Biomimetic Control of a Hexapod Robot

Georgi Tsatsev

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2014

Abstract

Biomimetics in robotics offers an enormous potential for future technologies. Its main idea is to create a new class of robots that mimic biological organisms whose models and systems are based on millennia of evolution. This project involves the evaluation, implementation and testing of a biomimetic controller of a hexapod robot, based on stick insect data. The robot's core design is based on Lewinger's [14] adaptive hexapod robot and its single leg controller is based on Ekeberg's [10] neural network controller, one of the most famous studies for single leg stick insect stepping locomotion. Its main idea is that every leg has its own sensory-motor loop that is only dependent on the information of this leg's joints. Yiming Yao's [23] implementation of this single leg neural network controller (middle leg) is researched, tested and evaluated, other five legs' control networks and their respective parameters have been derived and tested based on Ekeberg's neural data [10]. A brain module control walknet is discussed for whole robot walking and turning as a future project improvement.

Acknowledgements

I would like to thank everybody that supported me throughout the year. I would like to give special thanks to my supervisor, Barbara Webb, for her patience and helpful guidance and Georgios Petrou, who helped me in getting the project started and was willing to come and help me tackle problems that were not in my expertise. Also I would like to give my sincere gratitude to my peers, friends and especially my family for everything they have done for me.

Table of Contents

Chapter 1

Introduction

Biomimetics (from Greek words bios, which means life and mimesis, which means to imitate) is a field of robotics that researches real biological organisms in order to inspire a new class of robots that show a more advanced and natural behavior. This new set of biologically inspired robots shows better performance and stability under perturbing conditions. This is all based on the fact that biological organisms have evolved to be as effective as possible in different situations. Mimicking the control mechanisms of such organisms provides a very robust control mechanism for robot locomotion. One of the most complicated parts of robot control is to make the movement as natural, robust and real as possible and biomimetics, through analysis and imitation of the organisms that surround us, aims to provide such control mechanisms.

Walking for example is one of the most common forms of terrestrial locomotion and it is one of the most challenging and researched topics in robot control. This is due to the fact that walking robots can traverse rough terrain whereas wheeled robots are at a disadvantage. In situations where the precision and way the movement adapts to a changing situation is where a walking robot shows better performance than a wheel-based one. For example search and rescue missions in real-world scenario a legged-based robot can deal with obstacles and move better under different perturbing conditions such as going over a big rock or dealing with uneven terrain. Another reason why walking has been researched in robotics is because by implementing biomimetic walking controllers on robots its behavior can be analyzed and the provided data can give important information about the biological system of the organism that the controller has been based on. Walking robots can be divided into bipedal, quadripedal, hexapedal , etc. depending on how many legs are involved in the process of walking. The most common ones are the quadripedal but research into hexapedal walking robots and organisms has been increasing recently because of the robust movements and stability of some six-legged animals. There are many neural-biological locomotion control models that exist for example stick insects (single legs: Cruse [4]; Schumm and Cruse [5] and hexapods: Cruse et al. [21]; Beer et al [19]), cockroaches(Pearson and Iles [17]), lampreys (Grillner [12]). The most common challenges with such biologically inspired designs are the tuning of neural parameters, and the adaptation of the robot while traversing rough terrain or turning.

Figure 1.1: Stick Insect [11]

One of the best studied organisms for hexapedal walking control based on neural information is the stick insect (Carausius Morosus Figure 1.1) and this honors project focuses on the implementation and evaluation of a hexapod control mechanism based on stick insect data. Despite their small brains their walking proves to be highly efficient and robust. This is because they have adapted to walk on rough terrain while blending with the environment, hence achieving walking with as little effort as possible. Inter-leg coordination rules based on stick insect behavior data have been proposed [4, 8] and tested in simulation and on robots [9, 3]. Neural network controllers have been derived from the neural-biological data of the stick insect [10]. It has been tested and evaluated by means of simulations and real robot implementation [14]. Ekebergs single leg model in particular is one that has proved to be a very effective locomotion control that uses the neuro-biological data as inputs and a modular structure of the leg segmentation. Each leg has its own control loop and its joints movement relying on feedback that they have received from the movement. Based on that Arndt von Twickel derived and simulated a neural network controller for a single-leg of a stick insect.

This neural network controller has been implemented on a real hexapod robot, previously built by Georgios Petrou during his PhD, by Yiming Yao (former MSc student in UoE). The robot's design is based on William Lewinger [13] and will be discussed in detail in the background chapter. Yao implemented and tested the model for the middle leg of the robot and discussed the implementation of the other legs based on von Twickel's [22] suggested neural network. Yao also researched methods of implementing each leg's separate controller on the whole robot by coupling the individual legs' feedback control loops via a brain module controller. This is in contrast to standard control techniques like coupled Central Pattern Generators [7] or trajectory control [1].

This honors project can be divided into three main parts. The first part was to evaluate

Yiming Yao's implementation of the single-leg control mechanism for each of the six legs and to make sure that the neural networks are working reliable for all the legs. I was provided with the code for one of the legs (the middle left leg) and had to understand his implementation and evaluate if it is working reliably on the single leg. This involved implementing two types of hind leg's neural network controllers based on Twickel's paper, substantial amount of testing of Yao's middle leg implementation, deriving the front leg neural network parameters and also figuring out how to implement a mirrored version of these controllers.

The second part was after each of the six legs' control networks had been thoroughly tested and the data gathered was compared to biological data on the stick insect leg movement. It involved implementing a basic walking controller within a brain module. The walking mechanics are based on biologically derived rules for leg coordination [6] and are discussed in more detail in the implementation chapter. The resulting hexapod controller is similar to the alternating tripod gait, where three legs are on the ground at the same time but the difference is that in this implementation, since the single leg controllers are only dependent on their respective control loop an alternative non-gaited walknet is achieved. It is only dependent on individual leg sensor inputs and transitions during swing or stance phase. The third phase was the implementation of a more advanced walking and turning control methods. Afterwards the resulting controller was to be tested thoroughly under different perturbing conditions. Unfortunately the second and third phase of the project could not be completed but they have been discussed as future work on the biomimetic control of the hexapod robot.

Chapter 2

Background and Description

2.1 Robot Architecture

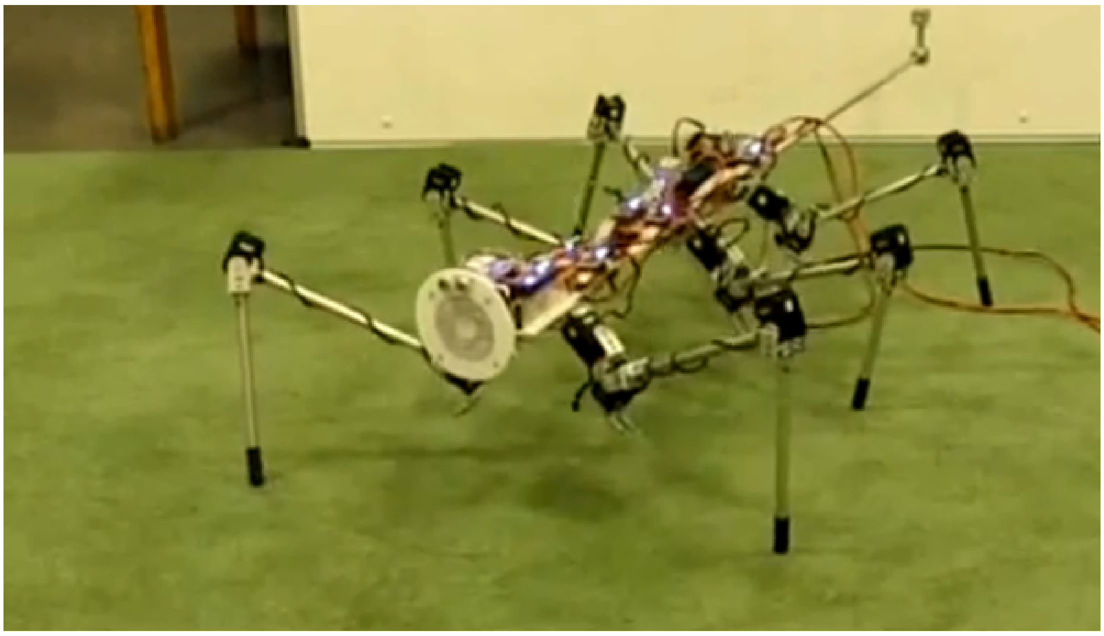


Figure 2.1: Hexapod Robot [18]

The hexapod robot used in this project is based on Lewinger's design [14] and was built by Georgios Petrou (Figure 2.1). He was a PhD student in University of Edinburgh and implemented a simple feed-forward controller on the robot. In contrast to his implementation of the robots walking and turning my project would a walknet feedback controller based on motor sensor information and leg inner control loops. The robot is 145 cm long and weighs around 3.7 kg and is 18 times the size of a real stick insect. The thorax (body) of the robot is 66 cm and its abdomen (tail) is 77 cm long. Its coxa is 5 cm, its femur is 22.5 cm and its tibia is 20.5 cm. The robot implementation does not have tarsus and antennae since their roles cannot be fully implemented on a robot. The hexapod robot has 18 MX-28 (see section 2.3 for details) servo actuators with 18 degrees of freedom in total that are controlled by 7 mbed (see section 2.2 for details) microcontrollers. The robot's design also includes six foot contact sensors that are installed on the tip of each of the six legs and are used to determine if the leg is in a stance or a swing phase. The legs are installed at a 33 degrees angle from the thorax and are controlled by their respective mbed. The seventh mbed controller is on the front part of the robot and would be later used to coordinate the movement of the six legs while walking and turning.

