Melanie v2.0
Robust hexapod robot of 3 dof[1]/leg for rough terrain

Alejandro Alonso-Puig – mundobot.com

April 2004

[1] DOF: Degrees of Freedom

Introduction

Melanie is an hexapod robot of 3 degrees of freedom by leg, that by the novel design of legs which it has, can transport several kilograms on its body without excessive power overload.

Some basic characteristics that define Melanie v2.0 are the following ones:

Mechanics

- Size (cm): 33 x 31 x 20
- Weight: 2.1 Kg
- Structure of PVC and Aluminum
- 6 Legs. 3 degrees of freedom by leg.
- Design of legs optimized to diminish consumption and to maximize power
- Driven by radio control servos (Twelve of 3kgcm and six of 5kgcm)

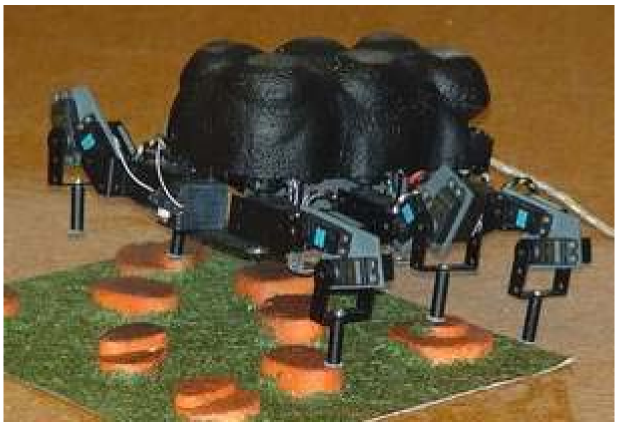
Electronics

- Servos control circuit with speed management, controlled via serial port.
- I/O circuit, with 28 A/D conversion ports and 8 digital I/O. All chips are controlled via I^2C bus. Measurement of the position of the 18 servos taking the signal of their potentiometers. Measurement of the consumption of energy of the servos by measuring the voltage drop in resistances of ½ ohm.
- Battery of 6v NiMH 3300mAh for the servos
- Battery of 9v NiCd for the control electronics
- Distance measurement infrared sensor located at the front and able to be oriented

Programming

- Control program in PC developed in Visual C++ 6.0
- Movement sequence recording via direct programming
- Movements generation by displacement of three simultaneous waves and use of inverse kinematics algorithms.
- Detection of obstacles on the ground by detection of the variation in the current consumption and adaptation of the legs and trajectories to resolve it.

In this work there will be mention to all the mechanical, electronic and programming aspects, as well as conclusions of the experience.



Mechanics

Body

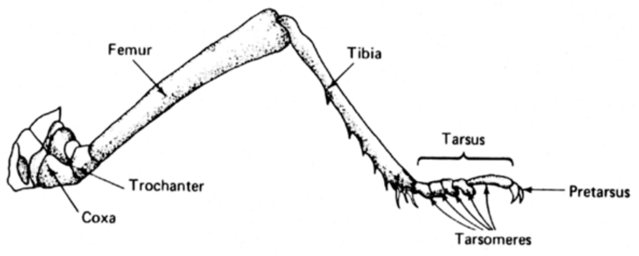
The trunk of Melanie is formed by a central chassis made out of two aluminum preperforated plates of 2mm, on which they are attached the six legs it has.

It has been procured the maximum symmetry possible in the body so that the center of gravity is the most centered possible and thus improves the physical behavior of the robot and distributes in a suitable way the effort throughout the legs.

The robot has been equipped with a shell of light porespan to protect the electronics and give it a more "animal" aspect.

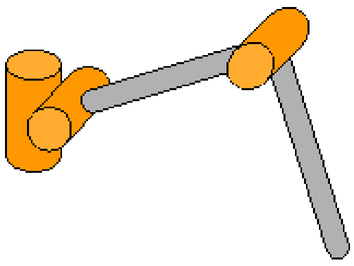
Legs

The legs are made of aluminum of 2 mm and PVC of 4 mm. The structure of them differs from the habitual distribution of joints of the leg of an insect.



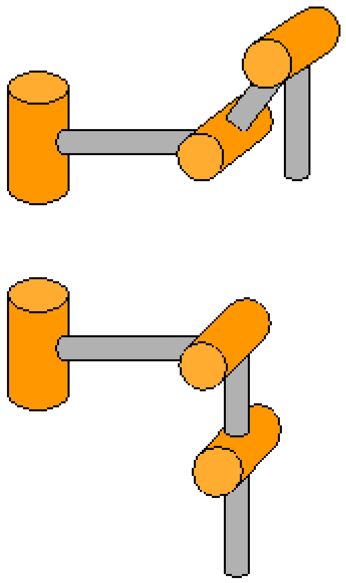
The Coxo-trochanter and trochanter-femur joints of an insect practically are fused, and they take care of the movements of femur in the space. The femur- tibia joint takes care of the movements of the tibia in the plane vertical to the ground.

The mechanical implementation of this structure usually is limited to three degrees of freedom and traditionally it implies the necessity to have an important torque in the servos due to the length of the femur. As longer it is, greater torque will be necessary in the motors. In addition, once carried out the elevation of the body, a great amount of energy is usually needed to maintain it in this position.



