                {
                        no_avance = 1;
                }
        }

        if ((buffer0 == 'a') && (no_avance == 0))// arriba
        {
                TX2_DATA[0] = '!';
                TX2_DATA[1] = 'a';
                TX2_DATA[2] = '1';
                TX2_DATA[3] = 'F';
                TX2_DATA[4] = salto;
                TX2_DATA[5] = '!';
                TX2_DATA[6] = 'B';
                TX2_DATA[7] = '1';
                TX2_DATA[8] = 'F';
                TX2_DATA[9] = salto;
                data_length_COM0 = 9 ;
                complete_RX_COM0 = 1;
        }
        else if (buffer0 == 'd') // abajo
        {
                TX2_DATA[0] = '!';
                TX2_DATA[1] = 'A';
                TX2_DATA[2] = '1';
                TX2_DATA[3] = 'F';
                TX2_DATA[4] = salto;
                TX2_DATA[5] = '!';
```

```
                TX2_DATA[6] = 'b';
                TX2_DATA[7] = '1';
                TX2_DATA[8] = 'F';
                TX2_DATA[9] = salto;
                data_length_COM0 = 9 ;
                complete_RX_COM0 = 1;
        }
        else if (buffer0 == 'D')    // izquierda
        {
                TX2_DATA[0] = '!';
                TX2_DATA[1] = 'A';
                TX2_DATA[2] = '1';
                TX2_DATA[3] = 'F';
                TX2_DATA[4] = salto;
                TX2_DATA[5] = '!';
                TX2_DATA[6] = 'B';
                TX2_DATA[7] = '1';
                TX2_DATA[8] = 'F';
                TX2_DATA[9] = salto;
                data_length_COM0 = 9 ;
                complete_RX_COM0 = 1;
        }
        else if (buffer0 == 'A')    // derecha
        {
                TX2_DATA[0] = '!';
                TX2_DATA[1] = 'a';
                TX2_DATA[2] = '1';
                TX2_DATA[3] = 'F';
                TX2_DATA[4] = salto;
                TX2_DATA[5] = '!';
                TX2_DATA[6] = 'b';
                TX2_DATA[7] = '1';
                TX2_DATA[8] = 'F';
                TX2_DATA[9] = salto;
                data_length_COM0 = 9 ;
                complete_RX_COM0 = 1;
        }
        else if ((buffer0 == 'S') || (buffer0 == 's') || ((buffer0 == 'a') &&
(no_avance == 1)) )
        {
                TX2_DATA[0] = '!';
                TX2_DATA[1] = 'a';
                TX2_DATA[2] = '0';
                TX2_DATA[3] = '0';
                TX2_DATA[4] = salto;
                TX2_DATA[5] = '!';
                TX2_DATA[6] = 'b';
                TX2_DATA[7] = '0';
                TX2_DATA[8] = '0';
                TX2_DATA[9] = salto;
                data_length_COM0 = 9 ;
                complete_RX_COM0 = 1;
        }
        else
        {
                TX2_DATA[0] = ' ';
                TX2_DATA[1] = ' ';
                TX2_DATA[2] = ' ';
                TX2_DATA[3] = ' ';
                TX2_DATA[4] = salto;
                TX2_DATA[5] = ' ';
                TX2_DATA[6] = ' ';
```

```
			TX2_DATA[7] = ' ';
			TX2_DATA[8] = ' ';
			TX2_DATA[9] = salto;
			data_length_COM0 = 0 ;
			complete_RX_COM0 = 0;
		}

	}


	// Programa Principal
	int main(void)
	{
		DDRB = DDRB | 0x20;
		PORTB = 0xDF;
		DDRE = DDRE | 0x08;
		PORTE = 0xF7;
		DDRH = DDRH | 0x08;
		PORTH = 0xF4;
		DDRL = DDRL | 0x08;
		PORTL = 0xF7;
		//TIMER1_setup();
		USART0_setup();
		USART2_setup();
		ADCInit();

		temp_ADC = 0;
		int i = 0;
		for (i=0 ; i<=(num_canal-1) ; i++)
		{
			sensores_ADC[i] = 0;
		}
		sei();
		while(1)
		{
			Lectura_ADC();

			if(complete_RX_COM0 == 1)
			{
				for(int i=0; i<=(data_length_COM0); i++)
				{
					if (TX2_DATA[i] != 0x00)
					{
						USART2_send(TX2_DATA[i]);
						_delay_ms(RETARDO);
						TX2_DATA[i] = ' ';
					}
					if (i==data_length_COM0)
					{
						data_length_COM0 = 0;
						complete_RX_COM0 = 0;
					}
				}
			}

			if(complete_RX_COM2 == 1)
			{
				for(int i=0; i<=(data_length_COM2); i++)
				{
					if (RX2_DATA[i] != 0x00)
					{
						USART0_send(RX2_DATA[i]);
```

```
                                        _delay_ms(RETARDO);
                                        RX2_DATA[i] = ' ';
                                }
                                if (i==data_length_COM2)
                                {
                                        data_length_COM2 = 0;
                                        complete_RX_COM2 = 0;
                                }
                        }
                }

        }
}
```

7.3 Anexo C: Hojas Técnicas

7.3.1 Anexo C.1 – Hoja Técnica - EMOTIV EPOC+

Emotiv EPOC

	EEG HEADSET
Number of channels	14 (plus CMS/DRL references, P3/P4 locations)
Channel names (International 10-20 locations)	AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4
Sampling method	Sequential sampling. Single ADC
Sampling rate	128 SPS (2048 Hz internal)
Resolution	14 bits 1 LSB = 0.51μV (16 bit ADC, 2 bits instrumental noise floor discarded)
Bandwidth	0.2 - 45Hz, digital notch filters at 50Hz and 60Hz
Filtering	Built in digital 5th order Sinc filter
Dynamic range (input referred)	8400μV (pp)
Coupling mode	AC coupled
Connectivity	Proprietary wireless, 2.4GHz band
Power	LiPoly
Battery life (typical)	12 hours
Impedance Measurement	Real-time contact quality using patented system

EMC and Telecom: Class B
ETSI EN 300 440-2 V1.4.1
EN 301 489-1
EN 301 489-3
AS/NZS CISPR22 :2009
AS/NZS 4268 :2008
FCC CFR 47 Part 15C (identifiers XUE-EPOC01, XUE-USBD01)

Safety:
EN 60950-1:2006
IEC 60950-1:2005 (2nd Edition)
AS/NZS 60950.1:2003 including amendments 1, 2 & 3
CB Certificate JPTUV-029914 (TUV Rheinland)

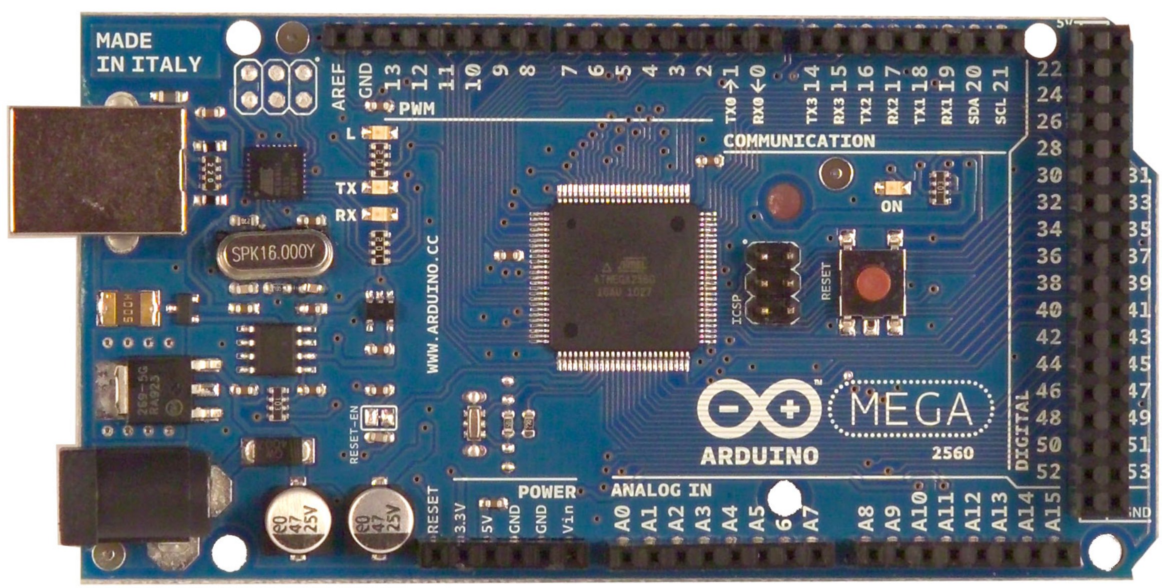
7.3.2 Anexo C.2 – Hoja Técnica – Arduino Mega ADK

www.robotshop.com

La robotique à votre service! - Robotics at your service!