2.2 Mbed Microcontrollers

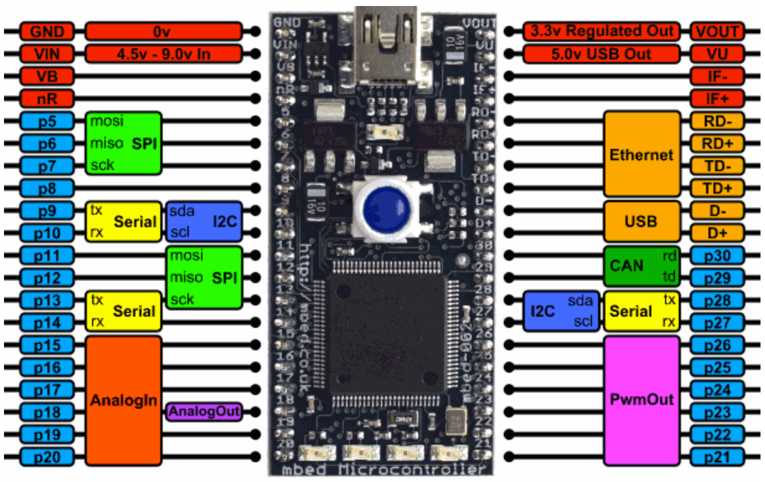
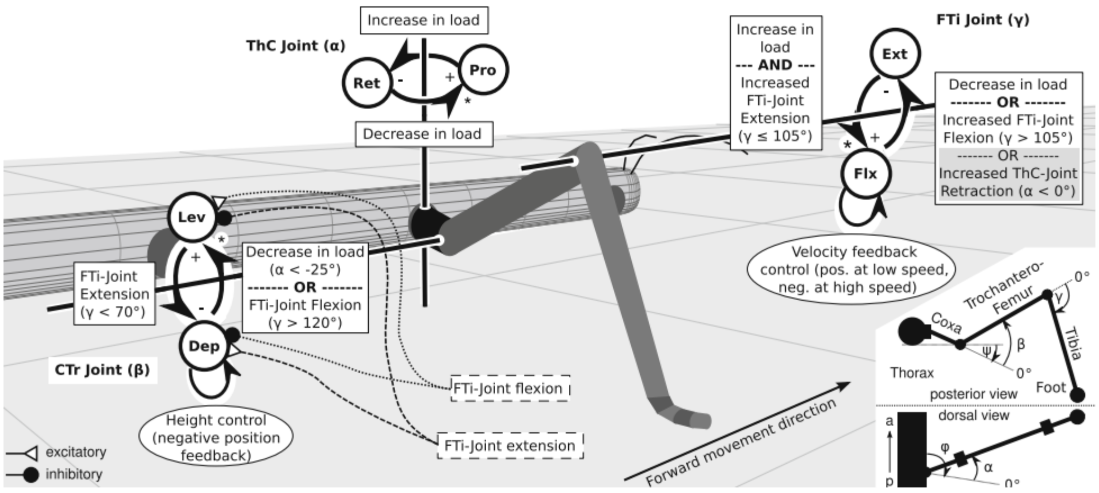


Figure 2.2: Mbed LPC 1768 [16]

There are seven mbed LPC1768 [16] microcontrollers installed on the robot. Six of them are used for each leg's neural network controllers and one additional mbed is used for the brain module. The mbed NXP LPC1768 Microcontroller in particular is designed for prototyping devises that use Ethernet, USB for communication. It is packaged as a small DIP form-factor for prototyping with through-hole PCBs, stripboard and breadboard, and includes a built-in USB FLASH programmer. It has a 96 Mhz 32-bit ARM Cortex M3 core, 32KB RAM and 512 FLASH and four programmable LED lights (Figure 2.2 shows the commonly used interfaces and their pins). The mbed programming is done via their online compiler [15] that uses C and the libraries can be imported within the online compiler. The libraries used for the communication between the mbeds and the MX-28 servos have been written by Georgios Petrou and have been uploaded on the mbed main website so they can be easily imported via the compiler's import option.

2.3 Motor Actuators

There are 18 MX-28 motors in total that act as actuators for the robot's movements and each of them has a unique motor ID (1-18). Each leg has three motors that act as the



Increase in load
ThC Joint (α)
Ret
Pro
Decrease in load
Increase in load --- AND --- Increased FTi-Joint Extension (γ ≤ 105°)
FTi Joint (γ)
Ext
Flx
Decrease in load ------- OR ------- Increased FTi-Joint Flexion (γ > 105°) ------- OR ------- Increased ThC-Joint Retraction (α < 0°)
Velocity feedback control (pos. at low speed, neg. at high speed)
Lev
Dep
FTi-Joint Extension (γ < 70°)
Decrease in load (α < -25°) ------- OR ------- FTi-Joint Flexion (γ > 120°)
CTr Joint (β)
Height control (negative position feedback)
FTi-Joint flexion
FTi-Joint extension
Forward movement direction
excitatory
inhibitory
Thorax
Coxa
Trochantero-Femur
Tibia
Foot
posterior view
dorsal view

Figure 2.3: Summary of Ekeberg's joint action rules for forward walking middle leg. The state transition rules marked with a * have priority. Optional conditions are shown with a gray background [10].

three joints, mimicking the real stick insect's leg segments. In each leg they represent the Thorax-Coxa (ThC), Coxa-Trochanter (CTr) and Femur-Tibia (FTi) joints. The ThC joint does protraction-retraction, the CTr joint does levation-depression and the FTi joint does flexion-extension movements of the legs (Figure 2.3). The MX-28 motor has three pins that are for digital input, power supply and ground (Figure 2.4). It has a led indicator that flashes once when voltage has been provided to the motor if the motor is operating normally. Its resolution is 0.29 degrees and the voltage required for normal operations is 12V-16V. Its running temperature is from -5 to 85 degrees cesium. The communication speed of this servo model is 7400bps - 1 Mbps. The goal angles of the motor available are from 0 to 4095 and motor's goal velocity is from 0 to 1023 and its current velocity is from 0 to 2047 (for clockwise turning: 0-1023 and for counterclockwise turning: 1024-2047). The unit for goal position is 0.088 so an angle of 90 degrees is: 90/0.088 = 1023 so the goal position of the servo for 90 degrees would be 1023. In each leg the three motors are connected by three cables (three wires

Figure 2.4: MX-28 [20]

each for: communication, power, ground). Pins 13 and 14 of the mbed microcontroller are used for the serial communication transmit (tx) and receive (rx) and are connected to the communication wire (9 and 10 for the test mbed) and the protocol used for the communication is serial asynchronous half-duplex. The power provided to the motors is 12V.

2.4 Sensors

Each of the MX-28 motors has sensors to determine the current position, temperature, velocity, load and input voltage. The sensors that are used in this project are to determine the position and velocity. In the neural network described in von Twickels's paper [22] it is suggested to use a load sensor for the ThC joint but in this implementation the load sensor is not taken into account for the controllers. Instead the change in load is solely determined by the activation or deactivation of a foot contact switch (Figure 2.5). The foot contact switch is also used to determine if the leg is in swing or stance phase.

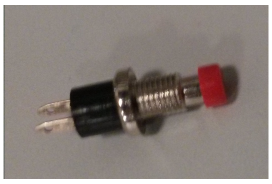


Figure 2.5: Foot contact sensor

2.5 Single Leg Neural Network

This honors project is about implementing and testing a biomimetic controller on the hexapod robot and the first part of this is the single leg controller. As mentioned before the single leg model is based on Arndt von Twickel's [22] paper, which is based on Ekeberg's model [10]. Ekeberg's controller is based on two hypotheses:

- The first is that each of the three main leg joints Thorax-Coxa, Coxa-Trochanter and Femur-Tibia has its own autonomous control module that generates alternating activity in the motor neuron pools through a bistable element. Each leg has three such bistable elements for flexion-extension (in FTi joint), levation-depression (in CTr joint) and protraction-retraction (in ThC joint).
- The second assumption is that central connections are not sufficient to generate stable phase to phase inter-joint coupling, hence with the help of sensory signals the generation of motor activity would be influenced by inducing transitions in the bistable elements (”timing influence”) and by modifying the magnitude of the motor output (”magnitude influence”)[2].

The FTi joint controller (Figure 2.3) for example can either be doing extension or a flexion, which is determined by sensory signals (this dependency is called ”timing influence”). If FTi-joint angle (gama) is less or equal to 105 degrees and an increase in load is detected (a foot contact is 1 in the case of this project) flexion state is activated. And the other case is if FTi- joint angle is more than 105 degrees or ThC joint angle (alpha) is less than 0 degrees or a decrease in load is detected then an extension state is activated. If conflicting states appear then extension to flexion is given priority. During flexion phase the sensor inputs have an additional magnitude influence on the motor outputs. At low speeds it gama is under a positive feedback control and at high speed it is under a negative one.

2.5.1 Model Description

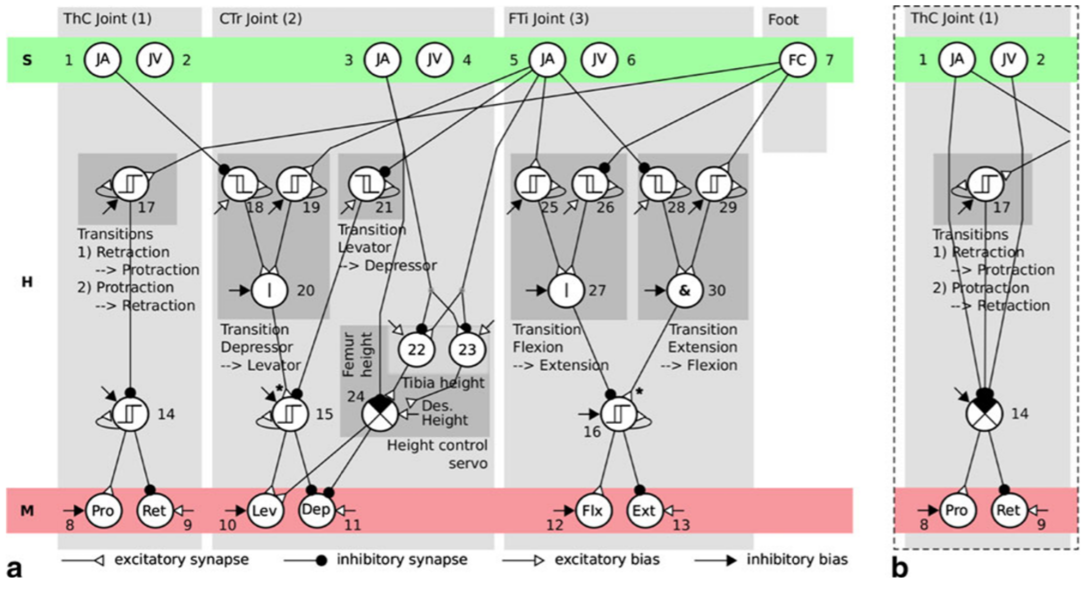


Figure 2.6: Complete neural network controller transfered from [10] for front and middle legs. S represents the sensor/input layer, H hidden layer, M motor or output layer, JA is the joint angle JV is the joint velocity and FC is foot contact. Two alternative models of the ThC joint module are given as part ”a” and ”b” in the figure.

