Biologically Inspired Joint Control for a Humanoid Robot

DAMIEN KEE
School of Information Technology and Electrical Engineering
University of Queensland,
St Lucia, Australia, 4072
damien@itee.uq.edu.au

GORDON WYETH
School of Information Technology and Electrical Engineering
University of Queensland,
St Lucia, Australia, 4072
wyeth@itee.uq.edu.au

JONATHON ROBERTS
Commonwealth Science and Industry Research Organisation
Information and Communication Technology
PO Box 883, Kenmore, Australia, 4069
Jonathon.Roberts@csiro.au

The GuRoo is a 1.2m tall, 23 degree of freedom humanoid constructed at the University of Queensland for research into humanoid robotics. The key challenge being addressed by the GuRoo project is the development of appropriate learning strategies for control and coordination of the robot's many joints. The development of learning strategies is seen as a way to side-step the inherent intricacy of modeling a highly non-linear multi-DOF biped robot. This paper outlines the approach taken to generate an appropriate control scheme for the joints of the GuRoo. The paper demonstrates the determination of local feedback control parameters using a genetic algorithm. The feedback loop is then augmented by a predictive modulator that learns a form of feed-forward control to overcome the irregular loads experienced at each joint during the gait cycle. The predictive modulator is based on a Cerebellar Modeled Articulated Controller (CMAC) architecture. Results from tests on the GuRoo platform show that both systems provide improvements in stability and joint control tracking.

1. Introduction

Humanoid robots typically have many joints, and each joint is subject to complex and varying loads as the robot moves about. These loads are mostly due to the effect of gravity on the robot's mass, but also include centrifugal and Coriolis forces from the robot's complex motion. In addition, joints may become loaded and unloaded as the robot lifts and places its feet.

It is challenging to design controllers that maintain a high level of tracking and stability performance under such a range of load conditions. Typically the problem is addressed by calculating the forward model of the torques on the joints and appropriately compensating[1]. However humanoid robots are difficult to model mathematically and hence analytical determination of feed forward dynamics for model based control can be both a complicated and time consuming process. In addition, contact with the unpredictable loads from the real world and human interaction further complicates the modeling problem.

Biological controllers do not use an accurate model of the system rather incremental adjustment of control parameters is performed, based on the experience of the system. Initial response may be quite crude, but over time appropriate control parameters are learnt. Neural networks hold some promise in the field of trajectory control with the ability to learn system dynamics without an explicit representation of a robot's configuration.

This paper presents a biologically inspired control scheme as a two-stage approach. A Genetic Algorithm determines a set of feedback control parameters based on a fitness function minimizing both tracking error and vibration experienced by each joint. This process is performed offline on a simulator. The tuned feedback system is augmented with a feedforward system that uses an on-line learning algorithm based on a Cerebellar Modeled Articulated Controller (CMAC) neural network. The GuRoo humanoid robot with its high degree of freedom and non-linear dynamics forms a suitable platform to apply the system.

1.1. Previous Work

There are numerous traditional methods available to the control engineer to tune feedback controllers. A vast amount of material has been published in the area. A summary of the most popular techniques is given in O'Dwyer[2]. It should be noted however, that most of these deal with the tuning of a single controller in isolation. Since the early 1990's, Genetic Algorithms (GAs) have been successfully used to tune PI and PID controllers[3,4,5]. As with more traditional approaches to tuning, most of these GA methods have only been applied to single controllers although Bomfin *et al.*[6], use GAs to simultaneously tune multiple power system damping controllers.

This use of GA's within the humanoid field of research differs from other approaches in its tuning of control parameters as opposed to the generation of complete walking gaits. Endo *et al*[7] employ a GA to determine the necessary joint velocities required to maintain stable walking. Shan *et al*[8] makes use of a central pattern generator to provide the required joint commands. The CPG consists of a set of neurons for each joint, with each neuron comprising a pair of neural oscillators. The weights between connecting neurons are determined through the use of a genetic algorithm.

The use of a cerebellum models for motion control has been studied in the past. Infants of approximately 5 months of age display multiple accelerations and decelerations when moving an arm[9]. This series of sub-movements eventually guides the arm to the desired position. Over time, and with more experience, the child learns the required muscle movements to smoothly guide the arm. This shows that the human body is not born with a perfect plant model, but in fact learns it through experience.

The Cerebellum Model Articulated Controller (CMAC) developed by Albus[10] is an artificial neural network architecture based on the human cerebellum. It is well suited to the application of robot motor control with a simple algorithm, fast learning and ability to generalize.

Fagg *et al*[11] implemented a CMAC control system on a 2 degree of freedom arm, actuated by three opposing sets of muscles. The CMAC is responsible for the co-ordination of these three actuators to control the two joints. When the CMAC does not bring the arm to the required position, an additional external CMAC was engaged that produces short sharp bursts of motor activity until the target was reached. Once the desired position was reached, the trial was terminated and a new trial initiated. Over time, the external CMAC was made redundant as the original CMAC correctly learned the required muscle commands.

Collins and Wyeth[12] used a CMAC to generate the required velocities needed for a mobile robot to move to a waypoint. Significant sensory delay was introduced that would cripple a traditional control system. The CMAC was able to learn the system dynamics, compensate for this delay and produce the required signals necessary to move to the waypoint with a smooth velocity profile.

1.2. Paper Overview

Section 2 describes The GuRoo, the humanoid platform constructed at the University of Queensland, to which the research is applied. Section 3 presents the genetic algorithm approach used to obtain a set of PI control parameters for each joint. These parameters are compared to a set of hand tuned parameters. Section 4 outlines the CMAC neural network implemented on the robot. A crouching experiment is undertaken and results from before and after the implementation of the system are presented. The final section draws conclusions from these results and discusses where these results may lead.

2. GuRoo Project

GuRoo is a fully autonomous humanoid robot (Figure 1) designed and built in the University of Queensland Robotics Laboratory[13]. The robot stands 1.2 m tall and has a total mass of 40 kg, including on-board power and computation. GuRoo is currently capable of a number of demonstration tasks including balancing, walking, turning, crouching, shaking hands and waving.

