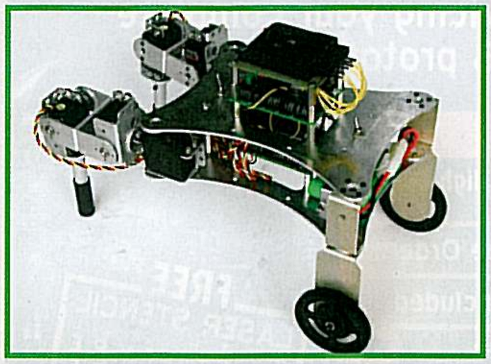
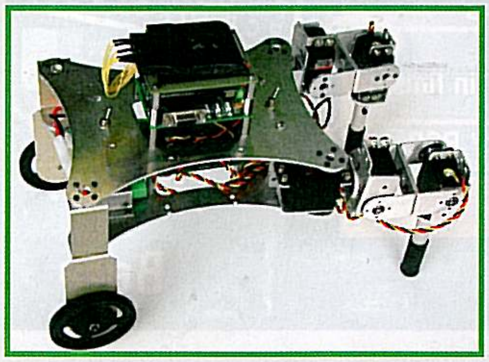
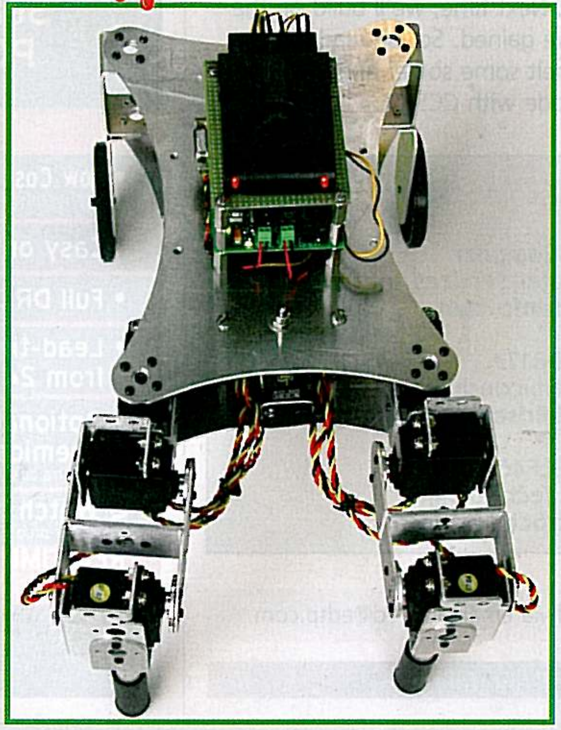
Creepy Hybrid







Part 2

By Alan Marconett

Last time, we discussed the mechanical aspects of Strider, the "Creepy Hybrid" — a low-cost platform for exploring the operation and construction of a legged robot. We've also got it mostly assembled. This time, we'll discuss how to program the microcontroller and the servo controller board.

For Starters

For our learning experience, we will compile our own "stripped down" and modified version of the Lynxmotion (LM) AH3-R Basic program that will run on the BB2 board with an Atom Basic module installed. The Atom module sends serial commands to the SSC32 servo driver board. For our own fully commented Creepy Hybrid control program, check the *SERVO* website for downloads (**www.servomagazine.com**).

