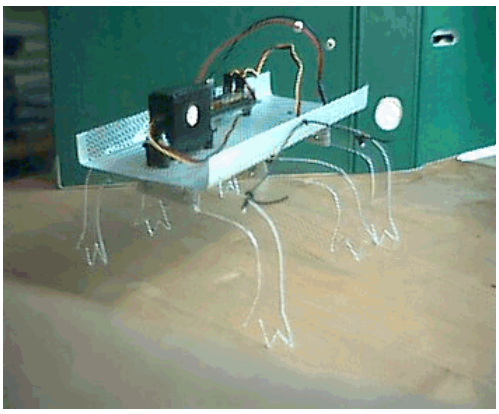
Y.A.C.H : Yet Another Cool Hexapod

A Little ‘’ How -To ‘’



History & Background:

I would like to add ‘’yet another hexapod’’ to the already countless others on the ‘net. As I’m no ‘’professional’’ my project naturally had to be as simple as possible, which on the other hand makes it a good beginner’s project for others too. So this text is also intended to be a little ‘’How-To’’ for the beginner.

Now, there are several ways to build a ‘’Hexapod’’ (which actually means ‘’Six-Feeter’’ in Greek), depending on the versatility of movements it is designed for.

A Robot’s legs could move back & forth, up & down, rotate around their own axles or maybe move in a circular way ... and those movements could be combined in any way. Each movement demands it’s own type of ‘’joints’’ and ‘’muscles’’.

Human legs for example basically use three large joints to manage the movements they’re capable of: The hip, the knee and the ancle-joint. Each and every joint has it’s own type of movement, all together make all the functions possible: Walking, Running, Jumping, Dancing ...

Of course there’s the ‘’fine tuning’’ with tiny joints and muscles such as in the foot which help making a human a rather ‘’All Terrain Vehicles“.

The more joints one uses the more sophisticated and variable are the possible types of movement. But also the more difficult are the mechanics and the necessary controll systems.

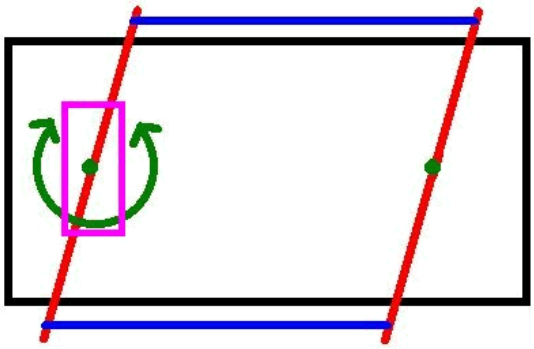
So there are Hexapods with one or two or even more joints and servos per leg. There are on the other hand also designs which couple two or more legs and therefor use *less* servos, thereby making the design simpler (and the project cheaper).

These designs will also be lighter, which is a big point to me (take a close look at all those beautiful hexa - and whatever - pods You can find on the net. You’ll often be able to notice an inconspicious little cable running away from the vehicle, providing it’s power supplies and sometimes also it’s control signals ... And I think a robot should be independant of this).

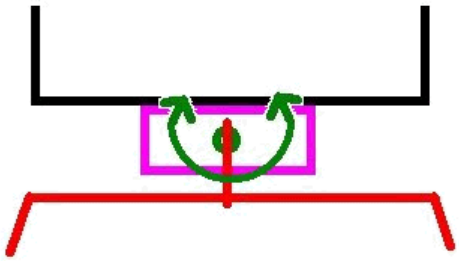
On the other hand the simpler designs (like this, for example ...) will not be ‘’All Terrain Vehicles’’of course. Because, for example, missing ‘’Degrees of Freedom’’ will have to be substituted by other factors, e.g. ‘’Friction’’. In fact, *this* vehicle is very much a ‘’linoleum’’ or ‘’wooden floor’’ vehicle.

The hexapod design I'm proposing uses just two servos. Take a look:

Robot as seen from above. The **chassis** is kept in black, **hind- and fore-legs** are in red. Both hind and fore-legs are coupled, they **rotate around the axles** (green dots). Both pairs of legs are coupled by **strings or wires** (blue line. Stiff or flexible connections, no matter. I used flexible, isolated wires). Only the front pair of legs is actually powered by a **servo** which is mounted on the chassis, with the axle facing downwards, the control horn directly connecting to the foreleg axles. The hind-legs are just ''pulled along''. Make sure the strings are not to tight and not to loose.



Now, this arrangement as shown above would make the robot's chassis just swing from left to right but not actually move it forward. But we haven't talked about the middle pair of legs yet ...



Robot as seen from the front. Colors as above. The **second servo** is mounted on its side, **axle** connected to the **middle pair of legs** with a little lever to enlarge it's range of movements.

And this is the principle of this hexapod's way to move:

Lift up the left side. Move the fore- and hindlegs on the left side forward and at the same time those legs on the right side backward.
Lift up the right side. Move the fore- and hindlegs on the right side forward and at the same time those legs on the left side backward.

Do it again. Voilà, it walks.