The eventual aim of the project is to construct a robot that can play a game of soccer with or against human players or other humanoid robots. The current aim is to achieve reliable and robust locomotion. GuRoo has been designed to mimic human form and function to a limited extent, taking into consideration conflicting factors of function, power, weight, cost and manufacturability.

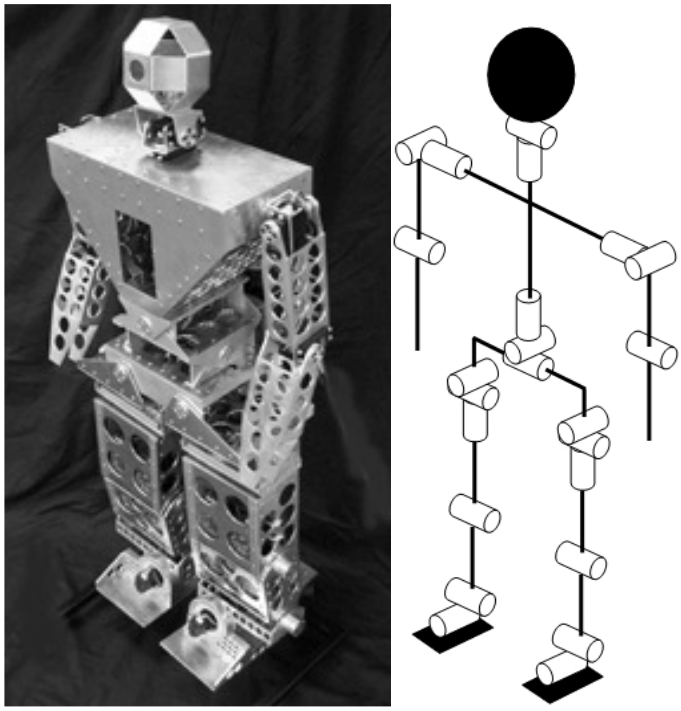

Figure 1 : The GuRoo humanoid robot with a schematic showing the degrees of freedom.

2.1. Electro-Mechanical Design

The robot has 23 joints in total. The legs and spine contain 15 joints that are required to produce significant mechanical power, most generally with large torques and relatively low speeds. The other 8 joints drive the head and neck assembly, and the arms with significantly less torque and speed requirements.

Table 1 outlines the type and axis of actuation of each motor. Due the high power / low velocity nature of these joints, large gearboxes are used which contribute to the length of the actuators which in turn lead to unnaturally wide legs.

The centre of gravity of each leg lies outside the line of the hip rotation, and as such, the legs naturally swing inwards. The motors that drive the roll axis of the hip joints are each supplemented by a spring with a spring constant of 1 Nm/degree. These springs serve to counteract the natural tendency of the legs to collide, and help to generate the swaying motion that is critical to the success of the walking gait.

Table 1: Type and axis of each DoF. "2 x" indicates a left and right side.

Joint	Type	Axis	No.
Head/Neck	RC Servo	Pitch + Yaw	2
Shoulder	RC Servo	Pitch + Roll	2x2
Elbow	RC Servo	Pitch	2x2
Spine	DC Brushed	Pitch + Roll + Yaw	3
Hip	DC Brushed	Pitch + Roll + Yaw	2x3
Knee	DC Brushed	Pitch	2x1
Ankle	DC Brushed	Pitch + Roll	2x2
		TOTAL	23

2.2. Distributed Control Network

A distributed control network controls the robot (Figure 2), with a central computing hub that sets the goals for the robot, processes the sensor information, and provides coordination targets for the joints. The joints have their own control processors that act in groups to maintain global stability, while also operating individually to provide local motor control. The distributed system is connected by a CAN network. In addition, the robot requires various sensor amplifiers and power conversion circuits.

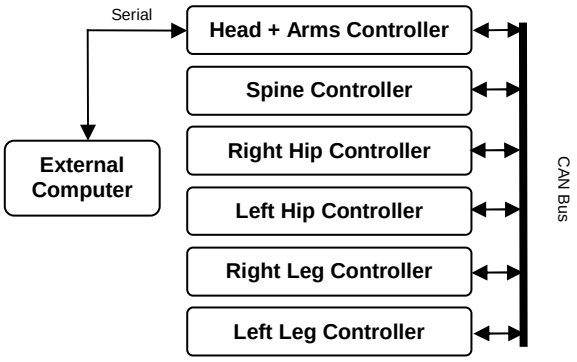

Figure 2: Block diagram of the distributed control system

2.3. Sensors

The GuRoo currently has encoders on each of the high powered DC motors, able to provide rotational position to a resolution of 0.001 of a degree. An inertial measurement unit consisting of 3 rate gyroscopes and 3 accelerometers has been obtained that is currently being integrated into the system. Provision has been made for the future inclusion of pressure sensors on the soles of the feet and a stereo vision system.

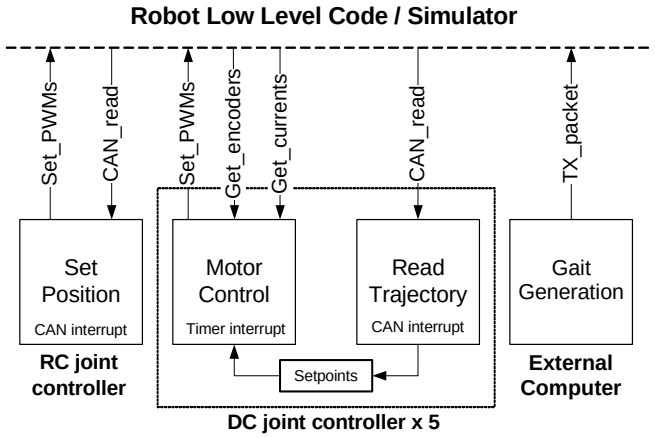

Figure 3 : Block diagram of common software modules and the interface used by both the real robot and the simulator.