Creepy Code

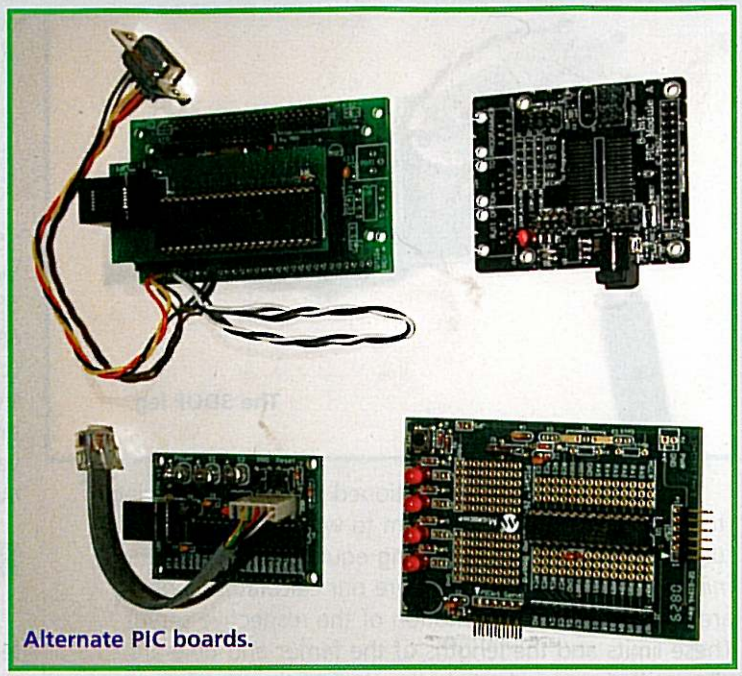
Our BASIC program will only use the joysticks of the wireless PS2 controller and few — if any — of the buttons as it is just intended to get us started. We made an effort to minimize the code length in order to simplify the program. Feel free to expand it!

The basic (no pun intended) tasks of our program can be summarized:

- Read PS2 joystick (or a COM port).
- Interpret the joystick data into a "motion vector" (which way we want to go).
- Calculate IK (Inverse Kinematics) for the leg moves.
- Move the legs.

Using some "canned" peripheral functions available in Basic Micro's Atom Basic makes it easy to read the PS2 interface from a joystick or a COM port. The *shiftout* and *shiftin* commands you see first initialize the PS2 interface for us (we want the joystick in analog mode) and then allow us to read information from the joystick. We're primarily interested in the data from the right joystick. This joystick will allow us to control or drive the robot forward or backward, and make left or right turns. But you knew that!

To keep this program similar to the original LM code so that you can also start understanding it, I'm going to use the same coordinate systems as the AH3-R Basic program created by the free LM PowerPod utility. That is, X is forward and backward, Z is left to right, and Y is up and down. Confused yet? Don't worry, you really don't have to understand any of the IK code in order to program and run the robot. I'm also going to keep most of the variable names, as well. After you are familiar with our version of the code, you should have a good start on understanding the full AH3-R code.


Alternate PIC boards.

Orders Received!

The PS2 wireless joystick is our primary control device for this robot (Creepy2ps2.BAS version). It's how we receive our marching orders. The control program running on the BB2 reads the PS2 and saves the data into the PS2 variable array called *DualShock*. From here, we get the *XSpeed* and *YSpeed* parameters, and then calculate a vector angle (*DAngle*) and a distance (*DCoord*) which are the components of our motion or "command" vector. When we detect a command that we need to act on, we'll set a value in *MovesDelay* which will allow us to take a few steps. We'll also watch for a few other buttons, such as the circle and triangle buttons, and the left joystick. We have a height control (left vertical joystick, *LegUpShift*) and a steering control (left horizontal joystick, *Steering*). We will read the digital joystick left and right buttons to control our gait speed (*GaitSpeedTmp*). Oh yes, we can also sound a horn by pressing the left joystick button!

Inverse Kinematics

It's relatively easy to do the FK (Forward Kinematics) for a leg. That's just "I have the angles of the hip, thigh, and knee joints, and the lengths of the tibia and femur. How do I figure out where the foot will go?"

What's harder is figuring out how to set the servo angles to move the foot where we want. That's where IK (Inverse Kinematics) comes in. It does the calculations of "How do I move my leg joints to get the foot to this position?" We'll do this with a little pythagorean theorem, a sine and cosine calculation here and there, and the good old cosine law. If you don't care for more high school trig, feel free to skip to the next section.

We'll calculate a hip angle first. Our current *Xpos* and *Zpos* (foot distances from 0) for one leg are two sides of a right triangle. The hypotenuse of this triangle is calculated (Distance) and then we figure out the angle between them using the Arc Cosine function. Atom Basic doesn't have an acos function, so the original program cleverly uses a subroutine to do so using a look-up table. The result is stored in *HipH_Angle* (horizontal).