Traditional legs structure

The structure of legs of Melanie varies of the common structure shown in the above figure, modifying the position of the joints as it is shown the following figure:



Legs structure of Melanie

With this structure two important advantages are obtained with respect to the traditional one shown previously:

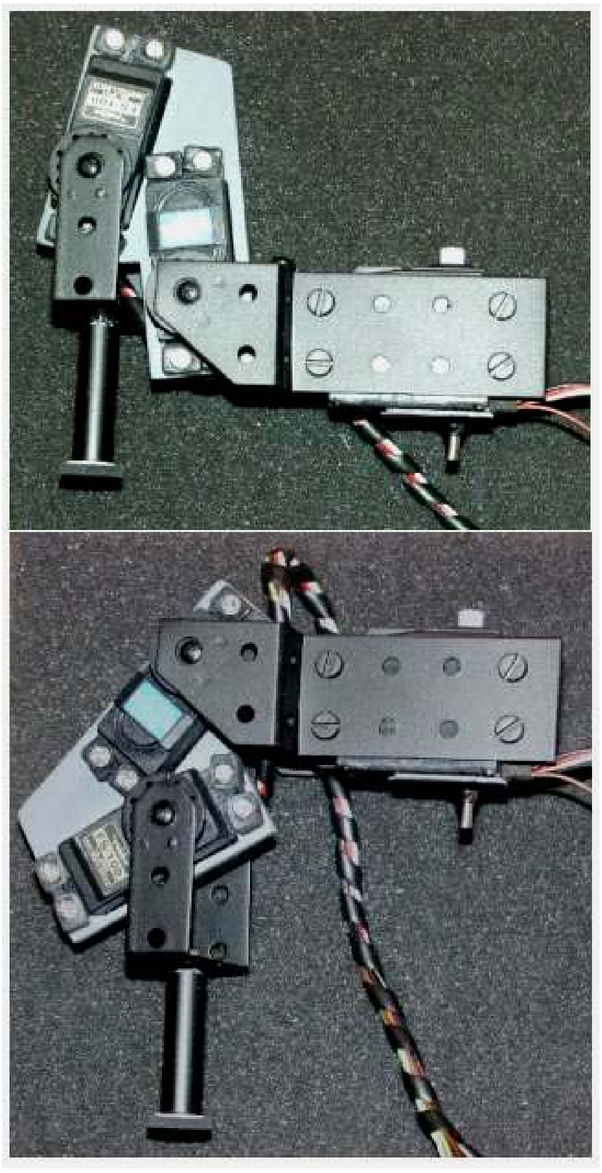
1. Once elevated the body they are aligned the joints that allow the displacement in the vertical plane of the legs, which implies a

drastic reduction of the energy consumption due to a reduction on the load in the motors of these joints. The final implementation has shown that in raised state the provision of energy to the motors can be cut and the robot will remain elevated, even with an additional charge (test with 1kg on its body). It is for that reason that will also be able to carry important additional charges in this position.

2. When moving the joint of the trochanter-femur zone to the femur-tibia zone, the mass center of the motors is moved towards the tibia, which releases of weight the trunk increasing the capacity to support a greater weight over it. The tests made have allowed verifying that each leg is able to elevate a weight of 800 grams in addition to the own weight of the leg. Reason why with 6 legs a trunk of 4,800 grams would be elevated. In the test bench the robot, of 2,1kg has risen with an additional charge of 4Kg.

The servo used are of 3Kgcm of torque for coxa and tibia joint and 5kgcm for the intermediate joint.

All the servos used have been mounted with false back shaft to assure a suitable attachment that avoids the torsions on the servo shaft.

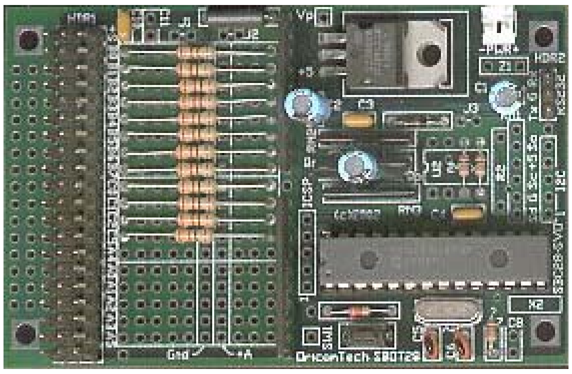

Leg structure of Melanie

Electronics

Melanie v2.0 is based fundamentally on two cards: A commercial circuit of management of servos controlled by serial port RS232C of low cost and a I/O card with 28 A/D conversion ports and 8 digital ports.

Servos control card

Card MSCC20 from Oricom Technologies (www.oricomtech.com) allows the control of up to 20 servos and in addition it allows to control the speed of each servo independently with just a single command.



For additional information on this card it could be accessed its Internet web page, in which it will be found explanation of the usable Set of commands:
http://www.oricomtech.com/svo-cc.htm

The card is fed by a battery of 9v, being the servos fed by a battery of 6v NiMH 3.300 mAh

The card is connected via three wires cable to a PC serial port (RS232C) from which the control commands are sent.

