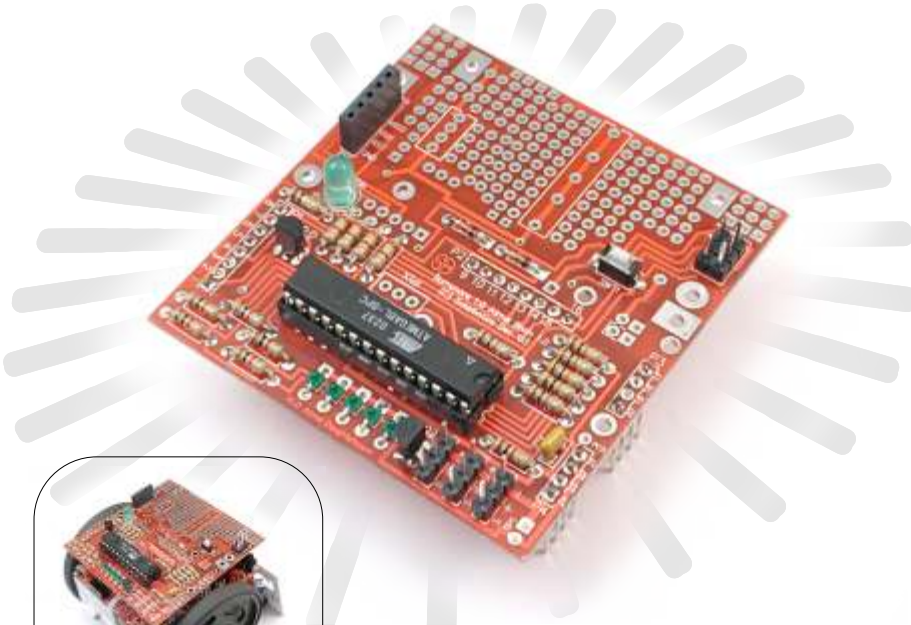
The Solarbotics SUMOVORE

Atmel ATmega8 Version 1 Brainboard Add-on



This brainboard uses the popular and powerful Atmel ATmega8 microcontroller (included!) to take over the functions of the Discrete Brain that comes with your Sumovore.

There are many programming languages (some free!) you can use to program your Brainboard via your computer's parallel port or Atmel STK-500 development system.

It's fast, inexpensive, and very powerful. An ideal mate to the Sumovore!

(Sumovore Sumo robot kit and DB25 printer cable/connector req'd)

Produced by

SOLARBOTICS® Ltd

Document Release: October 26, 2004

We strongly suggest you inventory the parts in your kit to make sure you have all the parts listed. Use a pen, pencil, pricked finger, chocolate bar - anything to mark off the items. If anything is missing, contact us for replacement parts information.

Disclaimer of Liability

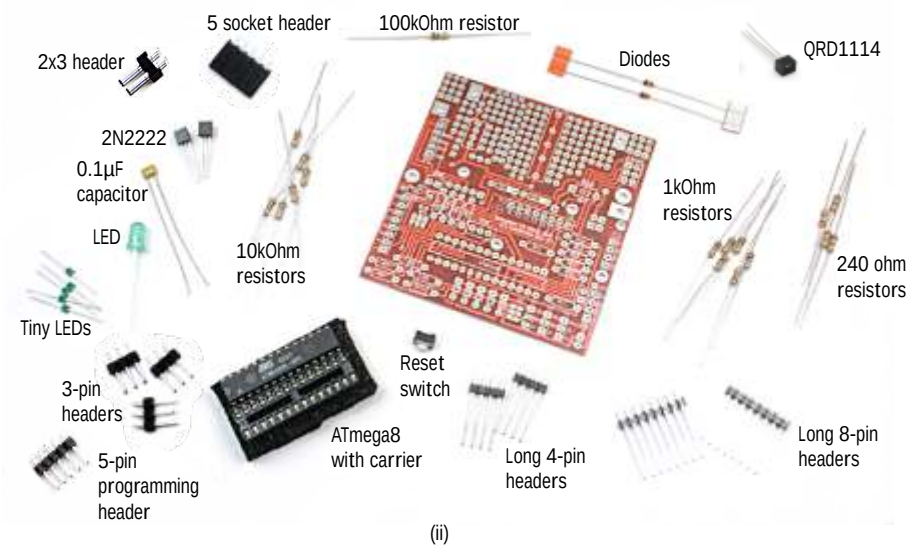
Solarbotics Ltd. is not responsible for any special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, good-will, damage to or replacement of equipment or property, and any costs or recovering of any material or goods associated with the assembly or use of this product. Solarbotics Ltd. reserves the right to make substitutions and changes to this product without prior notice. (Sorry, gotta make the lawyer happy)

ATmega8 Brainboard Components

1 - Printed Circuit Board (PCB)
1 - 0.1µF capacitor (labled '104')
2 - Diodes
5 - Tiny LEDs
1 - Programming indication LED
2 - 2N2222 NPN Transistors
4 - 240 Ohm Resistors (Red/Yellow/Brown)
6 - 1k Resistors (Brown/Black/Red)
6 - 10k Resistors (Brown/Black/Orange)
1 - 100k Resistor (Brown/Black/Yellow)
1 - ATmega8 28 pin Carrier
1 - Atmel ATmega8 (or 8L) Microcontroller
1 - SPST Push Button Switch
1 - 5-Socket programming header
1 - 6-Pin (2 rows of 3 pins) STK-500 programming header
1 - 5-Pin Header (for building optional programming cable)
2 - 4-Pin Sumovore interface long headers
2 - 8-Pin Sumovore interface long headers
3 - 3-Pin Headers (for optional servo headers)
1 - QRD1114 edge sensor (for Sumovore's middle sensor)