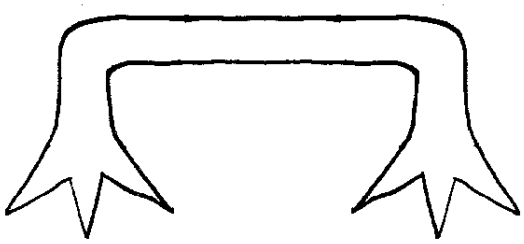
The Chassis:

Anything goes, really. As the first law of building small robots would probably be to *keep the weight down* one can use materials like foamed plastics, thin plexiglass boards, plywood and the like. My preferred chassis materials are Aluminium plates (0,7mm thick), with small perforations. They are used for decorative purposes mainly (I get them at the Do-It-Yourself-Shop).
To me those plates are ideal: they offer the necessary rigidness while being light enough, they are easy to work with (just make Your holes with a hand drill or use an old screwdriver if You like, cut them with an old pair of scissors, bend them by hand over the edge of Your working bench). And they look very robotic, too :)

The Legs:

The material is a bit of a problem. Depending on the way of movement of Your robot and the intended terrain it is supposed to walk on very different characteristics of the used material must be considered. There's the friction, the resistance against torsion or twisting, robustness, weight ...

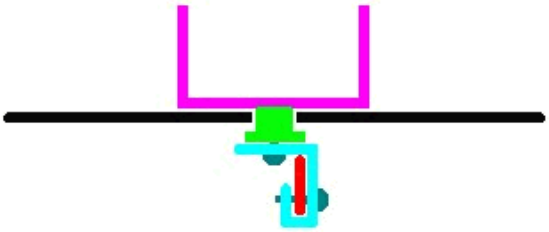
I used a 2mm thick plexi-glasslike material because I figured it was easy to work on. I drew the outline of the legs on the plastic sheet and cut them out with a fret-saw. The plastic I used showed a slight tendency to splinter at the edges. Another type of saw blade might help here. The legs are just a little bit ''floppy'' but after long test marches they are still in good working (or walking) order.



I decided to give my legs a reptile sort of look ...

... which from a more technical point of view is probably not the optimum. For example, walking on a carpet does not work very well because the feet will get caught in the slings. A ''foot'' or ''stump'' type of design would have been an alternative. The feet I used do add some *character*, though ...

The middle pair of legs must be slightly shorter than the two other pairs because it's center of rotation is a bit lower than the other two's. For connecting the legs to the servos I once again used an aluminium plate: I wrapped it around the plastic legs and used it to create the angles necessary to connect the fore- and hind-legs and the little lever described above to lower the turn point of the middle legs. Then I put the screws through both aluminium and plastic, thereby giving the plastic a little extra support. A little graphic will show what I mean:

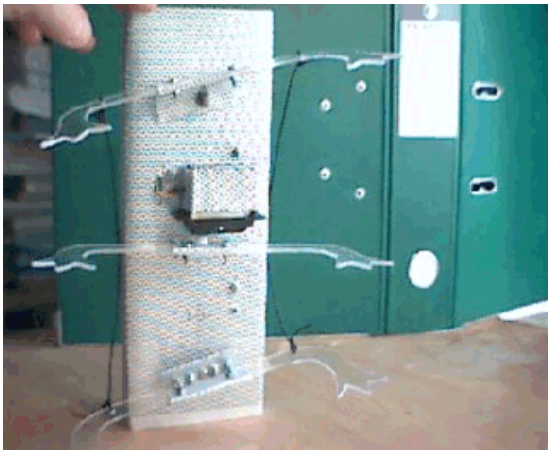


The fore-leg-pair, *seen strictly from the side*. The **control horn** is connected to the **servo** which is mounted upright on the **chassis** . The bent **aluminium plate** is fastened to the **control horn** with **screws**, it is wrapped around the **plastic leg** and also fixed with **screws**.

The back pair of legs is fixed quite similary (since there's no servo providing an axle I substitute it with a screw which is fastened by a nut and counter nut so it can turn easily without being to ''shaky'').

Now to the middle pair, again *seen strictly from the side*, with the **servo** underneath the **chassis**. The **control horn** is connected to the **servo** through an **aluminium plate** which again ''wraps in'' the **leg pair**, again tightened with several **screws**.

For connecting the hind- and fore-legs I used a flexible wire which is kept out of the way of the middle legs by another short piece of wire forming a sort of eyelet, tightened to the chassis (take a close look at the picture).



The Motors:

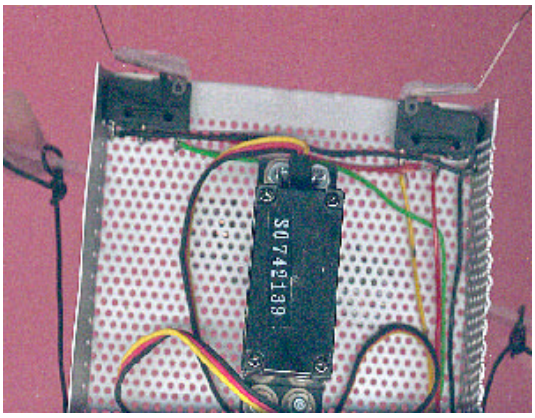
I'm using standard servo motors like they are used in RC models. There are lots of different brands around (Futaba, Hitec, Robbe, ...). I took the cheapest I could get. The torque will always be enough, using the light weight design I present. They should be able to feed from 5V.
But let me talk about how a servo is driven and what it *is* anyway.

A servo is a motor with a gearhead attached to it, which provides some torque and also reduces the motor's speed. Because of it's mechanics a servo's range of movement is restricted to about 180-240°. By using the control line of the servo You can tell it where on this potential radius it is supposed to turn to. The further away from the designated position the faster it moves to get there. As soon as it gets there it will try to remain there, no matter what forces (gravity, mainly) are trying to remove it from there, *as long as the power is switched on.* With the power switched off the servo will usually not be able to hold it's position!