Next, we're going to calculate the knee angle. Remember "c^2 = a^2 + b^2 - 2ab cos(C)" — the cosine law?

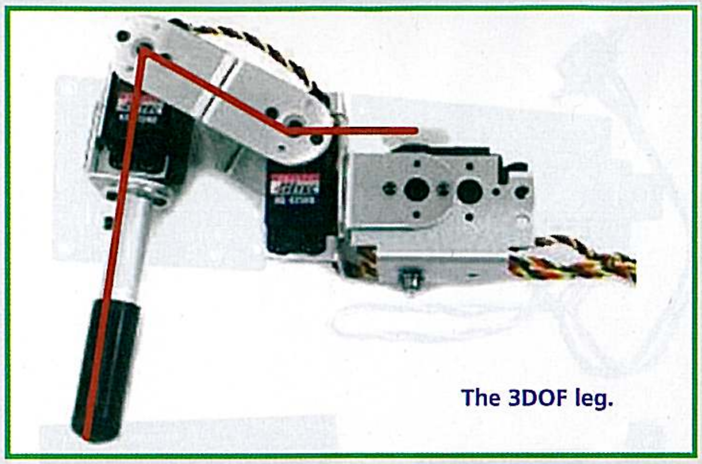
Think of the tibia as the "b" side of the triangle and the femur as the "a" side. We'll square the tibia length and the femur lengths, and then subtract the square of our previous triangle's hypotenuse. (Still with me?) We still need a 2ab term, which is two times the femur length times the tibia length. All we have to do now is take the arc cosine of this partial result, and we've got our knee angle (*Knee_Angle*). Two calcs down, one to go.

We'll use the cosine law again, only this time we'll add two angles together: one formed by the femur and tibia (*HipVertAngle*), and one formed on the ground (*TempAngle*). The result is the vertical hip angle (*HipV_Angle*). I won't go through all that again; that'll be left as an exercise for the reader!

The program to run Strider is available online at **www.servo magazine.com**. Get the appropriate BAS control program (PS2 or COM), and then compile and download the file to the BB2 with the appropriate IDE available from Basic Micro (**www.basicmicro.com**). I used the 5.3.1.3 Atom IDE. Refer to the BB2 and Basic Atom documentation to accomplish this.

Builders will find it useful to download the Power Pod program from the Lynxmotion website at **www.lynxmotion.com**, generate code for the CH3-R Hexapod robot, and follow along in it for comparison. Many of the variables mentioned here have been designed to be the same as in the CH3-R robot code. When building the CH3-R program, several parameters you'll be interested in are:

```
IDE: Basic Micro IDE V05.xx
Control:   PS2
H3 Leg:    3FOFA
H3 Body:   Round
PS2 Controller/BB2 connections: BB2 (pins 12, 13, 14, and 15)
PS2 sticks dead zone:  small
Tibia Angle:   vertical
SSC32 on (pin 8)
```


The 3DOF leg.

Notice that I haven't mentioned some of the scaling (the 127s) used by the program to work with Atom Basic functions, nor the range limiting equations that use the *min* and *max* functions to insure our calculated angles are within the range of motion of the respective servos. These limits and the lengths of the femur and tibia are all specified as constants in the start of the program. Change them if you change the leg component dimensions.

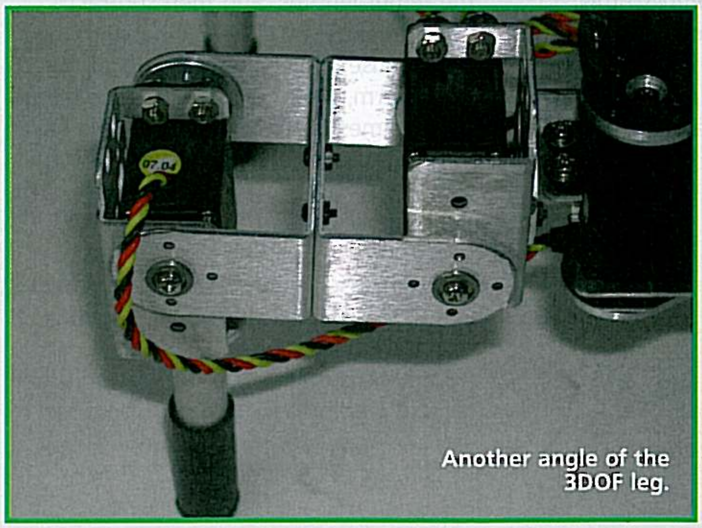
We'll do these calcs for both legs. Imagine all the calcs that need to be done for a hexapod or an octapod!