Tools Required

Soldering equipment
Side-cutters or fine snips
Computer for profurther programming



The ATmega8 Brainboard

Introduction

Looking for a more flexibility out of your Sumovore? Well, welcome to the next in our series of brainboards - the ATmega8. The ATmega 8 offers an impressive list of features, including (but not limited to):

- 8kB flash memory
- Three Pulse-width modulation (PWM) channels
- Six Analog-to-digital converters (ADCs)
- In-circuit programmable (which we make use of)
- Internal RC oscillator (a resonator is needed to surpass 8MHz)
- 23 programmable Input/Output (I/O) lines
- Single clock execution up to 1MIPs/MHz (this means "it's QUICK", like 250 times faster than a standard Basic Stamp 2!)

We've made good use of the capabilities of the ATmega8, but there's lots more you can do with it, including program it with the free open-source GCC C-compiler (our weapon of choice) or other compilers for BASIC, JAL, Java, Assembler, Pascal, Forth,... it goes on and on!

This is not a kit for a microcontroller beginner. Anybody using this brainboard should have the appropriate skills, or be ready to learn the techniques that make a microcontroller... microcontrol!

This kit lets you swap out the default discrete brainboard for a programmable version. If you run into any problems, it's a simple process to swap a different brain back in. Didn't you ever have days where that'd be a handy feature for you to have (umm...for the robot, we mean).

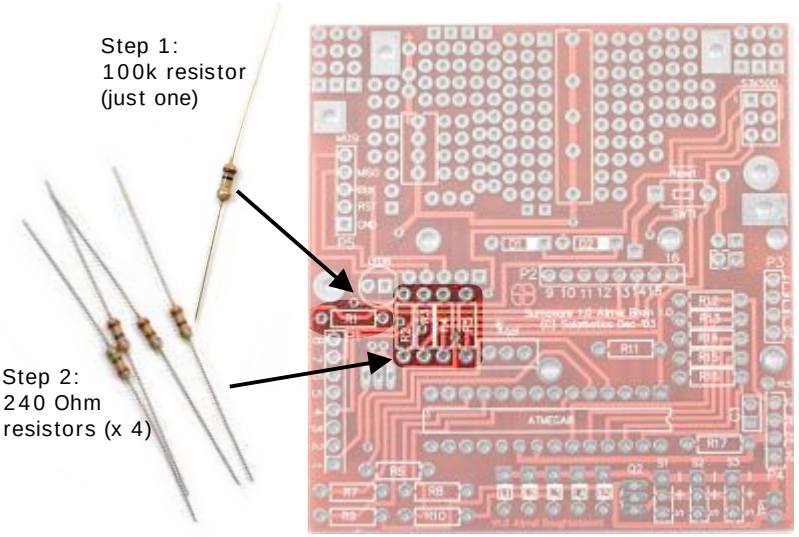
This kit features:

- Atmel ATmega8 (or 8L) microcontroller
- 5 indicator LEDs
- 1 "Programming-in-progress" indicator LED
- STK500 and 5-pin programming headers
- Three servo (or similar peripheral) headers
- Extra breadboarding space and hard-point mounts
- Microprocessor Reset Switch

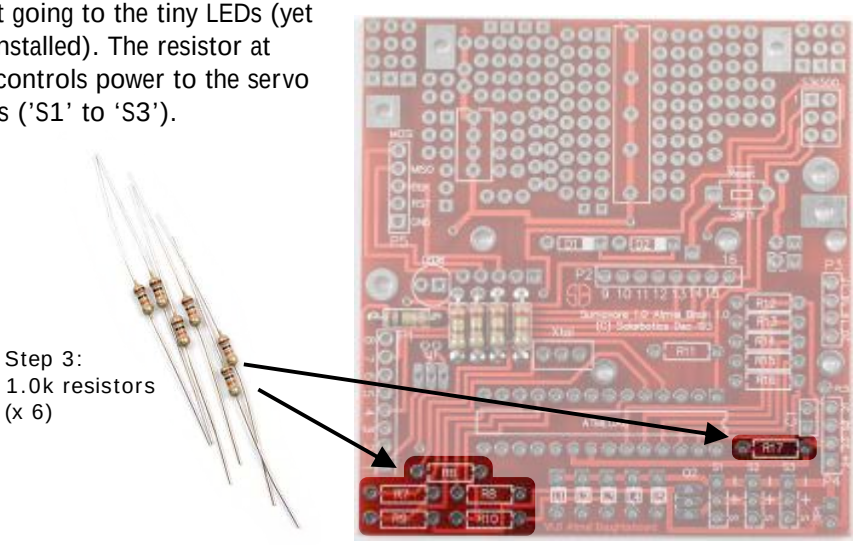
We designed the breadboarding space to accommodate extra ICs and support electronics, or simply as a place to mount a servo with double-sided sticky tape! It's a flexible area - use it for whatever strikes your fancy.

Step 1 - 100kOhm Resistor (Brown / Black / Yellow): Bend the leads of the 100k resistor over and insert it position 'R1'. This resistor is part of the programming LED indicator circuit.