How is the position sent over the single control line? A servo expects a pulse on it's control line about *every 20 msec* (this value is not too critical and may be varied - trial and error!).

The *length* of this pulse *is* critical. It will in fact tell the servo where to go (a pulse of 1.5msec will usually position it somewhere in the middle of it's range of movement). Again, the exact values have got to be found out experimentally. Remember, just one pulse will often not be enough to reach the designated position because the servo will take some time to get there. If one removes the pulse before the servo has reached the position it might not get there at all.

The Sensors:



At the front of the robot I have attached two ''cherry-type'' switches, with piano wires fixed to them. Bumping into something will switch one or both switches and make the robot react accordingly.

Eventually I'll supplement the sensors by an IR design (the PCB is in the make ...).

The Electronics:

Ah, now comes the easy part :) . I have decided to use Atmel AVR microcontrollers, the AT90S2313 to be precise. It is fast, easy to use for the beginner and (apart from temporal shortages ...) is widely available. And it offers a lot of features like 15 programmable I/O lines, a full duplex UART, an on-chip analog comparator, an SPI serial interface, 128 byte of RAM, 128 byte of EEPROM and -very important- 2Kbyte of FLASH ROM program memory which makes reprogramming it real fast and simple.Please do take a look at the specs:

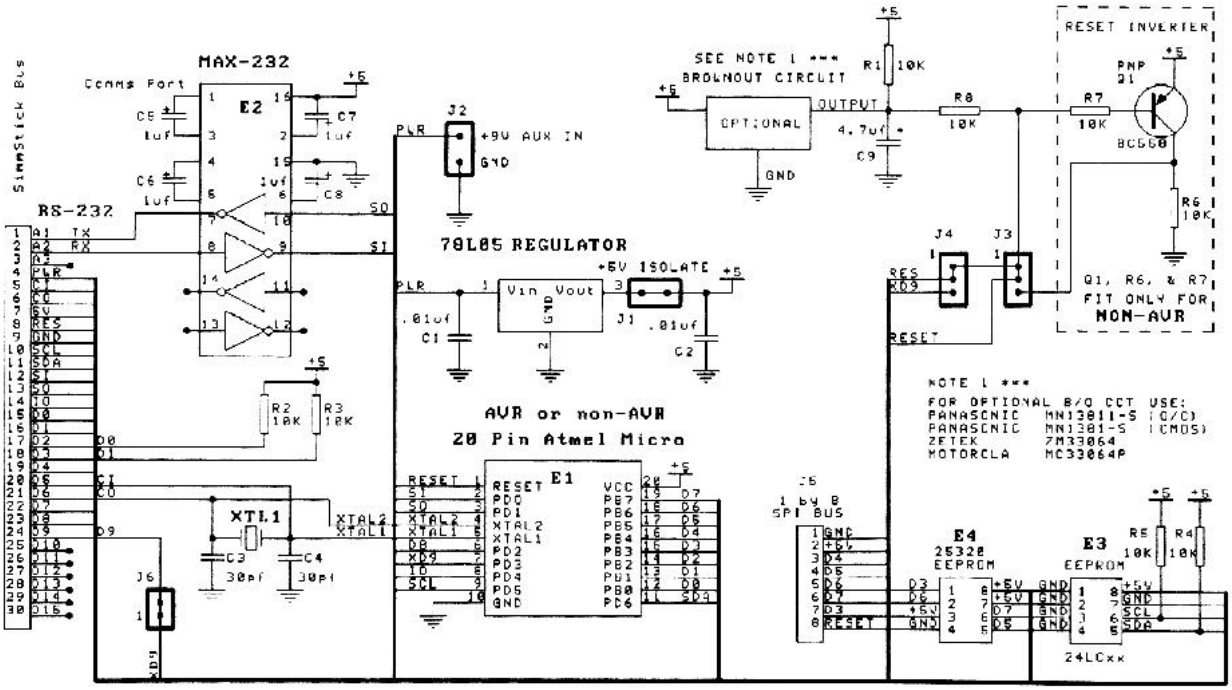
http://www.atmel.com/atmel/products/prod200.htm (or thereabout)

As a platform for the circuitry I use the DT104 by DonTronics / Australia which is cheap, done nicely and is good to solder. The layout and photo of the DT104 board are given below (most pictures taken from DonTronic's webpage http://www.dontronics.com , courtesy of Don McKenzie).

The DT104 is part of a program of PCBs with different layouts for different micros. All of these PCB's have one thing in common: They offer a bus with more or less standardized pins which are led out of the PCB using a ''SIMM'' type connector (which is why they're called ''SIMM Sticks''). Thereby one's given a simple way of connecting different PCB's, according to the function required.

So far I haven't used this feature: I am using the DT104 as a stand-alone circuit (but might change that when I eventually run out microcontroller I/O ports or want to add custom PCBs, which will sure happen some time ...).