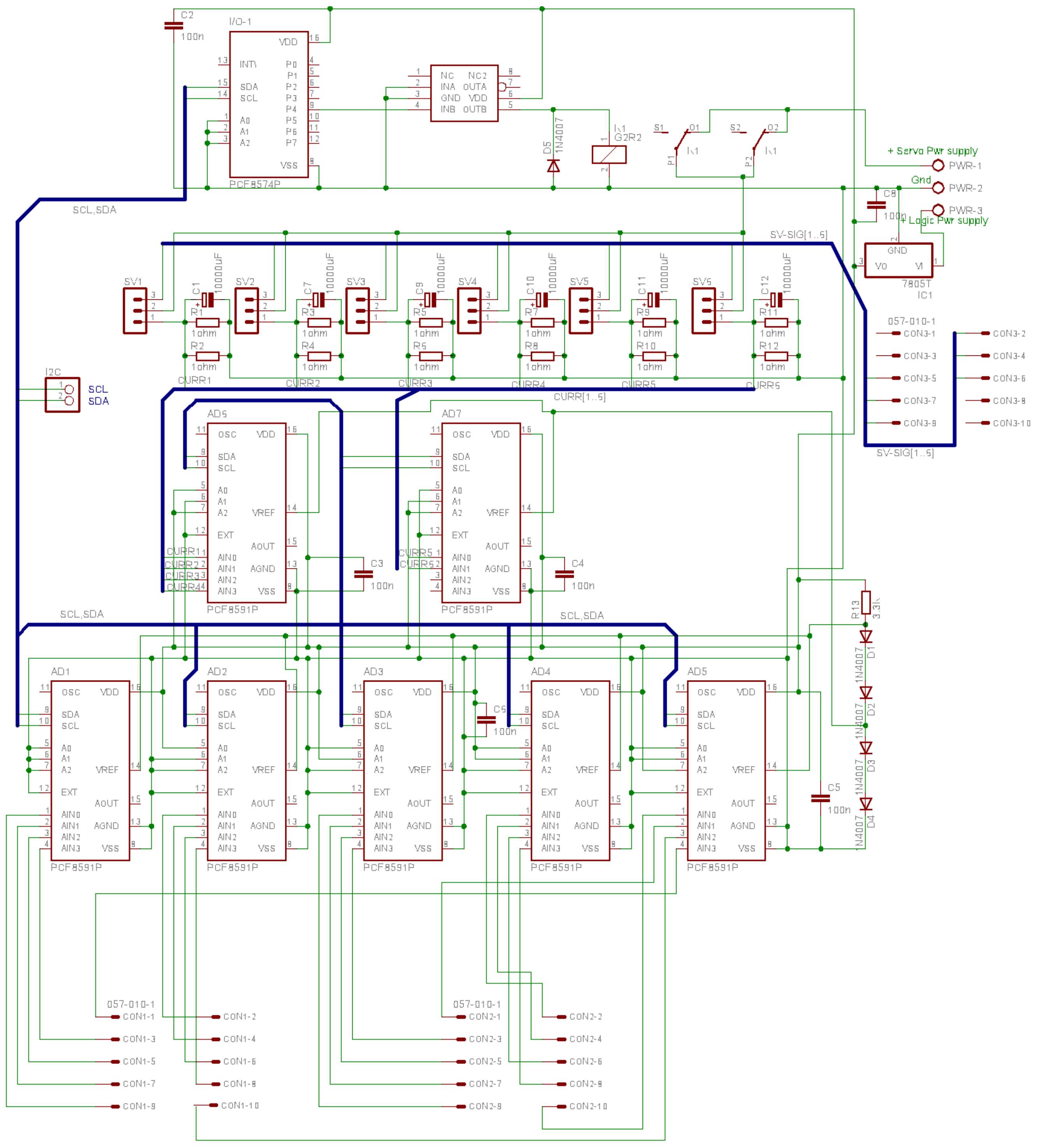
I/O card

I/O Card is compound basically of 7 Chips PCF8591 (quadruple A/D of 8 bits controlled by I^2C) and a chip PCF8574 (port expander of 8 bits by I^2C)

The 28 A/D channels are used for the measurement of the position of the 18 servos taking the signal from their potentiometers, for the measurement of the energy consumption of the 5Kgcm servos, measuring the fall of tension in resistance of ½ ohm and for the measurement of batteries voltage. As reference voltage it has been used 2,4v for the chips which measure the position of the servos and 1,4v for the chips which measure the consumption of the servos and the voltage of the batteries. For more details see the circuit diagram that is in the following figure.

Of the 8 digital channels only it is used one (channel 4). This channel activates the driver TC4425 that feeds a relay. This relay is used to cut the tension that goes to the 5Kgcm servos. This is used to reset servos since they are digital servos, that even maintain the position if we cut PWM signal. Causing a short fall of the tension in servo after stopping PWM signal they remain deactivated. Normal servos do not require this current cut as it is enough to stop the PWM signal in order to stop them. This deactivation is used for two aims: To deactivate by program the servos after a period of operation and to deactivate selectively servos to be able to move them manually and thus to make the direct programming.

Since I/O card is controlled by I^{2}C bus, it is used an interface card between the robot and the external computer from which it is controlled. This card is the Velleman K8000. It is a commercial interface from parallel port to bus I^{2}C. Additional information could be obtained at: www.velleman.be or consulting the work of the author on the compatible KI^{2}C card of low cost in http://www.mundobot.com/tecnica/ki2c/spki2c.htm



Following it is the relation of elements connected to connectors CON1, CON2 y CON3:

CON1-1	Nothing	CON1-2	Potentiometer Servo 12
CON1-3	Potentiometer Servo 11	CON1-4	Potentiometer Servo 13
CON1-5	Potentiometer Servo 6	CON1-6	Potentiometer Servo 5
CON1-7	Potentiometer Servo 4	CON1-8	Potentiometer Servo 3
CON1-9	Potentiometer Servo 2	CON1-10	Potentiometer Servo 1