Move Those Legs!

Now that we've got our three servo angles per leg, we'll scale them into pulse ranges that our servos can understand. We'll also use the *min* and *max* functions again to insure that the calculated values won't send a servo into the stops. (You wouldn't like that!) Some min/max constant examples:

```
HipH_PulseMax - HipH_PulseMin
HipV_AngleMax - HipV_AngleMin
```

You'll also see three more variables: *HipH_Pulse*, *HipV_Pulse*, and *Knee_Pulse*. You guessed it! These are the pulse values we'll send to the servos later. For the calculation of each of these three variables, we'll use


Another angle of the 3DOF leg.

constants for the allowed range of the servos, such as *HipV_PulseMin* and *HipV_PulseMax*, for example.

```
serout SSC32,i38400,
[ "#",FRHH,FRHH2,"P",DEC HipH_Pulse(2)
```

With these values in hand, we'll send them to the SSC-32 with the special Atom Basic command *serout*. We send out a string of characters and values that the SSC-32 interprets for us and continuously positions our servos. Believe me, the SSC-32 saves a LOT of work (calculations) getting our servos to their positions! We send the servo # (character constants defined as *RRHV,RRHV2* and others), a 'P' for a position command, and a decimal value which is the position of the servo we just calculated. We do this for all six of our servos. Whew!

But Wait!

Before we can do those IK calculations and move the legs, we need to back up a bit and figure out which leg to lift and when. In the Tripod subroutines, we basically alternate between the two legs — lifting one of them, moving the lifted leg forward, and the "grounded" leg back to take a step. The "*Tripod*" and "*Steps*" variables are used to walk us through dividing up the gait into little steps to make the moves smooth, and alternating which leg is up.

```
YPos(Index) = -Tibia_Length + Height
XPos(Index) = XPos(Index) + (XPos2(Index)
- (XPos(Index) - (HipV_HipH +
Femur_Length)))/StepFlag

ZPos(Index) = ZPos(Index) + (ZPos2(Index)
- ZPos(Index))/StepFlag
```

We've now updated the X, Y, and Z positions for the feet by a little bit each time through the loop.

This entire gait action is controlled by a big loop; it proceeds when we have a reason to move the legs (*MovesDelay*) and have finished the last move of the legs. We can't very well issue new move commands to the legs while the SSC-32 is still moving them from the previous commands, now can we?

Did I Miss Anything?

There are some little subroutines that set up the trig functions that I mentioned. I won't go over them here. We have some initialization routines for the leg's starting positions (*H3Init* and *InitPos*), but that's about it. There are also three LEDs that can optionally be wired in to allow monitoring of the PS2 or COM activity (yellow), and the two legs (red and green).

Alternate Code

A program very similar to the Atom Basic program can be written and compiled by MBasic Pro from Basic Micro.

This is a conventional compiler that can generate executable code that will run on a PIC chip installed in either of the previously mentioned PICkit 2 demo board or SchmartBoard's "A PIC" board. The advantage here is that you can write code for any of several different blank PICs and use them in a PCB (printed circuit board) of your own choosing. (Some of us like to make our own boards!)

Besides the Atom Basic, there is also the Atom Pro Basic, which is a much faster processor with more functions. It requires a slightly different Basic program to run it (Creepy2p.BAS), which is also available for download. The "Pro" IDE for this module allows compiling C programs that can be run on the Pro module.

PC COMS

Oh yeah, you will have a different routine to replace the PS2 interface if you elect to use a COM port connected to a PC instead. On the PC end, you can use the PS2 simulation program (Serial_CP_H3.exe) supplied with PowerPod to talk to the bot. The COM version (Creepy2com.BAS) has the appropriate serial interface code. In this case, the bot sends the characters "Rd" to the PC to tell it when it is ready to receive a new command. There is also the addition of a checksum calc to insure the data is okay.