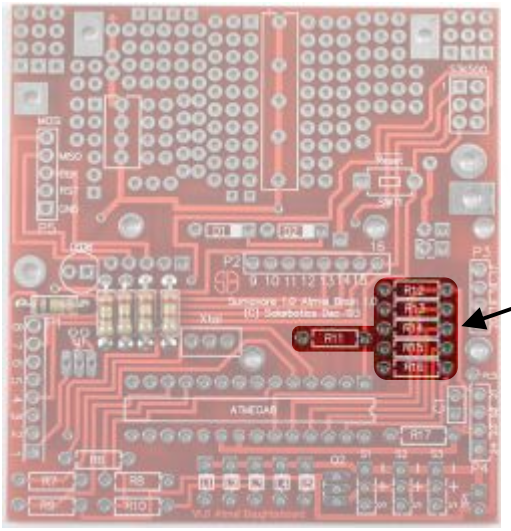
Step 2 - 240 Ohm Resistors (Red / Yellow / Brown): These resistors get installed as a group in positions 'R2', 'R3', 'R4', and 'R5'. These are parallel port programming protection resistors.

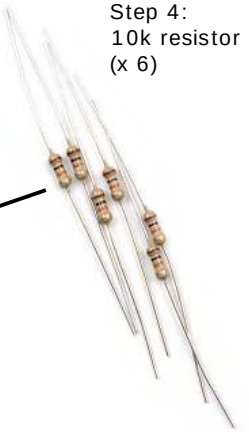


Step 3 - 1kOhm Resistors (Brown / Black / Red): These 6 resistors are installed in positions 'R6', 'R7', 'R8', 'R9', 'R10' and 'R17'. The first set of 5 limit the current going to the tiny LEDs (yet to be installed). The resistor at 'R17' controls power to the servo headers ('S1' to 'S3').



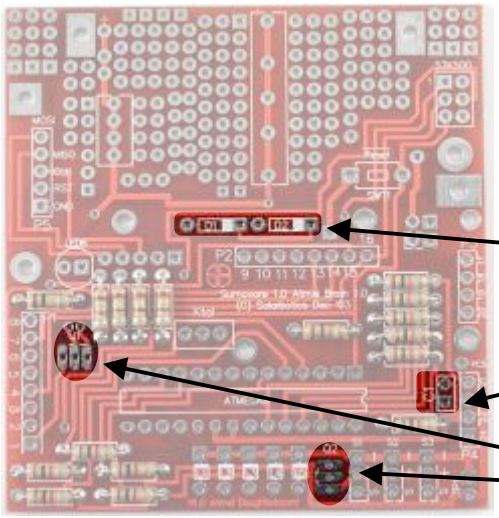
Step 4 - 10k Resistors (Brown / Black / Orange): Bend and install the 10k resistors into positions ‘R11’, ‘R12’, ‘R13’, ‘R14’, ‘R15’, and ‘R16’. R11 is the reset pull-up, while the rest are the front edge detector sensor pull-ups.





Step 4:
10k resistor
(x 6)

Step 5, 6, & 7 - Diodes, Capacitor, and Transistors: Bend and install the two diodes into positions ‘D1’ and ‘D2’, but make sure it goes in the right way! The band on the diode must match the position of the band on the PCB. The 0.47µF capacitor (labeled ‘474’) is installed at position ‘C1’, and the two transistors are installed at ‘Q1’ and ‘Q2’. Make sure the curve of the transistor matches the one printed on the PCB!

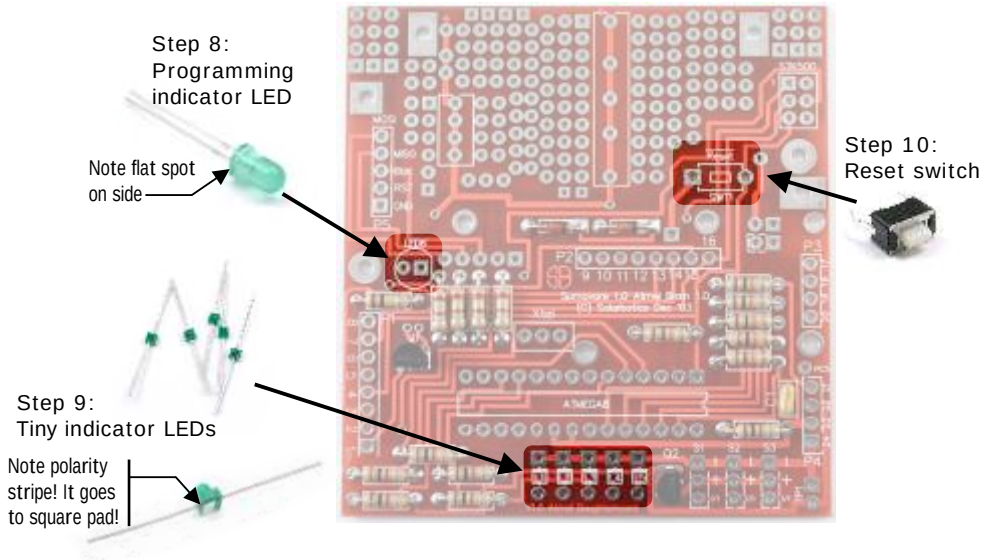


Step 5: Diodes (x 6)
Make sure the diode goes in with the bar on the right!