Let's have a look (see a slightly larger image at the end of this text):



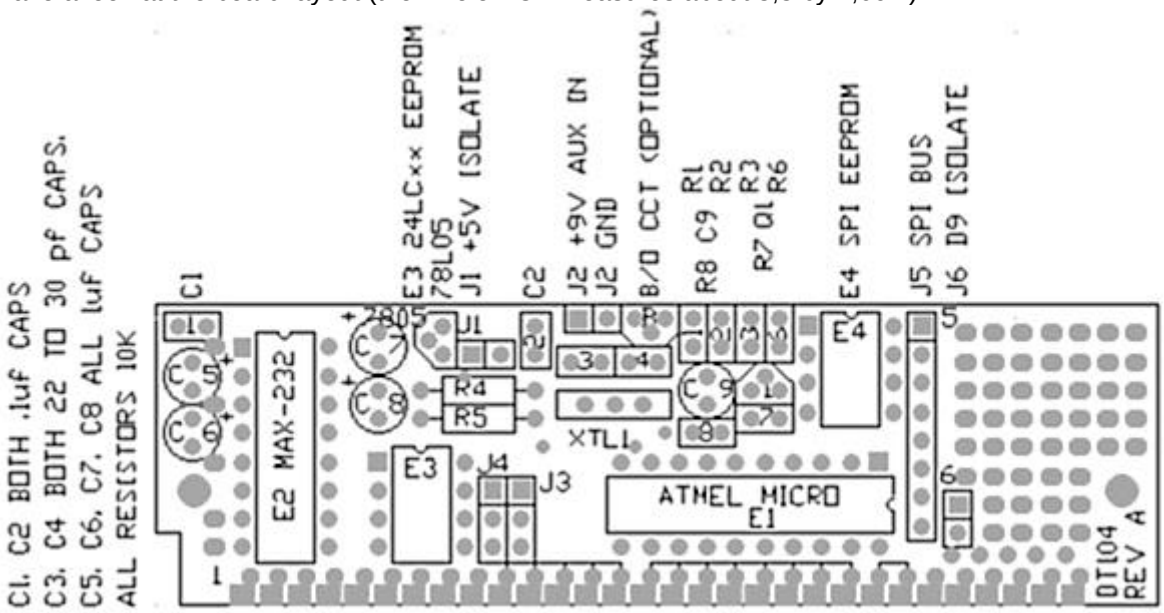
Of course there's all the necessary stuff to drive Your microcontroller like the quartz and a few other parts like capacitors and the like (see the '' Bill Of Material '' below).

As You can see there is the possibility to connect Your circuit to a PC's serial port via the MAX232 to the left (it's not necessary in my project but it might be useful to debug Your programs later on, as You can hand out messages to a PC).

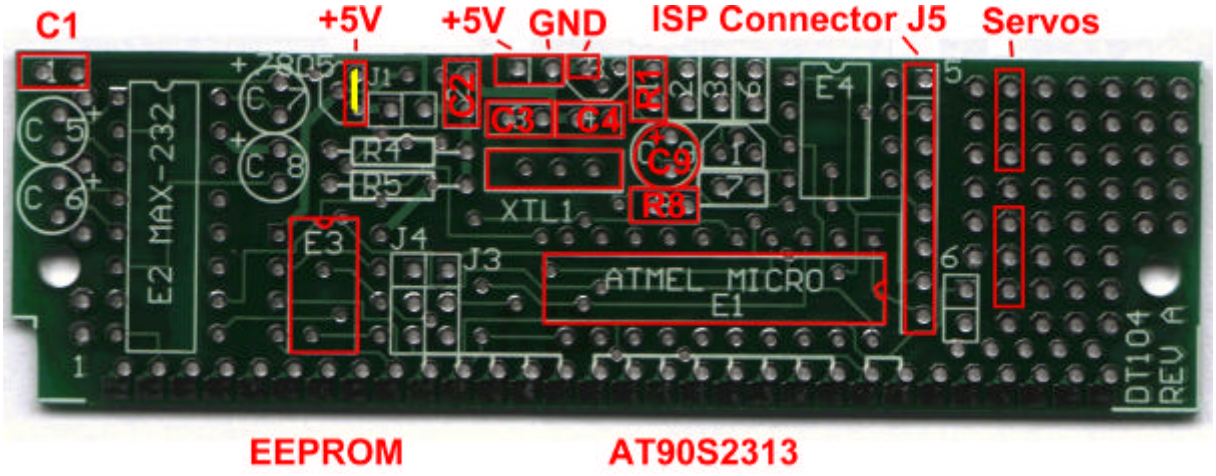
There is also space for a 5V voltage regulator. I don't need this feature either since I use 4 type AA batteries, providing 6V for both the circuit and the servos (which works well). Instead I use the regulator's solder pads to provide some extra voltage supply pins for external circuitry.
The brownout circuit and the reset inverter are not needed in my design, too.

The possibility to use one of two different external EEPROM chips (to be programmed serially) might eventually prove valuable, for example if one planned to collect data in some way. For the actual design -again- it's not necessary.

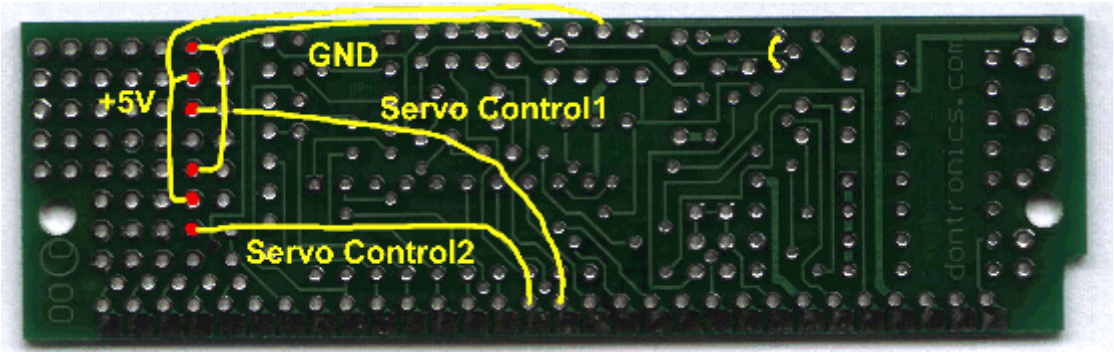
Have a look at the board layout (the whole PCB measures about 8,8 by 2,6cm):