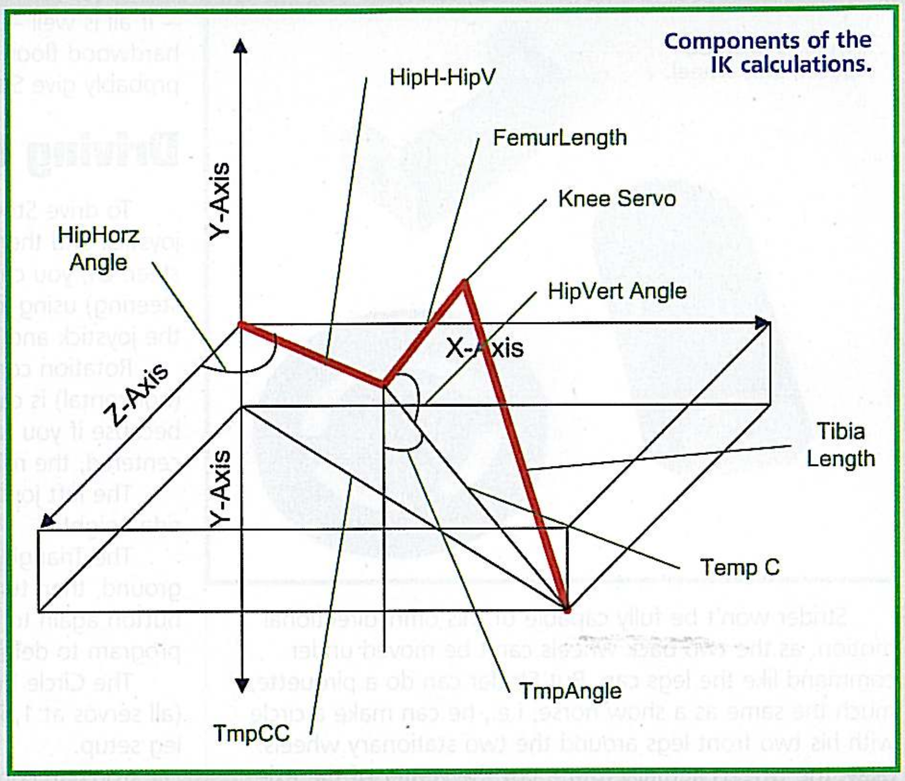
PC Direct to SSC-32

For a really simple implementation, it's also possible to directly control the SSC-32 board from a Basic or C program running on the PC via a trailing COM wire (or use Blue Smirf). While I haven't actually done this yet, it looks like an interesting possibility for those inclined. The same basic calculations would be done, but this time on the PC instead. The PC program would send the same servo commands out over a serial port to the SSC-32. You'd be on your own to work out a joystick.