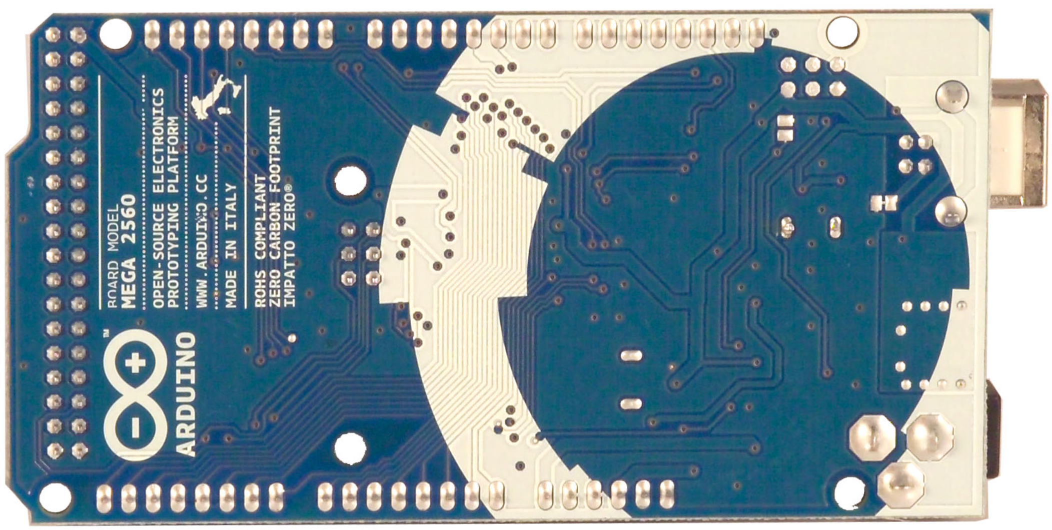
ARDUINO

Arduino Mega 2560 Datasheet



RobotShop www.robotshop.com

La robotique à votre service! - Robotics at your service!



Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

Schematic & Reference Design

EAGLE files: arduino-mega2560-reference-design.zip

Schematic: arduino-mega2560-schematic.pdf

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Power

The Arduino Mega can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode() , digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH

value, the LED is on, when the pin is LOW, it's off.

- **I^2C: 20 (SDA) and 21 (SCL).** Support I^2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I^2C pins on the Duemilanove or Diecimila.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega8U2 on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).
A SoftwareSerial library allows for serial communication on any of the Mega2560's digital pins.
The ATmega2560 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website for details. For SPI communication, use the SPI library.

Programming

The Arduino Mega can be programmed with the Arduino software (download). For details, see the reference and tutorials.
The ATmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It

communicates using the original STK500 protocol (reference, C header files).
You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.
This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.
The Mega2560 contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Mega2560 has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega2560 PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega2560 is designed to be compatible with most shields designed for the Uno, Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega2560 and Duemilanove / Diecimila. *Please note that I_2C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).*

7.3.3 Anexo C.3 – Hoja Técnica – Silla de Ruedas Mecatrónica

JIANGNAN MACHINERY CO.,LIMITED
No.6 xinxiang road,xianglu village,wangting town,suzhou,china
TEL:+86-512-66700071 FAX:+86-512-66709733 Mobile:+86 18868715511

Electric wheelchairs

To: Pontificia Universidad Catolica del Peru
Av. Universitaria 1801, San Miguel,Lima 32 - Peru

Photos	Item No.	Technical specifications		
	**Maximus EW-001**	Control system	PG imported from UK	
		Dimensions	1150*730*1370mm	
		Weight with batteries	197kg	
		Load Capacity	150kg	
		Tyre size	Ø400mm	
		Seat size	16”/18”/20”	
		Seat to floor	580mm	
		Chassis to floor	120mm	
		Batteries	12V,27AH,4pcs	
		Driving motor	1350W x 2pcs	
			Imported from Taiwan	
		Level rack motor	300W x 1pcs	
		Max speed	7.0km/h(10km/h available)	
		Max mileage	15-18km	
		Operation time	2.0-3.0h	
		Charging time	6-8h	
		Turning radius	0°	
		Step climble angle	max 25°	
		Slope climble angle	max 45°	
		Brake	Electromagnetic braking	
		Drive system	Four wheel drive	
		Stair track	Optional	
		Tilt seat	Optional	

7.3.4 Anexo C.4 – Hoja Técnica – Driver RoboteQ AX2550

RoboteQ

AX2550
AX2850

Dual Channel High Power Digital Motor Controller

User's Manual

v1.9b, June 1, 2007

visit www.roboteq.com to download the latest revision of this manual

SECTION 3

AX2550 Motor Controller Overview

Congratulations! By selecting Roboteq’s AX2550 you have empowered yourself with the industry’s most versatile, powerful and programmable DC Motor Controller for mobile robots. This manual will guide you step by step through its many possibilities.

Product Description

The AX2550 is a highly configurable, microcomputer-based, dual-channel digital speed or position controller with built-in high power drivers. The controller is designed to interface directly to high power DC motors in computer controlled or remote controlled mobile robotics and automated vehicle applications.

The AX2550 controller can accept speed or position commands in a variety of ways: pulse-width based control from a standard Radio Control receiver, Analog Voltage commands, or RS-232 commands from a microcontroller or wireless modem.

The controller's two channels can be operated independently or can be combined to set the forward/reverse direction and steering of a vehicle by coordinating the motion on each side of the vehicle. In the speed control mode, the AX2550 can operate in open loop or closed loop. In closed loop operation, actual speed measurements from tachometers or optical encoders are used to verify that the motor is rotating at the desired speed and direction and to adjust the power to the motors accordingly.

The AX2550 can also be configured to operate as a precision, high torque servo controller. When connected to a potentiometer coupled to the motor assembly, the controller will command the motor to rotate up to a desired angular position. Depending on the DC motor's power and gear ratio, the AX2550 can be used to move or rotate steering columns or other physical objects with very high torque.

The AX2550 is fitted with many safety features ensuring a secure power-on start, automatic stop in case of command loss, over current protection on both channels, and overheat protection.

The motors are driven using high-efficiency Power MOSFET transistors controlled using Pulse Width Modulation (PWM) at 16kHz. The AX2550 power stages can operate from 12 to 40VDC and can sustain up to 120A of controlled current, delivering up to 4,800W (approximately 6 HP) of useful power to each motor.

The many programmable options of the AX2550 are easily configured using the supplied PC utility or one-touch Program and Set buttons and a 7-segment LED display. Once programmed, the configuration data are stored in the controller's non-volatile memory, eliminating the need for cumbersome and unreliable jumpers.

The AX2850 is the AX2550 controller fitted with a dual channel optical encoder input module. Optical Encoders allow precise motor speed and position measurement and enable advance robotic applications.