Below is a picture of the PCB's upper side. In red I marked those parts of the circuit I actually use:



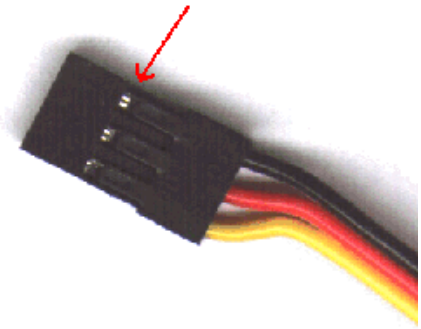
The Jumper J5 on the left will be needed for programming the chip ''in circuit'', using the PC's parallel port (see the ''Software Section below''). Next to J5 I add two three pin-rows for connecting the servos. Those pins are connected to the appropriate pins by stiff isolated wires on the solder side of the PCB.
C1 and C2 are ceramic condensators with 100nF capacity, C3 and C4 are 22pF condensators. C9 is an electrolytic condensator with 10µF / 63V.
R8 and R1 are 1kΩ resistor.
E3 is an 8pin socket for an external EEPROM. E1 is for the 20pin socket for the microcontroller (using sockets is definitely recommended ...).
In- and output pin pads of the 7805's part of the PCB are shorted (yellow line, see also picture of the solder side of the PCB below!) and used for additional +5V pins. Also, for another GND pin I use the left one of the three pads of the brown out chip.
For the XTAL I am using IC-socket-typed connectors so that I can eventually change the quartz type.
The rest of the board is not used so far.



These are the connections added on the solder side (stiff isolated wires in different colors will do nicely: Black for GND, red for +5V and yellow for the controll lines).

The red dots are the two servo connectors which both carry the power supply and the control lines. These connect to the microcontroller's I/O pins PORTB 0 and PORTB 1, respectively. Please count the pins of the SIMM bus thoroughly (the appropriate pins should be Nos.15 and 16 - look for the position of pin1 !).

Just to have more possibilities for connecting stuff I usually solder a row of pins into the SIMM bus connector pads. If You intend to use the SIMM module *as such* You won't do that of course ...



Please take care: The actual position of the three signals +5V, GND and Control on the plug may vary, depending on the type of servo You use. In some cases You will have to swap the wires in the servo's connector by carefully pressing on the little clamp with a small screwdriver and then removing the wires:

In any case the red and black wires will stand for plus and minus, respectively whereas the yellow (or white) wire will be the control line.

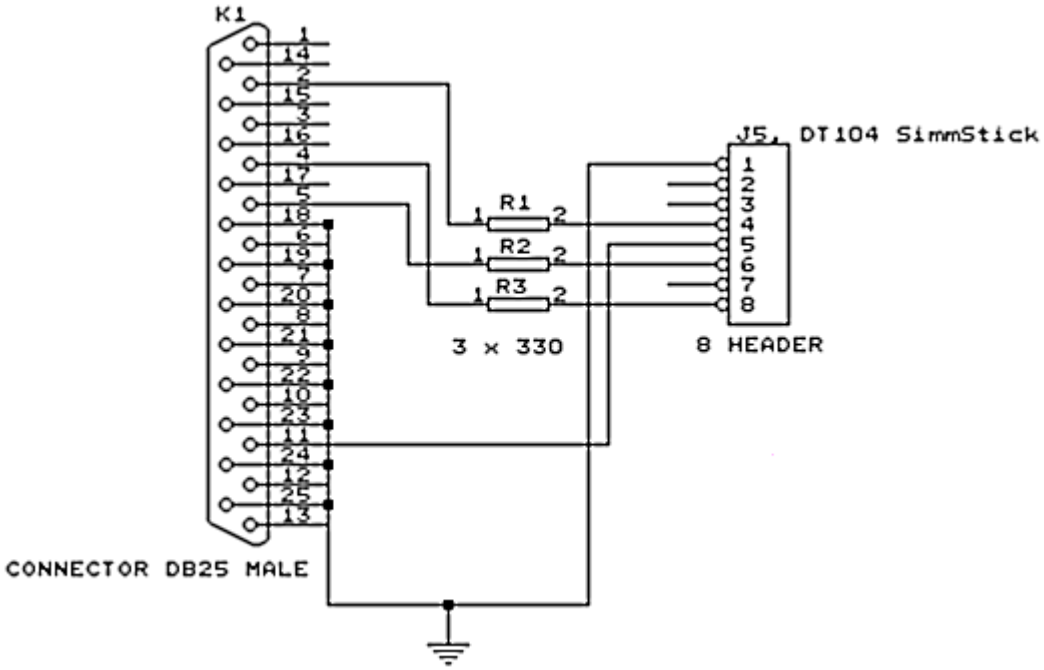
A battery holder for the four AA-type batteries and an OFF-switch complete electronics. The next step would be the connection of sensors and maybe actuators, too.

The Software:

What the DT104 is for the electronic's side of the project, the BASCOM basic compiler is for the software department. The Dutch Firm MCS Electronics is offering this really useful tool for a very reasonable price. And above, they are distributing their product in a (free) demo version which is fully functioning and is only restricted in the size of programs it will compile. This maximum code size is fully sufficient for the '2313, though !