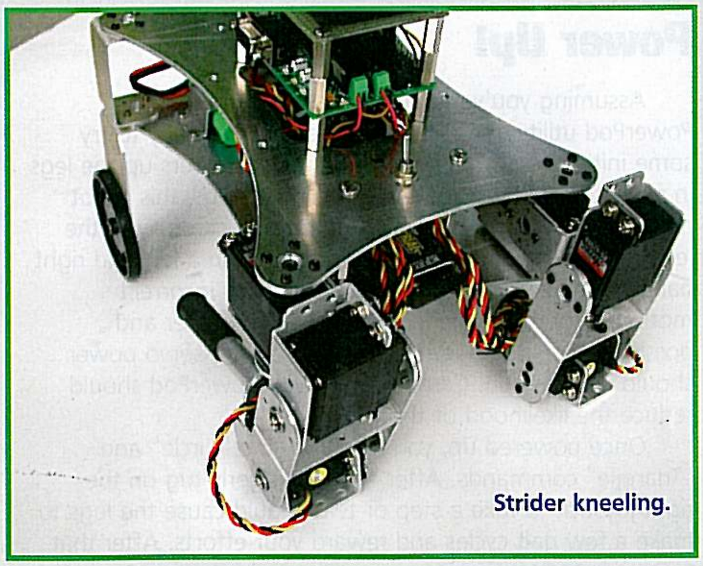
More on the Creep Gait and Steering

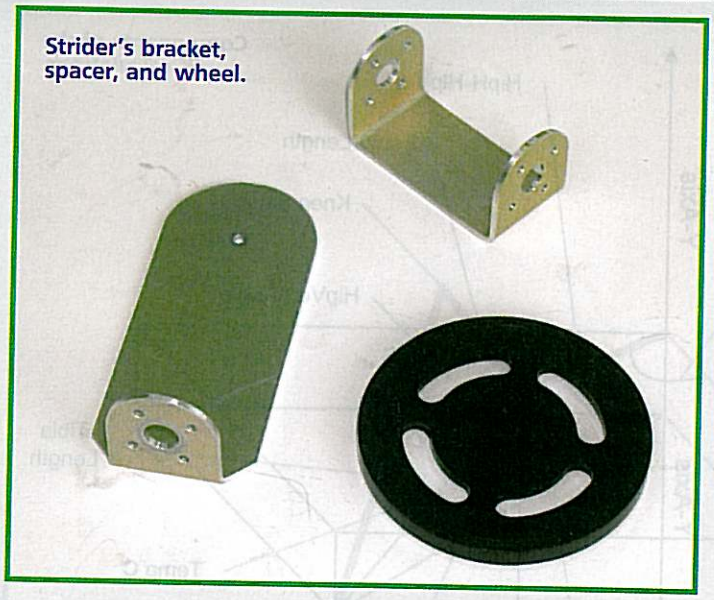
Actually, there are two ways to move a basically rectangular robot forward. One can simply put half the legs/wheels on one side of the robot on one "stick" (control channel) and the other half of the legs/wheels on the other side on the other stick. This is a basic tank drive (also called skid-steer) which can still be seen in use today in tracked vehicles. The operator controls the power/braking applied to each track individually, and the balance of the power or braking causes the vehicle to turn or go straight. Many R/C controlled four wheel drive vehicles work in this manner, although they've combined the two sticks of the old caterpillar tractor into our modern joystick. Our legs would simply move forward/backward at the same speed, or slightly different speeds to make turns (just like with tracks).

The other way (much more fun) for a legged robot to move is to move each leg so that it gains ground in a "vector" (angle and distance) relative to its chassis. The robot is thus able to "translate" in any desired direction. Translation is defined as "Motion of a body in which every point of the body moves parallel to and the same distance as every other point of the body." That is a good definition of the motions my round hexapod robot is capable of. A quadruped robot with splayed legs should also be capable of this motion (I'll find out when I build a quadruped for my next project).



Components of the IK calculations.


Strider kneeling.


Strider's bracket, spacer, and wheel.

Strider won't be fully capable of this omni-directional motion, as the two back wheels can't be moved under command like the legs can. But Strider can do a pirouette, much the same as a show horse, i.e., he can make a circle with his two front legs around the two stationary wheels. okay, the wheels actually rotate but the center of the axle between the two wheels stays in place. So, what we have is the two legs being capable of moving in any direction, with the wheels basically just following.

Both models of steering can be simplified to a "bicycle" model (front steerable and rear non-steerable wheels); this method of steering only allows us to turn in circles of about twice our length. The round hexapod (or octapod, for that matter) could be considered a "unicycle" model, and capable of rotating around its own center.

Warning!

Keep fingers clear of the legs when first powering up. The servos can move very fast and can pinch! Legged robots have also been known to "jump" off of tables when first powered on.

Power Up!

Assuming you've adjusted your legs using the PowerPod utility as described earlier, you're ready to try some initial moves. The control program powers up the legs in steps to avoid damage. With only two legs, this is not much of a chore, but should be observed carefully. If the legs are assembled incorrectly or other than a left and right pair (and on the proper sides), subsequent incorrect motions may cause the legs to strike each other and possibly cause damage. Be ready to remove servo power should this happen. Careful setup with PowerPod should reduce the likelihood of this.

