

Melanie v1.0

Robust hexapod robot of 3 dof¹/leg

Alejandro Alonso-Puig – mundobot.com
March 2004

Introduction.....	2
Mechanics	3
Body.....	3
Legs.....	3
Electronics.....	5
Software	6
Use of the application	6
Application design	7
Conclusions and future investigations	9
Bibliografy and interesting links.....	10

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.

The version 1.0 that is in this technical report refers the simplest version of the implementation of the robot, in which the robot is connected to an external PC by serial port (RS232C) for its control.



Some basic characteristics that define Melanie v1.0 are the following ones:

- Measures (cm): 33 x 31 x 12
- Weight: 2,2Kg
- Six legs
- Three degrees of freedom
- Driven by radio control servos (Twelve of 3kgcm and six of 5kgcm)
- "smoothed" Control of servos by using a driver with speed management.
- Communication with master PC by serial port RS232C
- Distance measurement infrared sensor located at the front and able to be oriented
- Structure of PVC and Aluminum
- Battery of 6v NiMH 3300mAh for the servos
- Battery of 9v NiMH for the control electronics
- Control program in PC developed in Visual C++ 6.0

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

Mechanics

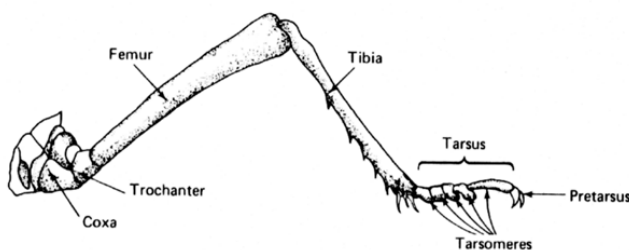
Body

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.

Legs

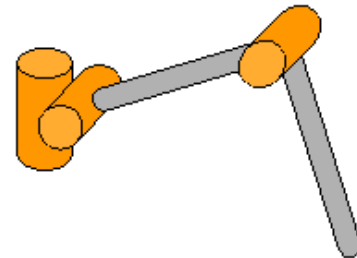
The legs are made of aluminum of 2 mm and PVC of 4 mm. The structure of them differs from the habitual distribution of joints of the leg of an insect.



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.

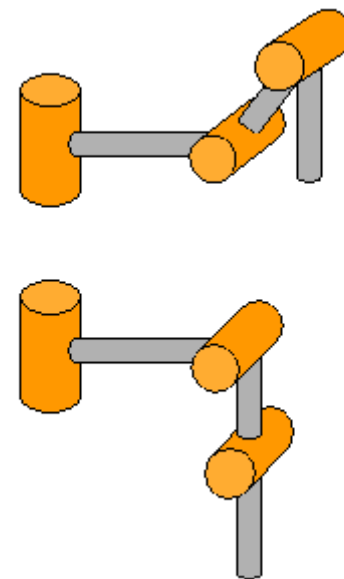
The mechanical implementation of this structure usually is limited to three degrees of freedom and traditionally it implies the necessity to have an important torque in the servos due to the length of the femur. As longer it is, greater torque will be necessary in the motors. In addition, once carried 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:

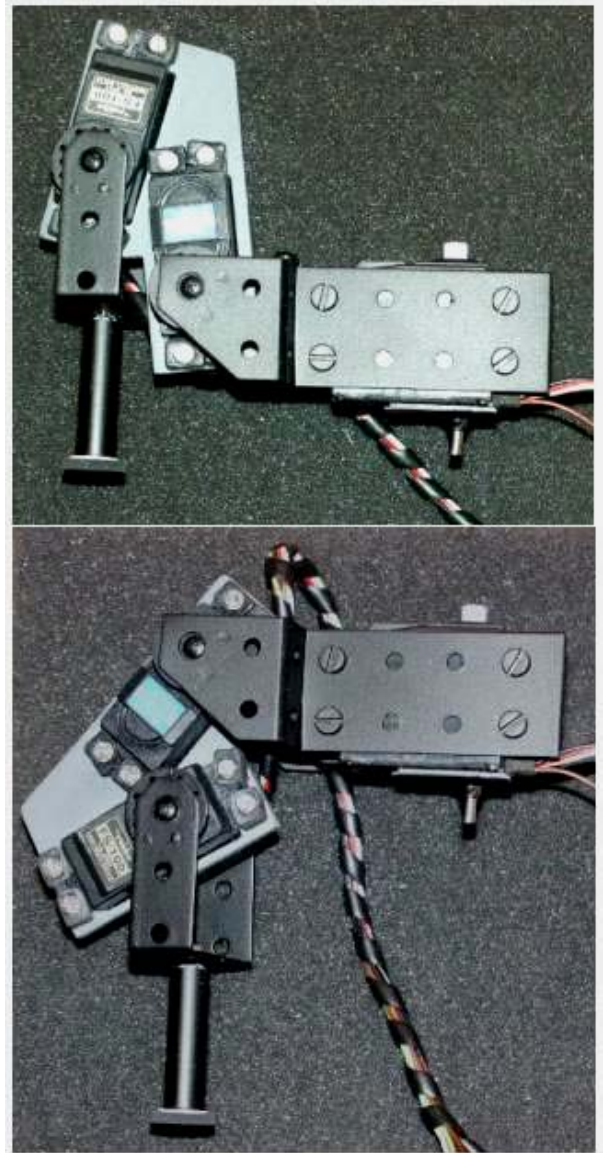
1. Once elevated the body they are aligned the joints that allow the displacement in the vertical plane of the legs, which implies a drastic reduction of the energy consumption due to a reduction on the load in the motors of