Technical features

Fully Digital, Microcontroller-based Design

- Multiple operating modes
- Fully programmable using either built-in switches and 7 segment display or through connection to a PC
- Non-volatile storage of user configurable settings
- Simple operation
- Software upgradable with new features

Multiple Command Modes

- Radio-Control Pulse-Width input
- Serial port (RS-232) input
- 0-5V Analog Command input

Multiple Advanced Motor Control Modes

- Independent operation on each channel
- Mixed control (sum and difference) for tank-like steering
- Open Loop or Closed Loop Speed mode
- Position control mode for building high power position servos
- Modes selectable independently for each channel

Automatic Joystick Command Corrections

- Joystick min, max and center calibration
- Selectable deadband width
- Selectable exponentiation factors for each joystick
- 3rd R/C channel input for accessory output activation (disabled when encoder module present)

Special Function Inputs/Outputs

- 2 Analog inputs. Used as:

    - Tachometer inputs for closed loop speed control
    - Potentiometer input for position (servo mode)
    - Motor temperature sensor inputs
    - External voltage sensors
    - User defined purpose (RS232 mode only)
- 2 Extra analog inputs (on RevB hardware). Used as:
    - Potentiometer input for position while in analog command mode
    - User defined purpose (RS232 mode only)
- One Switch input configurable as
    - Emergency stop command
    - Reversing commands when running vehicle inverted
    - General purpose digital input
- One general purpose 24V, 2A output for accessories
- Up to 2 general purpose digital inputs

Optical Encoder Inputs (AX2850 only)

- Inputs for two Quadrature Optical Encoders
- up to 250khz Encoder frequency per channel
- two 32-bit up-down counters
- Inputs may be shared with four optional limit switches per channel

Internal Sensors

- Voltage sensor for monitoring the main 12 to 40V battery system operation
- Voltage monitoring of internal 12V
- Temperature sensors on the heat sink of each power output stage
- Sensor information readable via RS232 port

Low Power Consumption

- On board DC/DC converter for single 12 to 40V battery system operation
- Optional backup power input for powering safely the controller if the motor batteries are discharged
- Max 200mA at 12V or 100mA at 24V idle current consumption
- Power Control wire for turning On or Off the controller from external microcomputer or switch
- No power consumed by output stage when motors are stopped
- Regulated 5V output for powering R/C radio. Eliminates the need for separate R/C battery

High Efficiency Motor Power Outputs

- Two independent power output stages
- Optional Single Channel operation at double the current
- Dual H bridge for full forward/reverse operation
- Ultra-efficient 2.5mOhm (1.25mOhm on HE version) ON resistance (RDSon) MOS-FET transistors
- Synchronous Rectification H Bridge

- 12 to 40 V operation
- High current 8 AWG cable sets for each power stages
- Temperature-based Automatic Current Limitation
  - 120A up to 15 seconds (per channel)
  - 100A up to 30 seconds
  - 80A extended
  - High current operation may be extended with forced cooling
- 250A peak Amps per channel
- 16kHz Pulse Width Modulation (PWM) output
- Auxiliary output for brake, clutch or armature excitation
- Heat sink extruded case

Advanced Safety Features

- Safe power on mode
- Optical isolation on R/C control inputs
- Automatic Power stage off in case of electrically or software induced program failure
- Overvoltage and Undervoltage protection
- Regeneration current limiting
- Watchdog for automatic motor shutdown in case of command loss (R/C and RS232 modes)
- Large, bright run/failure diagnostics on 7 segment LED display
- Programmable motor acceleration
- Built-in controller overheat sensor
- Emergency Stop input signal and button

Data Logging Capabilities

- 13 internal parameters, including battery voltage, captured R/C command, temperature and Amps accessible via RS232 port
- Data may be logged in a PC, PDA or microcomputer

Sturdy and Compact Mechanical Design

- Built from aluminum heat sink extrusion with mounting brackets
- Efficient heat sinking. Operates without a fan in most applications.
- 7” (178mm) long (excluding mounting brackets) by 5.5” wide (140mm) by 1.8” (40mm) high
- -20o to +85o C case operating environment
- 3.3 lbs (1500g)

SECTION 4

Connecting Power and Motors to the Controller

This section describes the AX2550 Controller's connections to power sources and motors.

Important Warning

Please follow the instructions in this section very carefully. Any problem due to wiring errors may have very serious consequences and will not be covered by the product's warranty.

Power Connections

The AX2550 has three Ground (black), two Vmot (red) power cables and a Power Control wire (yellow). The power cables are located at the back end of the controller. The various power cables are identified by their position, wire thickness and color: Red is positive (+), black is negative or ground (-).

The power connections to the batteries and motors are shown in the figure below.

Programmable Acceleration

When changing speed command, the AX2550 will go from the present speed to the desired one at a user selectable acceleration. This feature is necessary in order to minimize the surge current and mechanical stress during abrupt speed changes.

This parameter can be changed by using the controller's front switches or using serial commands. When configuring the controller using the switches (see "Configuring the Controller using the Switches" on page 171), acceleration can be one of 6 available preset values, from very soft(0) to very quick (6). The AX2550's factory default value is medium soft (2).

When using the serial port, acceleration can be one of 24 possible values, selectable using the Roborun utility or entering directly a value in the MCU's configuration EEPROM. Table 6 shows the corresponding acceleration for all Switch and RS232 settings.

Numerically speaking, each acceleration value corresponds to a fixed percentage speed increment, applied every 16 milliseconds. The value for each setting is shown in the table below.

TABLE 6. Acceleration setting table

Acceleration Setting Using RS232	Acceleration Setting Using Switches	%Acceleration per 16ms	Time from 0 to max speed
30 Hex		0.78%	2.05 seconds
20 Hex		1.56%	1.02 seconds
10 Hex		2.34%	0.68 second
00 Hex	0	3.13%	0.51 second
31 Hex		3.91%	0.41 second
21 Hex		4.69%	0.34 second
11 Hex		5.47%	0.29 second
01 Hex	1	6.25%	0.26 second
32 Hex	-	7.03%	0.23 second
22 Hex	-	7.81%	0.20 second
12 Hex	-	8.59%	0.19 second
02 Hex	2 (default)	9.38%	0.17 second
33 Hex	-	10.16%	0.16 second
23 Hex	-	10.94%	0.15 second
13 Hex	-	11.72%	0.14 second
03 Hex	3	12.50%	0.128 second
34 Hex	-	13.28%	0.120 second
24 Hex	-	14.06%	0.113 second
14 Hex	-	14.84%	0.107 second
04 Hex	4	15.63%	0.102 second
35 Hex	-	16.41%	0.097 second
25 Hex	-	17.19%	0.093 second

TABLE 6. Acceleration setting table

Acceleration Setting Using RS232	Acceleration Setting Using Switches	%Acceleration per 16ms	Time from 0 to max speed
15 Hex	-	17.97%	0.089 second
05 Hex	5	18.75%	0.085 second

When configuring the acceleration parameter using the Roborun utility, four additional acceleration steps can be selected between the six ones selectable using the switch, extending the slowest acceleration to 2.04 seconds from 0 to max speed. See "Power Settings" on page 182 for details on how to configure this parameter using Roborun.

Important Warning

Depending on the load's weight and inertia, a quick acceleration can cause considerable current surges from the batteries into the motor. A quick deceleration will cause an equally large, or possibly larger, regeneration current surge. Always experiment with the lowest acceleration value first and settle for the slowest acceptable value.

Command Control Curves

The AX2550 can also be set to translate the joystick or RS232 motor commands so that the motors respond differently whether or not the joystick is near the center or near the extremes.

The controller can be configured to use one of 5 different curves independently set for each channel.

The factory default curve is a "linear" straight line, meaning that after the joystick has moved passed the deadband point, the motor's speed will change proportionally to the joystick position.

Two "exponential' curves, a weak and a strong, are supported. Using these curves, and after the joystick has moved past the deadband, the motor speed will first increase slowly, increasing faster as the joystick moves near the extreme position. Exponential curves allow better control at slow speed while maintaining the robot's ability to run at maximum speed.

Two "logarithmic" curves, a weak and a strong, are supported. Using these curves, and after the joystick has moved past the deadpoint, the motor speed will increase rapidly, and then increase less rapidly as the joystick moves near the extreme position.

The graph below shows the details of these curves and their effect on the output power as the joystick is moved from its center position to either extreme. The graph is for one joystick only. The graph also shows the effect of the deadband setting.

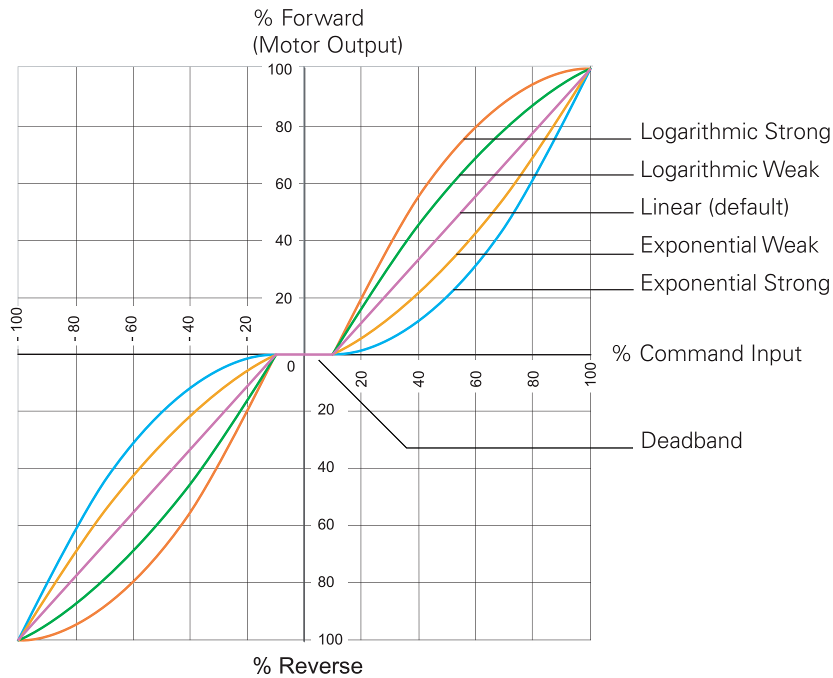


FIGURE 21. Exponentiation curves

The AX2550 is delivered with the "linear" curves selected for both joystick channels. To select different curves, the user will need to change the values of "**E**" (channel 1) and "**F**" (channel 2) according to the table below. Refer to the chapter "Configuring the Controller using the Switches" on page 171 or "Using the Roborun Configuration Utility" on page 177 for instructions on how to program parameters into the controller.

