

Cerberus the Humanoid Robot: Part III – Software Development and System Integration

Salim Nasser, Shusheng Ye, Mehmet Ismet Can Dede, and Sabri Tosunoglu

Florida International University
Department of Mechanical Engineering
10555 West Flagler Street
Miami, Florida 33174
305-348-6552
snass001@fiu.edu

ABSTRACT

Software development for the Cerberus humanoid robot as well as the initial tests conducted on the prototype humanoid are addressed in this work. The software package developed for this project includes a capability to describe the robot's gait on the screen graphically by the click of a mouse, which is then transparently converted into the P-Basic code to control the servos. This approach significantly saves time and eases software development in defining new gaits for the robot. Various developed gaits have been experimentally tested to assess the capabilities and limitations of the robot.

Keywords

Humanoid Robot, Cerberus, Biped, Quadruped, Gait, PBasic.

1. INTRODUCTION

Creating a defined walking gait for bipedal robot is a complex process dependent on several factors such as service of torque capacity, robot design (center gravity, link lengths), and the type of walking desired (static or dynamic) [1]. The process involved in moving from a theoretical walking gait to the physical realization of such gait by the robot involves several steps. First, there is the need to develop a walking pattern or gate that is appropriate for the given robot [1]. Secondly, the gate has to be translated into an algorithm or code that can be interpreted properly by the given processor being used to control the robot's servos [6]. Finally, the algorithm/code must be compiled and read on to the processor by means of some type of software interface that allows communication between the controller and the computer. It was our intent to create a software package that would integrate the first two steps of this process, which are the most complex and time-consuming of the three. Through the use of a graphical interface for gait creation and automatic code generation, the amount of time spent developing a specific motion or gate and then producing the corresponding code will be dramatically reduced by use of this software application. This program was developed specifically for the Cerberus humanoid robot, using its geometry and range of motion to develop the graphical on-screen representation. Nevertheless, its general algorithm can ultimately be adapted for use with robots/mobile platforms of varying characteristics. The integration between the software and the robot was achieved using a Basic Stamp 2 homework board. The programs created using the gate generator

software were compiled using the Parallax Basic Stamp Editor and loaded onto the homework board using a serial connection. Figure 1 shows how physical robot links can be represented graphically.



Figure 1. (Right) Cerberus humanoid lower body with position markers. (left) Graphical representation of links throughout the gait using APPAS program

2. ALGORITHM FOR WALKING

Robotic studies and applications have shown a great increase in the past thirty years. Robots are started to be used for industrial purposes in assembly lines. Then as they evolved and as they became more intelligent, their interaction with humans in daily life increased.

The controller being used to operate the servos on the Cerberus robot is the Basic Stamp BS-2 model. PBasic is the programming language used by this type of controller and so the type of algorithm defining the walking gait of the robot is determined by said language. The method of control for standard servos involve using a simple *for* loop, which includes the *PULSOUT* command, used to active the servo [4]. There are two types of loops that can be used in order to produce a prescribed motion; point- to-point and stepped increments [4]. As the names implies, a point-to-point loop simply moves a servo from an initial angular position to the next desired position.

```
For i=0 To 100
PULSOUT left_hip, 1080
PULSOUT left_knee, 750
PULSOUT left_ankle, 740
PAUSE 17
Next
```

The code above is an example of a point-to-point loop. The servos for the left leg will rotate from their current position to positions prescribed. The rate at which the microchip processes information line by line is in the order of one millisecond per line, therefore, this change will happen rather abruptly even with the pause command included. Because of the lack of control with regards to the rate of change in the angular positions of the servos that the point-to-point method was replaced by the stepped increment approach in programming the walking gaits. Using step increment loops, one can control the rate at which angular position varies. The following is an example of how one would change the position of a servo(s) using this approach.

```

For i=0 To 100 Step 4
PULSOUT left_hip,      1080 - ( i * 3 )
PULSOUT left_knee,    750 - ( i * -2 )
PULSOUT left_ankle,   740 - ( i * 0 )
PAUSE 17

```

Next

The stepped increment method provides the capability to control both the speed and the final position. An increase in the value of the steps will increase the rate of change and decreasing the steps size will do the opposite. By using whole numbers as multipliers of the loop variable, the total change in angular position can be control. As seen from the sample code, the left hip would decrease by a factor of 300, the left knee increases by 200, and the left ankle would maintain its current position.

It is this type of loop that is the bases for the walking algorithms used to program the Cerberus humanoid robot.

3. SOFTWARE DESCRIPTION AND DESIGN

The process of developing a proper gait and writing the corresponding code can be very time consuming. This is especially true when trial and error is involved in the process. In theory, devising a gait or prescribed motion involves the measurement of angles throughout the motion at the defined critical positions [1]. Combining this process of “reverse engineering” along with writing, what often ends up being hundreds upon hundreds of lines of code, the trial and error approach becomes almost impossibility. It was because of this that a software package, which simplifies and integrates these two processes, was developed using Visual Basic 6.0.

The basic idea behind the Cerberus Gait Solver (CGS) program is to allow the user to develop a specific walking gait or motion using a graphical representation of the robot [3]. Once the user has created said gait or motion, the entire corresponding PBasic code is generated with the click of a button.

3.1 Graphical Interface

The CGS program was developed to be as a user-friendly as possible. A two dimensional representation of the Cerberus robot is shown on the screen as links. The black lines represent the left leg while the blue lines represent the right leg. The interface was designed in such a way as to allow the user to change the positions of any of the joints by simply clicking the corresponding hip, knee, or ankle plus or minus buttons on the left-hand side of the interface [2].

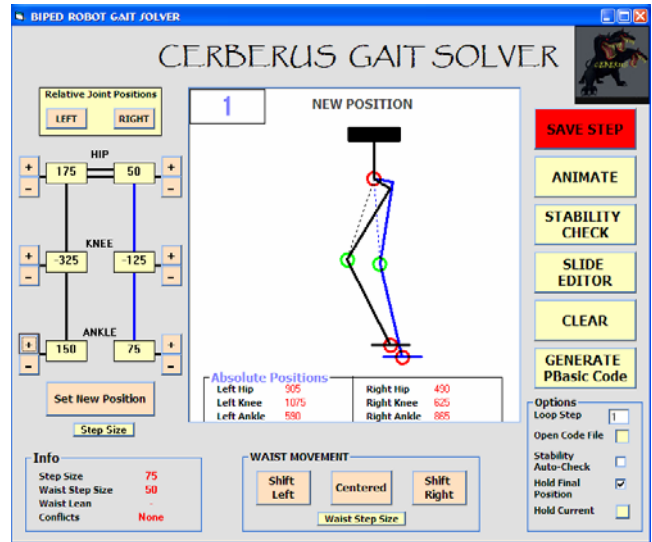


Figure 2. CGS main window

The values entered here are relative joint positions with respect to the defined default position, which in our case is when the robot is standing with its legs perfectly straight. The absolute position values for each joint can be found on the lower part of the picture screen. The units that define both the absolute and relative position values are a result of the servos resolution and range of motion. The servos used have a maximum range of motion of 180 degrees and a resolution of 0.18 degrees per unit increment. That is to say, a relative change of 100 is equivalent to and 18-degree change in the position of a servo. The program offers the user the ability to change the steps size or increment value in order to fine-tune the desired motion. Due to the fact that the Cerberus robot walks by means of shifting its waste position from left to right, the ability to include this in the development of the gait is made available in the *Waist Movement* box. The user can shift the waist to the left, right, or center as well as controlling the waist steps size. The reason for including this is to produce a code that is complete in terms of overall control of the robots gait. Given that the interface is currently two dimensional, the *Stability Check* button was included to check whether the waste is being shifted in the proper direction and/or to remind the user to shift the waist in the proper direction if he/she has failed to do so.

There are five general steps involved in producing the PBasic code for a desired gait or motion. They are: key frame/slide creation, stability check, animation, generation of PBasic, and slide editing.

3.2 Key Frames/Slide Creation

Creation of a key frame is a fairly simple process. Once the user sets the virtual robot figure in the desired position, by clicking on the *Save Step* button, the key frame is created and saved. Each key frame represents a critical end point position in the motion of the robot.

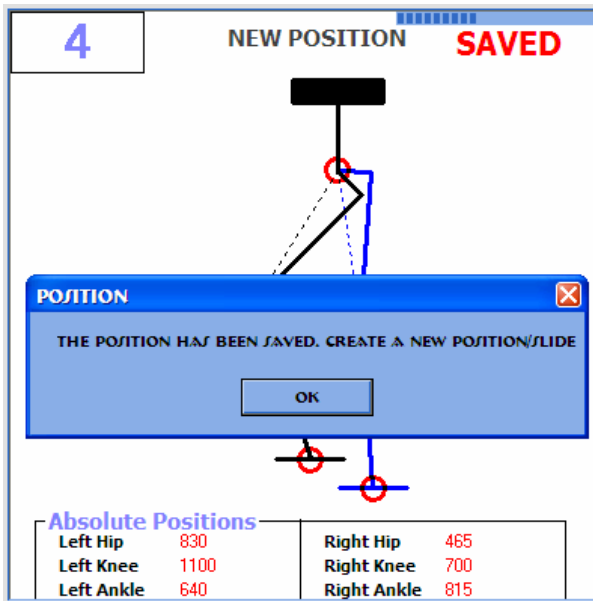


Figure 3. Saving a key frame

The speed at which the robot will produce this motion is dependent on the steps size, the relative change in position, and the loops step. The loop Step is also user defined and is found under the options menu. This value determines this step increment within the loops in the code that the program will generate and by default this set to one. If a value for the loops that is not a multiple of the steps size is entered, the program will alert the user to enter a proper value.

For each key frame the user has the option of holding that position for a period of five seconds. This is done by pressing the *Hold Current* button found in the options menu.

A similar option called *Hold Final Position* is found in the options menu in the form of a check box. If active, a line of code is added at the end of the program, which causes the robot to hold its final position for the duration of 10 seconds.

3.3 Stability Check

Part of creating a key frame includes not only the position of the legs, but also the position of the waist. The user has two options in terms of stability check, he/she can manually check whether the hip is in the correct position or can use the *Stability Auto-Check* feature found in the options menu. By enabling this feature, the program checks for stability every time the user clicks on the Save button. If the position is deemed unstable, a message box alerts the user that the robot is not stable and displays the correct waste position should be.

As long as the position is considered unstable, the key frame will not be saved. It is important to note that the stability check option simply checks that the waist is shifted in the proper direction when a leg is raised. This does not necessarily mean that the robot will be stable when the code produced is tested. Parameters such as the steps size of the waist and the positions of the feet are critical in defining whether the robot is stable or not.

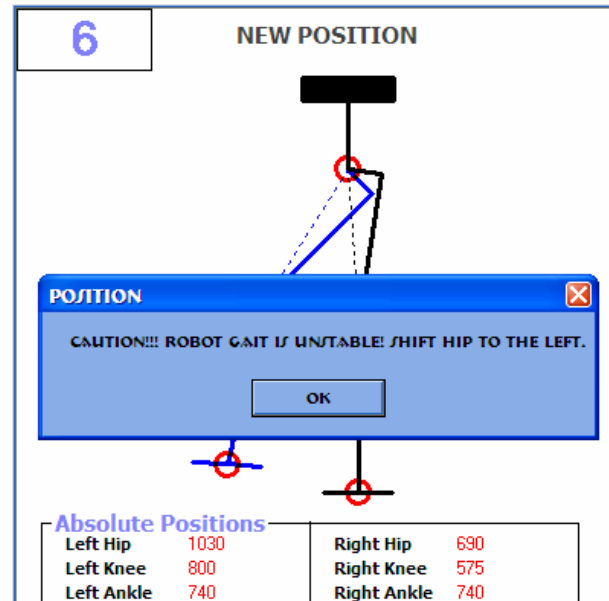


Figure 4. Error type displayed while running stability check

3.4 Gait Animation

Once the key frames/ slides that define the gait or motion are created, the user can visualize the gait created by clicking on the *Animate* button. This feature displays the slides one by one, in 1/2 second intervals from beginning to end. It is a rudimentary "movie" of the gait pattern created and is useful in visualizing whether not the gait is appropriate without having to test it directly on the robot.