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. Also it will 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.

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.

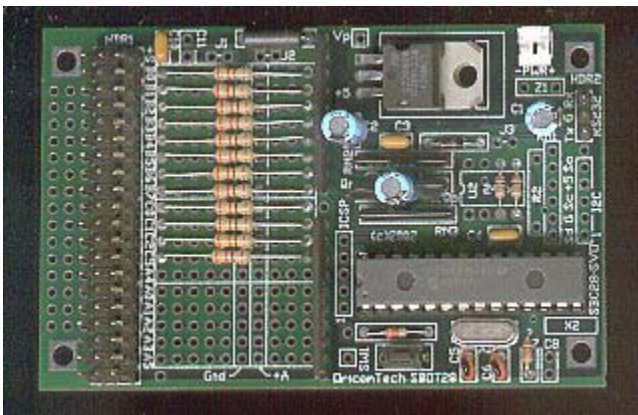


Leg structure of Melanie

Electronics

The electronics used in Melanie v1.0 is very simple as it is based exclusively on a commercial circuit for servo management of low cost controlled via serial port RS232C.

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.

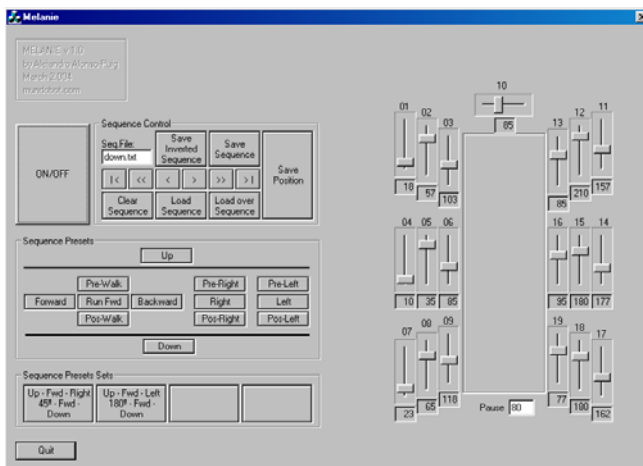
The robot has a GP2D12 infrared distance sensor in the frontal part, but in the present version it is not used.

Software

For the control of the robot it has been developed the application "Melanie" in Visual C++ 6.0. This application is located in an external computer (Pentium II - 233Mhz, 64Mb ram, Windows 98) and it sends the suitable commands to the servo control card of the robot via serial port.

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.

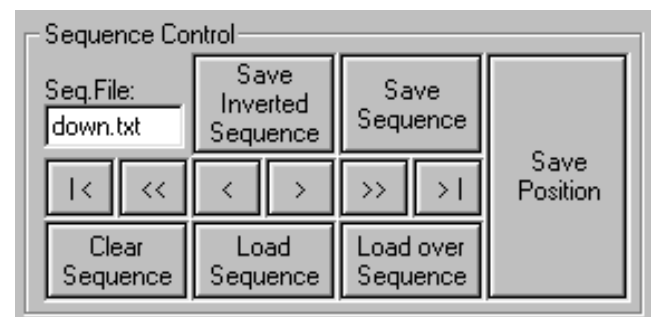
In order to activate the robot the button ON/OFF should be pressed, that will place the legs tight to the body.

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.