TABLE 7. **Exponent selection table**

Exponentiation Parameter Value	**Selected Curve**
E or **F** = 0	Linear (no exponentiation) - default value
E or **F** = 1	strong exponential
E or **F** = 2	normal exponential
E or **F** = 3	normal logarithmic
E or **F** = 4	strong logarithmic

Left / Right Tuning Adjustment

By design, DC motors will run more efficiently in one direction than the other. In most situations this is not noticeable. In others, however, it can be an inconvenience. When operating in open loop speed control, the AX2550 can be configured to correct the speed in one direction versus the other by as much as 10%. Unlike the Joystick center trimming tab that is found on all R/C transmitters, and which is actually an offset correction, the Left/Right Adjustment is a true multiplication factor as shown in Figure 22

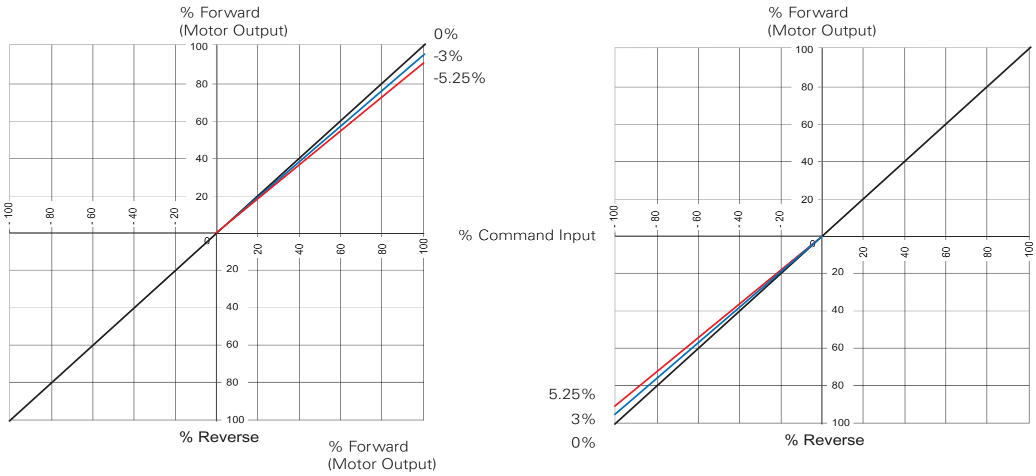


FIGURE 22. Left Right adjustment curves

The curves on the left show how a given forward direction command value will cause the motor to spin 3 or 5.25% slower than the same command value applied in the reverse direction. The curves on the right show how the same command applied to the forward direction will case the motor to spin 3 to 5.25% faster than the same command applied in the reverse direction. Note that since the motors cannot be made to spin faster than 100%, the reverse direction is the one that is actually slowed down.

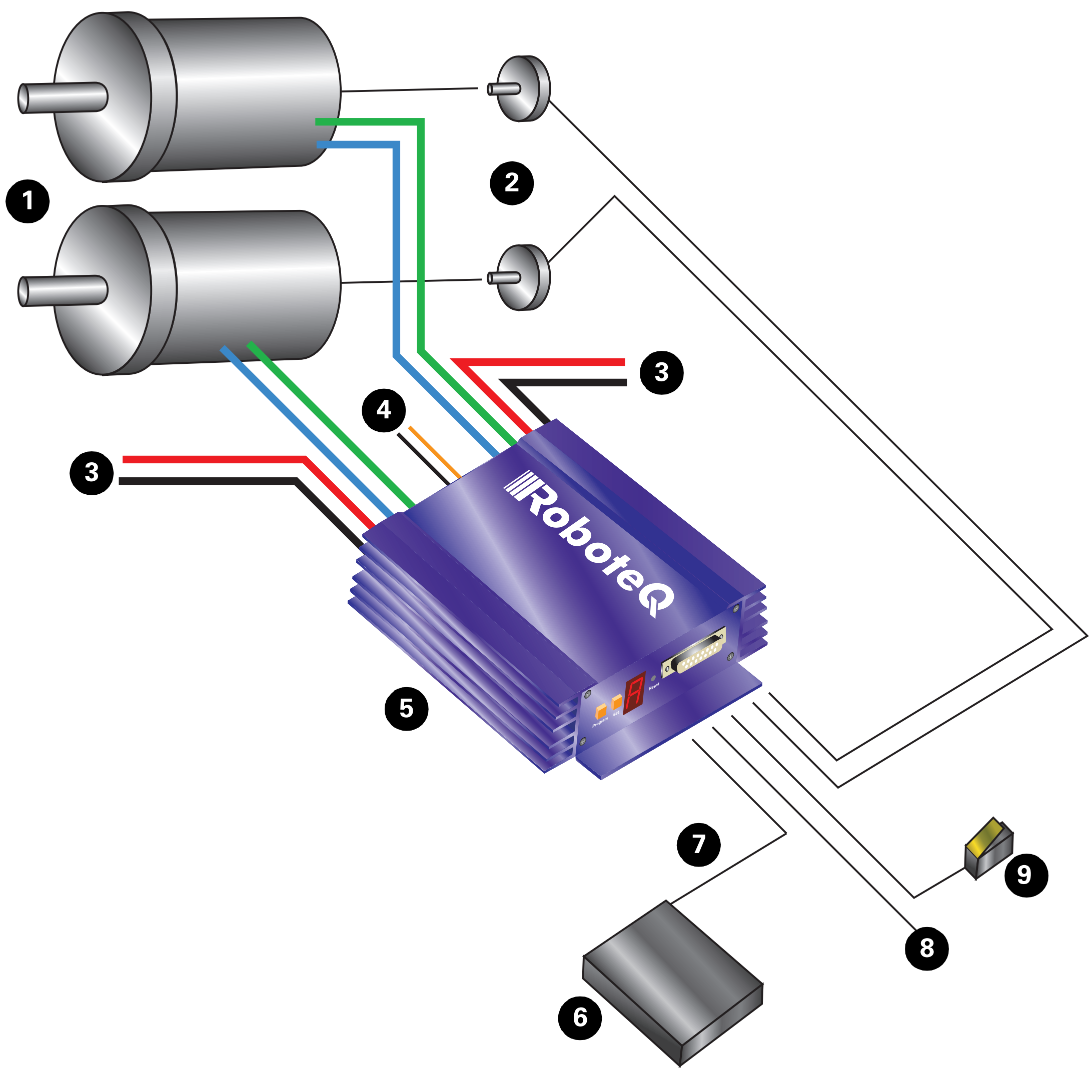
In applications where two motors are used in a mixed mode for steering, the Left/Right Adjustment parameter may be used to make the robot go straight in case of a natural tendency to steer slightly to the left or to the right.

The Left/Right adjustment parameter can be set from -5.25% to +5.25% in seven steps of 0.75%. See “Programmable Parameters List” on page 174 and “Loading, Changing Controller Parameters” on page 181 for details on how to adjust this parameter.

The Left/Right adjustment is performed in addition to the other command curves described in this section. This adjustment is disabled when the controller operates in any of the supported closed loop modes.

TABLE 8. **Left/Right Adjustment Parameter selection**

Parameter Value	Speed Adjustment	Parameter Value	Speed Adjustment
		7	None (default)
0	-5.25%	8	0.75%
1	-4.5%	9	1.5%
2	-3.75%	10	2.25%
3	-3%	11	3%
4	-2.25%	12	3.75%



1- DC Motors

2- Optional sensors:
- Tachometers (Closed loop Speed mode)
- Potentiometers (Servo mode)
- Optical Encoder (AX2850 only - all closed loop modes)

3- Motor Power supply wires

4- Power Control wire

5- Controller

6- R/C Radio Receiver, microcomputer, or wireless modem

7- Command: RS-232, R/C Pulse

8- Miscellaneous I/O

9- Running Inverted, or emergency stop switch

FIGURE 26. Typical controller connections

AX2550's Inputs and Outputs

In addition to the RS232 and R/C channel communication lines, the AX2550 includes several inputs and outputs for various sensors and actuators. Depending on the selected operating mode, some of these I/Os provide feedback and/or safety information to the controller.

When the controller operates in modes that do not use these I/O, these signals become available for user application. Below is a summary of the available signals and the modes in which they are used by the controller or available to the user.

TABLE 9. AX2550 IO signals and definitions

Signal	I/O type	Use	Activated
Out C	2A Digital Output	User defined	Activated using R/C channel 3 (R/C mode), or serial command (RS232 mode) Activated when any one motor is powered (when enabled)
Inp F	Digital Input	User defined	Active in RS232 mode only. Read with serial command (RS232)
		Activate Output C	When Input is configured to drive Output C
		Turn FETs On/Off	When Input is configured as "dead man switch" input
Inp E	Digital Input	Same as Input F - (Not available when encoder module present)	
EStop/Invert	Digital Input	Emergency stop	When Input is configured as Emergency Stop switch input.