BASCOM is very similar to Quickbasic so it should be easy to learn ... Most of the AVR's features and lots of external hardware are supported. There is a rich wealth of basic instructions for making sounds, driving LCD displays, using PWM or the various integrated interfaces. And there are constantly functions being added.

Another bonus is the possibility to program Your chip ''in circuit'' from out of the compiler. There's no need for a dedicated programming device, a simple self made cable is enough.
A small schematic (once again from DonTronic's webside) is added here for completeness:



I bought a parallel cable and cut off the surplus connector, then soldered on a little 8pin connector fitting into the pin row of J5. The 330Ω resistors can as it says on Don's webside be omitted. I did and it works well (the resistors provide a little bit of security for Your PC's port. I am using a cheap plugin-card for an extra parallel port which does the same ...).

The latest version of the software can be found on MCS Eletronic's homepage

http://www.mcselec.com

There is a rather lively mailing list where usually always someone is willing to answer silly questions. Besides, Mark Alberts, (co-?) programmer of BASCOM is one of the regulars readers too so there's information from first hand available too.

The program below provides a way of ''steering'' the vehicle. See how it's done ...
Eventually I will have to introduce a sort of operating system, maybe even with a simple multitasking ability so that I can read sensors *while* driving the motors.
The code is far from optimated, too. I've had slight problems already with the program space ...

(Somewhere on this PDF page there should be a ''note'' which contains this program in a form You can copy & paste into BASCOM easily ... uncommented version)

```
'*************************************************************************
'* ATTENTION PLEASE: this code compiles with BASCOM AVR 1.11b and up !   *
'*                                O.N.L.Y.                                *
'*************************************************************************
'*        YACH - Version 1.1         *
'*************************************
'*    Hexapod with two servos        *
'*        and two feelers            *
'*     "Steers like a cow" ;)        *
'*************************************
'* Portb.1: Middle Pair of Legs      *
'* Portb.0: Front- and Rear Pair     *
'* Portb.2: Left feeler              *
'* Portb.3: Right feeler             *
'*************************************
'*               Init                *
'*************************************

Init:
$crystal = 10000000                 ' Quartz value in Hz

$baud = 9600                        ' Baudrate for the serial port

Ddrb = &B11110011                   ' set portb pin direction for minm & b3
'                                   ' to "in", rest to out

Dim A As Word
Dim H As Byte
Dim B As Byte
Dim Dir As Byte                     ' contains the direction
Dim Newtouch As Byte                ' =1: move is called by a feeler sub
Dim Feeler As Byte                  ' feeler value, 0=No Contact,
'                                   ' 1= Left, 2= Right, 3= Both
Dim Speed As Integer                ' speed of movement
Dim Maxf As Byte , Minf As Byte     ' Maxi-/Minimum range (forelegs)
Dim Maxm As Byte , Minm As Byte     ' Maxi-/Minimum range (middlelegs)
Dim Count As Byte                   ' duration of current type of movement

Config Portb = &B11110011           ' configure PORTB as OUTPUT

'*************************************
'*              Main                 *
'*************************************

Main:                               ' Main Routine
    Newtouch = 0                    ' no Contact on Switch-on
    Dir = Rnd(7)                    ' randomly select a direction/action

    Gosub Feelme                    ' check feelers, if "contact" Dir will
'                                   ' be changed accordingly
    Gosub Action                    ' move depending on Dir
Goto Main                           ' keep on walking forever

'*************************************
'*               Action              *
'*************************************
```

```
Action:


On Dir Goto Fow , Lef , Rig , Dnc , Bck , Ffw , Pse , F_rgt , F_lft , F_bth

Fow:                                    ' forward
   Maxm = 110
   Minm = 230
   Maxf = 180
   Minf = 150
   Count = 20                           'it will walk forward for a longer time
   Speed = 5
   Gosub Moveme
Return

Lef:                                    'turn left
   Maxm = 145
   Minm = 230
   Maxf = 200
   Minf = 150
   Count = 16
   Speed = 3
   Gosub Moveme
Return

Rig:
   Maxm = 110
   Minm = 145
   Maxf = 165                           ' turn right
   Minf = 120
   Count = 16
   Speed = 3
   Gosub Moveme
Return

Dnc:                                    ' Step Dance :)
   Maxf = 165
   Minf = 165
   Count = 10
   Speed = 7
   Gosub Moveme
Return

Bck:                                    ' backward
   Maxm = 230
   Minm = 110
   Maxf = 180
   Minf = 150
   Count = 10
   Speed = 5
   Gosub Moveme
Return

Ffw:                                    ' fast forward
   Maxm = 110
   Minm = 230
   Maxf = 180
   Minf = 150
   Count = 10
   Speed = 7
   Gosub Moveme
Return

Pse:
    Wait 3
Return

F_non:                                  'no contact made - no action done
Return

F_lft:                                  ' left feeler touched
   Newtouch = 1                         ' move without testing feelers
   Gosub Dnc                            ' Panic !
   Gosub Bck                            ' shy back
   Gosub Rig
   Newtouch = 0                         ' try the right side
Return

F_rgt:                                  ' right feeler touched
```

```
   Newtouch = 1                     ' move without testing feelers
   Gosub Dnc                        ' Panic !
   Gosub Bck                        ' shy back
   Gosub Lef
   Newtouch = 0
Return

F_bth:                              ' both feelers touched
   Newtouch = 1                     ' move without testing feelers
   Wait 3                           ' BIG panic :)
   Gosub Dnc
   Gosub Bck                        ' shy back
   Gosub Lef
   Gosub Lef
   Newtouch = 0
Return


'************************************
'*               Feelme             *
'************************************

Feelme:
   Feeler = Pinb And &B00001100
   Feeler = Feeler / 4
   If Feeler = 0 Then Return
   Dir = Feeler + 6                 ' => 7,8,9: for the "On .. Goto .."

Return

'************************************
'*               Moveme             *
'************************************

Moveme:                             ' approx. values for the range:
'                                   ' 115-179: forward
'                                   ' 187-255: reverse
'                                   ' 180-186: zero position
While Count <> 0
   For H = 0 To 9
      Set Portb.1                   ' start pulse on portb, pin1
      For B = 1 To Maxm
         Waitus 4
      Next B
      Reset Portb.1                 ' end pulse on portb, pin1
      Waitms 10
   Next H

   For A = Minf To Maxf Step Speed  ' The value for "step" will
'                                   ' determine the speed.
'                                   ' counting upward means "forward"
      For H = 0 To 2                ' give the servo some time to turn
         Set Portb.0                ' start pulse on portb, pin0
         For B = 1 To A             ' "A" determines the actual position
            Waitus 4                ' the servo is supposed to go to
         Next B
         Reset Portb.0              ' end pulse on portb, pin0
         Waitms 10                  ' wait a bit ("every 20 msec" ...)
      Next H
      Set Portb.1                   ' start pulse on portb, pin1
      For B = 1 To Maxm
         Waitus 4
      Next B
      Reset Portb.1                 ' end pulse on portb, pin1
      Waitms 10
   Next A

   For H = 0 To 12
      Set Portb.1
      For B = 1 To Minm
         Waitus 4
      Next B
      Reset Portb.1
      Waitms 10
   Next H
   If Newtouch = 0 Then
      Gosub Feelme                  ' quick check: feeler contacted?
      If Dir > 6 Then Return        ' yes?
   End If
```