CON2-1	Potentiometer Servo 14	CON2-2	Potentiometer Servo 15
CON2-3	Potentiometer Servo 16	CON2-4	Potentiometer Servo 7
CON2-5	Potentiometer Servo 8	CON2-6	Potentiometer Servo 9
CON2-7	Potentiometer Servo 18	CON2-8	Potentiometer Servo 17
CON2-9	Potentiometer Servo 19	CON2-10	Nothing

CON3-1	Nothing	CON3-2	PWM signal Servo 2
CON3-3	Nothing	CON3-4	PWM signal Servo 12
CON3-5	PWM signal Servo 5	CON3-6	PWM signal Servo 15
CON3-7	PWM signal Servo 8	CON3-8	Nothing
CON3-9	PWM signal Servo 18	CON3-10	Nothing

On the other hand, to the connectors SV1,..SV6 they are connected the following digital servos (It would work the same with normal analogical servos):

SV1	Servo 2	SV2	Servo 12
SV3	Servo 15	SV4	Servo 18
SV5	Servo 8	SV6	Servo 5

I^2C addresses of the different chips are hardware set to the following decimal values:

AD1	144	AD2	146
AD3	148	AD4	150
AD5	152	AD6	154
AD7	156	I/O-1	64

Sensors

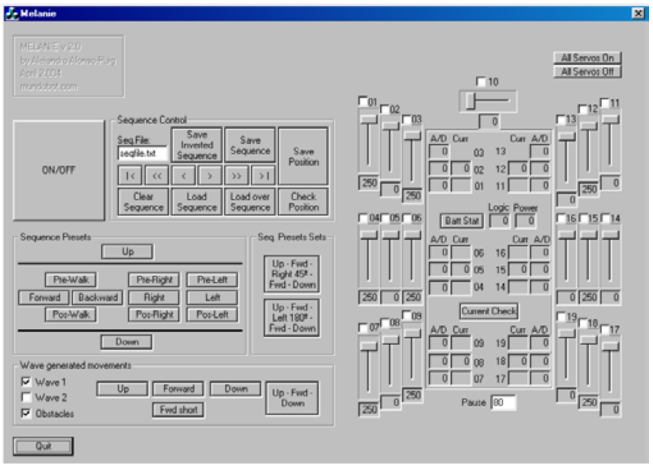
The robot has a GP2D12 infrared distance sensor in the frontal part, but in the present version it is not used.

Software

For the control of the robot it has been developed the application "Melanie v2.0" in Visual C++ 6.0. This application is located in an external computer (Pentium II - 233Mhz, 64Mb ram, Windows 98). It sends the suitable commands to the servo control card of the robot via serial port and to the I/O control card by bus I^2C. This control by bus I^2C is performed via the K8000 or KI^2C interface card connected to the parallel port that has been mentioned in the section of Electronics. The control of this card is carried out by specific libraries. For more details, see the work of the author in which its use is specified widely: http://mundobot.com/tecnica/ecoi2c/specoi2c.htm

Use of the application

The interface with the user consists basically of the following dialogue box:



The dialogue box shows a series of controls that allow controlling the robot in a manual way and generating sequences of movements that can be recorded and later reproduced.

In order to activate the robot the button ON/OFF should be pressed, that will activate the legs maintaining them where they are.

Recording/Reproduction of sequences

The recording of sequences can be carried out by two ways:

- Moving the servos by means of sliding controls and recording the positions.
- Moving manually the legs and recording the positions by means of direct programming

Sliding controls

The right part shows the positioning values of the servos. Moving each slider control a servo will move.

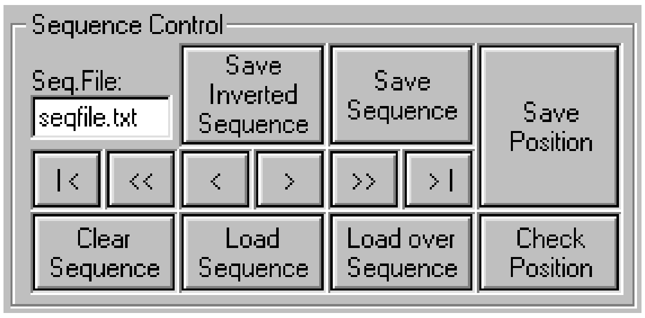
A sequence could be recorded by positioning servos as desired so that the robot has a determined position, establishing a value of pause (field underneath the scheme of the robot), that will determine the time that should happen between the position that previously it had and the new one requested. Depending on this value, that will come measured in second hundredth, a greater or smaller speed will be applied to the servos.

Once selected position and pause, would press the button "*Save Position*" of the "*Sequence Control*" area, that will store the new position in dynamic memory. Following this way it is possible to generate a complete sequence of movements. Putting a name of file in the field "*Seq.File*" and pressing "*Save Sequence*" a file with the generated sequence will be recorded in hard disk.

Direct programming

In order to carry out this programming it would be necessary to deactivate the desired servos to be able to move them manually. This is carried out unmarking check-box of the wished servos in the right part of the screen. After this they could be moved manually to the wished position.

Once selected position and pause, we should press the button "Check Position", that will verify the position of legs servos and then "Save Position" of the area "Sequence Control", that will store the new position in dynamic memory. In this way we could store a complete sequence of movements. Putting a name of file in the field "Seq.File" and pressing "Save Sequence" a file with the generated sequence will be recorded in hard disk.