		Invert Controls	When Input is configured as Invert Controls switch input.
		User defined	When input is configured as general purpose. Read with serial command (RS232).
Analog In 1	Analog Input	Tachometers input	When Channel 1 is configured in Closed Loop Speed Control with Analog feedback
		Position sensing	When Channel 1 is configured in Closed Loop Position Control with RC or RS232 command and Analog feedback
		User defined	Read value with serial command (RS232).
Analog In 2	Analog Input 2	Same as Analog 1 but for Channel 2	
Analog In 3	Analog Input 3	Position sensing	When Channel 1 is configured in Closed Loop Position Control with Analog command and Analog feedback
		User defined	Read value with serial command (RS232).
Analog In 4	Analog Input 4	Same as Analog 3 but for Channel 4	

I/O List and Pin Assignment

The figure and table below lists all the inputs and outputs that are available on the AX2550.

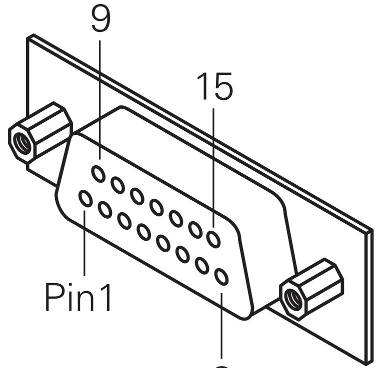


FIGURE 27. Controller's DB15 connector pin numbering

TABLE 10. DB15 connector pin assignment

Pin Number	Input or Output	Signal depending on Mode	Description
1 and 9	Output	Output C	2A Accessory Output C
2	Output	R/C: Data Out	RS232 Data Logging Output
		RS232: Data Out	RS232 Data Out
		Analog: Data Out	RS232 Data Logging Output
3	Input	R/C: Ch 1	R/C radio Channel 1 pulses
		RS232: Data In	RS232 Data In (from PC/MCU)
		Analog: Unused	Unused
4	Input	R/C: Ch 2	R/C radio Channel 2 pulses
		RS232/Analog: Input F	Digital Input F readable RS232 mode Dead man switch activation
5 and 13	Power Out	Ground	Controller ground (-)
6	GND In Unused in RevB Hardware	Ground Unused in RevB Hardware	Optocoupler GND Input, Connect to pin 5** Unused in RevB Hardware

TABLE 10. **DB15 connector pin assignment**

Pin Number	Input or Output	Signal depending on Mode	Description
7	+5V In Unused in RevB Hardware	+5V Unused in RevB Hardware	Optocoupler +5V Input. Connect to pin 14** Unused in RevB Hardware
8	Digital In and Analog In	R/C: Ch 3	R/C radio Channel 3 pulses - (Not available on AX2850)
		RS232: Input E / Ana in 4	Accessory input E Dead man Switch Input Activate Output C Analog Input 4 in RevB Hardware
		Ana: Input E / Ana in 4	Accessory input E Dead man Switch Input Activate Output C Channel 2 speed or position feedback input in RevB Hardware
10	Analog in	RC/RS232: Ana in 2	Channel 2 speed or position feedback input
		Analog: Command 2	Analog command for channel 2
11	Analog in	RC/RS232: Ana in 1	Channel 1 speed or position feedback input
		Analog: Command 1	Analog command for channel 1
12		RC: Unused	
		RS232: Ana in 3	Analog input 3
		Ana: Ana in 3	Channel 1 speed or position feedback input in RevB Hardware
14	Power Out	+5V	+5V Power Output (100mA max.)
15	Input	Input EStop/Inv	Emergency Stop or Invert Switch input

**These connections should only be done in RS232 mode or R/C mode with radio powered from the controller.

Connecting devices to Output C

Output C is a buffered, Open Drain MOSFET output capable of driving over 2A at up to 24V.

The diagrams on Figure 28 show how to connect a light or a relay to this output:

7.3.5 Anexo C.5 – Hoja Técnica – Baterías Sprinter

Para un suministro de energía ininterrumpido

Especificaciones

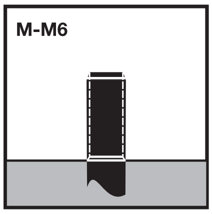
- Extremadamente potentes y compactas, las baterías AGM de la serie Sprinter P son una fuente de energía ideal para sistemas de alimentación ininiterrumpida y particularmente adecuadas para aplicaciones de UPS.
- Excelente rendimiento con altas corrientes además de una larga vida de servicio
- Libres de mantenimiento durante toda su vida de servicio
- Potencia (15 Minutos) de 570 – 4060 W / unidad de 12V
- Hasta 10 años de vida de diseño a 20°C de temperatura ambiente (80% de capacidad remanente)
- Material del recipiente según UL94-HB
- De acuerdo a IEC 896-2
- Placas planas fabricadas a partir de aleación de plomo-calcio
- Muy baja emisión de gas gracias a una recombinación de gases interna (eficiencia del 99%)
- Bajo índice de autodescarga
- Tiempo de recarga muy corto
- A prueba de descargas profundas de acuerdo a DIN 43539 Parte 5
- Completamente reciclables

Tipo	Código	Voltaje Nominal V	Potencia 15' 1.6 V/C 25°C W	Capacidad C_{10} 1.8 V/C 25°C Ah	Largo* (l) mm	Ancho* (b/w) mm	Alto* (h) mm	Peso Aprox. kg	Resistencia Interna m Ω	Intensidad de cortocircuito A	Terminal
P12V570	NAPW120570HP0MA	12	570	21	168	177	126	9.5	10.0	900	M-M6
P12V600	NAPW120600HP0MA	12	600	24	168	127	174	9.5	9.5	950	M-M6
P12V875	NAPW120875HP0MA	12	875	41	198	168	175	14.5	7.0	1350	M-M6
P12V1220	NAPW121220HP0MB	12	1220	51	234	169	190	19.5	6.2	1750	M-M6
P12V1575	NAPW121575HP0MB	12	1575	61	272	166	190	24.0	5.5	2200	M-M6
P12V2130	NAPW122130HP0MA	12	2130	86	359	172	226	33.0	4.0	2600	M-M8
P6V1700	NAPW061700HP0MA	6	1700	122	272	166	190	25.0	1.5	3200	M-M8
P6V2030	NAPW062030HP0MA	6	2030	178	359	171	226	32.5	1.2	4200	M-M8

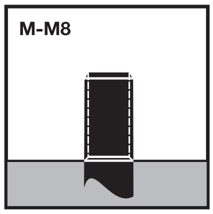
* +/-1mm

Recipiente, terminal y par de apriete

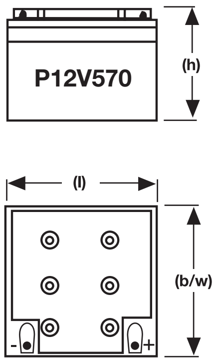
Recipiente: UL 94-HB
= Polipropileno (PP)

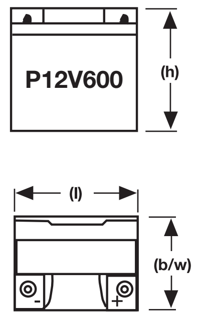


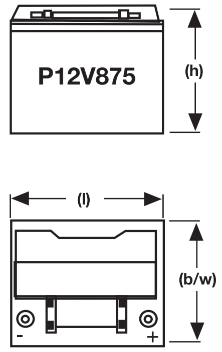
6 Nm

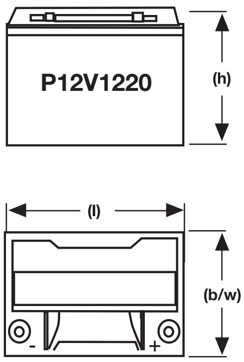


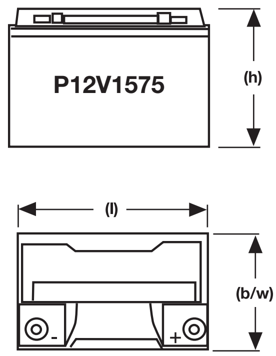
8 Nm

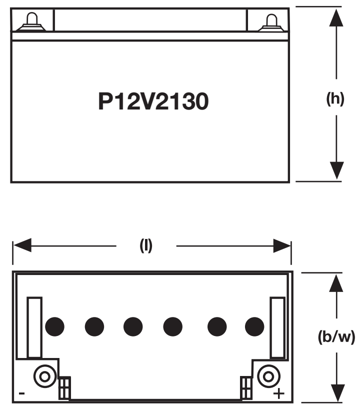

P12V570
(h)
(l)
(b/w)
-
+

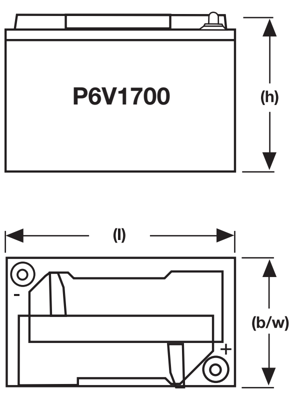

P12V600
(h)
(l)
(b/w)
-
+

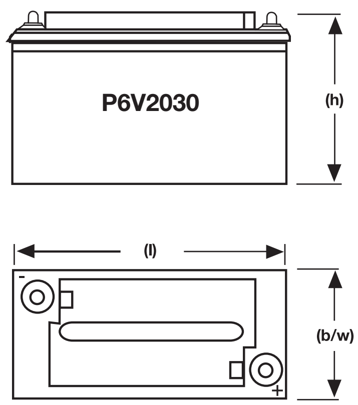

P12V875
(h)
(l)
(b/w)
-
+


P12V1220
(h)
(l)
(b/w)
-
+


P12V1575
(h)
(l)
(b/w)
-
+


P12V2130
(h)
(l)
(b/w)
-
+


P6V1700
(h)
(l)
(b/w)
-
+


P6V2030
(h)
(l)
(b/w)
-
+

¡No a escala!