Step 6: 0.47µF capacitor

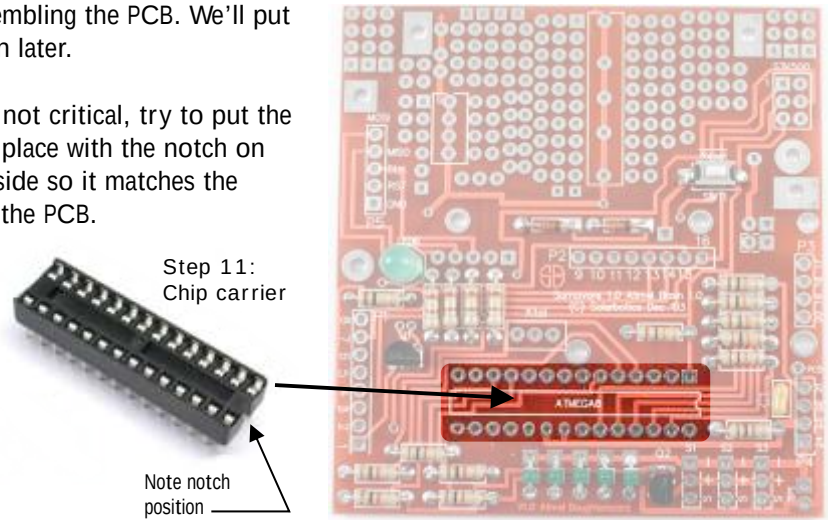
Step 7: 2N2222 transistors (x 2)

Step 8, 9, & 10- Regular and Tiny LEDs; Reset Switch: Everybody likes LEDs. They let you know when the microcontroller is being programmed and when sensors are sensing! Just make sure you put them in the correct way at locations 'L1' through 'L5', and 'LED6' (backwards doesn't work). The reset switch is mounted at position 'SWT1'.

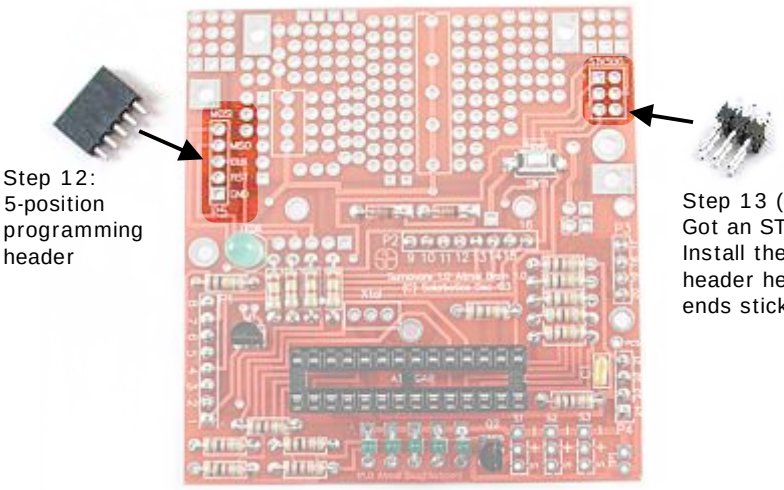


Step 11 - 28-pin Atmel Carrier: We use a carrier to protect the microcontroller from any strange funkiness going on while assembling the PCB. We'll put the chip in later.

Although not critical, try to put the carrier in place with the notch on the right side so it matches the shape on the PCB.



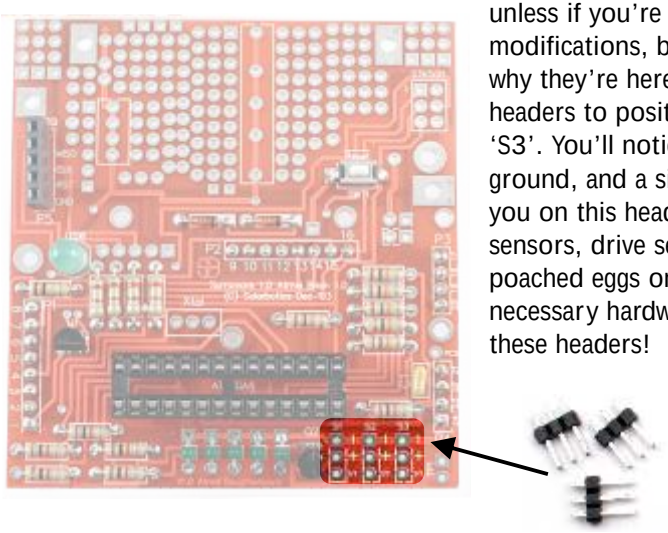
Step 12, 13 - Programming Headers: You'll most likely be using the 5-pin header for most of your programming with a modified parallel port cable. Simply solder the header to position 'P5'. If you have an Atmel STK-500 development board, you can install the optional 2x3 pin header to the position 'STK500' on the right side of the board as well.



Step 12:
5-position programming header

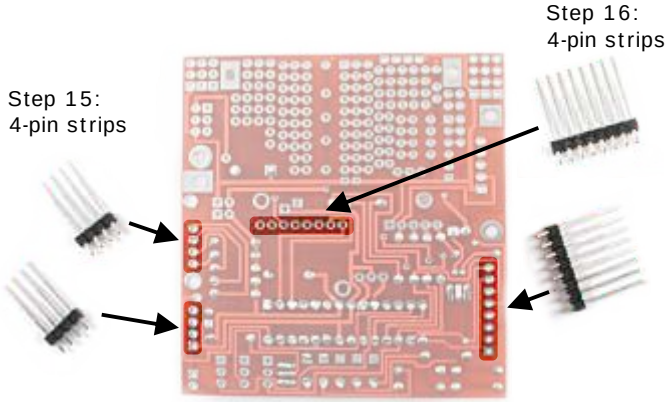
Step 13 (optional):
Got an STK500? Install the 2x3 pin header here (long pin ends sticking up)

Step 14 - Servo / Auxiliary Headers: You most likely not need these headers unless if you're planning some funky modifications, but then again, that's why they're here! Solder the 3-pin headers to positions 'S1', 'S2', and 'S3'. You'll notice that there's power, ground, and a signal line all available to you on this header, so you can read sensors, drive servos, or... or... make poached eggs on toast if you have the necessary hardware! Have fun with these headers!