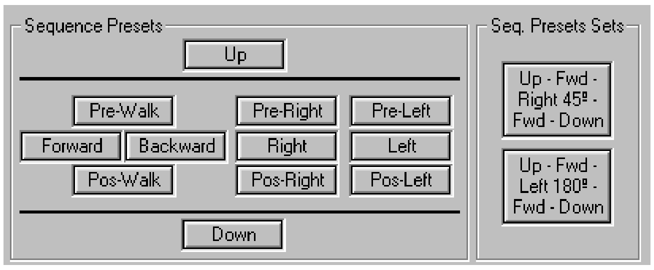
The other controls of the area "*Sequence Control*" carry out the following functions:

- **Save Inverted Sequence**: It records the sequence but in inverse way. Very useful when a movement is identical to another existing one but in the inverse way, like for example walking forwards and backwards
- **Clear Sequence**: Deletes the sequence that is in memory
- **Load Sequence**: Load in memory a prerecorded sequence. The name of the sequence file should be specified in the field "Seq.File"
- **Load over Sequence**: Load in memory a sequence from the present position of the pointer, that is to say, if we have a sequence in memory and we move to its end with the buttons we will explain later, we can press "*Load Over Sequence*" to load a sequence on the existing one in memory from the present position. This way we could connect sequences.
- "|<" Allows to be placed at the beginning of the existing sequence in memory
- ">|" Allows to be placed at the end of the sequence
- ">" Allows to advance a position in the sequence loaded in memory. This allows going step by step in the sequence for debugging.
- "<" inverse case of previous one
- ">>" Allows to reproduce the sequence from the present position to the end without interruption
- "<<" Just as the previous one but in inverse way.

With the previous controls it is possible to debug positions with advancing step by step, to correct the position wished with the slider controls and to press "*Save Position*".

It is necessary to consider that pressing this button overwrites the position of all servos in the present place of the sequence and the sequence pointer moves to the following position being prepared to overwrite it. If it is only wanted to overwrite a position, it will be done as indicated and then "<" and ">" should be clicked so the robot is forced to be placed according to the following position.

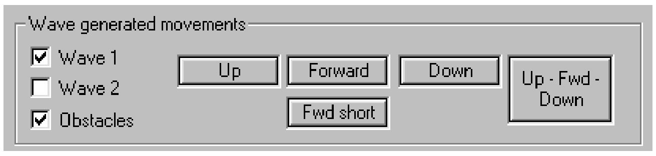
There is a series of prerecorded basic sequences that will allow moving the robot. These sequences are grouped under the set "*Sequence Presets*". They allow raising the robot, making it walk, turn...



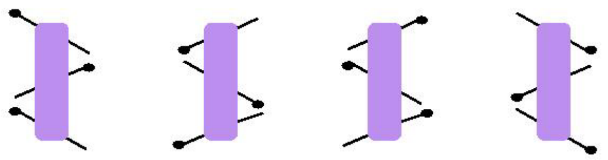
Additionally there is a group of sets of sequences prerecorded under the set "*Sequence Preset Sets*" that allow to execute several simple sequences of the indicated previously, to carry out a concrete task, demonstration, etc, like rising, walking, turning around and lying down.

Movements by displacement of waves over rough terrain

There is another area at the screen (*wave generated movements*) that allows controlling the robot not by playing recorded sequences, but by means of a system called waves displacement. The robot generate the movements based on triple overlapped waves.



The program is prepared for two kind of waves: Wave 1 and Wave 2. The first one makes the robot walk in the typical tripod mode, it means, walking with alternate triangles of legs so they are always three legs on the floor.



The second mode move the legs in a way so there are always five legs on the floor.

Also it is possible to make the robot sensible or not to obstacles on the floor by using the checkbox "*Obstacles*". In case this mode is used, the robot will be able to walk in rough terrain, detecting obstacles like stones, walking over them while maintaining the body horizontal to the floor.

Application design

The application has been generated in Visual C++ 6.0. It uses a library developed by P.J. Naughter for the control of the serial port with which the robot will communicate.

For more information on using the mentioned library the work of the existing author can be consulted in the direction:

http://www.codeproject.com/system/cserialport.asp

Additionally it uses a library for the control of the parallel-I^2C interface.

For more information on this library the following report could be reviewed:
http://mundobot.com/tecnica/ki2c/spki2c.htm

The interface with the user consists basically of a dialogue box associated to a *CMelanieDlg* class within which there are all the functions and variables used for the control of the robot.

The source code of the application contains numerous commentaries and can be acceded in the direction:
http://mundobot.com/projects/melanie/melanieprg.zip

In any case next we will explain the general structure of the application.

Recording/Reproduction of sequences

As it was explained previously, it is possible to store sequences of movements. In order to carry out this, pointers to structures stored in dynamic memory are used.

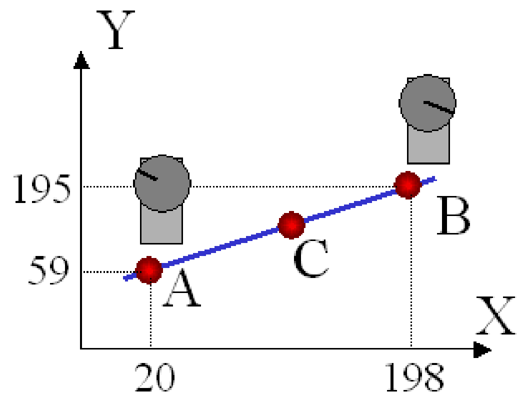
```
struct  stPosition
{
        unsigned char           cServoPos[21],
                                cPause;
        struct stPosition       *nextpos,
                                *prevpos;
};
```

In each element of this structure we can store a complete position, that includes the 20 positions of servos and delay that must have between the previous position and this one. Additionally they are stored pointers both to previous and next elements to have a linked list in which we could move freely forward and backward.