7.3.6 Anexo C.6 – Hoja Técnica – Sensor de Ultrasonido MAXSONAR

XL-MaxSonar® - WR/WRC™ Series

High Resolution, IP67 Weather Resistant, Ultra Sonic Range Finder

MB7052, MB7060, MB7062, MB7066, MB7067, MB7068, MB7070, MB7072, MB7076, MB7077, MB7078, MB7092

The XL-MaxSonar-WR and XL-MaxSonar-WRC sensor series provide users with robust range information in air. These sensors also feature high-power acoustic output along with real-time auto calibration for changing conditions (supply voltage sag, acoustic noise, or electrical noise), operation with supply voltage from 3V to 5.5V, object detection from 0-cm to 765-cm (select models) or 1068-cm (select models), and sonar range information from 20-cm out to 765-cm (select models) or 1068-cm (select models) with 1-cm resolution. Objects from 0-cm to 20-cm range as 20-cm or closer. The sensor is housed in a robust PVC housing, designed to meet the IP67 water intrusion standard, and matches standard electrical/water ¾" PCV pipe fittings. The user interface formats included are pulse-width (select models), real-time analog-voltage envelope (select models), analog voltage output, and serial output.

Features

- Real-time auto calibration and noise rejection
- High acoustic power output
- Precise narrow beam
- Object detection includes zero range objects
- 3V to 5.5V supply with very low average current draw
- Free run operation can continually measure and output range information
- Triggered operation provides the range reading as desired
- All interfaces are active simultaneously
- RS232 Serial, 0 to Vcc, 9600 Baud, 81N
- Analog, (Vcc/1024) / cm for standard models
- Analog, (Vcc/1024) / 2cm for 10-meter models (MB7066, MB7076)
- Sensor operates at 42KHz

Benefits

- Acoustic and electrical noise resistance
- Reliable and stable range data
- Sensor dead zone virtually non-existent
- Robust, low cost IP67 standard sensor
- Narrow beam characteristics
- Very low power excellent for battery based systems
- Ranging can be triggered externally or internally
- Sensor reports the range reading directly, frees up user processor
- Easy hole mounting or mating with standard electrical fittings
- Filtering allows very reliable operation in most environments

Applications and Uses

- Tank level measurement
- Bin level measurement
- Proximity zone detection
- Environments with acoustic and electrical noise
- Distance measuring
- Long range object detection
- Industrial sensor
- -40°C to +65°C (limited operation to +85°C)

About Ultrasonic Sensors

Our ultrasonic sensors are desired for use in air, non-contact object detection and ranging sensors that detect objects within a defined area. These sensors are not affected by the color or other visual characteristics of the detected object. Ultrasonic sensors use high frequency sound to detect and localize objects in a variety of environments. Ultrasonic sensors measure the time of flight for sound that has been transmitted to and reflected back from nearby objects. Based upon the time of flight, the sensor then outputs a range reading.

XL-MaxSonar-WR/WRC Pin Out

Pin 1- Leave open (or high) for serial output on the Pin 5 output. When Pin 1 is held low the Pin 5 output sends a pulse (instead of serial data), suitable for low noise chaining.

Pin 2- This pin outputs a pulse-width representation of range. To calculate the distance, use a scale factor of 58uS per cm. (MB7052, MB7060, MB7062, MB7066, MB7067, MB7068)

This pin outputs the analog voltage envelope of the acoustic waveform. For the MB7070 series and MB7092 sensors, this is a real-time always-active output (MB7070, MB7072, MB7076, MB7077, MB7078, MB7092)

Pin 3- AN-This pin outputs analog voltage with a scaling factor of (Vcc/1024) per cm. A supply of 5V yields ~4.9mV/cm., and 3.3V yields ~3.2mV/cm. Hardware limits the maximum reported range on this output to ~700 cm at 5V and ~600 cm at 3.3V. The output is buffered and corresponds to the most recent range data.

For the 10-meter sensors (MB7066, MB7076) Pin 3 outputs an analog voltage with a scaling of (Vcc/1024) per 2cm. A supply of 5V yields ~4.9mV/2cm., and 3.3V yields ~3.2mV/2cm. This Analog Voltage output steps in 2cm increments.

Pin 4- RX- This pin is internally pulled high. If Pin-4 is left unconnected or held high, the sensor will continually measure the range. If Pin-4 is held low the sensor will stop ranging. Bring high 20uS or more to command a range reading.

Pin 5- TX- When Pin 1 is open or held high, the Pin 5 output delivers asynchronous serial data in an RS232 format, except the voltages are 0-Vcc. The output is an ASCII capital "R", followed by ASCII character digits representing the range in centimeters up to a maximum of 765 (select models) or 1068 (select models), followed by a carriage return (ASCII 13). The baud rate is 9600, 8 bits, no parity, with one stop bit. Although the voltages of 0V to Vcc are outside the RS232 standard, most RS232 devices have sufficient margin to read the 0V to Vcc serial data. If standard voltage level RS232 is desired, invert, and connect an RS232 converter such as a MAX232.When Pin 1 is held low, the Pin 5 output sends a single pulse, suitable for low noise chaining (no serial data).

V+ Operates on 3V - 5.5V. The average (and peak) current draw for 3.3V operation is 2.1mA (50mA peak) and 5V operation is 3.4mA (100mA peak) respectively. Peak current is used during sonar pulse transmit.

GND-Return for the DC power supply. GND (& V+) must be ripple and noise free for best operation.

Auto Calibration

Each time before the XL-MaxSonar-WR takes a range reading it auto calibrates. The sensor then uses this data to range objects. If the temperature, humidity, or applied voltage changes during sensor operation, the sensor will continue to function normally. (The sensors do not apply compensation for the speed of sound change verses temperature to any range readings.) If the application requires temperature compensation please look at the HRXL-MaxSonar-WR sensor line.

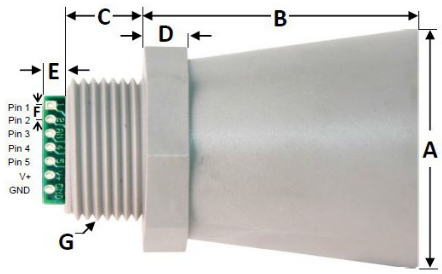
Supply Voltage Compensation

During power up, the XL-MaxSonar-WR sensor line will calibrate itself for the supply voltage. Additionally, the sensor will compensate if the supplied voltage gradually changes.

If the average voltage applied to the sensor changes faster than 0.5V per second, it is best to remove and reapply power to the sensor.

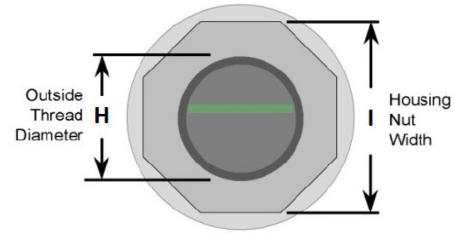
For best operation, the sensor requires noise free power. If the sensor is used with noise on the supplied power or ground, the accuracy of the readings may be affected. Typically, adding a 100uF capacitor at the sensor between the V+ and GND pins will correct most power related electrical noise issues.

XL-MaxSonar-WR Mechanical Dimensions

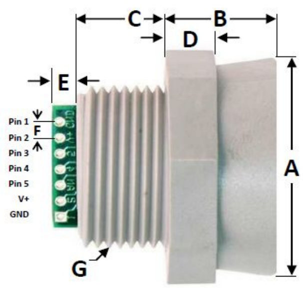


Values Are Nominal

A	1.72" dia.	43.8 mm dia.