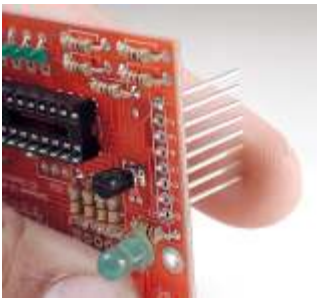


Step 14:
3-pin servo or auxiliary interface pins

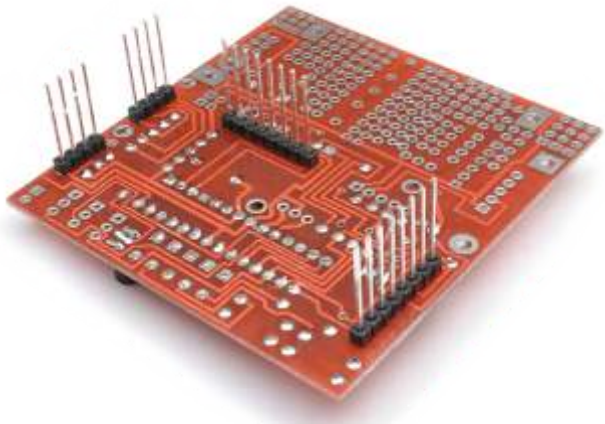
Step 15 & 16 - 4 and 8-pin strips: Gotta have a way to plug your ATmega8 brainboard into the Sumovore, right? Install these pins on the underside of the PCB, soldering only one pin per strip initially. This lets you eyeball and adjust them so they're straight up-and-down, which is important so they can mate with the sockets on the Sumovore.



Pin strip installed crooked - good thing you soldered only one pin!

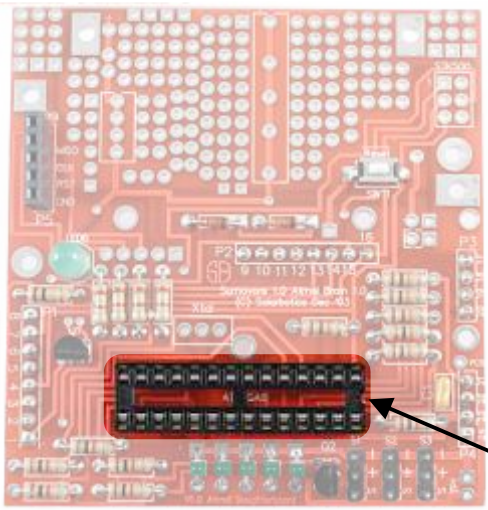


Remelt solder on top pin, and re-adjust pins so they sit properly, then solder the rest of the pins

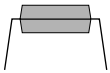


Competed pin stip installation on underside of PCB

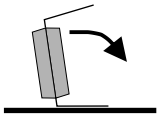
Step 17 Installing the ATmega8: Now you're ready to take the Atmel Mega8L out of the static-protection foam and insert it into the chip carrier. Make sure that the notch on the chip is to the right, matching the notch in the carrier and (more importantly) the notch printed on the circuit board.



Step 17: Install the microcontroller
Note: You may find it easier to insert the chip into the holder after gently bending all the pins inwards.

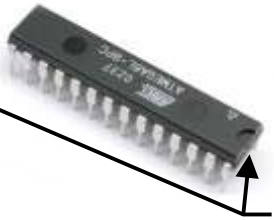


Starts like this...



(do for each side)

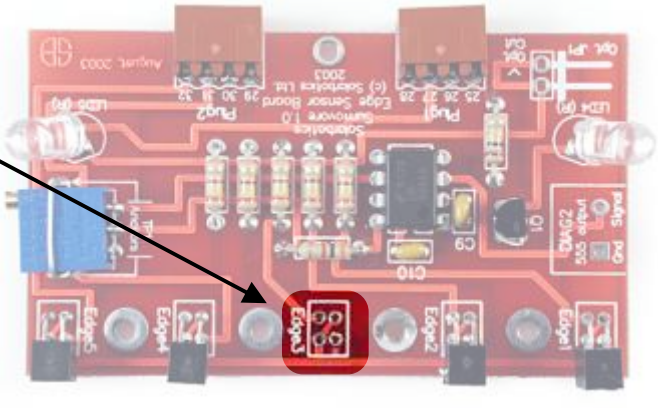
...ends like this!



Important! Notchs must be on right side!

Finished ATmega8 Brainboard, with all the options in place. Ready to start programming?

Step 18 - Installing the 5th line sensor: Yank the edge-sensor board out of your Sumovore, and install the included line sensor in position 'Edge3', just like you did when you originally built your Sumovore. You don't have to do this, but if you want to make the best use out of your Brainboard, it'd be a good idea!



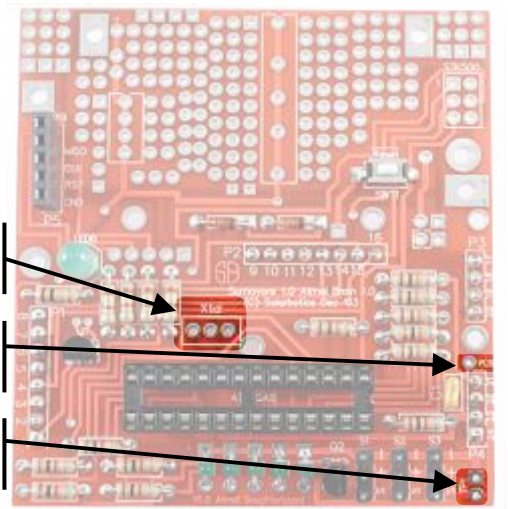
Step 18: Add the 5th line sensor

You will notice that on the Brainboard there's a spot labeled 'Xtal', which is only needed if you want to use an external resonator for greater timing accuracy.