The neural network model is divided into three layers (Figure 2.6). The first layer is the input layer that contains the sensory inputs such as joint angle, joint velocity and foot contact (JA, JV and FC respectively). Second is the hidden layer where there are four types of neurons: sigmoid neuron, ”AND” neuron, ”OR” neuron and a position and velocity control servo. The third layer is the motor layer (output) that executes the control of the servo actuators depending on the output of the hidden layer neurons. Since the synapse weights of each of the neurons connected to their respective servo actuators in the motor layer are the opposite only one of the two possible states can be activated at the same time. The structure of the neural network is the same for the middle and front legs, with different parameter values for neuron biases and weights, but had to be modified for the hind-leg controller. This is because the kinematics of the hind leg is different than the other four legs since the FTi extensor is active during the stance phase and the flexor is active during the swing phase. Because of this two types of neural network models (Figure 2.7) were implemented, for the hind legs, and tested within this project and are explained in detail in the implementation chapter.

2.5.2 Neuron Description

As I said in the above section there are four types of neurons in the hidden layer.

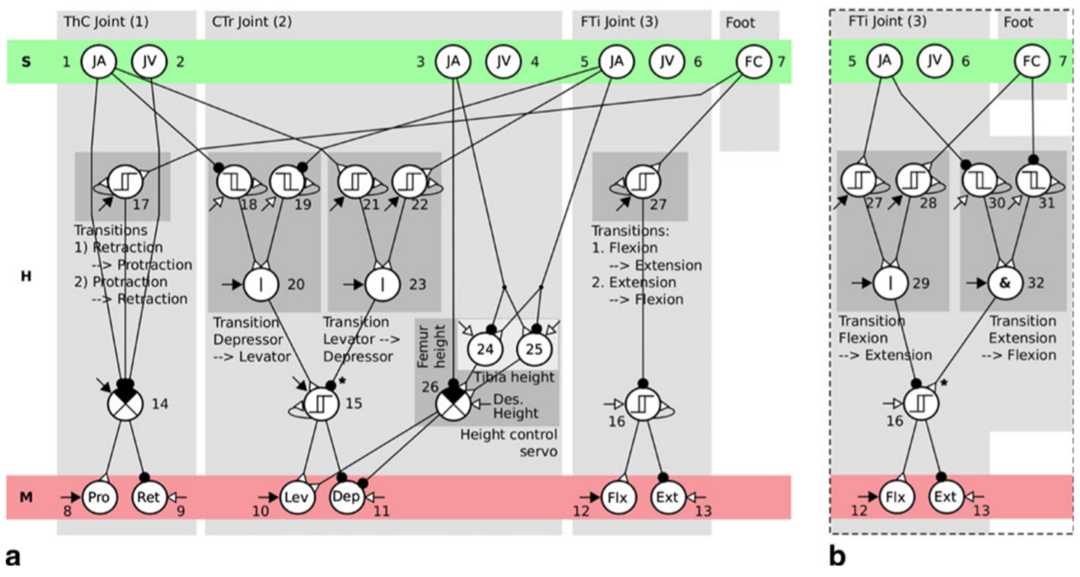


Figure 2.7: Complete neural network controller transfered from [10] for hind legs. Two alternative models of the FTi joint module are given as part ”a” and ”b” in the figure.

- The sigmoid neuron has its own bias and self-coupling weight. The bias is used to tune the threshold parameters that adjust the output towards the desired one. The self-coupling weight is there to meet two conflicting requirements: noise tolerance and fast switching. Larger self-weight means better noise tolerance but a delay in switching and a smaller one means the opposite. The sigmoid function is of a simple additive type:

$$\sigma(x) = \frac{1}{1+e^{-x}}, x \in \mathbf{R} \tag{2.1}$$

  The sensor neurons are an exception because they are using the unbounded identity function as transfer function and are given by:

$$o_k(t+1) = \sigma(\theta_k + \sum_{i=1}^{m} w_{ki} o_i(t)), k = 1, ..., m \tag{2.2}$$

  And output k is the output of neuron k,theta k is a bias term and w(ki) is the synapse from i to k

- The ”AND” neuron and the ”OR” neuron are basic logic computations that are used within the neural network to activate the bistable elements when certain conditions are met.

- There are also two servo modules in the ThC and CTr joint that are used to control the position and velocity of angles of the joints. The CTr servo, also called a height control servo is used to keep the body at certain height off the ground while the leg is in stance phase. It uses the angles of the CTr and FTi joints plus the offset angle at which the ThC joint is from the thorax to determine the current height and compares it to a desired height. Depending if it is higher or lower than the desired height a levation or a depression state is exited. The current height can be computed with the following equation:

Where beta is the angle of the CTr joint, gama is the angle of the FTi joint and psi is the angle of the ThC offset [22].

2.6 Brain Module

This section explains the design and workings of a suggested brain/central controller module that synchronizes the six legs for walking and turning locomotion of the robot.

2.6.1 Leg Phases

Every leg can be in one of two states. It can either be in a swing or a stance state. The swing phase is when the foot is in the air because it is doing limb advancement. Stance is when the foot is in contact with the ground and pulls or pushes the body depending on the leg and its position. A leg can go from swing to stance phase if it has reached the Anterior Extreme Position (AEP), which is the furthest position the leg should reach when walking. The leg can also go from stance to swing which happens when it is in Posterior Extreme Position (PEP), which is the furthest point a leg should go to towards the back (posterior) of the body.

2.6.2 WalkNet Controller

The walknet controller would be implemented in the seventh mbed that would be used to synchronize the movements of the six legs according to biological rules [6]. The initial walknet implementation was based on six rules determining the coordination between the legs. Rules from one to three are active when a leg is moving from AEP to PEP (stance). Rule four is active when a leg is moving from PEP to AEP (swing). Rules five and six are used for force and position control and are not used within this project since the single-leg neural network control loop has such controllers already implemented.

- Rule number one is active when a leg finishes its stance phase and sends an inhibitory signal for swing movement to its head directed leg. For example if the left hind leg has finished its stance phase the brain module according to this rule will prohibit the left middle leg from entering a swing phase. Also this rule does not apply to the front legs since they don not have a head directed neighboring leg.
- Rule number two is active when a retraction movement happens in the ThC joint. When such movement occurs the brain sends an excitory signal to the next head directed or contralateral leg to enter a swing phase.
- Rules three and four are used to prevent collision between legs and their tail-directed neighbors. They are also not applicable in this project since the single leg controller's angular thresholds prevent such collisions.

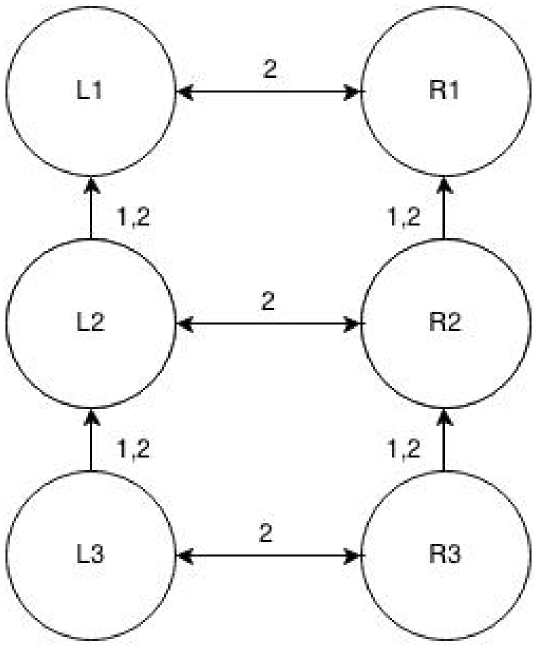


Figure 2.8: Simplified Walknet with only rule 1 and 2 as transition rules.

So in conclusion the only rules that are useful in this walknet are number one and two since each leg has its own control loop that removes the need for the other four rules thus resulting in a simplified version of Cruse et al.(Figure 2.8). There are two interfaces implemented within each leg's individual neural network controller [23]. One is used for performing swing and stance movements of the leg. The other is used for overriding the currently executed stepping motion. This is achieved by taking the input from the foot contact sensor to be zero and thus making the leg enter a new swing phase. These two interfaces depending on rules one and two can be used to implement the walknet control mechanism in the brain module.

Chapter 3

Implementation

In this section a detailed explanation of the single leg neural network will be presented. The Implementation of von Twickel's [22] single leg neural network controller is programmed on an mbed microcontroller and tested on a stick insect robot's leg.There is more information on how the initial stages of the project went and problems that arose during the project's development in the Appendix chapter.