B	2.00"	50.7 mm
C	0.58"	14.4 mm
D	0.31"	7.9 mm
E	0.23"	5.8 mm
F	0.1"	2.54 mm
G	3/4"-14 National Pipe Thread Straight	
H	1.032" dia.	26.2 mm dia.
I	1.37"	34.8 mm
Weight		50 grams

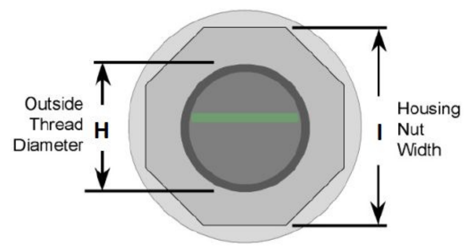


XL-MaxSonar-WRC Mechanical Dimensions



Values Are Nominal

A	34.7 mm dia.	1.37" dia.
B	17.9 mm	0.700"
C	14.4 mm	0.570"
D	7.9 mm	0.310"
E	5.8 mm	0.230"
F	2.54 mm	0.100"
G	3/4"-14 National Pipe Thread Straight	
H	26.2 mm dia.	1.032" dia.
I	34.8 mm	1.370"
Weight		32 grams



Range "0" Location

The XL-MaxSonar-WR and XL-MaxSonar-WRC reports the range to distant targets starting from the front of the transducer as shown in the diagram below.

The range is measured from the front of the transducer.

Device Comparison Chart

Part Number	AN Voltage	Serial Data (0 to Vcc level)	Pulse Width	Analog Envelope	Stability Filter	Most Likely Filter	Compact	7 meter range	10 meter range
MB7052	Yes	RS232	Yes		Yes	Yes		Yes	
MB7060	Yes	RS232	Yes					Yes	
MB7062	Yes	RS232	Yes		Yes			Yes	
MB7066	Yes	RS232	Yes						Yes
MB7067	Yes	RS232	Yes				Yes	Yes	
MB7068	Yes	RS232	Yes		Yes		Yes	Yes	
MB7070	Yes	RS232		Yes				Yes	
MB7072	Yes	RS232		Yes	Yes			Yes	
MB7076	Yes	RS232		Yes					Yes
MB7077	Yes	RS232		Yes			Yes	Yes	
MB7078	Yes	RS232		Yes	Yes		Yes	Yes	
MB7092	Yes	RS232		Yes	Yes	Yes		Yes	

Real-time Auto Calibration

The XL-MaxSonar-WR automatically calibrates prior to each range reading. The sensor then uses this data to range objects. If the temperature, humidity, or applied voltage changes during sensor operation, the sensor will continue to function normally. (The sensors do not apply compensation for the speed of sound change verses temperature to any range readings.) Detection has been characterized in the published sensor beam patterns.

Real-time Noise Rejection

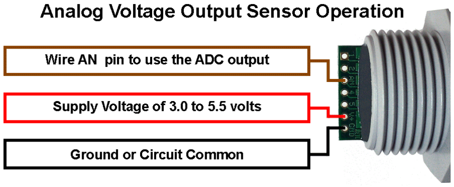
While the XL-MaxSonar-WR is designed to operate in the presence of noise, best operation is obtained when noise strength is low and desired signal strength is high. Hence, the user is encouraged to mount the sensor in such a way that minimizes outside acoustic noise pickup. In addition, keep the DC power to the sensor free of noise. This will let the sensor deal with noise issues outside of the users direct control (Even so, in general, the sensor will still function well even if these things are ignored). Users are encouraged to test the sensor in their application to verify usability.

XL-MaxSonar-WR Sensor Operating Modes

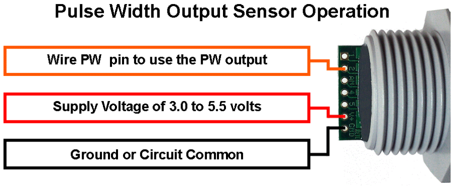
Independent Sensor Operation

The XL-MaxSonar-WR sensors are designed to operate in a single sensor environment. Free-run is the default mode of operation for all of the MaxBotix Inc., sensors. The XL-MaxSonar-WR sensors have three separate outputs that update the range data simultaneously: Analog Voltage, Pulse Width[1], and RS232 Serial. Below are diagrams on how to connect the sensor for each of the three outputs. Note 1 - select models output an Analog Envelope for end user processing (MB707X sensors and MB7092)

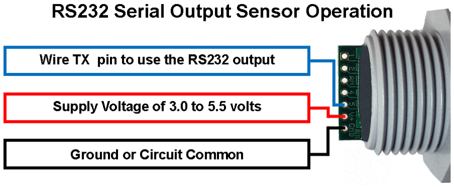
Analog Voltage Output Sensor Operation



Pulse Width Output Sensor Operation



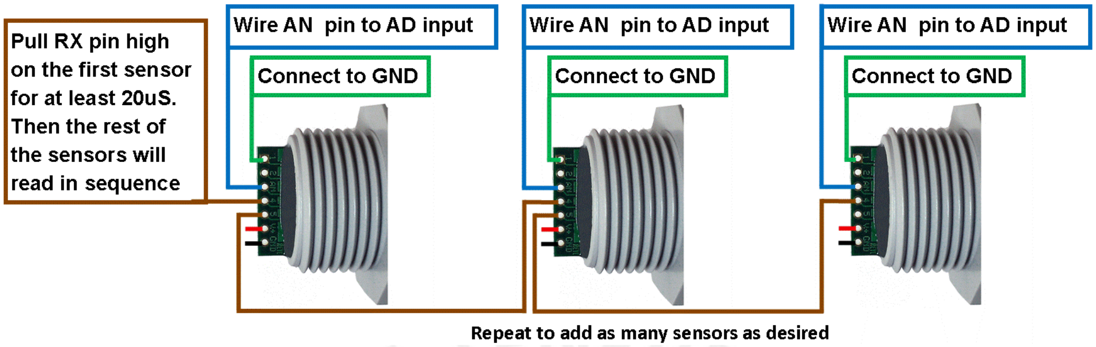
RS232 Serial Output Sensor Operation



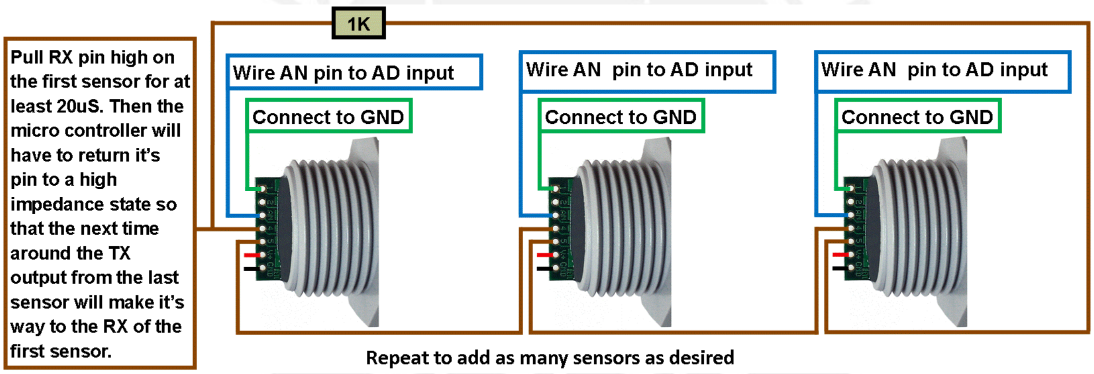
Using Multiple Sensors in a Single System

When using multiple ultrasonic sensors in a single system, there can be interference (cross-talk) from the other sensors. MaxBotix Inc., has engineered a solution to this problem for the XL-MaxSonar-WR sensors. The solution is referred to as chaining. We have 3 methods of chaining that work well to avoid the issue of cross-talk.

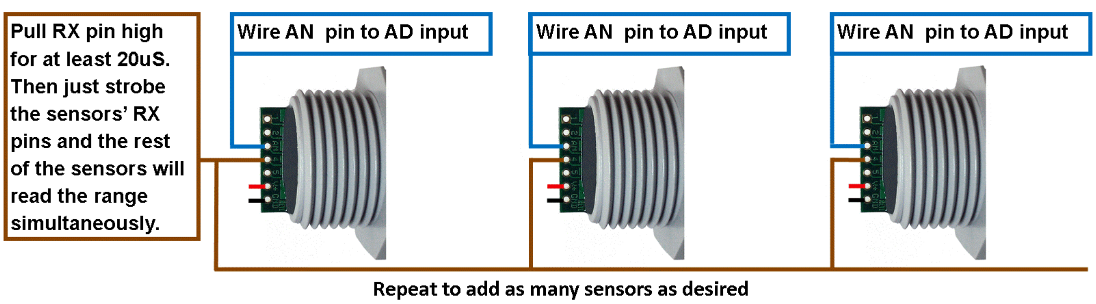
The first method is AN Output Commanded Loop. The first sensor will range, then trigger the next sensor to range and so on for all the sensors in the array. Once the last sensor has ranged, the array stops until the first sensor is triggered to range again. Below is a diagram on how to set this up.