Additionally, the pin for port C5 wasn't directly used on any Sumovore pins, so it is available as pad 'PC5' right next to the 'P4' quad-pin set on the right side of the Brainboard

You may also notice at the bottom right corner a pair of pads labeled 'TP1'. Test Point 1 is for monitoring the unregulated voltage from the bottom four AA batteries. This is so you can access a full 6V straight from the batteries if required for your modifications.

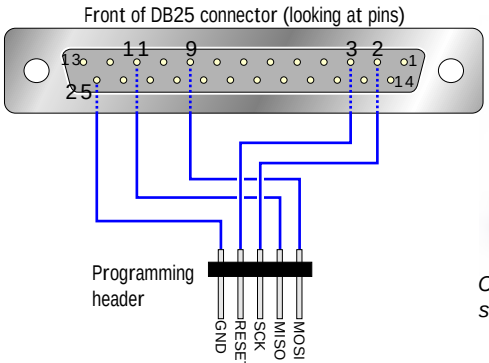
When using the bread-boarding space, remember that all square pads are connected to ground!



Unless if you're using an Atmel STK-500 development board, you'll be programming your ATmega8 Brainboard via your computer's parallel port. With the abundance of USB printers on the market now, it shouldn't take you much effort to find an old printer cable to hack into a programming cable. We've even included a 5-pin header for the brainboard side of the cable!

We're using what's called an SP12 serial programmer, released under the GNU license by Ken Huntington, Kevin Towers, and Pitronics. We've combined some of the parts (the resistors, transistor, LED) into the brainboard to make building the cable a simple project. If you want to learn more about the SP12 project, you can find it on the Internet at http://www.xs4all.nl/~sbolt/e-spider_prog.html.

SP12 Physical Wiring Diagram



Completed cable, with heatshrink sleeve and scrap plastic 5-pin header support

If you have Internet access, you should visit http://www.avrfreaks.net for as much Atmel AVR microprocessor information as you'll ever need - they even have a "newbie" section for absolute beginners! AVRFreaks hosts or links to practically all resources relating to the Atmel series of microcontrollers, so if the Solarbotics downloads page doesn't have what you need, try here.

Remember, there's a good many ways to program your ATmega8. We're going to show you our method of using WinAVR with GCC (GNU C Compiler) as a baseline (get the lastest software from http://winavr.sourceforge.net/).

Don't let this stop you from trying Assembler, or even some of the demonstration versions of other languages - have fun experimenting!