3.1 Design Overview

The neural network implementation is written in C plus plus and is made to control the robot's leg via the LPC1678 mbed micro-controller. In the beginning of the project I was given the code for the middle leg control network and the idea was to evaluate it and derive the other legs' controllers. In the process of my evaluation I decided that this implementation, which was based on von Twickel's model, had several faults (described in more detail in the discussion section) and decided to re-implement the design that was described in the paper [22]. Since the model described in there is modular an implementation in an object oriented language proved efficient and feasible. The resulting C plus plus design can be divided in the following parts/classes:

- The Joint class
- The Hidden class
- The Servo class
- The Bimodule class
- The Control class
- The Main program

3.1.1 The Joint Class

Each joint has its own inner control loop and therefore they can be described and controlled separately. The class joint as in the original model [22] has has its sensor inputs, represented as joint angle(JA) and joint velocity(JV) variables and an additional inner variable jointId that holds the unique id of the joint. Each joint has a unique id from 1 to 18 representing the 18 total joints within the six legs of the robot . Since the joint class is made to represent the Thorax-Coxa(ThC), Coxa-Trochanter(CTr) and Femur-Tibia(FTi) joints it has methods that a speed input execute the actions described in the motor layer (2.6) such as protraction and retraction for ThC, levation and depression for CTr, and flexion and extension for FTi.
The actions are executed by setting the respective mx28 motor's goal position to be a preset constant values representing the end position the motor should achieve in the particular direction. The variables that hold these values are defined as global variables and are:

- For the ThC joint - tcPEP and tcAEP representing the posterior and anterior extreme positions.
- For the CTr joint - ct_highP and ct_lowP representing the highest and lowest positions that the ctr joint should achieve.
- For the FTi joint - ft_StanceP and ft_SwingP representing the goal positions of the fti joint while in stance or swing phase.

The functions if called by a joint that should not execute that action does not do anything. This is achieved by having an if statement that checks the id of the joint that called it. Since all legs have three joints and all the joints/motors have unique id. Only the ThC joints' ids modulo 1 give 0, the CTr joints' ids modulo 2 give 0 and the FTi joints' ids modulo 3 give 0. By using this as a condition within the functions of the joints this ensures that no joint would execute another joint's action.
There is also a setId method that sets the joint variable's id. In this method according to the joint's id its angle limits are set as clockwise and counter-clockwise rotation limits of the servo actuators. With the use of a case statement that is used to determine by the id what joint has been defined it calls the functions of the mx28 servo's SetCWAngleLimit(id,limit) and SetCCWAngleLimit(id,limit). Both methods are implemented in the mx28 library and have been precoded by Georgios Petrou. The exact angle limits have been described in von Twickel's implementation [22] and some have been adjusted to my model of the network. The angle limits were translated as motor rotation limits and are represent the following:

- FTi joint - angle ranges from -25 to 50 degrees for all leg types
- CTr joint - angle ranges from 20 to 125 degrees for all leg types
- ThC joint - different angles for different legs
  - Front Legs - angle ranges from -60 to 10 degrees
  - Middle Legs - angle ranges from -40 to 50 degrees

– Hind Legs - angle ranges from 10 to -60 degrees

3.1.2 The Hidden Class

This class represents the neurons in the hidden layer of the neural network(see Figure 2.6). A neuron can be defined then as a Hidden object with parameters self weight and a bias:

$$Hidden\ neuron(selfweight, bias) \tag{3.1}$$

In the hidden class there are four methods that represent the information that the respective neuron holds.

- The Push_s method takes as inputs four values of type double. Two of them are the inputs that come from the synapses connected to that neuron and the other two are their weights of that pathway. The inner workings of this method are the calculation of a sigmoid neuron's output with by using these four variables and the neuron's self weight and bias and applying the sigmoid function to them:

  $$double\ input = -(input1 * weight1 + input2 * weight2 + selfWeight + bias);$$

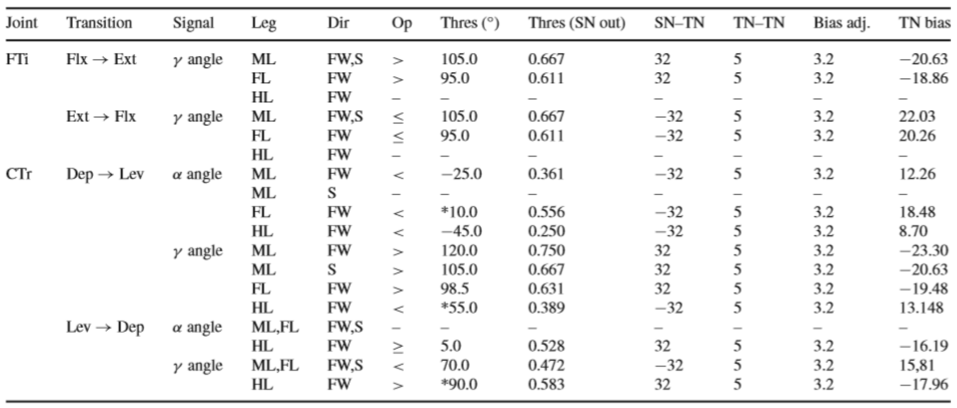
  $$double\ result = 1/(1 + std :: exp(input)); //The\ sigmoid\ function$$

- The Push_s_sensor is anther method that is used for the calculation of a sigmoid neuron's output. The reason for implementing a separate method is discussed in more detail in the discussion chapter (5.2) The difference is that this method takes input from only 1 synapse and an extra input is required as a parameter of this function - the SN_Thresh variable. It represents the angular thresholds for transition between bistable actions of a joint (Figure 3.1). Given this threshold it determines the threshold neuron bias as

  $$bias = -(w_{in} SN_{thres} + 0.5 w_{self}) + bias_{adjust} \tag{3.2}$$

  this calculation is based on von Twickel's neural network implementation [22] and the adjusting bias factor is approximately 3.2. The output of this sigmoid neuron function is a number between 0 and 1. If it is above the threshold then the sigmoid function would be above 0.5 and larger the difference the closer its output would be to 1. If it is below the threshold then the output of the function would be between 0.5 and 0 depending on how much lower is the actual angular input in comparison to the threshold given.

- The Push_and and Push_or methods are used for the or and and neurons within the network and are simple logic computation. For the "AND" method if both of the synapse inputs times their synapse weight plus the bias of the "AND" neuron are above 0 then the method returns 1, otherwise it returns 0. For the "OR" method it is almost the same but instead of requiring both of them to be above 0 it only needs one of them to be in order to output 1.



Joint	Transition	Signal	Leg	Dir	Op	Thres (°)	Thres (SN out)	SN–TN	TN–TN	Bias adj.	TN bias
FTi	Flx → Ext	γ angle	ML	FW,S	$>$	105.0	0.667	32	5	3.2	−20.63
			FL	FW	$>$	95.0	0.611	32	5	3.2	−18.86
			HL	FW	–	–	–	–	–	–	–
	Ext → Flx	γ angle	ML	FW,S	$\leq$	105.0	0.667	−32	5	3.2	22.03
			FL	FW	$\leq$	95.0	0.611	−32	5	3.2	20.26
			HL	FW	–	–	–	–	–	–	–
CTr	Dep → Lev	α angle	ML	FW	$<$	−25.0	0.361	−32	5	3.2	12.26
			ML	S	–	–	–	–	–	–	–
			FL	FW	$<$	*10.0	0.556	−32	5	3.2	18.48
			HL	FW	$<$	−45.0	0.250	−32	5	3.2	8.70
		γ angle	ML	FW	$>$	120.0	0.750	32	5	3.2	−23.30
			ML	S	$>$	105.0	0.667	32	5	3.2	−20.63
			FL	FW	$>$	98.5	0.631	32	5	3.2	−19.48
			HL	FW	$<$	*55.0	0.389	−32	5	3.2	13.148
	Lev → Dep	α angle	ML,FL	FW,S	–	–	–	–	–	–	–
			HL	FW	$\geq$	5.0	0.528	32	5	3.2	−16.19
		γ angle	ML,FL	FW,S	$<$	70.0	0.472	−32	5	3.2	15,81
			HL	FW	$>$	*90.0	0.583	32	5	3.2	−17.96

Figure 3.1: Transition rule switch parameters(joint angles) converted into neural parameters of sensor neuron (SN) and threshold neuron (TN).Table and content taken from [22]

3.2 The Servo Class

The servo class is used to describe the servo neurons in the network. In my implementation the neural network has two such neurons. The use of ThC servo neuron it is mentioned as an alternative joint controller for the front and middle legs in von Twickel [22] but I decided that it would provide better results than the simplified version (the reasoning is explained in more details in 5.2). A servo object is defined with a bias parameter and has two methods to calculate its output. The two types of servo neurons are:

3.2.1 Height Servo

The CTr height servo is as described in [22] used to adjust the height of the robot. Given a desired height and the three joints' angles it calculates the desired CTr joint angle and subtracts the current CTr angle and if the result from this subtraction is above or equal to zero it returns 1 otherwise -1. This means that if the desired height is more than the current height the servo will output -1 and would eventually excite a depression and inhibited a levator action (as described in [22]). If the output is 1 then a levation would be excited and a depression action would be inhibited. The calculation of the desired CTr angle is based on von Twickel's paper [22] height calculation:

$$d_{total} = d_{\beta_{eff}} - d_{\gamma_{eff}} \tag{3.3}$$

$$d_{total} = \sin(\beta - \psi) * L_{Femur} - \sin(\gamma - \beta + \psi) * L_{Tibia} \tag{3.4}$$

Where beta is the CTr angle, psi is the offset angle gamma is the FTi angle alpha is the ThC angle (see figure 15). In the original paper they do not take into account the offset angle at which the ThC joint is installed to the thorax(body) of the robot and the ThC joint current angle but in this implementation it affected the output of the controller (more details in chapter 5.1), hence resulting in the following calculations:

$$\sin(\beta - \psi) * L_{Femur} = (\sin(\beta)\cos(\psi) * L_{Femur} - \cos(\beta)\sin(\psi)) * L_{Femur} \tag{3.5}$$

$$\sin(\gamma-\beta+\psi)*L_{Tibia} = (\sin(\gamma+\psi)\cos(\beta)*L_{Tibia} - \cos(\gamma+psi)\sin(\beta))*L_{Tibia} \quad (3.6)$$

$$d_{total} = \sin(\beta-\psi)*L_{Femur} - \sin(\gamma-\beta+\psi)*L_{Tibia} + \sin(\psi)*LCoxa \quad (3.7)$$

from equations 3.4,3.5, 3.6 and 3.7 :

$$d_{total} = \sin(\beta)B - \cos(\beta)A \quad (3.8)$$

where A and B are :

$$A = \sin(\gamma+\psi)*L_{Tibia} + \sin(\psi)*L_{Femur} \quad (3.9)$$

$$B = \cos(\gamma+\psi)*L_{Tibia} + \cos(\psi)*L_{Femur} \quad (3.10)$$

So in order to get the desired ctr angle I use the following trigonometry:

$$C = desiredHeight/\cos(\psi)/\cos(\alpha) - \cos(\psi)*L_{Coxa} \quad (3.11)$$

$$X = \sqrt[2]{(C^2+B^2)/(B^2+A^2)} \quad (3.12)$$

$$CTr_{desired} = \arccos(X) \quad (3.13)$$

Where C is used to take into account the offset and ThC angles into account in the height calculations. After these calculations the output of the height servo is determined as 1 if the desired CTr angle is larger than the current one and -1 if not.