2.4. Software

The software comprises four major parts: the simulator, gait generation, joint controller software and low level firmware. The gait generation and joint control software have standard API's (Figure 3) and can be compiled to either the simulator or the low level firmware to be run on the actual robot. The joint velocities required by the joint controllers are provided by the gait generator module.

2.4.1 Gait Generation Software

Different gaits such as walking and crouching are used to evaluate the control architecture presented, and while not the focus of the paper, are briefly described below.

The gait generator yields walking gaits that are inherently Zero Moment Point (ZMP) stable, and are based on a constrained locus of motion[14]. The offline gait generator makes use of a low-resolution simulation and evolutionary computation. This approach involves learning a rough solution with an approximated model. Evolutionary computation is used for evolving walking gaits from a random population. Each gene is encoded to represent the motion sequence in terms of ankle positions. Limited information of the model on separation of legs and ratio of upper leg to lower leg is used for the calculation of inverse kinematics. An approximate mass distribution of the robot body is used for calculating the ZMP. The fitness is then measured by comparing the estimated ZMP trajectory of the playback of a motion sequence with a desired ZMP trajectory that meets stability criteria (with the projection of ZMP lying within the support polygon).

After generations of evolution, a solution for a stable gait pattern for the approximated model results. This pattern can then be used for the generation of trajectories for the legs, calculating the velocities of the joints at 50Hz.

The gait generator module can also generate smooth arbitrary motion of limbs for crouching, bowing and waving. Joint velocities follow a sinusoidal velocity of the form:

$$\omega = \frac{\theta}{T}\left(1 - \cos\left(\frac{2\pi t}{T}\right)\right)$$

where ω is the desired joint velocity, θ is the desired change in joint angle and T is the period of the movement. This profile ensures the joint experiences zero acceleration at the start and end of each movement which minimises the jerk and vibration. Zero velocity at the start and end of each trajectory allow for smooth transitions between trajectories.

2.4.2 Joint Controller Software

Located in the robot are two types of joint controllers, five DC brushed motor controllers and one RC servo motor controller. The CAN interrupt routine updates the desired joint velocities. The RC controller runs a single timer interrupt routine that sets the required pulse length for each RC servo motor based on the desired joint velocity.

An interrupt driven CAN routine updates the desired joint velocity for each motor controlled by a DC motor controller. A second timer interrupt occurs at 250Hz to run the motor control routine. The actual velocity is the change in encoder readings over time. The motor control routine calculates the error between the actual velocity and the desired velocity. A PWM value is calculated from this error using a Proportional-Integral control law. The PI control terms are software based and can by adjusted dynamically at any point in the gait.

The current is measured every control loop and the PWM waveform generated is modified to keep within a software set current limit.

2.4.3 Humanoid Simulator

The simulator is based on DynaMechs[15], a dynamic simulation tool for multi-chained, star configured robots. It has been adapted to include specific characteristics of the GuRoo, including the distributed nature of the control architecture and the CAN bus. The GuRoo's chest is modelled as a mobile base with 5 chains arranged in a star configuration representing the arms, legs and head. The modified Denavit-Hartenburg parameters and CAD surface area provide the graphical representation of the robot as seen in Figure 4. Mass distribution information in the form of inertia tensors is combined with actual motor characteristics to provide realistic interactions between links. The simulator provides the same interface as the firmware, with the ability to read encoders, measure current consumption and to transmit and receive CAN packets.

2.4.4 Firmware

The firmware present on each joint controller board provides direct access to the sensors, motor drivers and CAN bus. The firmware is accessed by a set of generic high level functions in the Joint Control module. The nature of these generic functions allows the control code to be microcontroller independent.

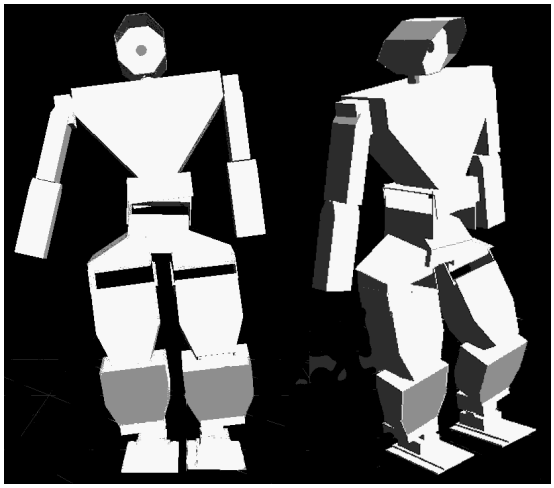

Figure 4 : Screenshot from the DynaMechs based simulator.

3. Control Parameters

The aim of the GuRoo project is to create a humanoid robot that can perform human-like motions, such as walking, crouching, reaching and kicking a ball. In order for these motions to be realised, it is essential that the lowest-level controllers, the joint controllers, perform well. Each joint controller is implemented as a simple PI controller around joint velocity. Each joint must therefore be tuned; good values of the P and I gains must be found for each of the 15 high-powered lower joints.

It should be realised that all joints must remain active during all phases of a motion. Even the act of standing up straight and remaining still requires all joints to be active. GuRoo simply slumps to the floor if it loses power. This implies that all 15 joint controllers must be tuned simultaneously in order to achieve the best outcome. An oscillation in one joint caused by a poorly tuned controller, will affect the performance of every other controller due to subsequent vibration and motion (generated at that joint) being transmitted through the robot to all the other joints.

3.1. Reducing the problem

Each of the 15 joints must be tuned simultaneously, each with two gains (a P and an I gain), making 30 gains in total. However, there is symmetry in GuRoo, and the legs can be considered to be identical. Hence, the problem maybe reduced to finding 18 gains (6 for the spine and 12 for a leg). In order to simplify the problem, we can restrict the possible values of the gains to a range that we know is roughly correct. It is unrealistic to tune all 15 controllers simultaneously without restricting the values of the gains. The robot must actually walk most of the time during the tuning experiments.

The PI controllers are implemented on the motor controller boards. These boards have limited processing power and hence the controllers have been implemented using a bit shifting strategy for speed. Hence, the P and I gains are not floats, but integers with unit values of 1, 2, 4 and 8 which corresponds to bit shifting values of 0, 1, 2 or 3. This allows gains to be represented as a 2-bit number. When the output of the PI control law is computed, the proportional error and integral error are shifted the number of bits corresponding to the P and I gains respectively. The 36 bit genome was constructed by

concatenating the 18 x 2 bit gains. From experimentation a range of gains that allow GuRoo to walk (at least very roughly) was gathered.

The GuRoo simulator provides a suitable environment on which to run a GA. Running the GA on the real robot would be difficult due to the constant need to move the robot back to the starting position every run and the problem that a poor run, where the gains are sub-optimal, may result in a highly unstable robot, which may fall over.

3.2. Fitness function — tracking and smoothness

As with all applications of GAs the key to success is finding the best fitness function that describes the performance of the system. In the case of GuRoo's joint controllers, of most interest is the optimization of two measures, tracking performance (minimising error) and tracking smoothness. The biggest single problem with the hand-tuned controllers was the noisy tracking performance which would give the robot a high frequency shake. The overall fitness function, *f*, used was therefore based on two separate fitness functions based on tracking error (f_t) and smoothness (f_s):

$$f = \frac{5.0}{f_t} + \frac{3.0}{f_s},$$

$$f_t = \sum_{j=1}^{N} \sum_{t=1}^{T} \left| p_{err}^{j}(t) \right| \frac{\Delta t}{T}$$

$$f_s = \sum_{j=1}^{N} \sum_{t=1}^{T} \left| \ddot{p}_{err}^{j}(t) \right| \frac{\Delta t}{T}$$

where $p^j_{\mathrm{err}}(t)$ is the joint tracking error of joint j at time t.

3.3. The GA and its parameters

The type of GA used for tuning was the 'simple GA' described in Goldberg[16]. The GAlib C++ library was used to run the GAs[17]. A simple GA uses non-overlapping populations and optional elitism, and creates an entirely new population of *f* individuals each generation. The GA parameters used were as follows:

- Population size: 100
- Number of populations: 100
- Crossover: 60%
- Mutation: 10%
- Genome: a 36-bit long bit string of concatenated 18 2-bit gains

The GA was run over a complete walking cycle which consists of eight elements:

- right foot strikes ground
- right foot on ground
- left foot toe off
- left knee straighten
- left foot strikes ground
- left foot on ground
- right foot toe off
- right knee straighten

The walking cycle is produced by the gait generation software described in Section 2.4.1. The simulation begins with the robot in the standing position. This pose is not part of the walking gait and so the robot must first move from the standing position into a walking pose. This takes place using an extra two elements. The first cycle of the walk is therefore 'non-standard' and hence the fitness function is only computed over the second walk cycle. Each walk cycle takes 6 seconds to run on a 2GHz PC (real-time) and so a complete run of a single GA individual takes 12 seconds. The total time to run the GA as described above was 1000 minutes (16.6 hours). Note that an exhaustive search for the optimal gains would take a little over 68,000 years to complete on the same PC.

3.4. Simulation Experiments

The gains generated by the GA tuning method and the original hand-tuned gains were run on the simulated GuRoo over a number of walking cycles. The results for both cases are compared in Table 2. This table gives the break-down of the components of tracking and smoothness performance. From this table it can be seen that both tuning methods track with approximately the same overall tracking accuracy (f_t). However, this is not the case for the smoothness measure (f_s) which shows that the GA tuning method is far superior to the hand tuning method by a factor of 2.

Table 2 : Comparison of fitness measures obtained on the simulator (a higher score is better).

Tuning Method	Tracking $\left(\frac{5.0}{f_t}\right)$	Smoothness $\left(\frac{5.0}{f_s}\right)$
Hand Tuned	0.116	0.152
GA Tuned	0.112	0.310

3.5. Real Robot Experiments

The gains generated by the GA tuning method and the original hand-tuned gains were run on the real GuRoo robot over a number of walking cycles. A comparison of the results is given in Table 3. This table gives the break-down of the components of tracking and smoothness performance. From this table it can be seen that the GA tuning method performs best with respect to tracking and smoothness.

Table 3 : Comparison of fitness measures obtained on the real robot (a higher score is better).

Tuning Method	Tracking $\left(\frac{5.0}{f_t}\right)$	Smoothness $\left(\frac{5.0}{f_s}\right)$
Hand Tuned	0.009	0.539
GA Tuned	0.011	0.699

It is interesting to note that the results differ significantly from the simulation case in that during the real experiments, the tracking performance was much worse, but the smoothness performance was slightly better. This can be explained by the fact that the real robot is much more compliant than the simulation model and hence 'wobbles' more when walking. This compliance has a natural smoothing tendency (hence smoother operation) but makes joint control challenging (hence larger tracking errors). Figure 5 shows a comparison of the joint error tracking performance for the right ankle pitch joint

during a section of the real walk. This is the same section of the walk used for the simulation results above. The figure for the right ankle shows a remarkable difference between the tuning methods. The comparative gains for this joint were P=1 and I=4 for the hand-tuned gains, and P=2, I=1 for the GA tuned method. This difference in gains produces very different results.

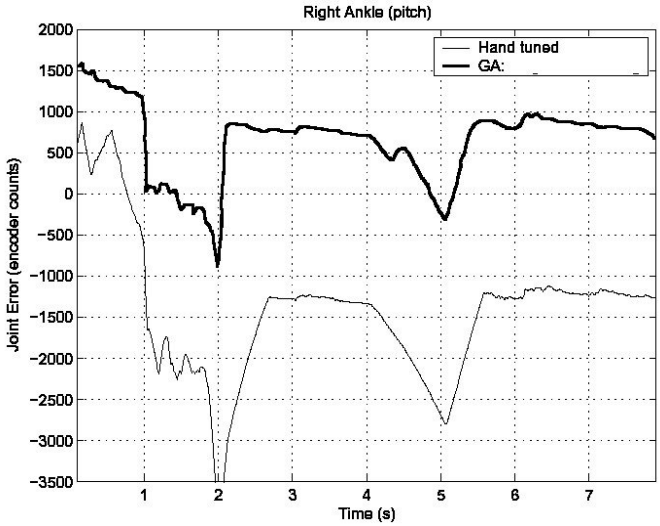

Figure 5 : Comparison of joint error for the hand-tuned and GA tuned control parameters for the ankle pitch joint.

4. CMAC Augmented Control

The joints of a humanoid robot experience disturbances of markedly different magnitudes during the course of a walking gait. Consequently, simple feedback control techniques poorly track desired joint trajectories. In this section, we present a CMAC augmented system that learns to compensate the changes in load that occur during a cycle of motion. The joint compensation scheme, called Trajectory Error Learning, augments the existing feedback control loop on the GuRoo.

4.1. CMAC Neural Network

The Cerebellar Model Articulated Controller (CMAC) was first described by Albus. The CMAC network can be viewed as a number of lookup tables. Each table, or Association Unit (AU), has the dimensions equal to the number of input variables.

The first step requires the inputs to the system to be quantized and scaled, giving a lookup address into each AU (Figure 6a). This address is mapped to a coarser address space in each AU which activates a single cell known as a State Space Detector (SSD) (Figure 6b). There is a weight associated with each SSD. Each network comprises of a collection of *n* AU's and as such for any unique set of inputs, there are exactly n SSD's activated (Figure 6c).

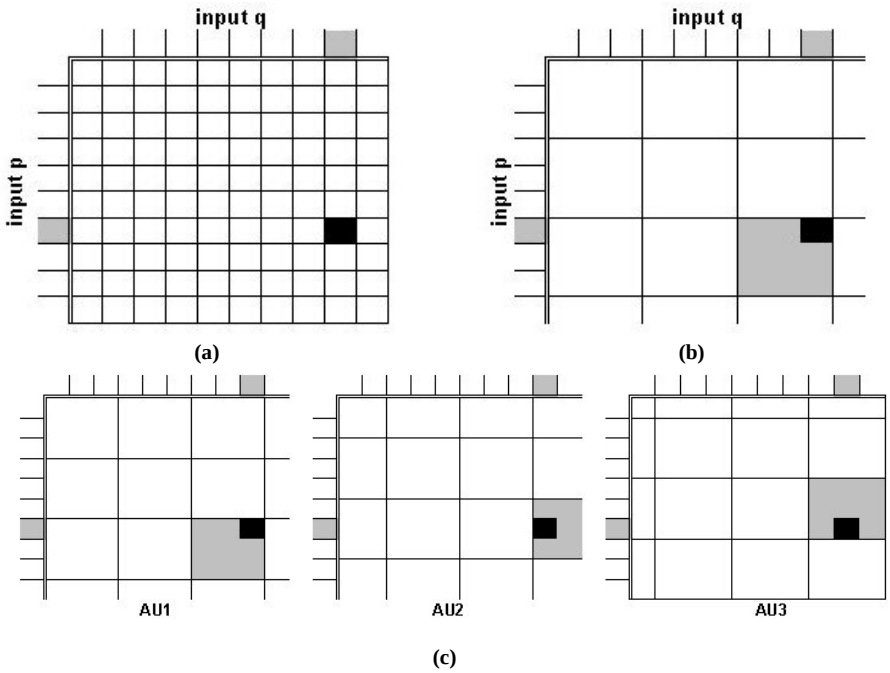

Figure 6 : A single Association Unit. (a) Inputs are scaled and quantised to give a single table entry. (b) This is then mapped to a coarser address space where a weight is obtained from the activated State Space Detector (SSD). (c) Multiple Association Units. The system consists of multiple AU's each which activate a single SSD. The weights from each SSD are summed to give an output response.

The output signal is calculated by finding the sum of the weights of all AUs at this lookup address (Figure 7). The AUs are structured such that should an input value change by 1 unit of resolution, all but one SSD will remain unchanged. For example, if there are 10 AU's, then a particular combination of input values will activate 10 SSD's and hence yield 10 independent weights. Should one of the inputs change by one step of resolution, nine of the original SSD's will remain activated, with only one SSD deactivated and a new one activated.

As the output result is the sum of all association unit's weights, a greater number of Association Unit's results in a system that is better able to generalize the input space. Small deviations in an input signal, do not greatly affect the output signal.

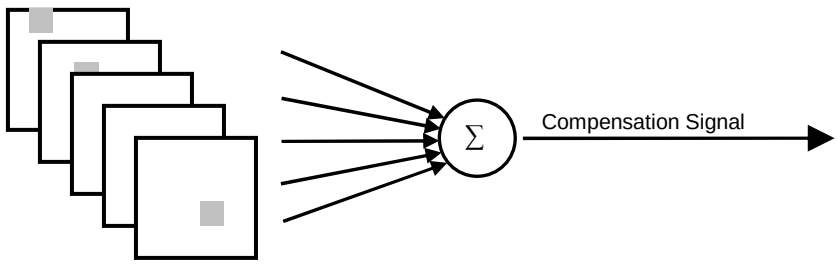

Figure 7 : The weights of each selected SSD is summed to give a compensating signal.

The input space is dominated by hyperplanes of plausible input combinations, with large empty spaces in each AU where real-life input combinations are not physically possible. Hashing techniques are used to reduce the memory requirements by mapping the complete set of SSD's to a smaller set of virtual SSD's using the modulo function. Hash collisions occur when two or more SSD's hash to the same virtual SSD. This is not necessarily fatal, as a large number of AUs will ensure the table weight in question to have a minor effect on the overall output.

Table weights are updated using the following rule:

$$\varpi_{new} = \varpi_{old} + \frac{\alpha}{\eta}(\theta_{des} - \theta_{act})$$

where:

ω_{new}	=	New weight value
ω_{old}	=	Original weight value
α	=	Learning rate
η	=	Number of association units
θ_{des}	=	Desired joint position
θ_{act}	=	Actual joint position

As the output of the response of the network is the sum of the selected table weights, the change in weight between iterations is divided by the number of association units to ensure the learning rate has the same effect regardless of the number of AUs.

4.2. Joint Position Error

Figure 8 demonstrates the error in the left knee joint during a typical walking gait cycle. At t=0, the foot is lifted from the ground and becomes the swing leg. The large error is indicative of the high gain necessary whilst in the single support phase, applied during the period of minimal load during the swing phase. The foot regains contact with the ground at t=0.4. At t=0.8 the knee enters the single support phase and is heavily loaded as reflected in the average joint error of -3 deg. The double support phase is active from t=1.2 until t=1.6 at which point the left leg again loses contact with the ground. This highlights the significant variations in load that prevent the PI control loop implemented on each of the GuRoo's joints from obtaining a satisfactory response.

TEL uses a CMAC network to supply a compensating signal to eliminate this position error. As a typical walking gait of a humanoid is periodic in nature any errors experienced by the robot are also typically cyclic in nature: for example, the joint error during the support phase. By observing the gait phase, the CMAC learns which parts of the gait require compensation.

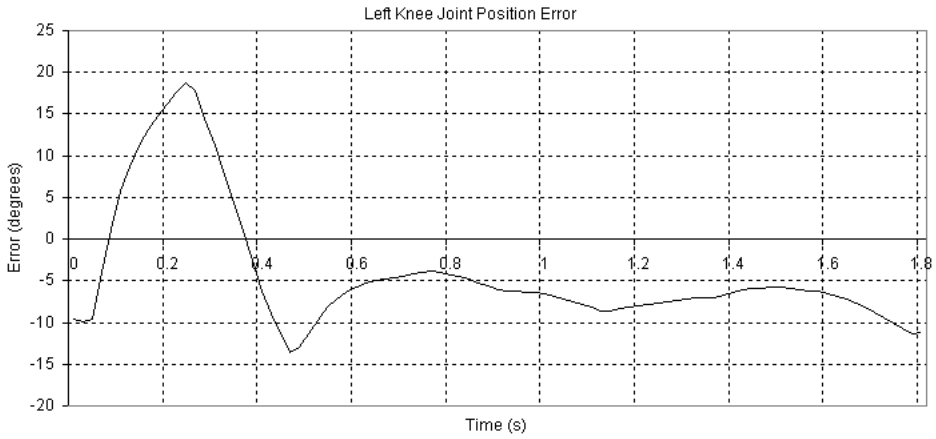

Figure 8 : Position error experienced in the left knee pitch joint over one complete walking cycle.

4.3. System Implementation

The method of compensating the joint error is illustrated in based on Trajectory Error Learning. Trajectory Error Learning (TEL) is a biologically inspired method of robot motion compensation based on the CMAC where learning is driven from the difference between the intended trajectory of the robot and the actual trajectory measured by the feedback sensor (possibly after some sensory delay)[18]. The system implemented on each joint of the robot is outlined in Figure 9. The trajectory of the limb is expressed as a stream of desired joint positions which are generated by the gait generator. As the motion of the robot is periodic, the state of the trajectory can be expressed as the gait phase. The gait phase is implemented as a periodic counter, incrementing every control loop and resetting at the beginning of each motion cycle.

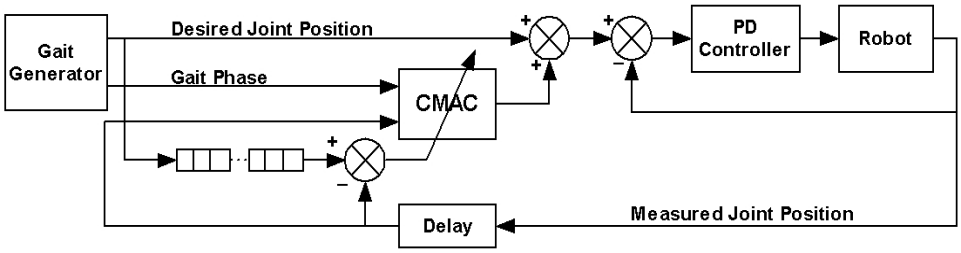

Figure 9: Control system diagram for a single joint on the robot. The Desired Joint Position is time delayed when calculating position error to account for sensor delay. Table weights are updated a set number of control loops after being used. This delay is equal to the sensory delay inherent in the system.

The desired joint position is augmented by the output of the CMAC, and passed to the feedback joint controller. The inputs to the CMAC consist of the gait phase and the measured joint position, where the measured joint position will be subject to some delay with respect to the desired joint position. In this form, the CMAC is used as a predictive modulator; seeking to eliminate the error it expects to see based on the errors that it has already seen at the same point in previous cycles. The sawtooth wave of the gait phase gives the CMAC the point in the cycle that it is currently compensating, while the

measured joint position accounts for different disturbance conditions that may occur at that point in time. If a typical gait movement is 500 gait phase units long, then a gait phase of 0 and 500 differ by only 1 input unit. The CMAC is modified to reflect this.

The error in joint position is used to train the CMAC network. A history of previous desired position commands is kept to compensate for the sensory delay experienced in the system. This history buffer also ensures weight updates are performed with the correct time delayed error. The error signal used to train the CMAC is as follows:

$$\xi_k = \theta_{des(k-t)} - \theta_{act(t)}$$

where:

ξ_k	=	Error training signal (k)
$\theta_{des(k-t)}$	=	Desired Joint Position (k - t)
$\theta_{act(k)}$	=	Actual Joint Position (k)
t	=	Sensory Delay

Thus weights are updated t control loops after they are used.

4.4. Crouching Experiment

To test the validity of the cerebellar modeled compensation system, a simple crouching motion is performed. Motion is generated using the sinusoidal generator described in section 2.4.1. This initial experiment conducted is based on a slow crouching motion run over a period of 6 seconds. The pitch axis motors of the hip, knee and ankle joints follow a synchronised sinusoidal profile with a magnitude of 18, 35 and 22 degrees respectively to reach the bottom of the crouch position. This test exposes the joints to a range of dynamic and static loads, and can be repeated many times without moving the robot around the laboratory.

For this experiment, the CMAC parameters outlined in Table 4 were chosen. The number of receptive units and field width were chosen to provide the necessary discrimination, while also providing local generalisation. The Joint Position Receptive Units are scaled to only the angular range experienced during the crouching motion. The hashing ratio was chosen to reduce memory requirements while still keeping a low probability of hashing collisions. The learning rate was tuned to provide rapid learning of the compensation, without learning from noise. The measured joint positions were subject to a delay of 40 ms, which corresponds to a delay of 3 control cycles.

Table 4: CMAC system parameters.

CMAC Parameter	Value
Joint Position receptive units	1000
Gait Phase receptive units	1000
Field width (AU's)	100
Global Address Spaces	216204
Virtual Address Spaces	10001
Learning rate	0.005

4.5. Results

The learning algorithm was initially applied only to one type of joint at a time to negate co-evolutionary effects between different joint types. Figure 10, Figure 11 and Figure 12 outline the uncompensated and compensated response of each joint used in the crouching motion.

Without TEL, the joints involved in the crouching motion experience an error in position which is cyclic, these errors do not change from cycle to cycle and are dependent on the current phase of gait, with larger errors present in the second half of the cycle as the robot accelerates itself upwards. In contrast, the error present in the compensated system steadily decreases to an acceptable noise floor.

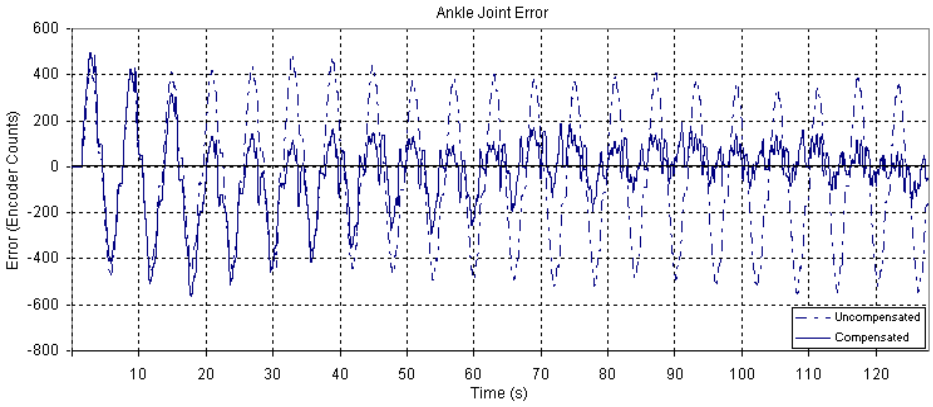

Figure 10 : Compensated and Uncompensated ankle pitch joint error during a series of crouching motions.

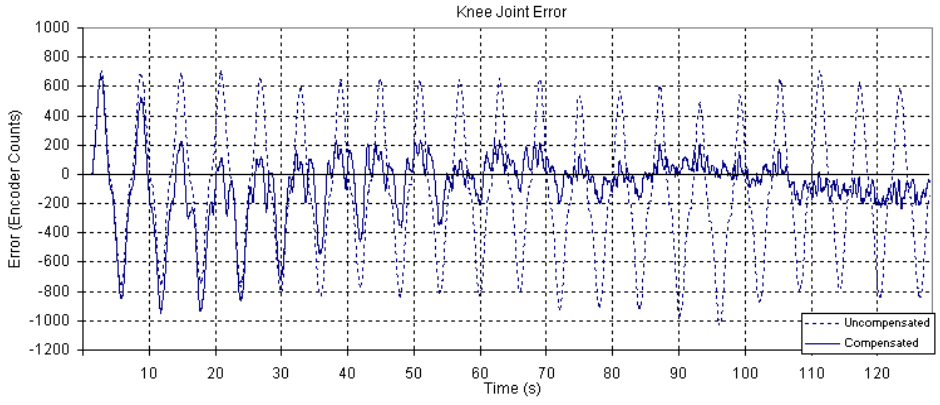

Figure 11 : Compensated and Uncompensated knee joint error during a series of crouching motions.

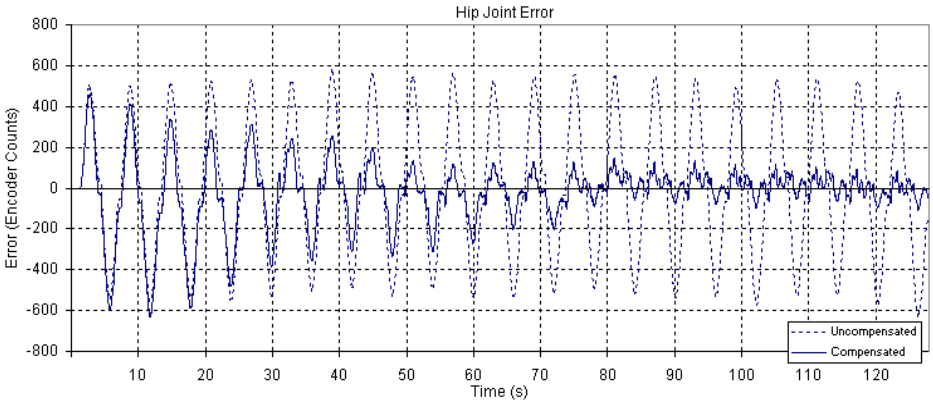

Figure 12 : Compensated and Uncompensated hip pitch joint error during a series of crouching motions.

Figure 13 outlines the compensation signal generated by the CMAC for the left knee and is indicative of all other joints. As can be seen the compensation signal is initially not present but over time increases to a cyclic signal. As the error in position is reduced, so too is the change in compensation signal.

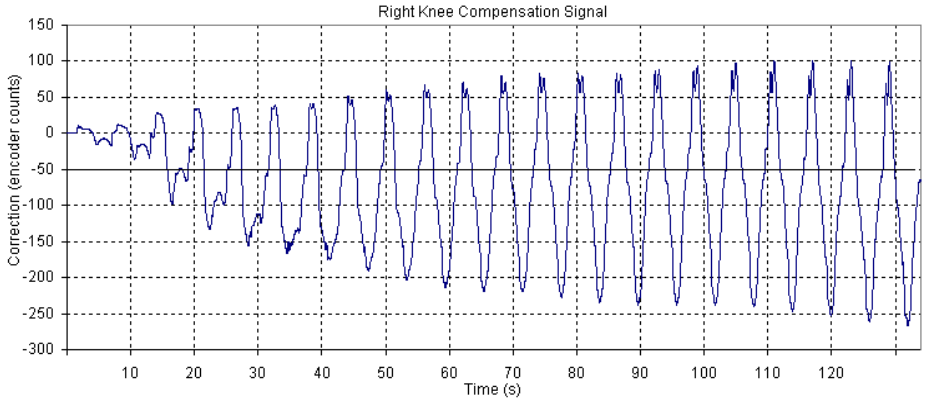

Figure 13 : Compensation signal generated for the right knee during the crouching experiment.

Table 5 outlines the results achieved for all pitch movements involved in the crouching motion. It can be seen that a similar performance increase was obtained on all six joints involved in the motion.

Table 5: Joint Error Reductions using the TEL system summarising peak error in position for all joints before and after learning. Peaks errors are averaged over several cycles.

Joint	Peak Before	Peak After	Reduction
Left Hip	500	100	80%
Right Hip	500	100	80%
Left Knee	800	200	75%
Right Knee	800	200	75%
Left Ankle	500	100	80%
Right Ankle	500	100	80%

5. Conclusion

Hand tuning control parameters is time consuming and the resulting system is not guaranteed to be optimal. By applying a genetic algorithm approach to the parameter selection, a set of control values with better tracking and smoothness properties was obtained.

Feedback control techniques alone however, are unsuitable for control of a humanoid robot. The extreme differences in load throughout the gait, and particularly during the swing phase versus the single support phase, make feed-forward compensation necessary. Modeling the plant dynamics of a mobile body with so many degrees of freedom is a difficult task. In this paper we have shown a system that learns to provide suitable feed forward compensation in an on-line and real-time fashion.

The simple crouching experiment demonstrates the existing control loop's inability to compensate for changes in load as a result of gravity. The error between the desired and actual joint position, the trajectory error, is used by the cerebellar system to learn a response capable of decreasing the peak error by 80%. The experiments were conducted with a crouching motion on a real 23 degree of freedom humanoid and show marked reduction in position error of all joints with the implementation of the TEL system.

5.1. Further Work

In this implementation, suitable compensation of tracking error has been achieved for a crouching motion. It will shortly be trialed on a walking gait for improvement of walking performance. In a walking gait, a humanoid encounters three distinct stages, single support, double support and swing phase. The use of a genetic algorithm to determine an initial three sets of control parameters is shortly to be trialed.

The TEL system used as the basis of this work is suited to many control problems where a tracking error is present. Within a humanoid robot, there are many trajectories that can be improved to enhance stability. Torso inclination, location of the Zero Moment Point and centre of foot pressure all follow a ‘desired’ path. Deviations to this path can be measured and a trajectory error calculated. This error can be used to train a separate TEL structured CMAC to improve balance and walking gaits.

Bibliography

1. J.J Craig. *Introduction to Robotics: Mechanics and Control*, Third Edition. Pearson Prentice Hall, 2004

2. O'Dwyer. *Handbook of PI and PID Controller Tuning Rules*. Imperial College Press, 2003.

3. Porter and A. H. Jones. Genetic tuning of PID controllers. *Electronics Letters*, 28(9):843–844, 23 April 1992.

4. P. Wang and D. P. Kwok. Optimal design of PID process controllers based on genetic algorithms. *Control Engineering Practice*, 2(4):641–648, 1994.

5. J. Herrero, X. Blasco, M. Martinez, J. Salcedo. Optimal PID tuning with genetic algorithms for non linear process models. *IFAC 15th Triennial World Congress*, Barcelona, Spain, 2002.

6. L. Bomfin, G. N. Taranto, and D. M. Falcao: Simultaneous tuning of power system damping controllers using genetic algorithms. *IEEE Transactions on Power Systems*, p163–169, 2000.

7. K. Endo, T. Maeno, H. Kitano: Co-evolution of morphology and walking pattern of biped humanoid robot using evolutionary. *ICRA* 2003: 1362-1367

8. J. Shan, C. Junshi, C. Jiapin: Design of Central Pattern Generator for Humanoid Robot Walking *Intelligent Robots and Systems*, pp.1930–1935, 2000.

9. Barto: Learning to reach via corrective movements, *Self-Learning Robots III Brainstyle Robotics*, pp. 6/1, 1999

10. J.Albus: A Theory of Cerebellar Function, *Mathematical Biosciences* Vol: 10, pp. 25-61, 1971

11. Fagg, N. Sitkoff, A. Barto, J. Houk: A model of Cerebellar Learning for Control of Arm Movements Using Muscle Synergies, *Computational Intelligence in Robotics and Automation*, pp. 6-12, 1997

12. Collins, G. Wyeth: Fast and accurate mobile robot control using a cerebellar model in a sensory delayed environment, *Intelligent Robots and Systems*, pp. 233-238, 2000

13. Kee, G. Wyeth, A. Hood, A. Drury: GuRoo: Autonomous Humanoid Platform for Walking Gait Research, *Autonomous Minirobots for Research and Edutainment*, 2003

14. Gordon Wyeth, Damien Kee, and Tak Fai Yuk. Evolving a locus based gait for a humanoid robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, Las Vegas, USA, October 2003.

15. S. McMillan: Computational Dynamics for Robotic Systems on Land and Underwater, PhD Dissertation, Ohio State University, 1995.

16. E. Goldberg. *Genetic Algorithms in Search and Optimization.* Addison-Wesley Pub. Co, 1989.

17. M. Wall. GAlib C++ library. http://lancet.mit.edu/ga/, 2000.

18. D. Collins: Cerebellar Modeling Techniques for Mobile Robot Control in a Delayed Sensory Environment, PhD Dissertation, University of Queensland, 2003