The next method is AN Output Constantly Looping. The first sensor will range, then trigger the next sensor to range and so on for all the sensor in the array. Once the last sensor has ranged, it will trigger the first sensor in the array to range again and will continue this loop indefinitely. Below is a diagram on how to set this up.

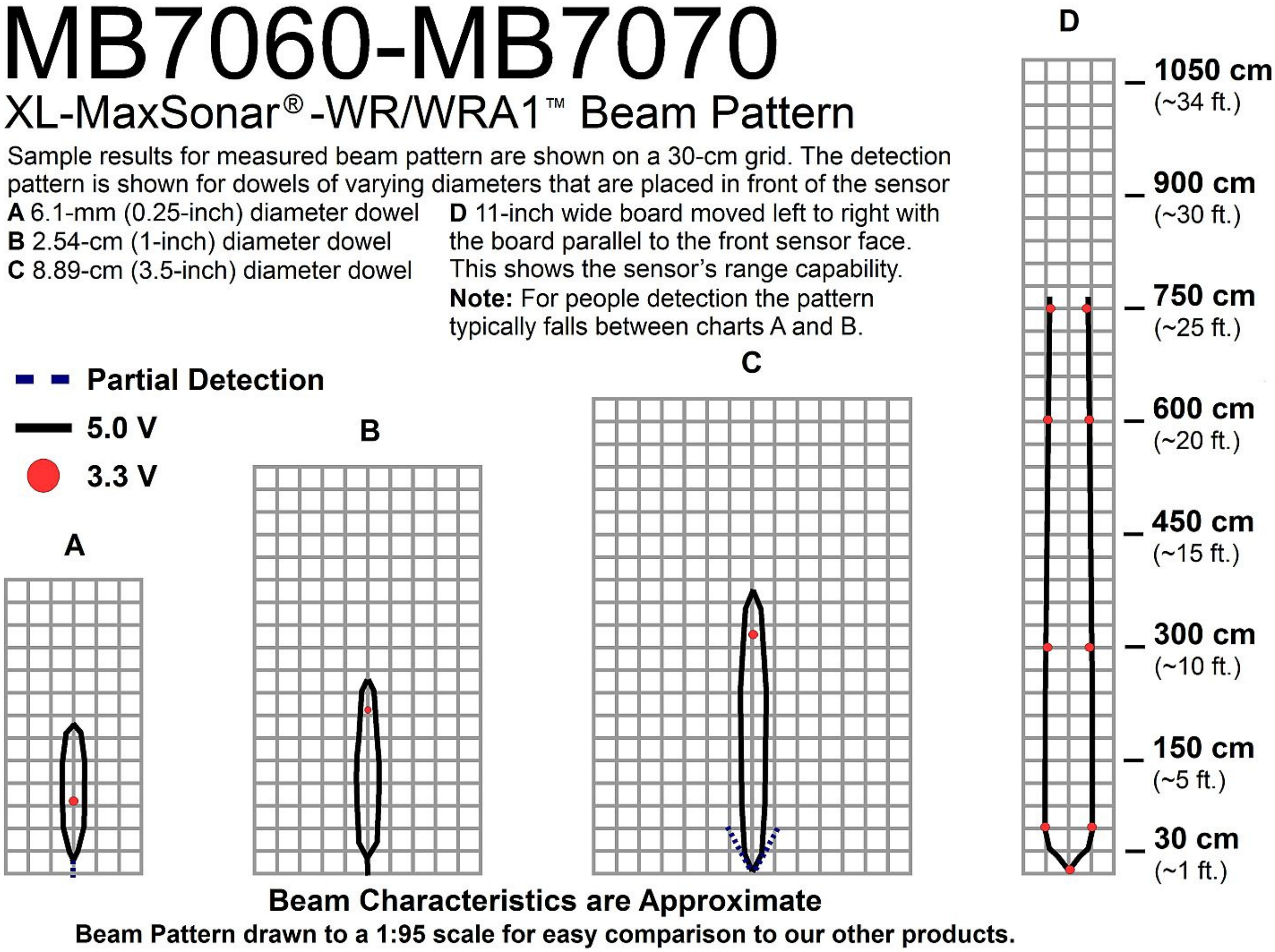


The final method is AN Output Simultaneous Operation. This method does not work in all applications and is sensitive to how the other sensors in the array are physically positioned in comparison to each other. Testing is recommend to verify this method will work for your application. All the sensors RX pins are connected together and triggered at the same time causing all the sensor to take a range reading at the same time. Once the range reading is complete, the sensors stop ranging until triggered next time. Below is a diagram on how to set this up.



MB7060-MB7070 XL-MaxSonar®-WR/WRA1™ Beam Pattern and Uses

The XL-MaxSonar-WR/WRA1 reports the range to the first detectable target. The MB7060 and MB7070 sensors are the most recommended XL-MaxSonar-WR sensor. This is a good starting place when unsure of which XL-MaxSonar-WR to use.



MB7060-MB7070
Features and Benefits

- Real-time calibration, and noise rejection for every ranging cycle
- Readings can occur up to every 100mS (10Hz)
- Analog voltage (Vcc/1024) / cm
- Precise narrow beam
- Continuously variable gain

MB7060-MB7070
Applications and Uses

- Applications where a stability filter is not needed or desired
- Multi-Sensor Arrays
- Distance Measuring
- People Detection

Have the right MaxSonar for your application?

Check out our MaxSonar Product Lines

Indoor Use
(or protected environments)

1 mm Resolution
HRLV-MaxSonar-EZ

1 in Resolution
LV-MaxSonar-EZ
LV-ProxSonar-EZ

1 cm Resolution
XL-MaxSonar-EZ
XL-MaxSonar-AE
XL-MaxSonar-EZL
XL-MaxSonar-AEL
I2CXL-MaxSonar-EZ

1 mm Resolution
HRUSB-MaxSonar-EZ

1in Resolution
USB-ProxSonar-EZ

Outdoor Use
(or rugged environments) IP67

1 mm Resolution
HRXL-MaxSonar-WR
HRXL-MaxSonar-WRS
HRXL-MaxSonar-WRT
HRXL-MaxSonar-WRM
HRXL-MaxSonar-WRMT
HRXL-MaxSonar-WRL
HRXL-MaxSonar-WRLT
HRXL-MaxSonar-WRLS
HRXL-MaxSonar-WRLST

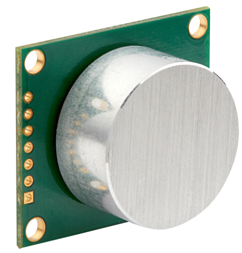
1 cm Resolution
XL-MaxSonar-WR
XL-MaxSonar-WRL
XL-MaxSonar-WRA
XL-MaxSonar-WRLA
I2CXL-MaxSonar-WR

F-Option Available for all WR models. For additional protection when necessary in hazardous

Chemical Shield

1 mm Resolution
HRXL-MaxSonar-WRC
HRXL-MaxSonar-WRCT

1 cm Resolution
XL-MaxSonar-WRC
XL-MaxSonar-WRCA
I2CXL-MaxSonar-WRC



1 cm Resolution
UCXL-MaxSonar-WR
UCXL-MaxSonar-WRC

Accessories-More information available online

MB7954 - Shielded Cable

The MaxSonar Connection Wire is used to reduce interference caused by electrical noise on the lines. This cable is a great solution to use when running the sensors at a long distance or in an area with a lot of EMI and electrical noise.

MB7950 - XL-MaxSonar-WR Mounting Hardware

The MB7950 Mounting Hardware is selected for use with our outdoor ultrasonic sensors. The mounting hardware includes a steel lock nut and two O-ring (Buna-N and Neoprene) each optimal for different applications.

MB7955 / MB7956 / MB7957 / MB7958 / MB7972 - HR-MaxTemp

The HR-MaxTemp is an optional accessory for the HR-MaxSonar. The HR-MaxTemp connects to the HR-MaxSonar for automatic temperature compensation without self heating.

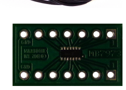
MB7961 - Power Supply Filter

The power supply filter is recommended for applications with unclean power or electrical noise.

MB7962 / MB7963 / MB7964 / MB7965 - Micro-B USB Connection Cable

The MB7962, MB7963, MB7964, and MB7965 Micro-B USB cables are USB2.0 compliant and backwards compatible with USB 1.0 standards. Varying lengths.

MB7973 CE Compliance Widget



The MB7973 adds protection for the CE requirement for Lightning/Surge IEC61000-4-5