3.2.2 ThC Servo

The ThC servo module takes as inputs the ThC joint angle, velocity and the output of neuron 17. In order to determine if a protraction or a retraction action is needed it applies a sigmoid function to the sum of the current ThC angle times its synapse weight, the desired position and its self bias value. The need of the joint velocity in this calculation was determined detrimental to the results hence the velocity is not used for determining the output of the ThC servo neuron (more details in chapter 5.2). The desired position depends on the output of neuron 17. If the value of neuron 17 is more than 0.5 then the desired position is the posterior extreme position, otherwise the desired position is the anterior extreme position both extreme positions taken from the global constants containing their values.

3.3 The Bimodule Class

The bimodule class works as the motor/output layer of the neural network and is described by each joints bistable module. A bimodule object is instantiated for protraction,retraction,levation,depression,flexion and extension with their respective bias values. The bimodule has three methods that execute the control of each of the three joints. Each of these methods take as input the output and weight of the neurons connected to it from the hidden layer and the speed at which the action needs to be executed. Within the body of the methods an output is calculated with the following calculation :

$$output = input * weight + bias \tag{3.14}$$

Where bias is the bistable object's self bias. After the output is determined two if statements are used to determine which of the two actions of the bistable module is to be executed. As in the original paper [22] the bistable module is used in order to ensure that only one of the joint's action will be excited. In my implementation this is achieved with the help of two if statements that cannot be true at the same time. The conditions depend on the output from equation 3.14 and the input coming from the hidden layer. In order for an action to be excited it requires the output to be a positive number and the input to be greater or equal than 0.5 for one of the actions and less than 0.5 for the other action(for example levation and depression actions in the CTr joint). This ensures that even if the input coming from the hidden layer is excitatory (greater or equal to 0.5) the output determined by equation 3.14 will be negative for one and positive for the second of the two possible actions of the bistable module. This results in a bistable module for each of the three joints that are independent from each other and each module executing one of its two possible actions as described in the original paper[22].

3.4 The Control Class

The control class is the main body of the control network(for the full version of the control class' code for see Appendix A)and represents the synapse connections between each joint's neurons within the three layers. It contains a while loop that executes until two conditions are met. One of the conditions is that the isWalking global variable is set to true and the other is a handler variable that needs to be larger than 0. The handler variable is instantiated to be equal to 10 at before the beginning of the loop and decrements during the execution of the while loop. The network controller model is designed to run at a rate of 100Hz so in order to simulate that a global variable called ctrlInterval is set to 10000 and an update interval variable is set to 100000(it represents the step size in milliseconds). Also there is another variable ctrlTimer that keeps track of the time that each step takes. The while loop has an inner if statement that checks if the step has taken more than the desired time (ctrlTimer ¿= ctrlInterval). This ensures that the whole controller would be executed exactly once every 10000 milliseconds. The control network is simulated within this while loop and when the if condition is met. The inner body of the if statement can be divided into the following four parts.

3.4.1 Sensor Layer

The first part is reading the inputs that represent the sensor layer of the neural network. It takes each of the three joints' angle and velocity and the foot contact as variables that are later handled by the second layer(the hidden layer). There is also the need of adjusting the angle variables to be the same as the ones depicted in figure 15. Because the joint angles are in fact a position of the mx28 motor. The position ranges from 0 to 4096. In order to get the angle from the motor's position it needs to be multiplied by 0.088 since each of the position units represent 0.088 degrees. The servo actuators are installed at a certain offset angle in the three joints so in order for them to represent the same angles as in figure 15 the following adjustments need to be done for each joint angle:

- ThCAngle = (2048-InitialThCAngle)*0.088
- CTrAngle = (3072 - InitialCTrAngle)*0.088
- FTiAngle = (InitialFTiAngle - 1024)*0.088

  These modifications remove the offset and change the servo positions to represent the same angles as in figure 15.

3.4.2 Hidden Layer - ThC

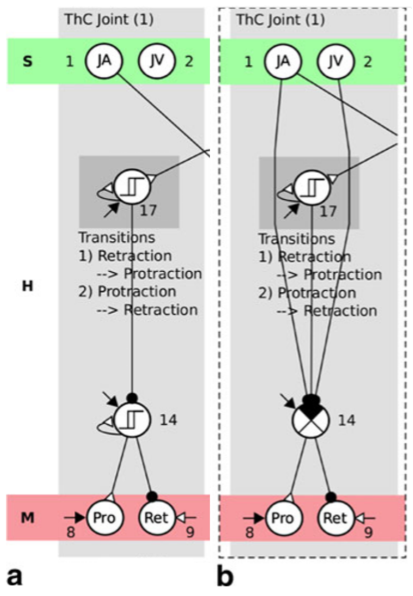


Figure 3.2: Two alternative representations of the ThC control module(a and b). A is the simplified version for front and middle legs. B is the alternative version for front and middle legs and the only version for the hind leg [22].

The second part represents the ThC joint controller (Figure 3.2). There are two implementation of the ThC joint control module described in von Twickel's paper [22] and in my implementation it is the same in all of the legs (with different neural network parameters for different legs - more details about the weight and bias parameters in Chapter 4). As depicted in the original paper's alternative implementation the thc controller has one sigmoid neuron (neuron17) and a servo module (ThCServo14). The synapses of the neural network are represented as input parameters of the different methods called to determine the output of each neuron. for instance to get the output of a sigmoid neuron the Push_s or the Push_s methods of the hidden class that is called and the parameters given to the method represent the input synapses that are connected to this neuron and their respective weights. The output of the neuron is then given as a parameter along the neural pathway. After the neurons within the hidden layer have calculated their respective outputs the last neuron in the pathway (the ThCServo in this case) sends its signals to the protraction and the retraction neurons in the motor layer with the joint controller methods of the bimodule class. Since the neural pathway leading to protraction are excitatory the weight parameter of the protraction.tcJointCtrl is a positive number. The neural pathway to retraction is inhibitory, hence the weight parameter of the retraction controller is a negative number.

3.4.3 Hidden Layer - CTr

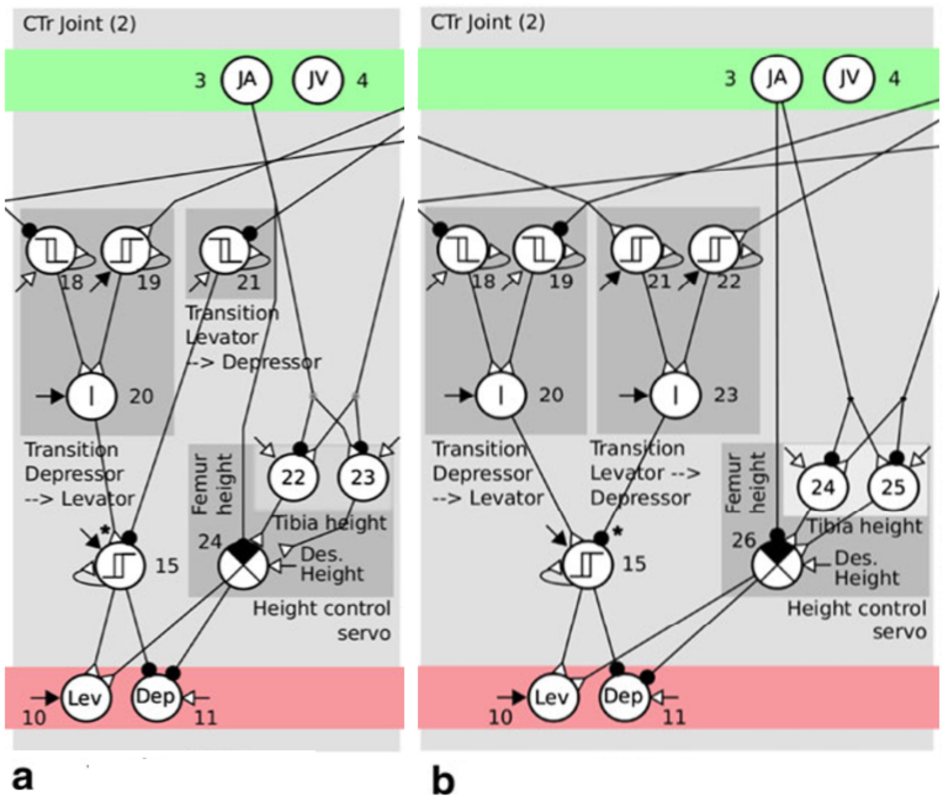


Figure 3.3: CTr joint control module. Part A is the front and middle leg controller and part B is the hind leg's controller [22].

The third part is the CTr joint controller (Figure 3.3). The Ctr controller is the same for the front and middle legs, but different for the hind legs. The difference is represented in a different structure and partially different synapse signs.