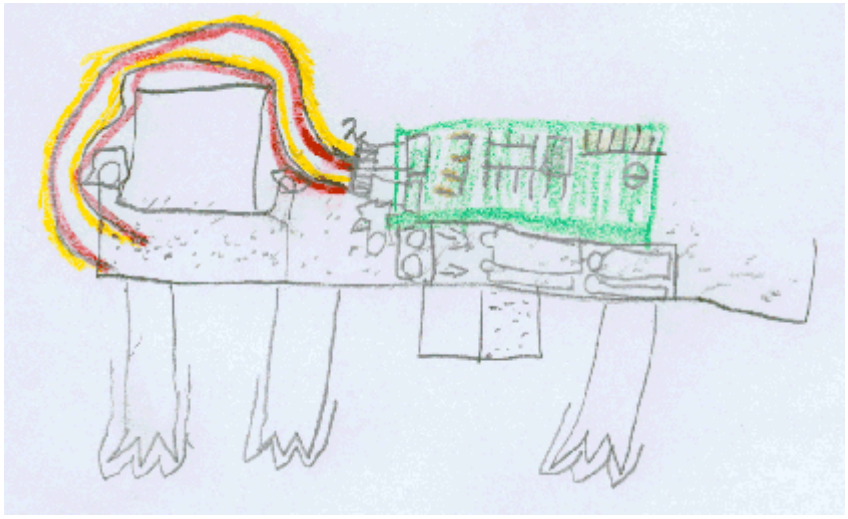
```
   Speed = Speed * -1                ' change direction
   For A = Maxf To Minf Step Speed   ' counting downwards means
'                                    ' moving backwards
'                                    ' all is relative, of course :)
      For H = 0 To 2
         Set Portb.0
         For B = 1 To A
            Waitus 4
         Next B
         Reset Portb.0
         Waitms 10
      Next H

      Set Portb.1
      For B = 1 To Minm
         Waitus 4
      Next B
      Reset Portb.1
      Waitms 10
   Next A
   Count = Count - 1                 ' repeat curr. direction until COUNT=0
   Speed = Speed * -1                ' change direcion again
Wend
Return

' And Yes, I would hate to take part in a dead reckoning contest ;)

End
```

Thanks to everybody on the web who helped me!



And thanks to my son Jonas for helping me along with his instructive paintings :)

Friday, 27. October 2000,

Christoph Klein (hyla@mayn.de)

Literature

Build Your Own Robot!
by Karl Lunt
publ. by AK Peters
ISBN 1-56881-102-0

The Robot Builder's Bonanza
by Gordon McComb
publ. by McGraw-Hill
ISBN 0-07136-296-7

BASCOM-AVR User manual
by Mark Alberts
publ. on http://www.mcselec.com
ISBN -

AVR 8-Bit RISC - Data Sheets AT90S2313
Atmel
publ. on http://www.atmel.com
ISBN -