Once powered up, you might try the "Circle" and "Triangle" commands. After that, a gingerly tug on the right joystick to take a step or two should cause the legs to make a few gait cycles and reward your efforts. After that – if all is well – a full-fledged walk on a kitchen or hardwood floor is recommended. Deep pile rugs will probably give Strider problems – he gets his feet caught!

Driving the Creepy Hybrid

To drive Strider like a car, pull back a little on the right joystick and then use the left joystick Y-axis (left-right) to steer. Or, you can translate (go in various directions without steering) using just the right joystick. A push forward on the joystick and Strider will back up.

Rotation controlled by the left joystick X-axis (horizontal) is called steering. It's not really steering because if you try to turn while both right joysticks are centered, the robot will rotate in place (pirouette).

The left joystick's Y-axis (vertical) changes the robot's ride height.

The Triangle button lowers the front end (legs) to the ground, then turns off all the servos; press the Triangle button again to turn the servos back on and reset the program to default settings.

The Circle button returns the robot to setup position (all servos at 1,500 mS) which is useful for checking leg setup.

You can control the speed of walking via left and right keys on the D-Pad. Up and down on the D-Pad increases or decreases the amount of leg lift.

The original AH3-R code running on a hexapod will, of course, have many more moves. The chassis can be tilted and rotated considerably, and some additional changes to the gait can be made. There are even "FLY" and "ATTACK" modes! Most of these moves are not possible with only two

Links to Resources

Legs, Servos, PS2, Battery, BB2, SSC-32, PowerPod
www.lynxmotion.com

Atom Basic, Atom Pro Basic, MBasic Pro Compiler
www.basicmicro.com

Bluetooth UBW
www.sparkfun.com/commerce/categories.php?cPath=16_115

SchmartBoard jumpers, Eight-bit PIC Module A
www.schmartboard.com

Hi-Tech PICC-18 Compiler
www.htsoft.com

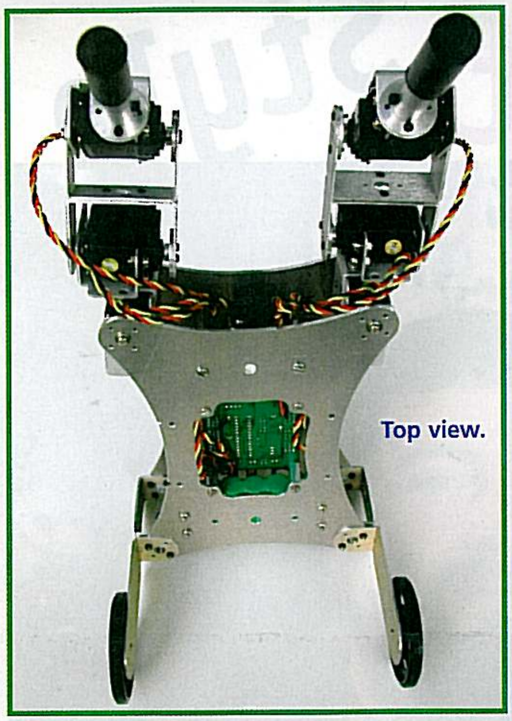
DM164120-3 PICkit 2 28-pin PIC Demo Board
www.microchipdirect.com