The ATmega8 Brainboard

Default Program Listing ("sumoline.c")

```
/*****************************************************************************
 Author:   Grant McKee, Solarbotics Ltd. (C) 2004
 Date:     Apr 2004
 Software: AVR-GCC 3.3.1
 Hardware: ATMEGA8L at 1 Mhz int OSC

Description:
Minisumo/Linefollower program Ver 1.0

If outside edge sensors see black during startup the program will branch to sumo.
If outside edge sensors see white during startup the program will branch to Linefollower.

Sumo mode is fairly basic:

- Wait 5 seconds before moving
- After 5 seconds go straight forward
- If a white line is detected on outside Left sensor
    * Reverse both motors briefly
    * Stop left motor
    * Reverse Right motor
    * After set time continue straight forward
- If a white line is detected on outside Right sensor
    * Reverse both motors briefly
    * Stop Right motor
    * Reverse Left motor
    * After set time continue straight forward
- If opponent detected on right Side
    * Stop Right motor
- If opponent detected on Left Side
    * Stop Left motor
- If both sensors detect opponent
    * Turn on both motors forward

Linefollower mode:

- Start immediately
- If center sensor sees black all is well- go straight forward
- If center right sensor sees black, make a gentle right turn by slowing down right motor
- If center left sensor sees black, make a gentle left turn by slowing down left motor
- If far right sensor sees black, make a very sharp right turn by reversing right motor
- If far left sensor sees black, make a very sharp left turn by reversing left motor

- If all is white, all is lost - go looking for the line!

*****************************************************************************/

#include <avr/io.h>

int value,left,mleft,middle,mright,right,channel,IRin; //Defined variables
long x,z; //Variables for delay timers

int ADCIN(int channel); //Function prototype for ADC function

#define thresh 128 //White line sensitivity higher is more sensitive (255 max)
#define start_time 150000 //Startup delay time constant 150000 ~ 1 sec (at clk = 1Mhz)
#define reverse_time 60000 //Time to reverse

int main(void) //Start of main
{
outp(0xFC, DDRD); //set 5 LED's on port D as outputs
outp(0xFF, DDRB); //set outputs to motor enables/direction

/***************************** Line sensor switch **********************/
   left = ADCIN(0);  //Read Left line sensor
   right = ADCIN(4);  //Read Right line sensor

/****************** Startup routine (am I Sumo or Linefollower) ********/
if ((right < thresh) & (left < thresh))  //If both left and right sensors see white
   {
        LINEFOLLOWER(); //Go to the linefollower loop
   }

SUMO();
}
/************************** 5 second startup ***************************/
SUMO()
{
sbi(PORTD, 2);    //Turn on L1
for(x=0; x<start_time; x++); //Pause for a second
sbi(PORTD, 3); //Turn on L2
for(x=0; x<start_time; x++); //Pause for a second
sbi(PORTD, 4); //Turn on L3
for(x=0; x<start_time; x++); //Pause for a second
sbi(PORTD, 5); //Turn on L4
for(x=0; x<start_time; x++); //Pause for a second
sbi(PORTD, 6); //Turn on L5
for(x=0; x<start_time; x++); //Pause for a second

cbi(PORTD, 2); //Turn off L1
cbi(PORTD, 3); //Turn off L2
cbi(PORTD, 4); //Turn off L3
cbi(PORTD, 5); //Turn off L4
cbi(PORTD, 6); //Turn off L5
/************************ End 5 second startup **********************/

while(1) //Do this sumo loop forever
   {
   left = ADCIN(0);  //Read line sensor (Left)
   //mleft = ADCIN(1); // (Center Left) - Not used for Sumo
   //middle = ADCIN(2); // (Center) - Not used for Sumo
   //mright = ADCIN(3); // (Center Right) - Not used for Sumo
   right = ADCIN(4); //Read line sensor (Right)

   sbi(PORTB, 1);//Enable left motor
   sbi(PORTB, 2);//Enable Right motor
   sbi(PORTB, 4);//Forward left motor
   sbi(PORTB, 5);//Forward Right motor

if(right < thresh) //Seeing white line on right sensor
   {
        for(x=0; x<6000; x++) //slam both motors into reverse to prevent drifting over the
line
        {
        cbi(PORTB, 4); //Reverse left motor
        cbi(PORTB, 5); //Reverse right motor
        }

        for(x=0; x<reverse_time; x++) //Turn around with duration given by "reverse_time"
        {
        cbi(PORTB, 4); //Reverse left motor
        cbi(PORTB, 2); //Stop right motor
        }
   }

if(left < thresh) //Seeing white line on left sensor
   {
        for(x=0; x<6000; x++) //slam both motors into reverse to prevent drifting over the
line
        {
        cbi(PORTB, 4); //Reverse left mo tor
     cbi(PORTB, 5); //Reverse right motor
        }

        for(x=0; x<reverse_time; x++)
        {
        cbi(PORTB, 1); //Stop left motor
        cbi(PORTB, 5); //Reverse right motor
        }
   }

IRin = inp(PIND); //Read inputs from port D
IRin = (IRin & 0x3); //Mask bits (1100000) PD0, PD1

if(IRin == 2) //See opponent with Right sensor
{
sbi(PORTD, 2); //Turn on L1
cbi(PORTD, 6); //Turn off L5
cbi(PORTB, 1);    //Disable Right motor
}

if(IRin == 1) //See opponent with Left sensor
{
sbi(PORTD, 6); //Turn on L5
cbi(PORTD, 2); //Turn off L1
cbi(PORTB, 2);    //Disable Left motor
}

if(IRin == 0) //Both sensors see opponent
{
sbi(PORTD, 2); //Turn on L1
sbi(PORTD, 6); //Turn on L5
sbi(PORTB, 2);    //Enable Left motor
sbi(PORTB, 1);    //Enable Right motor
}

if(IRin == 3) //Neither sensor sees opponent
{
cbi(PORTD, 2); //Turn on L1
cbi(PORTD, 6); //Turn on L5
}
}
}

/****************************** Linefollower start *********************************/
LINEFOLLOWER()
{
while(1) //Do loop forever
{
```

```
TCCR1A = BV(WGM10) | BV(COM1A1) | BV(COM1B1); //Setup PWM for left and right motors
TCCR1B = BV(CS10) | BV(WGM12); //ditto

 left = ADCIN(channel = 0);  //Read line sensors (Left)
 mleft = ADCIN(channel = 1); //(Middle Left)
 middle = ADCIN(channel = 2); //(Middle)
 mright = ADCIN(channel = 3); //(Middle Right)
 right = ADCIN(channel = 4); //(Right)


if(right < thresh) //Seeing white on right sensor
    {
    sbi(PORTD, 6);    //Turn on LED5
    }
if(right > thresh) //Seeing black on right sensor
    for(z=0; z<200; z++)
 {
    cbi(PORTD, 6);        //Turn off LED5
    cbi(PORTB, 5);    //Left motor backward
 }
if(mright < thresh) //Seeing white on middle right sensor
    {
    sbi(PORTD, 5);    //Turn on LED4
    }
if(mright > thresh) //Seeing black on middle right sensor
    for(z=0; z<200; z++)
 {
    cbi(PORTD, 5);        //Turn off LED4
    OCR1B = 127;        //Slow down Right motor
 }

if(middle < thresh) //Seeing white on middle sensor
    {
    sbi(PORTD, 4);    //Turn on LED3
    }
if(middle > thresh) //Seeing black on middle sensor
    {
    cbi(PORTD, 4);        //Turn off LED3
    sbi(PORTB, 4);    //Forward Right motor
    sbi(PORTB, 5);    //Forward Left motor
    OCR1B = 255;        //Set speed of Right motor
    OCR1A = 255;        //Set Speed of Left motor
    }

if(mleft < thresh) //Seeing white
    {
    sbi(PORTD, 3);    //Turn on LED2
    }
if(mleft > thresh) //Seeing black
    for(z=0; z<200; z++)
 {
    cbi(PORTD, 3);        //Turn off LED2
    OCR1A = 127;        //Set Speed of Left motor
 }
if(left < thresh) //Seeing white
    {
    sbi(PORTD, 2);    //Turn on LED1
    }
if(left > thresh) //Seeing black
    for(z=0; z<200; z++)
 {
    cbi(PORTD, 2);        //Turn off LED1
    cbi(PORTB, 4);    //Reverse right motor
 }

}
}

/********************************** ADC function *********************************/
 int ADCIN(int channel)
 {
 //Left adjust result for 8 bit res  //Set reference to AVCC //Set channel
 ADMUX = BV(ADLAR) | BV(REFS0) | channel;

 //Enable ADC //Start Conversion
 ADCSRA = BV(ADEN) | BV(ADSC);

 for(x=0; x<10; x++); // Pause for channel change

 //Wait until conversion is complete
 loop_until_bit_is_clear(ADCSRA, BV(ADSC));

 //Write value to ADCH
 value = ADCH;

 return value;
    }
/************************************* END ************************************/
```