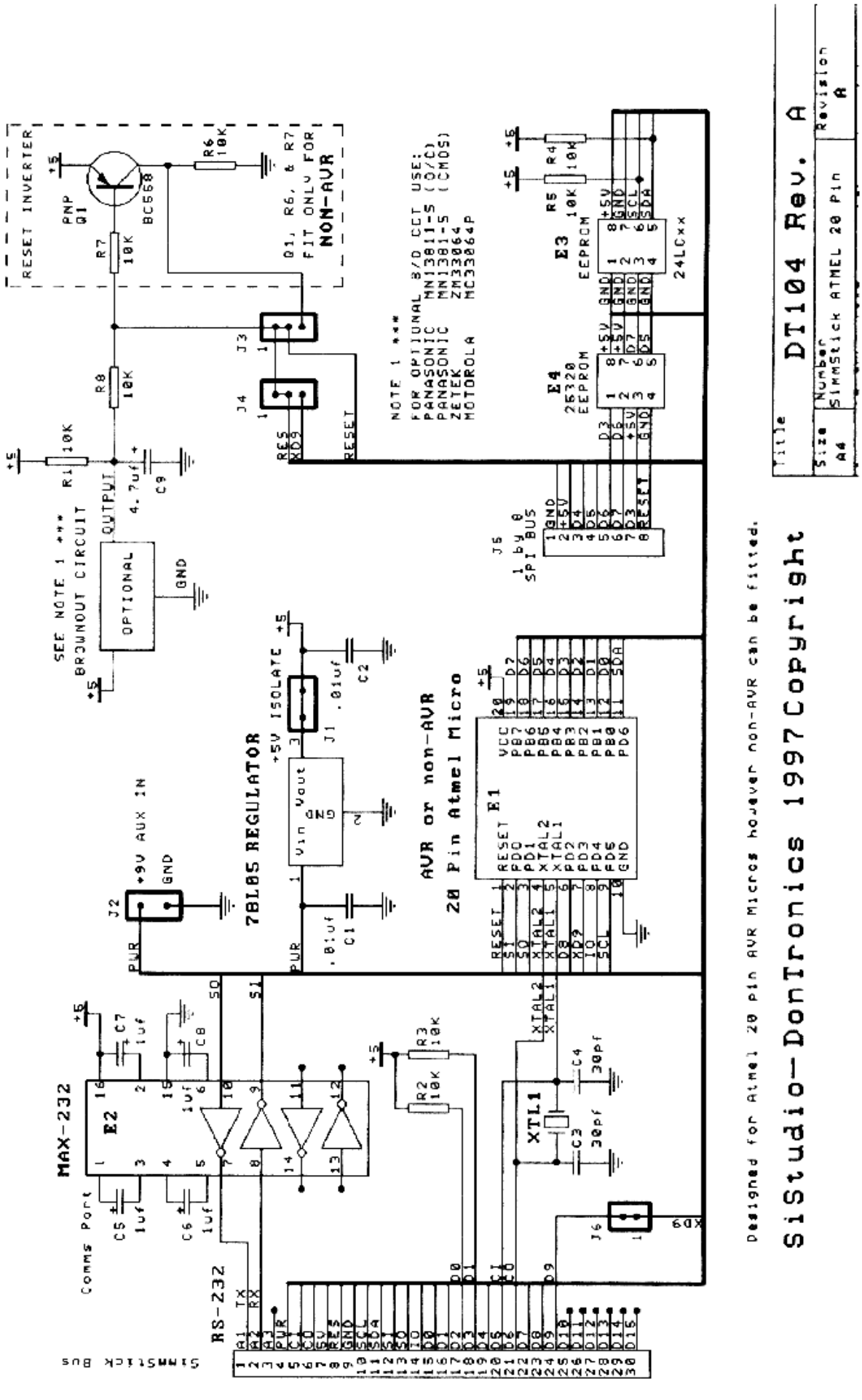
DonTronics product descriptions
Don McKenzy et al.
publ. on http://www.dontronics.com
ISBN -

And various, not to say countless other websides with clever tips and projects all around the world ...

Bill Of Material:

- Material for a chassis
- Material for the legs
- 2x standard servos
- a battery holder for 4 type AA batteries
- a modified parallel cable for ‘’in circuit programming’’
- a DT104 PCB (DonTronics and others, who can also tell You about a distributor in Your vicinity)
- 2x 100nF ceramic condensators
- 2x 22pF ceramic condensators
- 2x 1kΩ resistor
- quartz 10MHz
- 10µF / 63V (or so) electrolyte condensator
- 1x AT90S2313 microcontroller (Atmel)
- 1x 20pin ‘’narrow type’’ IC socket
- solder pins (the ones usually used for jumpers! Sold in rows of 50 or so to be separated on will ...)
- various isolated wires
- metall screws / nuts / washers
- BASCOM AVR compiler (MCSElectronics) or the ATMEL assembler if You feel strong enough

... Not a lot, really. I must have forgotten something :-)


SiStudio—DonTronics 1997 Copyright
Designed for Atmel 20 pin AVR Micros however non-AVR can be fitted.
DT104 Rev. A
Title
Size A4
Number SIMMStick ATMEL 20 Pin
Revision A
AVR or non-AVR
20 Pin Atmel Micro
78L05 REGULATOR
+5V ISOLATE
+9V AUX IN
MAX-232
RS-232
Comms Port
SIMMStick Bus
RESET INVERTER
PNP Q1
BC558
NON-AVR
FIT ONLY FOR
Q1, R6, & R7
NOTE 1 ***
FOR OPTIONAL B/D CCT USE:
PANASONIC MN13811-S (C/C)
PANASONIC MN13811-S (CMOS)
ZETEK ZM33064
MOTOROLA MC33064P
SEE NOTE 1 ***
BROWNOUT CIRCUIT
OPTIONAL
OUTPUT
E1
E2
E3
E4
EEPROM
24LCxx
25320
SPI BUS
XTL1
RESET
GND
+5V

A Short Movie: