ECE 492 Winter 2016

Smart Robotic Quadruped

Using Machine Learning to Optimize Parameters and Control Motion

Cody Otto **Duncan Prance** **Brittany Lamorie**

Group # 5

Abstract

Our project is designed to teach a robot how to walk. It is composed of a set of four legs, each containing three servos and attached to a single chassis. We have setup and ran a separate computer simulation to train a simulated version of our device, which we pass on to the robot in the form of individual servo commands. We therefore have a physical embodiment of the simulation results, and the system is now capable of basic forward movement after its proper training. It also has 4 modes that include dance, walk, weight shift and simulation. These modes are changed using the dip switches on the board.

Contents

Functional Requirements

Our final project is a 4 legged robot controlled by servo motors that are connected through GPIO to the DE0 Nano Altera FPGA. There are 4 pre-programmed modes that show off the capabilities of the robot, these are a walk, a dance, repetitive weight shifting and a simulation mode. This final mode was created from the outputs of one of the final simulations we ran. These simulations used a physically accurate rendering of our robot with a neural networks and reinforcement learning background to learn and improve on a forward motion. This training was all done on PC and then converted to pulse widths to be read by the servo motors on the robot.

Originally we had intentions to have on board corrections with an accelerometer and gyroscope being used to tell the robot when it was off balance. This addition was not made due to time constraints. Our simulations ran for the majority of our time with varying reward systems; unfortunately we were unable to come up with a successful system to train a walking motion. Most of our simulations resulted in the rendering falling in the appropriate direction but did not succeed in teaching him how to take a step.

Powering our robot was a final problem and one that was partially solved. Our original intentions were to have 6V NiMH batteries attached to the chassis of the robot. We decided against attaching the batteries due to the dramatic change in weight that would affect the balance of our robot while walking. The hard-coded walk was made so precisely that any change in weight could result in imprecise steps causing the robot to become off balance and fall. We next tried a tether system with 3 AA batteries powering the DE0 Nano and 4 AA batteries powering the 12 servo motors. This solution allowed for the robot to be portable but still allowed for us to maintain the same walk. This solution worked well for the dance and weight shift modes but we found the current draw on the walk to be too high for the batteries to handle. In the end we had the board powered by the 3 AA batteries but used a power supply for our servo motors.

Design and description of operation

Data Flow

The data flow of this system consists largely of moving data between the server system and the robotic hardware.

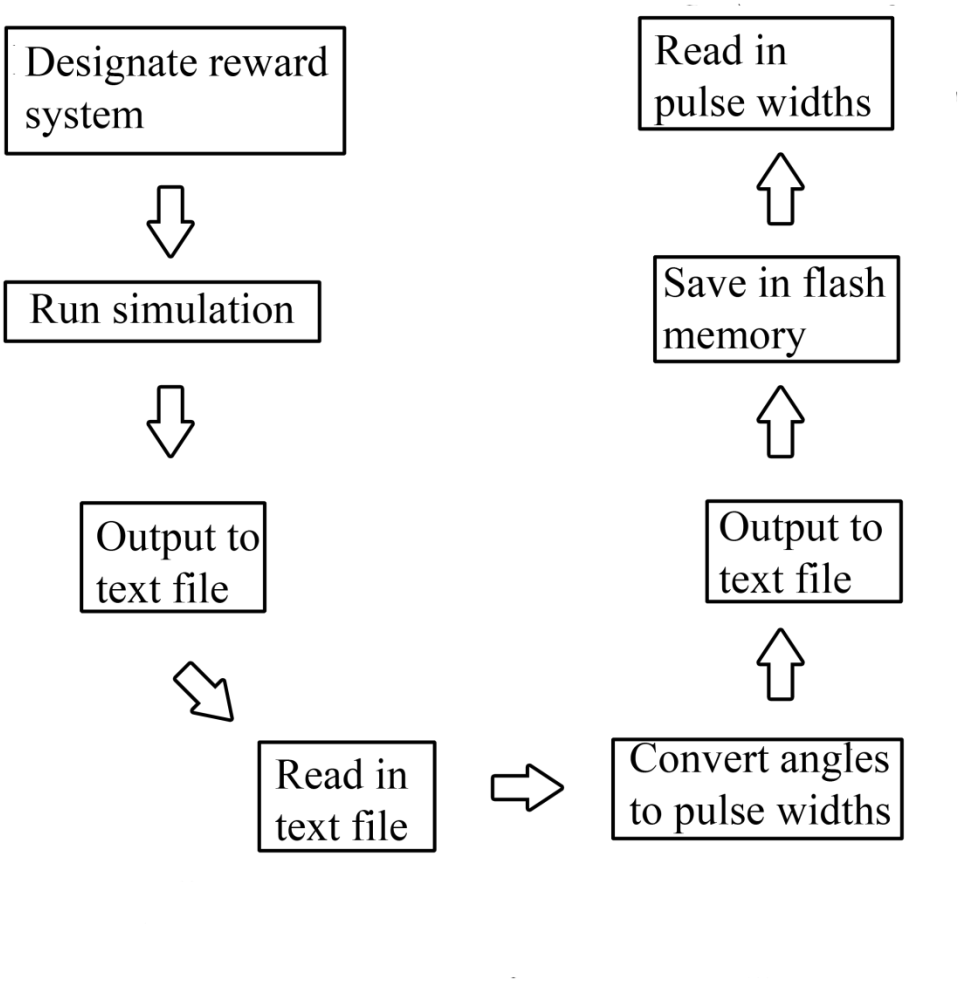

Figure 1: Data Flow

Hardware

The hardware of the system is largely separate from the software side, and involves fairly simple I/O. The DE0 Nano is the board we have chosen to use, and it is connected to 12 servo motors through the first 12 GPIO pins. Each of the signals is individually controlled for each joint in the robot. The robot framework is made of Lexan and aluminum components to keep the weight low. The control wires are directed to servo headers through a handmade wire-wrapped breadboard, to maintain a constant voltage and ground for each of the control signals.

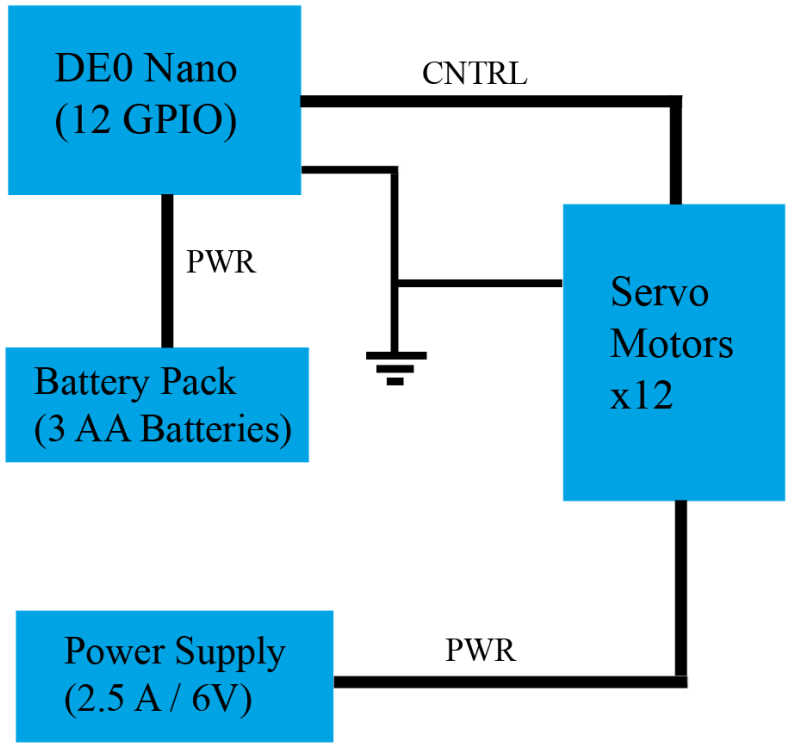

Figure 2: Hardware Diagram

Bill of Materials

Part	Supplier	Spec	Information Links	Cost (CAD)	Weight	Order Status
2x Lynxmotion Alum and lexan	Robot Shop	Hip Hor. to Hip Vert. = 38mm Hip Vert. to Knee Vert. = 57mm	http://www.robotshop.com/ca/en/lynxmotion-3dof-	$92.87 x2 (185.74)	~0.181 kg x2	Received (not all of the 4 kits were used, was

Leg(Pairs)		Knee to Foot = 124mm	aluminum-lexan-leg-pair.html			only required due to component issues)
2x Lynxmotion Alum and lexan Leg(Pairs) (Pieces)	Robot Shop		Only screws, spacers, washers, and a small part of the mounting brackets were used	~$5		
1x Lynxmotion Quadruped Body Kit Mini	Lynxmotion	Height = 2.125" Length = 7.250" Width = 3.000" Length = 5.250" Width = 5.250"	http://www.lynxmotion.com/p-435-quadrapod-body-kit-mini.aspx	$19.95 (27.93)	~0.181 kg	Received
DE0 Nano				Provided	~0.050 kg	Provided
6x HS-422 Servo Motor	Robot Shop	http://www.robotshop.com/media/files/pdf/hs422-31422s.pdf	http://www.robotshop.com/ca/en/hitec-hs422-servo-motor.html	$13.27x6 (79.62)	0.0455 kg x 6	Received
2x HS-311 Servo Motor	Robot Shop		http://www.robotshop.com/ca/en/hitec-hs311-servo.html	$9.99 x 2 (19.98)	0.043 kg * 2	Received
4x HS-635 Servo Motor	In Stock		http://www.servodatabase.com/servo/hitec/hs-635hb (pricing)	$29.99x4 (154.67)	0.0499 kg * 4	Received
Totals:				**$472.94**	**1.152 kg**	

Available Sources

From a software point of view, we used many different open source components to do the simulations and associated learning. To do this, we used the PyBrain library for Python. This library has dependencies including SciPy and Python 2.5 [3], which we included as well as the optional ODE physics engine [6] and PyOpenGL [7] to visualize the simulation. The sizes of these components are not relevant to the spec, as they were all run on desktop hardware. Therefore, size and processing power was not an issue.

Regarding the FPGA itself, we did not use any open source material. We used the microC libraries as our RTOS of choice. The size of the source is roughly 10MB [12].

Data Sheet

Operating Conditions:
In mode 4 (walking), ensure the robot has high friction with a surface such as an antistatic mat. There needs to be 4x2 AA batteries powering the servos to provide enough current. Ensure that the starting base of the robot is stable in the first position. It has a 1.5 second delay to give you time to do this. In modes 3 and 2 (dancing/weight shifting), ensure that the robot has a stable starting stance as above. You can run these modes off of only 4 batteries for the servos. Friction does not matter as much since it is not moving, but it is still recommended to use a high friction surface. In mode 1 (simulation output), run while held in the air. This takes a long time to run.

Power:
Our project used 12 servo motors at 6.0 V, with the DE0 nano board using 4.5V (3 AA batteries in series). The maximum current draw experienced is when all 12 servos move at the same time, and draws up to ~2.8A of current in the walking mode only. Since the battery pack can only provide ~2.0A at once, a parallel configuration would be needed. Powering the servos with an external battery pack was only tested with 4 AA batteries to provide the 6.0V required, but the current draw was too high to walk consistently. Therefore, a power supply was used instead. Our power was provided with AA non-rechargeable batteries, to provide a stronger voltage curve to maintain maximum power for the servos to help keep the robot stable. From online sources, we found the maximum current draw of a single servo while stalled to be ~700 mA (for HS-311 and HS-422, approximately the same for HS-635 as well). This would give an absolute maximum of 8.4 A of current if all 12 servos are running while stalled, which should never occur.

Background Research

As part of our research, we came across an article from the University of Texas at Austin [9]. This article detailed an experiment done using a commercially available four-legged robot, and their experiences with having it learn how to move. They initially proposed using a parabola as the general shape for a step, and using that as the framework for learning. Optimizing the parameters of the parabola was the ultimate goal, along with a few others such as the body height and amount of time each step takes. They used a hill-climbing algorithm, in conjunction with a few other learning methods, to train the machine.

In the paper written by Hornby and associates [10], they did a similar set of training with Sony robots. In this instance, they were aiming to deliver a system that required absolutely minimal human interaction. They quickly realized that, in a system as sophisticated as theirs, that things as minimal as the roughness of the surface affected the eventual gait of the robot in question. While not helpful in our case, this realization shows how even the smallest of factors can influence the outcome in unforeseen ways. Their paper also contains many base ranges for a variety of parameters that we may incorporate, such as step height and phase differences in opposing legs.

Software Design

In this project, data flows in a very non-continuous fashion. Due to the fact that most of the heavy calculations are done server side before the demo, commands only need to be transferred once. This creates a “stop and start” type of data flow. Refer to Figure 1 for more information about the data flow.

Server side, we have two separate components. One is the software simulation, which is designed to create an environment that is accurate enough to real world that the commands it generates can be converted into servo commands for the physical robot. The other piece of software is the hardware integration, where these commands are converted into pulse widths that are passed to the motors and the simulation is physically tested.

The simulation piece was done in Python and C++, using a combination of PyBrain and ODE/OpenGL for the physics aspect of it. PyBrain is used in an attempt to allow the simulation to learn general motion. We used a reward system to shape the movement, measured based on distance moved as well as overall height of the body. Every hundredth simulation has its commands saved to a text file, allowing us to recreate what occurred using the angles it measured.

The hardware integration piece is written partially in microC, with the rest being done in Python. The Python portion is responsible for taking the results from the simulation software and turning them into useable angles. This portion outputs a text file containing function calls, to a function within the microC environment, with the appropriate angles for each servo motor and appropriately spaced delays to allow for servo movement. These commands are then added to the simulation mode and the angle to pulse width function converts and sends the proper commands to the motors. By loading up a series of commands prior to the system being used, we limit the amount of ongoing calculations and signals needed by the board itself. This is similar in design to an airplane being given a flight plan, with very infrequent hand-holding required.

Test Plan

Software

In order to test the software for our project, we needed to do both hardware independent and dependent testing. For the independent version, the plan was executed as follows:

- Use PyBrain and ODE/OpenGL to construct a simulation of our robot. This simulation will generate commands in the form of angles, which will be associated with servos.
- Writing Python code to turn these high level commands into commands that a servo would require (pulse width modulation).
- Looking at the commands to see if they make logical sense. This can be done as the commands are communicated in terms of degrees, and it is easy to see if a joint or limb is doing something that it should not be.

By limiting the amount of errors in our software, we can help keep our hardware safe from these bugs. When we add the hardware into the equation, we are able to more accurately test the software results using a physical embodiment. This plan was as follows:

- We created sample movements that test a wide range of motion for each servo. Using these commands, it is easy to see if a servo is dying or if something else is wrong with the hardware.
- Each time we ran new movements, we used the sample movements first to ensure that everything was still working as expected.

Hardware

Hardware testing involved sending a signal to each of the servos prior to assembly of the robot. This caused some issues as we only saw that each servo was moving given a signal, not that they were operating correctly. We were unable to detect issues such as the servo missing teeth, or responding poorly to certain signals. This caused us to have to swap servos out part way through and reconfigure. Once the robot was constructed, we tested that the robot could stand on 4 legs, and then eventually on the 3 legs (similar to the final stance that was used). The servos were adjusted and configured to provide the required range of motion in each joint. Once we had everything working, we tested the battery packs for the DE0 Nano as well as the 12 servo motors. We found that the DE0 Nano was simple to power as long as the project was flashed onto the board, and that the servos could be powered with the 4 battery pack. We ended up using the power supply for our demo as the current draw during the walking was too much for the batteries and the power supply was simply more reliable. As it was late into the project we decided against testing out a new battery solution as it could have caused more issues than it solved. A 4x2 battery arrangement should be able to power all 12 servos correctly, but is currently untested.

Results of Experimentation and Characterization

We used our hardware testing to ensure all of our motors were working. Due to some other issues, we had to swap out some servos after the initial assembly. The construction of the robot took place, and was found to have issues in regards to the body not fitting together correctly. We originally wanted to go through with a mammalian walking design, but due to angle restrictions we went with a more spider type leg orientation, allowing us to use more of the hips. We tested the walking motion using a 4 AA battery pack, and found that the current draw during some motions was too high and would cause the robot to collapse. We assume that having 2 packs in parallel will be able to provide enough current, but due to time constraints we were unable to fully test this.

Our simulation results were not ideal though we were able to use them as a proof of concept for converting simulation angles to accurate pulse widths for the motors. The best result from the simulations were very frog like in nature, where the back legs would gently push the body forwards. We took the angles from this simulation and ran them through our conversion software, this results in a text file with proper angles and servo call for the motors. Adding these calls to the on board program allows the angles to be converted to pulse widths and sent to the motors with appropriate delays. This conversion was found to be somewhat accurate but the rendering of the body did not depict the angle of the hips properly and we were forced to hard code them. The direction of motion of the knee joints were also inconsistent but the movement itself was accurate. After all, we believe this conversion still proves this to be a reasonable way to transfer simulation results to physical motors.

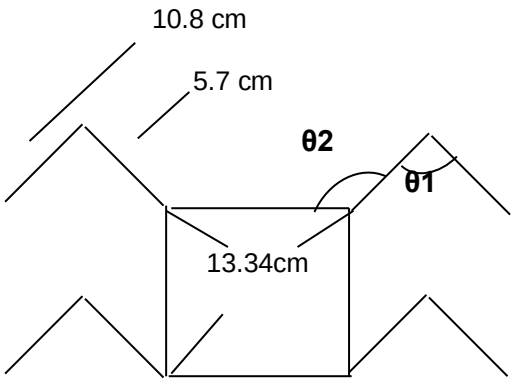

Figure 3: Distances measured from joints

Our total mass was 1.25 kg. At minimum our robot maintains a 3 point contact with the floor. We assume that each of the legs carries approximately even amounts of weight. The maximum torque on our robot will be experienced when the hip is pointed directly away from the body, the knee will be limited to this maximum torque by the equation:

*10.8*sin(x) = 5.7* (same distance as the hip)

This provides an x value of 31.86 degrees away from vertical as the maximum. Any of our load bearing legs will maintain a maximum angle of 30 degrees away from the vertical. This simplified calculation does not account for any mass that is directly over the rotational point of the torque, so this value is actually a slight overestimation. We will use kilograms and centimeters to be able to directly able to compare with the servo motor data sheets. This leads us to the equation:

*1.25 kg / 3 * 5.7 cm = 2.375 kg*cm*

This is well within the limit on our weaker weight bearing servos (HS-422 at 4.1kg*cm @6.0V).

Safety

The voltage supply of our project will be a 6V battery pack, this voltage and the current it supplies is not any particular danger to a human. This of course could damage the DE0 Nano and that is why the board is run off a separate battery pack.
The servos we are using have a maximum operating temperature of 60°C, this could cause third degree burns after only 5 seconds of contact. The robot will only be operating in short bursts to avoid these operating temperatures and we will be using caution when handling the servos after an extended run. The operating speed for no load is 0.16 sec/60°.

Regulatory and Society

Societal and regulatory concerns for this project are all hypothetical. Considering there is no wireless communication the possibility for hacking the robot is quite limited and its abilities do not lend themselves to dangerous control. The YouTube video [11] showing the robot Spot built by Boston Dynamics is a largely scaled up version of our project. Clearly a robot of that size and weight would be able to do much more damage than our small robot if it were to be hacked.

Society today has a love hate relationship with artificial intelligence. The automation of menial tasks is a huge step forward in society but when these tasks become more difficult and actually require a certain level of intelligence, people are wary of what else these robots are capable of in the future. Our project will have very limited brain power and its intelligence is completely stored on the desktop computer. The robot itself only follows the instructions given to it from the simulation data as well as hard-coded movements.

Environmental Impact

Our project has a very minor environmental impact when operating properly. The battery packs we use have disposable AA batteries in them. These have gotten safer in recent years and only using 3 for the board also minimize our impact. These batteries are Duracell AA which are Alkaline Manganese dioxide batteries, as long as the battery is not tampered with the caustic chemicals not be a danger to us or the environment. Recycling batteries is an important part of using them and will again minimize the impact we make while using our project.

Sustainability

Voltages: using a 6V power supply for the servos, going up to 600 mA (maximum) with a load on each servo. 4.5V supply for the DE0 Nano board with a current draw of up to 500 mA assuming it's used at 50 MHz (the maximum). This makes the total power consumption while all servos are active:

$$3.6\ W*12 + 2.25\ W = 45.45\ W$$

The servos use approximately 8 mA when idle, and the board draws around 50 mA, totalling only:

$$0.576\ W + 0.250\ W = 0.826\ W$$

We are assuming if the project is not idling or in use, it is off and using 0 power. Our project will be in idle mode approximately 90% of the time (plugged in but not used), and active mode the other 10% of the time. We expect to use our device actively (for demos or for

testing) approximately 3 hours this year, meaning it should be idling for approximately 27 hours. The weighted average of power used is:

*45.45 W * 0.1 + 0.826 W * 0.9 = 5.29 W*

CO_2 is generated by a ratio of 0.989 kg/kWh of power used. If it were to be used for an entire year in this ratio, you would generate:

*5.29 W * 24 hour/day * 365 day/year * 0.989 kg/kWh / 1000 kW/W = 45.8 kg CO_2/ year*

If the project is only used for the approximate 30 hours total of idle/active mode for this year, it would instead only generate:

*5.29 W * 1/1000 W/kW * 30 hours * 0.989 kg/kWh = 0.157 kg CO_2/ year*

References

[1] HS-422 Datasheet [Online]. Available: http://www.robotshop.com/media/files/pdf/hs422-31422s.pdf, Accessed on January 28th, 2016

[2] 6 DOF Gyro, Accelerometer IMU - MPU6050 Documentation [Online]. Available: http://www.robotshop.com/content/ZIP/documentation-sen0142.zip, Accessed on January 16th, 2016

[3] PyBrain Documentation [Online]. Available: http://pybrain.org, Accessed on January 16th, 2016

[4] DE0 Nano Documentation [Online]. Available: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=139&No=593&PartNo=4, Accessed on January 31st, 2016

[5] Rechargeable Nickel Metal Hydride Battery [Online]. Available: http://www.hobbyking.com/hobbyking/store/__25030__Turnigy_Receiver_Pack_2300mAh_6_0v_NiMH.html, Accessed on January 29th, 2016

[6] ODE Physics Engine Documentation [Online]. Available: http://www.ode.org/, Accessed on January 30th, 2016

[7] PyOpenGL Documentation [Online]. Available: http://pyopengl.sourceforge.net/, Accessed on January 30th, 2016

[8] Comparison Quadruped [Online]. Available: http://letsmakerobots.com/node/28077, Accessed on January 31st, 2016

[9] Nate Kohl and Peter Stone, “Machine Learning for Fast Quadrupedal Locomotion,”. [Online]. Available: http://www.aaai.org/Papers/AAAI/2004/AAAI04-097.pdf, Accessed on January 31st, 2016

[10] Gregory S. Hornby , Seichi Takamura, Takashi Yamamoto, and Masahiro Fujita, “Autonomous Evolution of Dynamic Gaits with Two Quadruped Robots” Robotics, IEEE Transactions on, Volume: 21 Issue: 3 June 2005

[11] Introducing Spot - Boston Dynamics [Online]. Available: https://www.youtube.com/watch?v=M8YjvHYbZ9w, Accessed on February 1st, 2016

[12] Jean J. Labrosse, “What You Need to Use uC/OS-II” in MicroC/OS-II The Real-Time Kernel Second Edition, Lawrence KS.

[13] Robert Hood, Barry Peyton, Max Marcus, “iOS Device Controlled RC Car Capstone Project”, Available: http://www.ece.ualberta.ca/~elliott/cmpe490/projects/2012w/g1_iOS_RC_Car/G01%20CapstoneProject--FinalReport.pdf

[14] Cadmium (NiCd) Vs Nickel-Metal Hydride (NiMH) Batteries - Environmental Impact & Recycling[Online] Available : http://www.globelink.co.nz/news/cadmium-nicd-vs-nickel-metal-hydride-nimh-batteries-environmental-impact-recycling/

[15] Servo Specification Forum. Alan T. http://www.rcgroups.com/forums/showthread.php?t=1073878

Appendices

Quick Start Guide

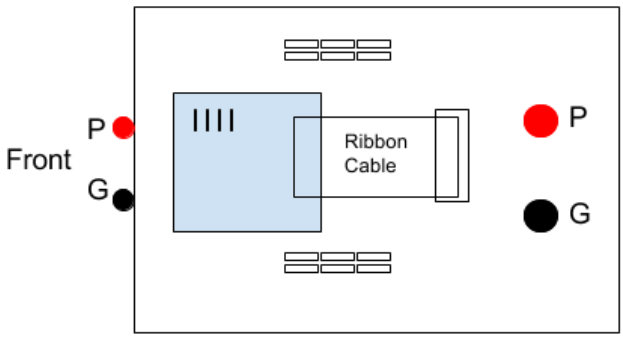

Figure 4: Overhead View of Board

The servo motors should be connected to the correct pins, the motions for the robot are already flashed onto the board and it only needs power to run.
If the DE0 loses the flash settings or the servo motors are disconnected the code can be updated to correct for the PWM signals / servo motors until the numbers work out. The numbers correspond to what is written in as the servo number when we call the motions. The front left leg should be: 12 HipX, 11 HipY, 10 Knee. Front right leg should be: 9 HipX, 8 HipY, 7 Knee. Back left should be: 6, 5, 4. Back right should be 3, 2, 1.

The board is powered at the front, to power it use 3 series AA batteries, the flashed program is already on board. The servos need to be powered with enough current, to ensure it will be enough we used a power supply at 6.0 V, with 2.8 A current available. This can be replicated by using 8 AA batteries connected 4 in series by 2 parallel (this was untested, but a singular 4 pack will not be able to provide enough current and the servos will fold over themselves).

The robot has 4 modes as indicated by the small dip switches on the DE0 board. When only one of the switches is high and the other 3 are low, it will be in the corresponding mode. The dip switches are labeled from 1-4 and these will be used to indicate the modes.
4: Walking mode, after 1.5 seconds delay will begin moving, in order for this mode to work properly, the robot has to be set down in the starting base (the first position it moves to)
3: Dancing mode, this mode can be set down at almost any time, will cycle quickly through lifting each leg.
2: Weight Shift mode, This mode should be able to be set down as long as it is not in motion, the base should remain almost constant and just move the body around the legs.
1: Simulation mode, this mode should not generally be used as it takes a long time to run through, but it comes from the simulation output from our simulation (would be falling in slow-motion).
Other modes could be programmed into the C code provided as future work.

Future Work

Basic operation was achieved but there are many improvements that could be made on our projects. The machine learning aspect was successful only in a proof of concept sense and more extensive work could be done to improve on it. A proper reward system was never found that resulted in a step and it would take significantly more work to achieve this.
The robot itself could have been improved with either more precise servo motors and/or less rigid joint options. This would improve the walk and make the weight shifting easier.

Hardware Documentation

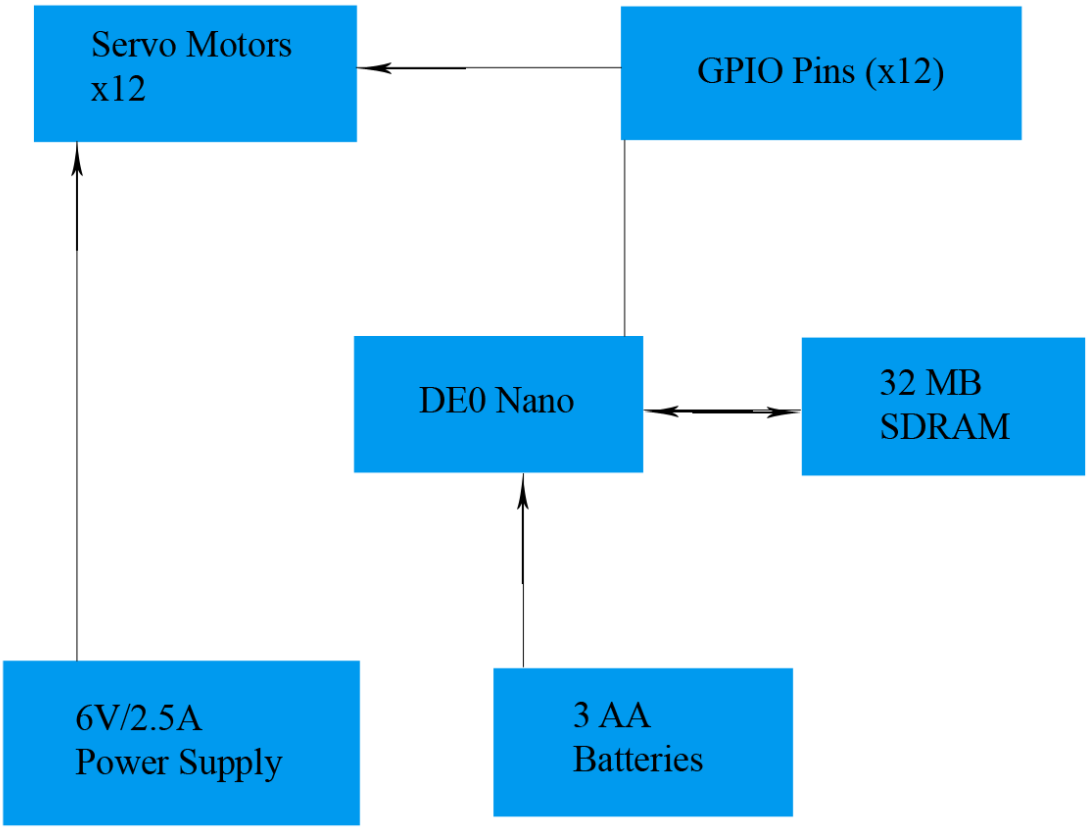

Figure 5: Hardware

Video

https://youtu.be/8WfGlfbsVwo

Source Code

See uploaded files for the remainder of source code.

multi_pwm.h

```
#ifndef MULTI_PWM_H_
#define MULTI_PWM_H_
// PWM channel adresses



#define PWM1(data)    IOWR(MULTI_PWM_0_BASE+(0<<2), 0, data)
#define PWM2(data)  IOWR(MULTI_PWM_0_BASE+(1<<2), 0, data)
#define PWM3(data)  IOWR(MULTI_PWM_0_BASE+(2<<2), 0, data)
#define PWM4(data)  IOWR(MULTI_PWM_0_BASE+(3<<2), 0, data)
#define PWM5(data)  IOWR(MULTI_PWM_0_BASE+(4<<2), 0, data)
#define PWM6(data)  IOWR(MULTI_PWM_0_BASE+(5<<2), 0, data)
#define PWM7(data)  IOWR(MULTI_PWM_0_BASE+(6<<2), 0, data)
#define PWM8(data)  IOWR(MULTI_PWM_0_BASE+(7<<2), 0, data)
#define PWM9(data)  IOWR(MULTI_PWM_0_BASE+(8<<2), 0, data)
#define PWM10(data)  IOWR(MULTI_PWM_0_BASE+(9<<2), 0, data)
#define PWM11(data)  IOWR(MULTI_PWM_0_BASE+(10<<2), 0, data)
#define PWM12(data)  IOWR(MULTI_PWM_0_BASE+(11<<2), 0, data)
#define control(data)  IOWR(MULTI_PWM_0_BASE+(12<<2), 0, data)
#endif /*MULTI_PWM_H_*/
```

multi_pwm.vhd

```
library altera;
use altera.altera_europa_support_lib.all;
-- Created by Darius Grigaitis 2009 www.grigaitis.eu
-- Repurposed for use by Group 5, uAlberta ECE 2016
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity multi_pwm is
generic( W:integer :=15);
   port (
      -- Avalon MM-----------
       clk : in std_logic;
     reset_n : in std_logic;
     readas : in std_logic;
     writas : in std_logic;
     chipselect : in std_logic;
     address : in std_logic_vector(5 downto 0);
     readdata : out std_logic_vector(31 downto 0);
     writedata : in std_logic_vector(31 downto 0);

    PWM1, PWM2, PWM3, PWM4, PWM5, PWM6, PWM7, PWM8, PWM9, PWM10,
PWM11, PWM12: out std_logic
      );

end multi_pwm;

architecture PWM of multi_pwm is
 signal  pwm_counter, pwm_value1, pwm_value2, pwm_value3,
 pwm_value4, pwm_value5,pwm_value6, pwm_value7, pwm_value8,
pwm_value9, pwm_value10, pwm_value11, pwm_value12 :
std_logic_vector(W downto 0);
 signal control_reg:  std_logic_vector(7 downto 0);
begin
process (clk, reset_n, chipselect)
begin
if reset_n='0' then
pwm_counter<=(others=>'0');
pwm_value1<=(others=>'0');
pwm_value2<=(others=>'0');
pwm_value3<=(others=>'0');
pwm_value4<=(others=>'0');
pwm_value5<=(others=>'0');
pwm_value6<=(others=>'0');
pwm_value7<=(others=>'0');
pwm_value8<=(others=>'0');
pwm_value9<=(others=>'0');
pwm_value10<=(others=>'0');
pwm_value11<=(others=>'0');
pwm_value12<=(others=>'0');
elsif clk'event and clk='1' then
```

```
        ---------------------------------- PWM set ----------------
---------
           if address = "000000" and writas = '0' then -- PWM UPDATE
COUNTER

               pwm_value1<=writedata(W downto 0);
             end if;

              if  address = "000001" and writas = '0' then -- PWM
UPDATE COUNTER
               pwm_value2<=writedata(W downto 0);
             end if;

              if address = "000010" and writas = '0' then -- PWM
UPDATE COUNTER
               pwm_value3<=writedata(W downto 0);
             end if;

              if address = "000011" and writas = '0' then -- PWM
UPDATE COUNTER
               pwm_value4<=writedata(W downto 0);
             end if;

              if address = "000100" and writas = '0' then -- PWM
UPDATE COUNTER
               pwm_value5<=writedata(W downto 0);
             end if;

              if address = "000101" and writas = '0' then -- PWM
UPDATE COUNTER
               pwm_value6<=writedata(W downto 0);
             end if;

                if address = "000110" and writas = '0'  then -- PWM
UPDATE COUNTER
                              pwm_value7<=writedata(W downto 0);
                            end if;
                if address = "000111" and writas = '0' then -- PWM
UPDATE COUNTER
                              pwm_value8<=writedata(W downto 0);
                            end if;
                if address = "001000" and writas = '0'  then -- PWM
UPDATE COUNTER
                              pwm_value9<=writedata(W downto 0);
                            end if;
                if address = "001001" and writas = '0'  then -- PWM
UPDATE COUNTER
                              pwm_value10<=writedata(W downto 0);
                            end if;
                if address = "001010" and writas = '0'  then -- PWM
UPDATE COUNTER
                              pwm_value11<=writedata(W downto 0);
                            end if;
                if address = "001011" and writas = '0'  then -- PWM
UPDATE COUNTER
```

```
                                pwm_value12<=writedata(W downto 0);
                              end if;

               if address = "001100" and writas = '0'  then -- PWM
UPDATE COUNTER
                 control_reg(7 downto 0)<=writedata(7 downto 0);
              end if;

              ----------------- PWM signal formation ------------

            pwm_counter<=pwm_counter+1;

            if (pwm_counter = "11110100001001000000") then
                     pwm_counter <= "00000000000000000000";
                end if;

                  if ((pwm_value1<pwm_counter)and (pwm_value1>0))
then
                   PWM1<='1';


                  else PWM1<='0'; end if;

                 if ((pwm_value2<pwm_counter)and (pwm_value2>0))
then
                   PWM2<='1';
                  else PWM2<='0'; end if;

                 if ((pwm_value3<pwm_counter)and (pwm_value3>0))
then
                   PWM3<='1';
                  else PWM3<='0'; end if;

                 if ((pwm_value4<pwm_counter)and (pwm_value4>0))
then
                   PWM4<='1';
                  else PWM4<='0'; end if;

                       if ((pwm_value5<pwm_counter)and
(pwm_value5>0)) then
                   PWM5<='1';
                  else PWM5<='0'; end if;

                       if ((pwm_value6<pwm_counter)and
(pwm_value6>0)) then
                   PWM6<='1';
                  else PWM6<='0'; end if;

                     if ((pwm_value7<pwm_counter)and
(pwm_value7>0)) then
                                       PWM7<='1';
                                     else PWM7<='0'; end if;

                     if ((pwm_value8<pwm_counter)and
(pwm_value8>0)) then
```

```
                                        PWM8<='1';
                                       else PWM8<='0'; end if;

                      if ((pwm_value9<pwm_counter)and
(pwm_value9>0)) then
                                        PWM9<='1';
                                       else PWM9<='0'; end if;

                      if ((pwm_value10<pwm_counter)and
(pwm_value10>0)) then
                                        PWM10<='1';
                                       else PWM10<='0'; end if;

                      if ((pwm_value11<pwm_counter)and
(pwm_value11>0)) then
                                        PWM11<='1';
                                       else PWM11<='0'; end if;

                      if ((pwm_value12<pwm_counter)and
(pwm_value12>0)) then
                                        PWM12<='1';
                                       else PWM12<='0'; end if;

end if;

end process;
end PWM;
```