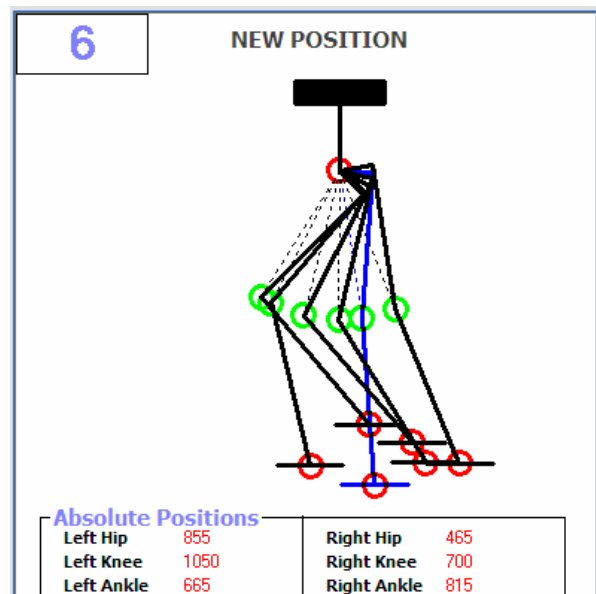


Figure 5. Overlapped Key-frames during animation

3.5 Code Generation

The simplest part of using the program is also the most import. Once the desired motion has been created and checked, the user may click on the *Generate PBasic Code* button. The program proceeds to create the complete code for the corresponding pattern and saves it as a text file under the name

"Pbasic_code.txt". The program contains everything necessary to be run through the compiler (Basic Stamp Editor) including definition of variables and assignation of ports to the corresponding servos. A caveat with respect to this program is that the steps size must remain the same throughout. Writing the Visual Basic code necessary to create the PBasic code involved calculating the counter (i) multipliers at each joint for every given key frame. Once this was done, it was a matter using the Visual Basic syntax to create the proper PBasic code and syntax [5]. Below is a small portion of a part of the program, which writes the PBasic code onto the text file.

```
Open "C:\pbasic_code.txt" For Append As #1
Print #1, " "
Print #1, "-----", pb
Print #1, "For i=0 to ", di, "Step", lstep
Print #1, "PULSOUT waist,", nh
Print #1, "left leg"
Print #1, "PULSOUT left_hip,", nlh, "-", "(i *", xlh(pb), ")"
Print #1, "PULSOUT left_knee,", nln, "-", "(i *", xln(pb), ")"
Print #1, "PULSOUT left_ankle,", nla, "-", "(i *", xla(pb), ")"
Print #1, "right leg"
Print #1, "PULSOUT right_hip,", nrh, "+", "(i *", xrh(pb), ")"
Print #1, "PULSOUT right_knee,", nrn, "+", "(i *", xrn(pb), ")"
Print #1, "PULSOUT right_ankle,", nra, "+", "(i *", xra(pb), ")"
Print #1, "PAUSE 17"
Print #1, "Next"
Print #1, " "
Close #1
```

```
nlh = nlh - (di * xlh(pb))
nln = nln - (di * xln(pb))
nla = nla - (di * xla(pb))
nrh = nrh + (di * xrh(pb))
nrn = nrn + (di * xrn(pb))
nra = nra + (di * xra(pb))
```

In the print statements, anything within quotations is considered text [5]. If not within quotations it is considered a variable, and the current corresponding value is written onto the text file [5]. After writing the PBasic loop for that given motion on the text file, the new values for the starting positions (nlh(pb), nln(pb), nla(pb), etc) are calculated before the start of the next Visual Basic loop.

The final result is a program that allows the user to control waist motion, the position of the legs, rate of change and servo positions as well as pauses or holds between motions.