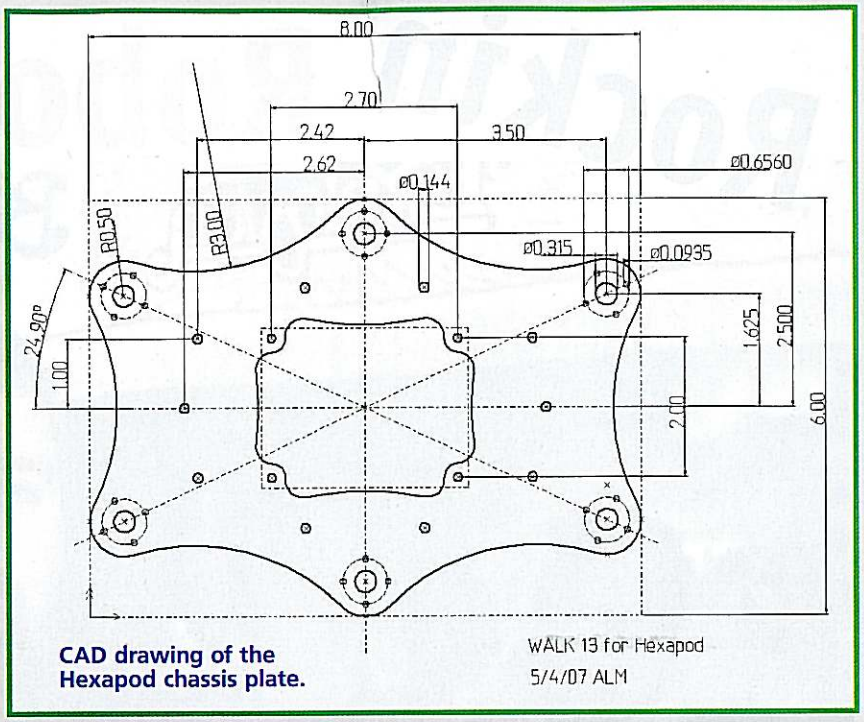
Microchip Datasheets, ICD 2, MPLAB
www.microchip.com

Parts, PICs
www.digikey.com

DT106 Development Board
www.dontronics-shop.com

Aluminum Stock
www.onlinemetals.com


Top view.



CAD drawing of the Hexapod chassis plate.

legs, but a lot has been learned, none the less!

Future Plans

The Creepy Hybrid can be thought of as an "introductory legged robot." With it, we've been able to learn a little about how to use legs on a robot. Having a little foresight, we started with either a quadruped or hexapod chassis, knowing that we can add additional legs later. As there is already code available for hexapods (AH3-R), this is a logical choice. For more of a challenge, new code for a quadruped can be written.

I'd also like to explore 2DOF legs on Strider. With fewer calcs to do (four vs. six servos), maybe just a single processor could run the entire bot!

Plus, with faster micros, it should be possible to combine the bot control and servo control program into a single program, and run six or even 12 servos on one processor board. Thus, a simpler (less expensive) bot could be built. **SV**

All the CAD drawings, as well as the .BAS files are available for download at **www.servomagazine.com**.

SOFTWARE Downloads:

Creepy2ps2.bas	Basic Atom 5.3 PS2 program
Creepy2com.bas	Basic Atom 5.3 COM (RS-232) control
Creepyp2.bas	Basic Atom Pro 8.0.1.8 PS2 or RS-232 control
Creepy2Mb.bas	Mbasic Pro 5.2 program PS2

Creepy Hybrid Parts List

- ❒ 3DOF leg pair
- ❒ Hitec HS475 servos (six)
- ❒ Hitec servo horns (six)
- ❒ 6V battery pack
- ❒ 9V battery
- ❒ 9V battery connector
- ❒ SPST toggle switches (one or two)
- ❒ 6V battery connector
- ❒ Quadruped chassis plate pair
- ❒ Chassis spacer pair
- ❒ Leg bracket pair
- ❒ Battery bracket
- ❒ Wheel pair
- ❒ PS2 plate
- ❒ PS2 wireless joystick
- ❒ Blue Smirf (alternate)
- ❒ BB2 (Bot Board 2)
- ❒ 28-pin PIC Demo board DS41301A (Microchip) (alternate)
- ❒ Eight-bit A-PIC board (SchmartBoard) (alternate)
- ❒ UBW Sparkfun (alternate)
- ❒ DT106 DonTronics (alternate)
- ❒ Basic Atom 28 module
- ❒ Basic Atom Pro 28 module (alternate)
- ❒ MBasic Pro from Basic Micro (alternate)
- ❒ SSC-32 Servo board
- ❒ SSC-12 (alternate)

HARDWARE

- ❒ 6-32 hex aluminum spacers (two or four)
- ❒ 4-40 hex aluminum spacers (eight or 12)
- ❒ 2-56 screws
- ❒ 6-32 screws
- ❒ 4-40 screws
- ❒ 4-40 nuts
- ❒ 6' DB9 serial cable M-F
- ❒ .025 jumpers 10