This is the default code that your ATmega8 ships with. If you mess something up, you can either re-download it from our website, or type it in from what you see here (ug!).

There are three major sections to the code, being the startup routine, the Sumo routine, and the Line-follower routine. If you start your Sumovore on a black surface (like a sumo ring), the startup routine reads the low inputs from the edge sensors and determines that it should start the sumo routine. When on a white surface (like line-follower usually is, with a black electrical tape line), then the startup routine kicks the Sumovore into running the line-follower routine.

Both of these programs are pretty decent, but there is much more room for creative optimizing. Feel free to modify and hack this code - we' presenting it to you as a good starting point.

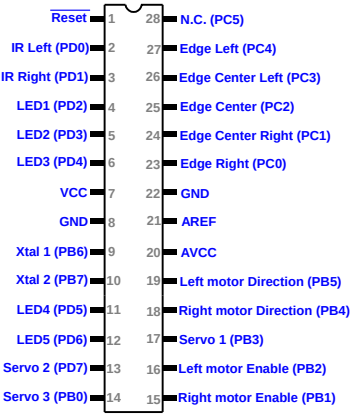
The ATmega8 Brainboard

Brainboard Schematics

For those of you wanting to do more customizing to your ATmega8 Brainboard, here are the microcontroller pin assignments and the PCB schematic.

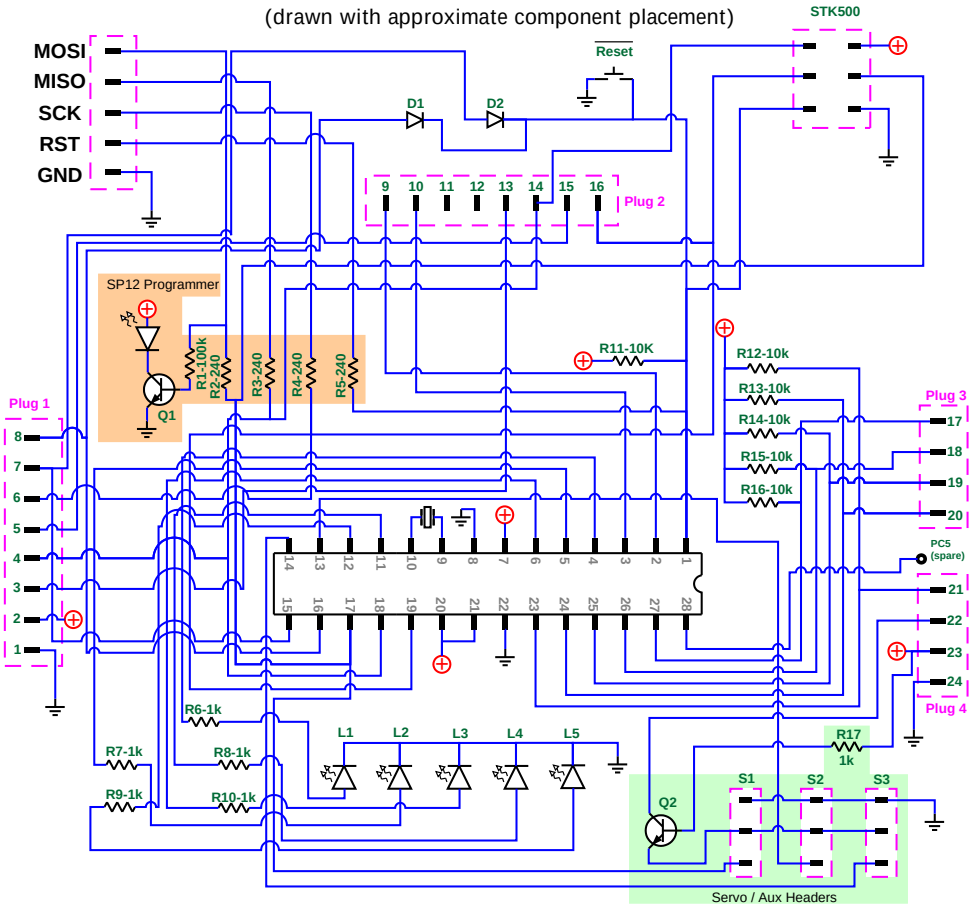
We've put the servo headers to good use by placing a servo on the top of our brainboard, and have it point an arrow in the direction it was going to move while doing line follower. It certainly adds character to a robot!

ATmega8 Microcontroller Pin Usage



Brainboard Schematic

(drawn with approximate component placement)



If you have any questions regarding this kit, please contact us!

Solarbotics Ltd.

179 Harvest Glen Way N.E.
Calgary, Alberta, Canada T3K 4J4
Toll Free: 866-276-2687 / 403-232-6268 Fax: (403) 226-3741
Website: http://www.solarbotics.com
Email: info@solarbotics.com