Different loops were created that write the sections of PBasic code that control the movement of the waist, hold any given position, and hold the final positions. These loops are appended to the text independently from each other. That is to say, the movement of the waist is written in a separate loop than that of the one that controls the legs. Therefore, leg and waist motions cannot be performed simultaneously. But, since the Cerberus robot is a static walker, this is of no importance anyway. The

figure below shows part of a PBasic program created using the CGS.

```
waist CON 3
right_hip CON 4
right_knee CON 5
right_ankle CON 6
arms CON 7
i VAR Word
j VAR Word
k VAR Word

'----- 1
FOR i=0 TO 25 STEP 1
PULSOUT waist, 750
'left leg
PULSOUT left_hip, 1080 - (i * 3)
PULSOUT left_knee, 750 - (i * -4)
PULSOUT left_ankle, 740 - (i * 1)
'right leg
PULSOUT right_hip, 440 + (i * 3)
PULSOUT right_knee, 750 + (i * -4)
PULSOUT right_ankle, 790 + (i * 1)
PAUSE 17
NEXT

'----- 2
FOR i=0 TO 250 STEP 1
PULSOUT waist, 750 + i
'left leg
PULSOUT left_hip, 1005
PULSOUT left_knee, 850
PULSOUT left_ankle, 715
'right leg
PULSOUT right_hip, 715
PULSOUT right_knee, 650
PULSOUT right_ankle, 815
PAUSE 17
NEXT
```

3.6 Slide Editor

Slide Editor Tool is a valuable feature that allows the user to modify saved key frames in a given gait. It allows the user in a sense to "tweak" or perfect each slide whether it is after having checked the gait through the animator or having tested it on the actual robot.

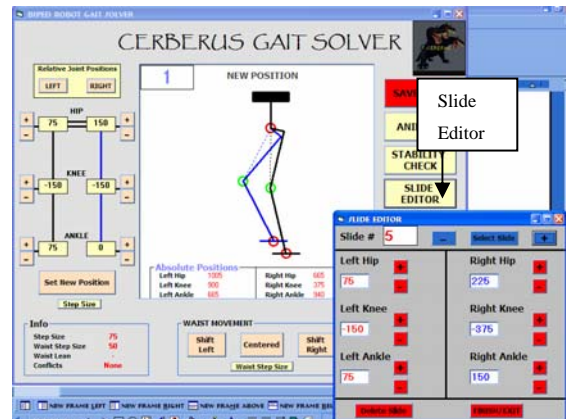


Figure 7. Slide Editor appears at bottom right of screen

This is an essential tool given that it allows for the optimization of a gait pattern after it has been tried directly of the robot. This further reduces the amount of time necessary to produce a proper walking gait or motion by allowing the user to go back and change key frames that may not be optimal.

The slide editor is simple and concise. The leg positions are controlled in the same way as in the main window. When a user chooses a slide, the current relative positions are shown. If changes are made, the new relative positions are displayed in the

main widow but not in the slide editor. This is done so that the user can always reference the initial position for that slide. The slide editor saves changes on the fly so there is no save button to push. Changes made to any slide are immediately updated in an effort to simplify the process.

The slide editor also allows slides/key frames to be “deleted”, though it does it in a very practical way. If the *Delete Slide* button is pressed, the current key frame position is replaced by that of the previous key frame. The reason for this is to allow the user to start over from the last valid position without changing the number of key frames.

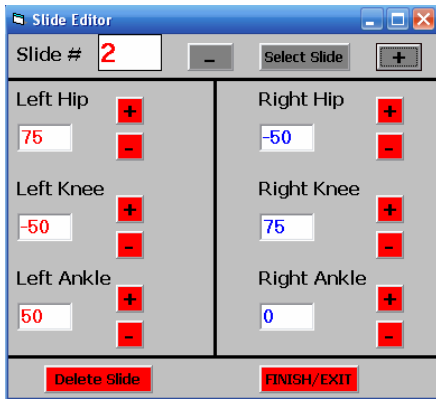


Figure 8. Slide Editor

3.7 Limiting Position Checks

Since the program is set up to represent the Cerberus Robot, it needs to take into account the range of motion (ROM) of each joint. When the user enters a position that is beyond the ROM of any of the 8 joints, it immediately alerts the user by means of a message box displaying the error as well as specifying which joint has gone beyond its prescribed ROM. The *info* box will also display the error.

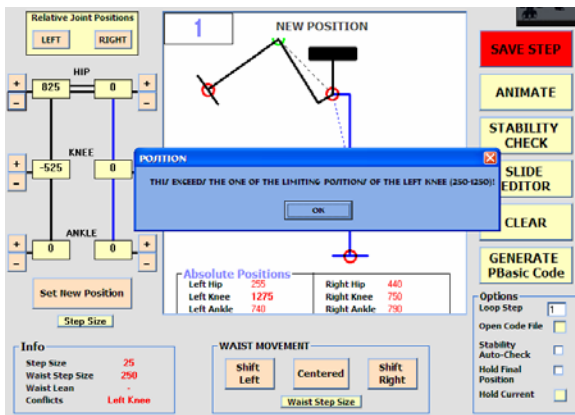


Figure 9. Shows the error message that appears when ROM of a servo has been surpassed. Info box also displays the error in Info-Conflicts

The user will not be able to save a position with conflicts. This is necessary in order to represent the robots capacities properly as well as to protect the actual servos from damage.

3.8 Program Flexibility and Ease of Use

The CGS program was designed to be very flexible in terms of creating a gait. The user can create small sections of the gait,

generate the code, and check to see if the robot responds in the desired fashion or simply check the animator. If changes need to be made, it can go back and make changes using the editor without having to start from the beginning. The program is set up so that you can continue adding slides to the gait after having tested the current ones. This allows for the creation of a gait or motion in one try or by segments. The flow chart below shows the flexible nature of the software.

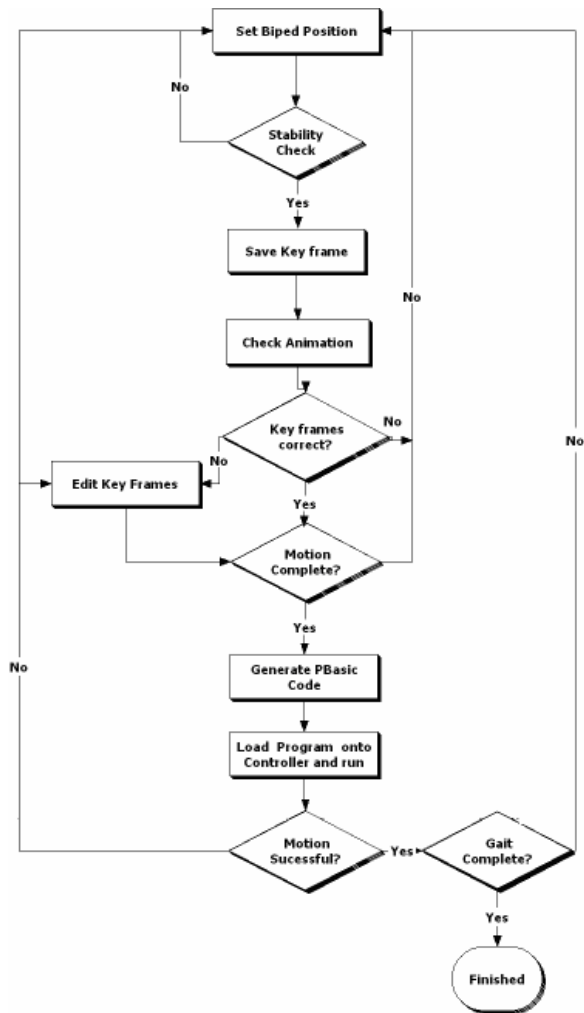


Figure 10. Flow chart of gait generation process

4. SOFTWARE AND HARDWARE INTEGRATION

At the heart of Cerberus's control system is the Basic stamp 2 homework board. The basic stamp uses a PIC 16C57 micro controller, which runs at about 20 MHz. It has a built-in EPROM with a capacity of 2K/500 lines of PBasic code and 32 bytes of RAM [4]. The BS-2 has 16 ports, which can be designated as either input or output, depending on the needs. It connects to the PC using the normal support interface. In the case of the Cerberus, the BS-2 controls all eight servos of the robots joints. The figure below shows how the controller was connected to both the servos as well as the power source. Once the code has been generated by the CGS, it is next copied and pasted into the compiler, which in

this case is the Basic Stamp Editor from Parallax. The following Picture shows the BS-2 mounted on the Cerberus humanoid.



Figure 11. Control board mounted on Cerberus

The following schematic shows how the servos and power source were connected.