The storage of the mentioned structure is performed when pressing the button "Save Position" which calls the function "*OnSavepos()*". This function takes the servos position values and store them in the structure.

If direct programming is being performed, before pressing the button “Save Position” it is needed to press the button “Check Position” which will call the function “*OnAD()*” and this to “*ChkPos(unsigned char cPos[20],unsigned char cAD[20])*” which takes the servos position values by reading the analog values of the servos internal potentiometers. This value is converted to a byte value that if sent with the positioning command to the servo control card would move the servo to the position it is right now. To get this, the program have a table (*byte bCal[21][4]*) which stores four values for each servo that correspond to two different positions of the servos (a couple of values for each of the two positions). Each couple have interrelated values. The first one is the value sent to the servo control card to move the servo to that position. The second one is the analog measurement of the position. Having this for two different positions A and B (see figure), it is possible to obtain the rest of the values of the line that cross both points. Lets see this with an example:

Servo number 1 has been positioned with the slider controls to the value 20 (showed in the dialog box), then potentiometer value has been measured (A/D conversion). The value obtained has been 59. The same thing has been done locating the servo to position 198 obtaining the A/D value 195. As there is a linear relation between the positioning value and the A/D value, using the equation of a line it is possible to



calculate the positioning value of a servo based on the A/D value measured. Then, if for example the value read from the potentiometer is, lets say 90, we will obtain the positioning value as showed here:

Position = X axis
AD = Y axis

Two points of the line = (20,59) y (198,195)

x-b = m(y-a) {Cartesian equation of a line}

Using the points: (x=20 , y=59, a=198, b=195)

59-195= m(20 - 198)

m = (59-195)/(20 - 198)

then :

y-195 = m(x - 198) = (59-195)/(20-198)*(x-198)

AD-195 = (59-195)/(20-198)*(POS-198)

POS = 198+(AD-195)/((59-195)/(20-198))

Therefore, for AD=90, POS=60’6

It has been taken therefore the values of A y B empirically for all servos and charged them in the matrix *bCal* as part of the program code. In case of changing the servos, it would be necessary to calibrate them and to update the values of this matrix again.

The existing sequence in memory can be saved to a file using the function "*OnSaveseq()*"and load a file in memory using the function "*LoadSequence(CString FileName)*" that is called from other several functions.

Only could be reproduced sequences that are loaded in memory, so the buttons of prerecorded sequences simply will call to functions that load in memory a prerecorded sequence and executes it.

The execution of each step of the sequence is carried out by calling the function "*SetGlobalPos()*", that will move servo and slider controls according to the values of the present pointer.

The delay (Pause) as it was mentioned, will indicate the time that must pass between a position and another one. So that died moments do not exist, function " *cSpeed(unsigned char cServo)*" will calculate the speed that should apply to each servo depending on the mentioned pause and the run to carry out. It is based on specifications from the manufacturer of the servo control card that indicates that the speed parameter has to be an integer value between 1 and 255 that can be calculated according to the following formula:

$$V = \frac{8(Pos1 - Pos0)}{T}$$

where T is the time in second hundredth that is desired to take the servo to go from a position to another one, being *Pos1* the position of greater value and *Pos0* the position of lower value. The values of *Pos1* and *Pos0,* according to specifications of the manufacturer of the servo control card, have to be integer values between 0 and 250.

The commands for the servo control card are mainly sent by using the function "*SendCommand(CString sCommand)*".

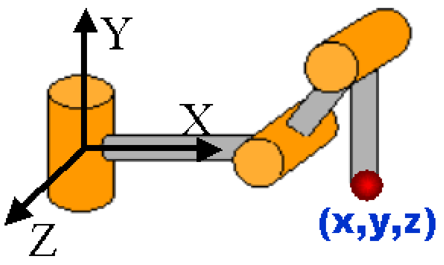
Although the servo control card admits great amount of commands, we will use only the following ones:

- **SO**: It turn all servos on
- **SF**: turn all servos off
- **SX1F9D**: Set excursion limits of all servos to 496µsg and 2.512µsg
- **SEss**: It activates the generation of pulses for servo "ss", value that should be in hexadecimal 00 - 13h (19d).
- **Mssddvv**: It moves servo "ss" (hexadecimal value) to the 8 bits position "dd" (hexadecimal value between 00 and FAh [250d]) at speed "vv" (hexadecimal value between 1 and FFh [255d])

For more information on the commands available and their use, go to the documentation of the manufacturer:
http://www.oricomtech.com/sub2/msc2-cmd.htm

Movement by displacement of waves

As told before, the robot can move generating himself the positions instead of using prerecorded positions. The way in which it makes this is by using three waves whose values will vary in the time and will define the values of coordinates (x, y, z) in the space for the end of each leg.



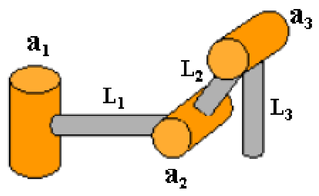
Having these values, a geometric algorithm of inverse kinematics will be used and the value of positioning of each servo will be obtained so that the end of the leg is placed in the indicated coordinates.

Inverse Kinematics

The program do all the inverse kinematics calculations needed by using the following function:

Int InvKinematic(double value[2][3], double result[3])

This function return 0 if it has not happened any error in the calculation (for example it could occur that the given coordinate is outside the reachable space of the leg).