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

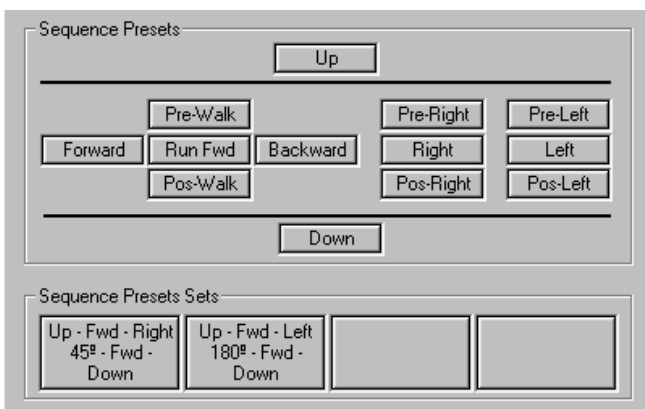
- **Save Inverted Sequence:** It records the sequence but in inverse way. Very useful when a movement is identical to another existing one but in the inverse way, like for example walking forwards and backwards
- **Clear Sequence:** Deletes the sequence that is in memory
- **Load Sequence:** Load in 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 in the plane. These sequences are grouped under the set "Sequence Presets". They allow raising the robot, preparing it to walk, making it run, turning...



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.

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>

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.

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 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 have to be integer values between 0 and 250.

The commands 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:

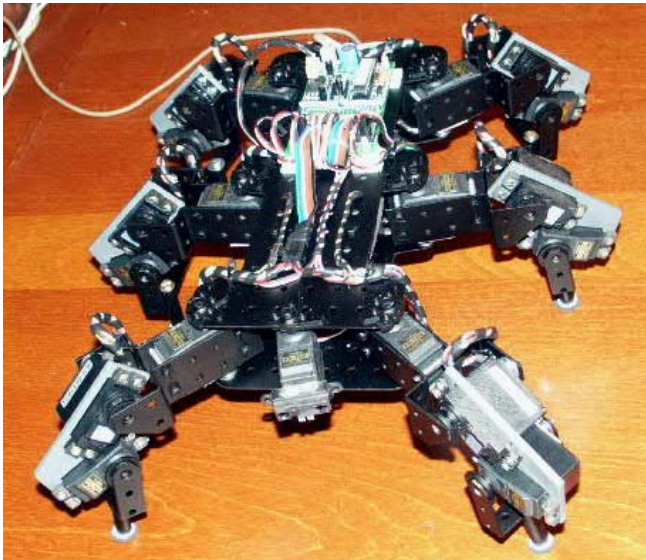
- **SO**: It turn all servos on
- **SF**: turn all servos off
- **SX1F9D**: Set excursión limits of all servos to 496µsg and 2.512µsg
- **SEss**: It activates the generation of pulses for servo "ss", value that should be in hexadecimal 00 - 13h (19d).
- **Mssddvv**: It moves servo "ss" (hexadecimal value) 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>

Conclusions and future investigations

In the present study it has been explained the design of an hexapod robot of optimal mechanical design, able to take additional charges with a suitable increase of effort.



The wished objective has been therefore obtained, consisting of creating an hexapod robot able to walk with prerecorded sequences.








The robot lacks sensorial elements that in later versions would be included for a suitable

perception of the surroundings and corresponding reactions. Between the recommended sensorial elements they would be the following ones:

- Perception of distance to obstacles by infrared. The robot has the sensor, but does not use it in the present version
- Detection of the real position of the joints of the legs, which would allow to a direct programming of the robot as well as detection of obstacles and overloads
- Detection of the consumption of current of the servo ones. Also the consumption will allow to detect obstacles when increasing the current needed by a servo over a normal value before arriving to its destination, or in converse case, detecting lacks of firm floor when detecting less current than normal when reaching the desired position.

The robot has been made quite robust, so it will have to allow many tests and improvements in a future.

Bibliography and interesting links

-  Agricultural Entomology. Daniel G. Pesante Armstrong. Chapter 7:
<http://www.uprm.edu/wciag/anscience/dpesante/4008/cap-7.PDF>
-  Puchobot. Quadrupe Robot. Andrés Prieto Moreno:
<http://www.iearobotics.com/personal/andres/proyectos/pucho/pucho.html>
-  Legs. General design considerations: <http://www.frasco.demon.co.uk/rodney/mechan/design.htm>
-  The Walking Machines Catalogue: <http://www.walking-machines.org/>
-  Silo4. Quadrupe Robot developed by CSIC. Spain: <http://www.iai.csic.es/users/silo4/>
-  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>