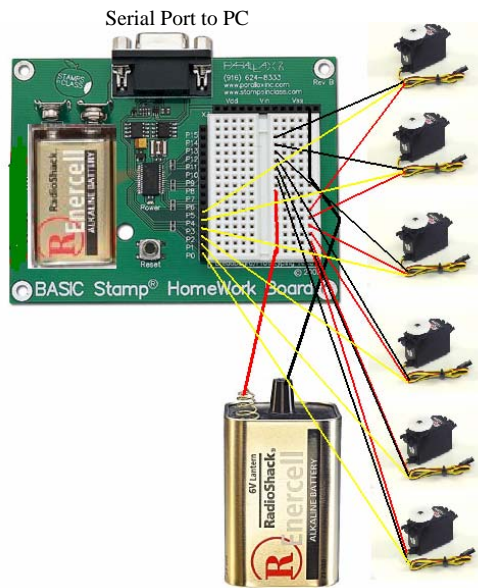


Figure 12. Schematic for connecting board to servos and power source

After the code is entered into the editor, the Cerberus robot's Control Board is connected to the computer via a serial cable. The code is then run to the Editor and loaded onto the control board's EPROM. Once this is done, the robot will perform instructions programmed.

5. RESULTS

Using CGS produced gaits; it was possible to test Cerberus's capacities. Motions created using CGS's graphical interface were reproduced accurately with a high trial and error rate. Ultimately, Cerberus took its first steps using a gait generated from the CGS. This initial gait produced a program that exceeded the processor's memory capacities. Hence, the gait was optimized

for speed, accuracy, and stability while minimizing the size of the code using the slide editor. The final result was a gait that maximized the physical capacities of the robot, producing a human-like walking pattern.

6. CONCLUSIONS

The Cerberus Robot was completed several weeks before the concept of the CGS program was born. Hence, many attempts were made at developing a feasible gait. This process was time consuming, difficult, and imprecise. Gait positions were drawn, joint angles were measured, the equivalent joint positions calculated, and then the corresponding code was written. This was an extremely inefficient way of developing a walking gait especially considering that this process had to be repeated many times.

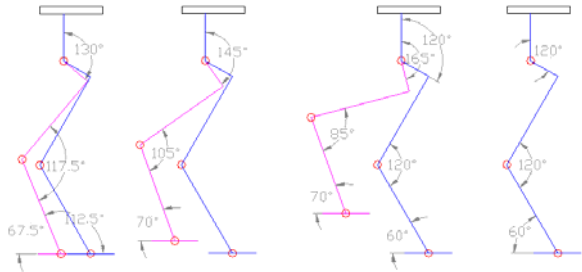


Figure 13. Initial attempts at developing a gait by sketching key frames

The development of the Cerberus Gait Solver made the trial and error process quick and simple. The time necessary to develop and test a given gait was reduced from hours to minutes. The program capabilities have proven to be invaluable in producing a robot with human-like walking gaits.

7. REFERENCES

- [1] Qiang Huang, Kazuhito Yokoi, Shuji Kajita. "Planning Walking Patterns for a Biped Robot," IEEE Transactions on Robotics and Automation, Vol. 17, No. 3, June 2001.
- [2] "Bi-Ped gait simulations." <http://www.havingasoftware.nl/robots/BiPed/BiPedSim.htm> accessed March 2005.
- [3] Krister Wolff, and Peter Nordin. "Evolution of Efficient Gait with An Autonomous Biped Robot Using Visual Feedback," Chalmers University of Technology Department of Physical Resource Theory, Complex Systems Group.
- [4] "BASIC Stamp Editor / Development System." http://www.parallax.com/html_pages/downloads/software/pbasic_2_5.asp, accessed March 2005.
- [5] Michael Halvorson, Jenny Moss Benson, and Emma Gibson. "Microsoft Visual Basic 6.0 Professional Step by Step," Microsoft Press, 1998.
- [6] Ruixiang Zhang, and Prahlad Vadakkepat. "An Evolutionary Algorithm for Trajectory Based Gait Generation of Biped Robot," Department of Electrical and Computer Engineering, National University of Singapore.