The input data for the function will be loaded in the matrix *value[2][3]* so that the first row of data will contain the lengths in millimeters of the three parts of the leg (L_1, L_2, L_3) and the second row will contain the coordinates (x, y, z) that is desired that it has the end of the leg respect to his origin of reference. As result the function loads in the matrix *result[3]* the values in degrees of the angles the joints should have (a_1, a_2, a_3)

The algorithm used for the calculation of the inverse kinematics use the geometric method.

Later the values of angles (in degrees) of each servo can be converted to positioning values accepted by the servo control card by using the function *unsigned char cConvDegToPos(char cServo, double Deg)*, where *cServo* is the number of the servo we are referring to and *Deg* is the value in degrees we desire this joint to have. The value returned by the function is the value to be sent to the servo control card.

In order to obtain this, values of calibration for each servo are used that have been taken empirically and are loaded at the beginning of the program in matrix cPosConv[20][2].

The first index indicates the servo and the second index indicates a position of 0 degrees and one of 45 degrees. For example, for leg 1 the following values are loaded:

cPosConv[1][0]=203;	*cPosConv[1][1]=140;*
cPosConv[2][0]= 77;	*cPosConv[2][1]=136;*
cPosConv[3][0]=103;	*cPosConv[3][1]=160;*

The first row includes the values for servo 1. 203 is the value that requires the servo control card for positioning the servo so that the angle of the joint is 0 degrees and 140 is the value that requires the servo control card for positioning the servo so that the angle of the joint is 45 degrees.

With these values and, given to the linear relation between value of positioning and angle in degrees of the joint, we can always obtain the value of positioning of the servo based on the value of the angle of the joint by using the equation of a line.

It exists an inverse function to this called *double cConvPosToDeg(char cServo, unsigned char cPos)*

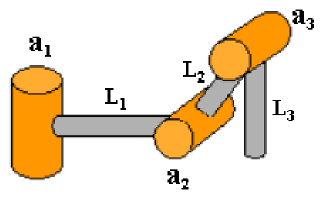
Direct Kinematics

The program also carries out in some cases direct kinematics calculations. For it it uses the function:

int FwdKinematic (double value[2][3], double result[3])

This function returns 0 if it has not happened any error in the calculation.

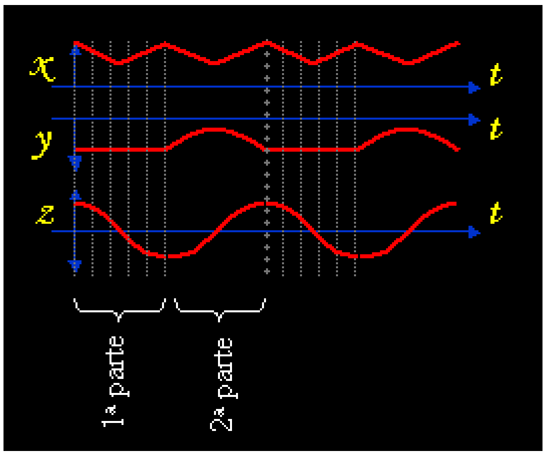
The used algorithm is also of geometric type.



The input data for the function will be loaded in the matrix *value[2][3]* so the first row of data will contain the lengths in millimeters of the three parts of the leg (L_1, L_2, L_3) and the second row will contain the angles in degrees of the joints (a_1, a_2, a_3). As result the function will load in the matrix *result[3]* the values of the coordinates (x, y, z) that the end of the leg will have respect to its origin of reference.

Generation of waves

As told before, they are three waves those that define the coordinates that each leg will have at every moment.



Coordinate X is calculated by means of a triangular wave, the coordinate Y by means of a half-wave and coordinate Z by means of a complete sine wave. The frequency of the triangular wave has to be double of the other two so that the leg moves suitably.

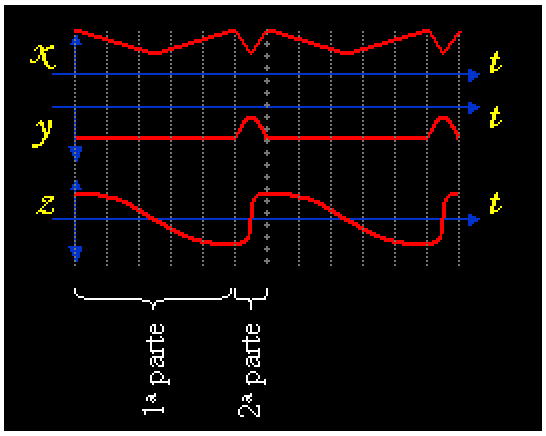
Varying the phase angle between the waves of the different legs and the relation between the length from the first (*1ª Parte*) and second part (*2ª Parte*) of the cycle of the wave we will obtained different "gaits".

The first part of the cycle corresponds to the moment at which the end of the leg is touching the ground and moving backwards (body advancing), whereas the second part of the cycle corresponds to the elevation and displacement ahead in arc of the leg (positioning for a new step). Thus for example in the previous figure, the first and second part of the cycle is of the same length (in time), so it will take the same time in moving backwards the leg that in positioning it again ahead. If we used this type of waves
in all the legs, but we apply a phase angle of 180 degrees to three of the legs, we will obtain a tripod "gait", so the body always will be supported by three legs.

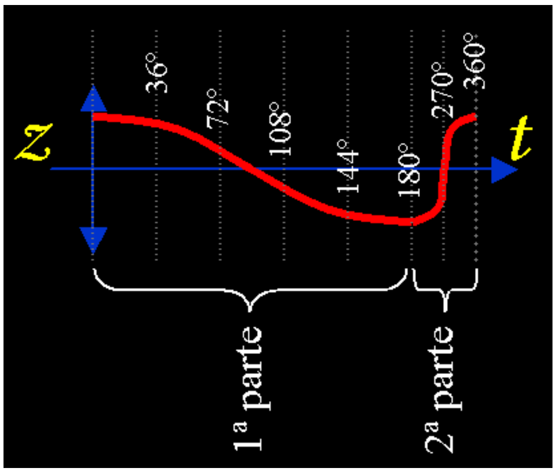
If for example we use waves of the previous type, but with the 2nd part of the cycle of a length 1/5 of the one of the 1st part the leg will rise and place ahead in a determined time, whereas it will move backwards 5 times slower.

If we combine this situation with a phase angle of 36º (180/5) between the waves of the different legs we obtain a "gait" according to which always there will be 5 legs on the ground and one in the air, but the body will be moving ahead with a continuous speed.

This is not accidental. If we wish that the body moves at continuous speed, it always has to have a sufficient number of legs in contact with the ground being moved backwards, not being anyone stopped. If we want that always there are 5 legs in contact with the ground, it will be necessary to raise the legs one by one, so that the 5 legs should have enough time to rise and be placed at the front in consecutive way in the time that the sixth leg moves backwards. So the time for the elevation and displacement of a leg in the air to the front will have to be 1/5 of the time that takes in moving the leg backwards. This is indeed the length in time of 2nd part of the cycle. On the other hand, as it is desired that only one leg is in the air at each moment (5 legs in the ground), the waves of the legs will have to be moved 1/5 of 180º, since 180º is the point in which the wave change from the 1st to the 2nd part of the cycle.



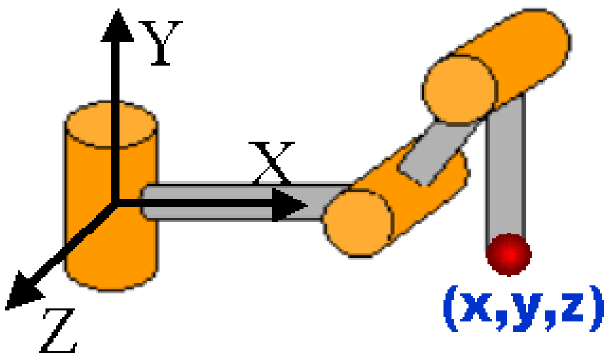
It is important in this point to stand out that the waves used are those indicated at the beginning of this section, so triangular, half-wave and complete sine wave and that the variation of the length in time of 1st or 2nd part of the cycle is just a kind of "stretching" of the time and therefore the values in degrees also "stretch". Lets see what we are talking about for the previous case of the sine wave (Cosenoidal) that generates the value of coordinate Z:



In the program it is the function *void OnWaveFwd()* that will make the robot walk forward according to the method of displacement of waves explained.

Following the same method of displacement of waves the *MoveLeg* function has been generated, that moves the end of one or several legs from a point in the space to another one, either with a straight line displacement between both points or making an arched movement of rise, displacement and go down.

It must be remembered that the coordinates handled for each leg are always relative to the point of coordinates of each leg, as showed in the following figure:



Obstacles detection

The program is prepared for the detection of obstacles on the ground and to adapt the "gait" to allow the robot to walk over these obstacles.

The way it is done is by measuring the consumption of current by the servos at the moment the leg is going down to put its end on the ground.

The measurement of the current consumption is done by the function *void ChkCurr()* that sends the needed commands to carry out the reading of A/D converters by bus I^2C and load in the *Current[6]* matrix the values of measurement of current for the 6 servos that more force have to make to support the body.

If the consumption measurement is over a given threshold, the leg stops, its position is obtained and the length of the second part of the cycle of the waves of this leg is modified temporarily. Also a small phase angle is added to assure that they move with coordination with the rest of legs. The values of phase and length of the second part of the cycle of this leg return to its normal values in the following cycle, with no disruption in the continuity of the waves during the transition.

Bibliography and interesting links

- Agricultural Entomology. Daniel G. Pesante Armstrong. Chapter 7: http://www.uprm.edu/wciag/anscience/dpesante/4008/cap-7.PDF
- Puchobot. Quadruped Robot. Andrés Prieto Moreno: http://www.iearobotics.com/personal/andres/proyectos/pucho/pucho.html
- Cube Reloaded. Worm robot. Juan González: http://www.iearobotics.com/personal/juan/doctorado/cube-reloaded/
- Direct & inverse kinematics course (Univ.Guadalajara-Mexico):: http://proton.ucting.udg.mx/materias/robotica/r166/r78/r78.htm
- Legs. General design considerations: http://www.frasco.demon.co.uk/rodney/mechan/design.htm
- The Walking Machines Catalogue: http://www.walking-machines.org/
- Silo4. Quadruped Robot developed by CSIC. Spain: http://www.iai.csic.es/users/silo4/
- An introduction to robot kinematics: http://www.generalrobotics.org/ppp/Kinematics_final.ppt
- MSCC20. credit card-sized control computers for driving up to 20 R/C servos. Oricom Technologies: http://www.oricomtech.com/svo-cc.htm
- CSerialPort v1.03 - Serial Port Wrapper. PJ Naugter: http://www.codeproject.com/system/cserialport.asp