- For the front and middle leg implementation there are three double variables representing the thresholds to the sensor inputs:
  - dep_to_lev_ft = ftAngle - X; the FTi angle threshold for depression to levation transition
  - dep_to_lev_tc = tcAngle - Y; the ThC angle threshold for depression to levation transition

- lev_to_dep_ft = ftAngle - Z; the FTi angle threshold for levation to depression transition

Where X is 98.5 for front legs and 120 for middle legs; Y is 10 for front legs and -25 in middle legs; Z is 70 in both front and middle legs (The threshold angles can be seen in Figure 3.1). In order to represent the neural pathways of the CTr joint described in the paper (see Figure 2.6 for more details)the following things are used:

- The excitatory synapse pathways are represented by a negative weight given as input for the neuron's output calculation function.(The possible functions are: Push_s; Push_s_sensor; Push_or; Push_and and are described in section 3.1.2)
- A priority synapse pathway (such as path 20-¿15 has a priority over 21-¿15, depicted with * in Figure 3.3) are given by a larger absolute value of the synapse weight parameter.
- The outputs of neuron 18,19 and 21 are given by the sensory sigmoid function (Push_s_sensor) and are based on the state transition rules described in Ekeberg's paper [10].
- The output of neuron 20 is given by the "Push_or" method of the hidden class.
- Output of sigmoid neuron 15 is calculated with the "Push_s" method of the hidden class and has a priority over the path from neuron 20 to 15 given by a larger absolute value of the input weight parameter.
- Since there are two parallel pathways to the motor layer in the CTr joint (from the height control servo and neuron 15) in my implementation this is represented as the output of neuron 24 which is servo neuron that is used when a foot contact is detected. This condition is not present in the original paper but its necessity is apparent since a height calculation of any sort would prove inaccurate while in a swing state. Hence the height controller is only triggered when the foot is on the ground. In both pathways an excitatory and an inhibitory signal is sent to the motor layer via the levation.ctJointCtrl() and depression.ctJointCtrl() methods respectively.

• The hind leg controller's CTr joint is similar to the front and middle legs but has different reverse synapse weight and bias signs for neuron 19. Because in the hind leg the body of the robot is being pushed there is a need for an extra condition that would trigger a levation to depression transition. When the leg is depresssing it is actually pushing the body forward when it is in a stance phase and the ThC angle is showing that the leg is behind its original position. This different behavior is achieved by the following changes:

  - The X,Y and Z values explained previously are now 55, -45 and 90 respectively and also a new threshold is given by the ThC angle for levtion to depression transition (lev_to_dep_tc = tcAngle - 5) and is fed to sigmoid neuron 21 as input parameter.

- There for there are two extra neurons: sigmoid neuron 22 and "OR" neuron 23 and neuron 21 now takes as an input the ThC joint angle (with an excitatory synapse weight).
- Neuron 22 is excited by the FTi joint angle and the output of "OR" neuron 23 is dependent on the outputs of neuron 21 and 22.
- The priority of output 15 input neural pathways is reversed and the transition from levation to depression has a priority over depression to levation.

3.4.4 Hidden Layer - FTi

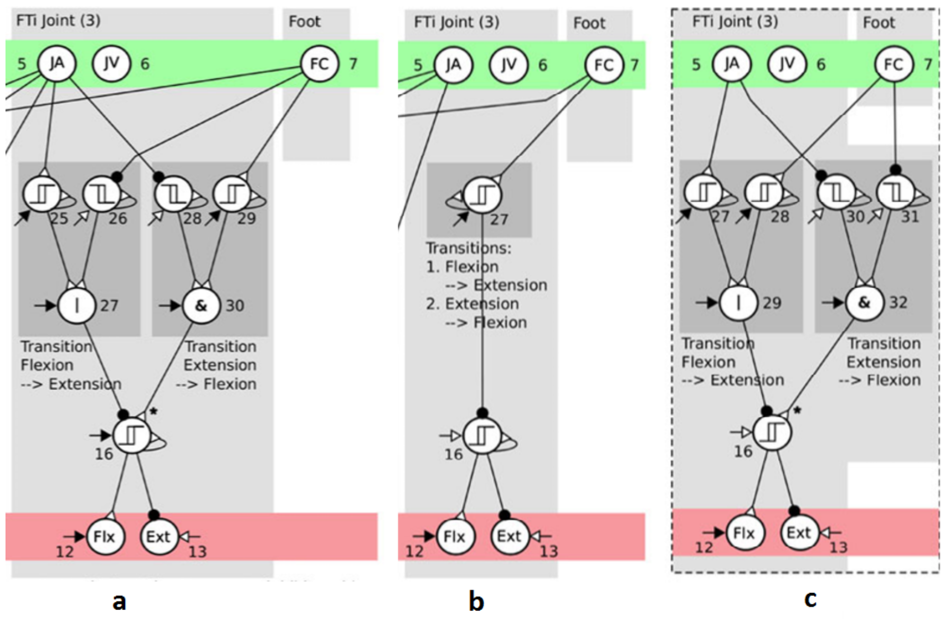

FTi Joint (3)
Foot
JA
JV
FC
Transition Flexion --> Extension
Transition Extension --> Flexion
Transitions: 1. Flexion --> Extension 2. Extension --> Flexion
Flx
Ext
a
b
c

Figure 3.4: Three alternative representations of the FTi control module. Part A is the front and middle legs' design and Part B and C are two alternative implementations for the hind leg's FTi control module [22].

The fourth part is the FTi Joint controller (Figure 3.4). For the front and middle legs the implementation of the controller is the same with different FTi angle thresholds for flexion to extension and extesion to flexion transitions:

- flx_to_ext_ft = ftAngle - X; the FTi angle threshold for flexion to extension transition
- ext_to_flx_ft = ftAngle - Y; the FTi angle threshold for extension to flexion transition

X and Y are 95 degrees for front legs and 105 for middle legs. These angular thresholds are fed to neurons 25 and 26 as a condition for a flexion to extension and extension to flexion transition respectively. These transitions also depend on the foot contact switch with excitatory weight for flx-ext transition and an inhibitory pathway weight for ext-flx.
Neuron 16 is the last neuron in the FTi controller hidden layer pathway and it takes as

inputs the outputs from neuron 27 and 30 (inhibitory and excitatory synapse pathway weights respectively), with the input from neuron 30 having a priority over the one from neuron 27. After the output of neuron 16 is being calculated with the simple sigmoid function it is send to the FTi joint motor layer with:

```
Flexion.ftJointCtrl(output16,32,step\_speed)
and
Extension.ftJointCtrl(output16,-32,step\_speed)
```

,where 32 and -32 represent the excitatory and inhibitory neural pathways leading from neuron 16 to neurons 12 and 13 respectively.
In the original paper are two alternative representations of this controller for the hind legs and for reasons discussed in more detail in chapter 5.1 I have implemented the alternative design (Figure 3.4 part C). So that makes the FTi controller of the three legs similar but with differences in some synapse weight and bias signs. More specifically neurons 28 and 31 in the hind leg's FTi joint controller (neurons 26 and 31 in the front and middle legs) have the signs of their input synapse weights and biases. Also the bias of sigmoid neuron 16 is excitatory as opposed to the front and middle leg's which is inhibitory.

3.4.5 The Main Function

The main function sets the baud rate to be 115200 which is the speed at which the serial communication packets are being sent. In order to get output in the terminal application used for communicating over the usb with the mbeds the baud rates need to be set the same.
The control class is instantiated with the respective joint angle ids and within an endless internal loop a switch case statement awaits for an input to be received via the communication channel(the terminal in case of that a terminal application is used to communicate via the usb port). In case it receives an input of '1' it starts executing the neural network and if input '2' is received the execution is halted. Once the positive command is received the isWalking flag is set to true and the controller is reset and a control_all function is called over and over at each iteration of the loop. This concludes the implementation of the single leg neural network controller.

3.4.6 Contra-Lateral Legs

For Contra-Lateral legs there are a few parameters that need to be changed in order to get the reverse action of the respective leg.

- The posterior and anterior extreme positions need to be swapped. This is achieved by subtracting their current value from the maximum value(0x1000 in hex and 4096 in decimal).
- The angle limits for the ThC joint that are set when setting a joint's id class need to be reversed (the same subtraction as above).

- The conditions in the tcJointCtrl for protracting and retracting need to be swapped. More specifically the condition for the input now needs to be less than 0.5 in order to protract and greater or equal to 0.5 for the retraction action.
- The ThC angle in the sensor layer is with an opposite sign after it has been read as an input(the beginning of the while loop inside the control class)
- Finally the ThC angle input needs to be fed to neuron 14 with the opposite sign.

Chapter 4

Results and Evaluation

This chapter discusses the results obtained from the implementation of the neural network controllers for the three legs of the hexapod robot (Only the front,middle and hind left legs are discussed because the right legs' behavior is the same but with the opposite direction of movement). Thorough analysis of the parameter influence have been conducted and the best values are given.
In the model described in the paper [22] they suggest some of the parameters of the neural network but others that are not specifically discussed, the ”Free” parameters' influence on each legs' controller is discussed in detail in the below sections. Figures are given for each leg's optimal behavior and figures representing the different ”Free” parameter testing were not included, instead their role for the different legs' controller action is discussed in detail. The sensory neurons' weight and biases are given in figure 3.1 and were used accordingly during the the tests and in my implementation of the controller. The following neuron parameter values were not changed:

- neurons: 17, 18, 19, 21, 25,28 in the front and middle legs' controllers.
- neurons: 17, 18, 19, 21, 22, 27, 30 in the hind leg's controller.
- synapse pathway weights are not considered unless one of them has a priority over the other and lead to the same neuron. In this case their influence is discussed respectively. This is because the resulting behavior of changing an excitatory and an inhibitory synapse weights leading to a single neuron does not influence its output since their absolute value is still the same and the output of the neuron does not change.
- The ThC's joint bias and synapse parameters were not changed since they were not considered as free parameters in the original paper [22].
- Motor neuron bias values' influence was too little to discuss since they were a part of a bistable module with no priorities over one another. Hence for pro-ret, lev-dep and flx-ext they are the same but with different signs. The only influence of these values was that with larger values the controller took more time to recover from a misstep and with smaller values the behavior was not stable hence an absolute value between 10 and 16 was given for these biases.

4.1 Front Leg

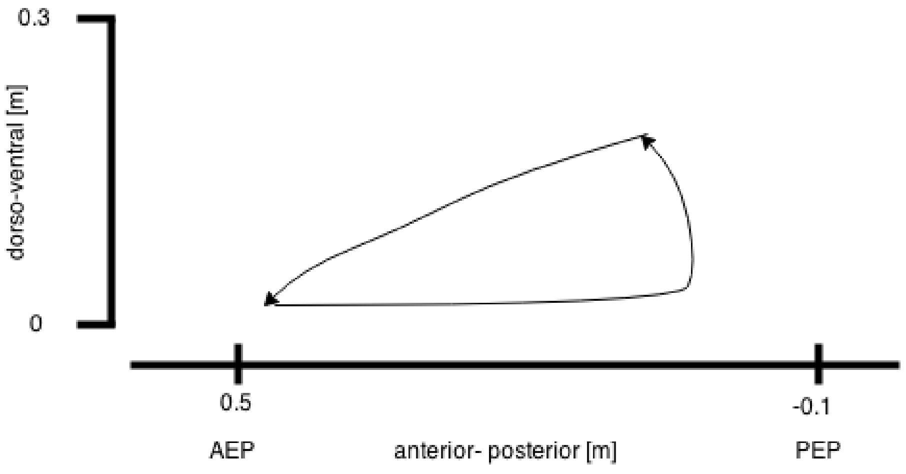


Figure 4.1: Side view of front leg's movement.

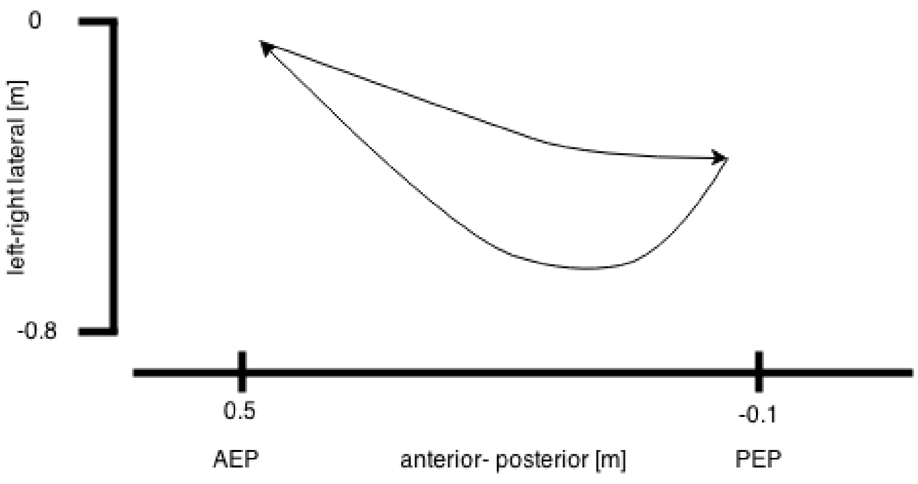


Figure 4.2: Bird view of front leg's movement.

The front leg's end behavior with the optimal parameter values is shown in figures 4.1, which represents the side view of the leg's movement and figure 4.2 which is a bird eye's view of the movement. They represent the end results of the controller implementation with the best derived weight and biases for the neurons and the synapse pathways.

4.1.1 Bias Influence

The bias of a neuron in a neural network shifts the activation function to the desired position. Hence the behavior of my controller changed as tested it with different bias values for the "Free" neurons. First I would like to mention that the further the position of a neuron is in the neural pathway the more its bias influenced the behavior of each leg hence the neurons' output that were send to the motor layer affected the results the most. Second the bias' influence of the "Free" neurons for front leg controller are the following:

- Neuron 20 - This neuron affects the transition from depression to levation in the CTr joint and within my evaluation its bias value influence was that with a higher bias value the activation of the levation neuron was reduced. Since the front leg's behavior is to reach further forward this bias value after extensive testing was determined to be as low as possible in order to achieve optimal results. The optimal value derived was 0.1.
- Neuron 15 - The last neuron in the hidden layer of the CTr joint hence its value influenced the activation of the levation and depression actions the most. Larger values of this neuron reduced the activation of the depression and increased levation. At extreme large value it resulted in no depression and with small extreme values it was not activating the levation neuron. Since the front leg's depression state is used to pull the robot its activation is more important than the levation state but on the other hand larger levation meant reaching further in the front. The balancing value that resulted in reaching the furthest and still managing to pull backwards was -2.
- Neuron 26 - With higher values of this bias the behavior of the controller was less and less stable (unstable behavior is when the is leg stuck at one place). This neuron influences the extension of the FTi joint and in the front legs when the leg is near or at the anterior extreme position it needs to activate the flexion neuron in order to help with the pulling of the body. The less this value was the less it flexed and the value that produced the best results was 7.
- Neuron 27 - This neuron was of type "OR" and was influencing the transition flexion to extension. With smaller values the protraction neuron could not be activated and with larger values the tip of the leg did not reach as much as the expected behavior. Hence the optimal value was -29.
- Neuron 30 - Influenced the extension to flexion activation and with smaller values the tip of the leg was spreading further sideways but the behavior was unstable and the optimal value of this bias was determined to be -29.
- Neuron 16 - The last of the FTi joint's hidden layer synapse pathway neurons affected the flexion and extension the most. With smaller values it produced the results shown in figure 4.3. The optimal value was -8.

4.1.2 Synapse Weight influence

The synapse weights of non-prioritized pathways were not affecting the controller and hence will not be discussed. The CTr joint's excitatory input synapse pathway for neuron 15 has priority over the inhibitory input pathway and with changing these weight parameters the behavior of the system changed. The output of neuron 15 was influ-

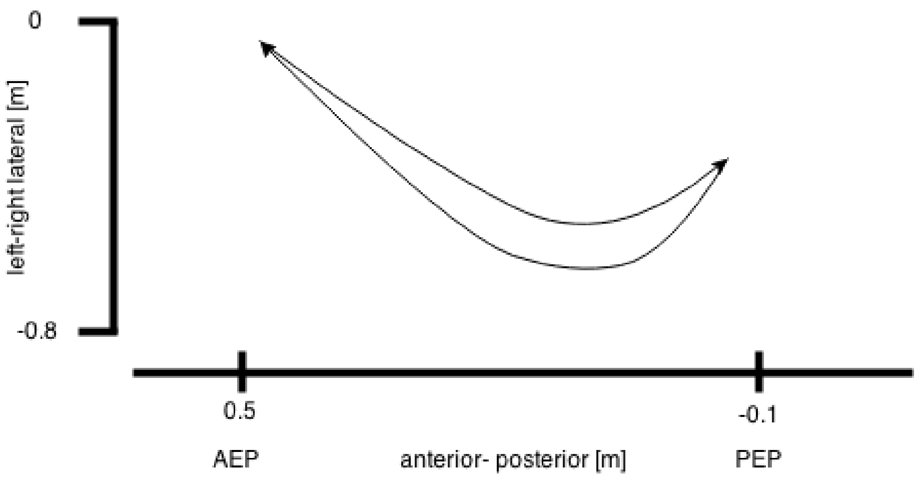


Figure 4.3: A bird eye's view of the front leg controller with smaller neuron 16 bias value.

enced by the difference in the absolute values of the input pathways. The larger the difference of their absolute values the more the prioritized action was activated. Hence for the front legs this was the depression to levation transition. With large difference it resulted in the levation action being activated more often and with smaller - seldom. There was also a need to balance this value since in the front leg the levation and depression action are both important for the front leg since its AEP is further in the front and the controller needs to pull the body back afterwards. When their synapse weights' absolute value differed by just 2 it produced the best results. The FTi joint's has a priority for the flexion action represented by synapse pathway weight of neuron 30 to neuron 16 having a larger absolute value than the pathway from neuron 27 to neuron 16. As in the case with the CTr joint larger difference resulted in flexion action being activated more. An optimal absolute difference was again determined to be 2.

4.2 Middle Leg

The middle leg's controller is similar to the front leg's and its optimal behavior is shown in the figures 4.4 giving the side view of movement of the leg and 4.5 showing a view from the top of the leg's tip. Since the controllers of the middle and front legs are not that much different their actions are more dependent on their angular thresholds and their parameter values. The angular thresholds are given in [22] and the middle leg's bias and synapse weights influence is discussed in the this section.

4.2.1 Bias Influence

Since the controller of the middle leg is the same as the front leg's the free neurons are the same with different optimal values:

- Neuron 20 - The best value for this neuron's bias was derived from neuron 18 and

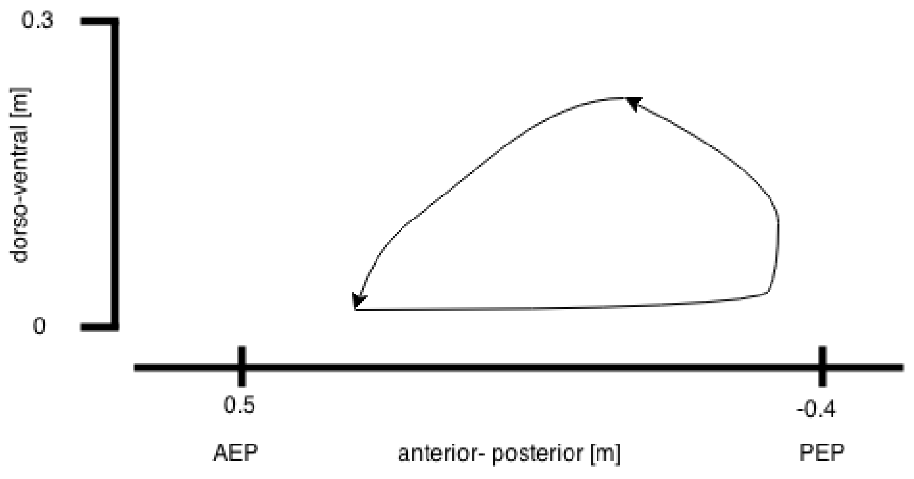


Figure 4.4: Side view of middle leg movement.

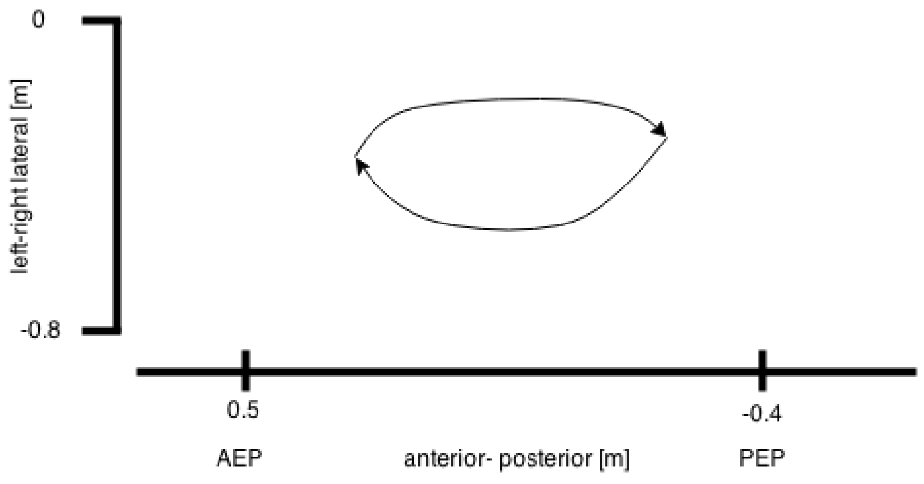


Figure 4.5: Bird eye's view of middle leg movement.

19 bias values. This is because in the middle leg's movement does not depend so much on the activation of a levation action as the front leg and a value that balanced the outputs of these sensory neurons was needed. I determined that it was a value between these two neuron's biases and the tests that I performed with different values of neuron 20's bias confirmed my assumption. The optimal value with which the leg was behaving as depicted in figures 4.5 and 4.4 was -17.

- Neuron 15 - The best value for this neuron was -5 and it produced a good balance between the activation of levation and depression neurons.

- Neuron 26 - Its influence was little in the middle leg since the leg's movement did not depend on the flexion and extension as much as it did in the front leg. This is because with the middle leg controller the tip did not extend that much towards the anterior as in the front leg and hence the foot contact was lost less times. Therefore it did not influence the FTi joints actions that much.

- Neuron 27 - With smaller bias values it resulted in the tip reaching further on the side and with a bigger value it flexed too much as it happened with the front leg controller. The best values were determined to be between -3 and -13 and the actual value set for the figures given was -5.
- Neuron 30 - As the value of this neuron's bias got close to 0 it produced highly unstable behavior and with it being a smaller number it almost never flexed. The best value that produced good flexion and extension activations and a very stable behavior was -15.
- Neuron 16 - As in the front leg controller it had a great influence on the leg's movement with higher value it activated the extesion neuron more often and the best results were achieved with it being set to -10.

4.2.2 Synapse Weight influence

The synapse weight influence was determined to be the same as in the front leg's controller since their neural networks are the same. The only difference is that the middle leg's behavior did not get stuck as often as the front leg's controller when testing with extreme values of the prioritized synapse weights.

4.3 Hind Leg

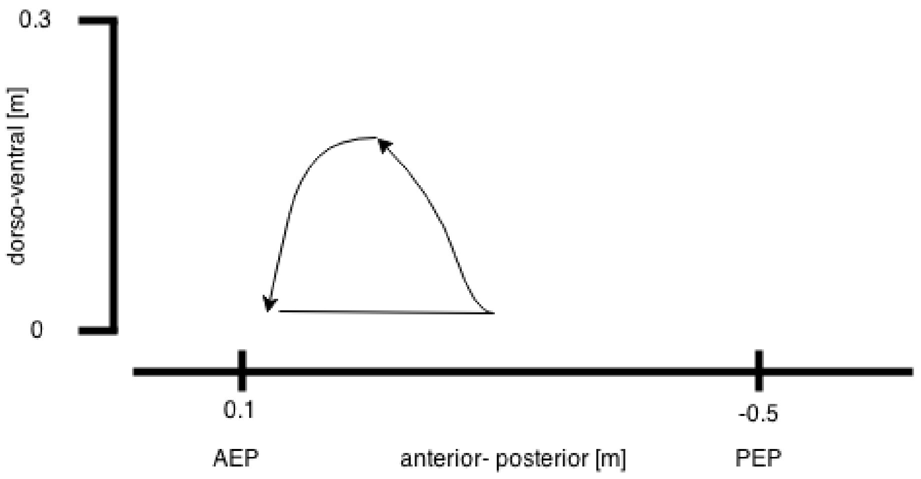


Figure 4.6: Side view of hind leg movement. Without extra weight applied on the leg.

The hind leg controller CTr and FTi joint control modules have a different structure and partially different synapse signs. Therefore the influence of its bias values was different than in the front and middle legs. Also it is important to mention that there were a lot of issues that arose with the hind leg's implementation. The most important one being the constant loss of the foot contact while in a stance phase. Because of this the hind leg's movement did not go as further back as it should have and produced

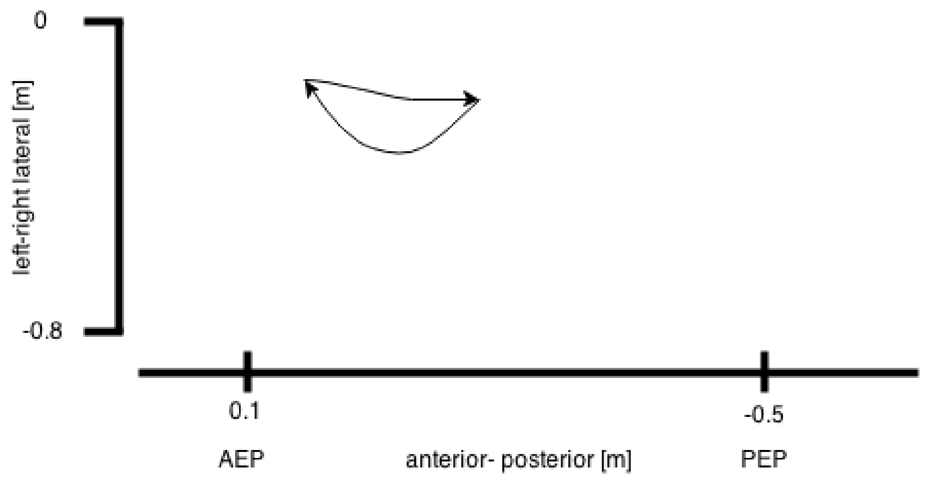


Figure 4.7: View from the top of hind leg movement. Without extra weight applied on the leg.

only a small amount of movement(shown in figures 4.6 and 4.7).This problem was partially removed by applying extra pressure/weight on the leg while in stance phase and produced the results shown in figures 4.8 and 4.9. Results from the implementation of the simplified version of the hind leg controller 3.4 part B were not included since in most cases the leg ended up stuck because its FTi joint flexed in such way that because of the leg's position of that moment it could not do an extension action. Therefore only the implementation of part C of the FTi joint was discussed in this section.

4.3.1 Bias Influence

- Neuron 20 - Its influence on the joint's actions was different than in the other two legs' controllers since in the hind leg's neural network neuron 19 receives an inhibitory signal from the FTi joint angle as opposed to the excitatory that was in the other two legs. In conducting my experiments I observed that in the hind leg this neuron's needed to have a small absolute value in order to capture fast changes in the neural network and in doing so provide a more stable behavior. A good value that was derived was:-0.6.

- Neuron 23 - Its bias had a large influence on the behavior of the leg since the hind leg pushes the body by depressing the CTr joint. This is the reason why the synapse pathway from this neuron to neuron 15 now has priority over the one from neuron 20 to neuron 15. The larger the absolute value that I tested for this bias the more the depression action was activated. A balancing value was determined to be -25.

- Neuron 15 - It influenced the behavior of the leg the most and even small changes of it produced very different leg movements. Larger values made the leg stuck because of over-depressing and smaller values made the leg not depress at all. Extreme values of this bias were not feasible and the best value determined was -2.

- Neuron 29 - Its optimal value was -17 and when being closer to 0 it flexed less times and when smaller it extended rarely.
- Neuron 32 - Optimal value was 16 and with bigger and smaller values influenced the behavior of the leg in the same way that they did in for neuron 29.
- Neuron 16 - It had the reverse effect than in the other two legs. This was due to the fact that the synapse pathways coming from the foot contact sensor have signs opposite from before. Its behavior was therefore the contrary - with higher absolute value it flexed more often and with smaller one it extended more frequently. The optimal value determined was 8.

4.3.2 Synapse Weight influence

The weight influence of the input pathways of neuron 15 were the same as in front and middle leg. In the case of neuron 16's ,where one of the synapse pathway of its inputs is prioritized. The larger the difference the more likely it was for the extension neuron to be activated and the smaller the difference was the more likely it is the FTi joint to produce a flexion action.

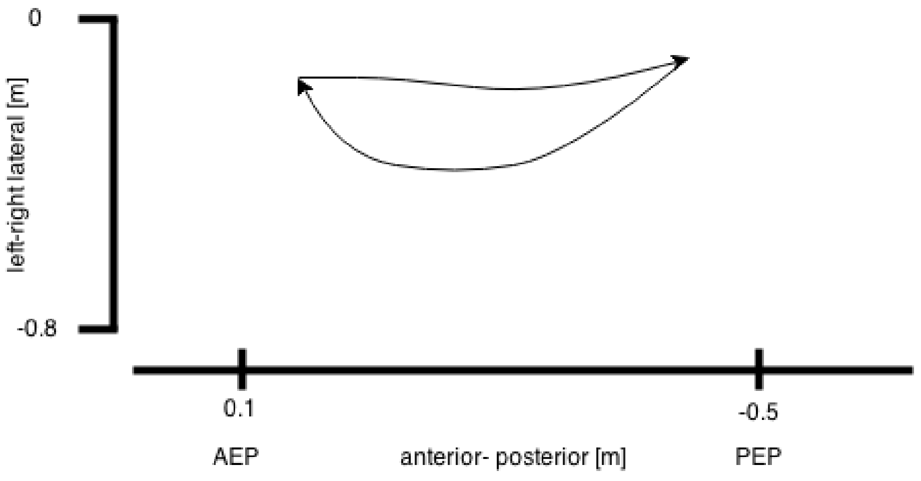


Figure 4.8: View from the top of hind leg movement with extra weight applied on the leg.

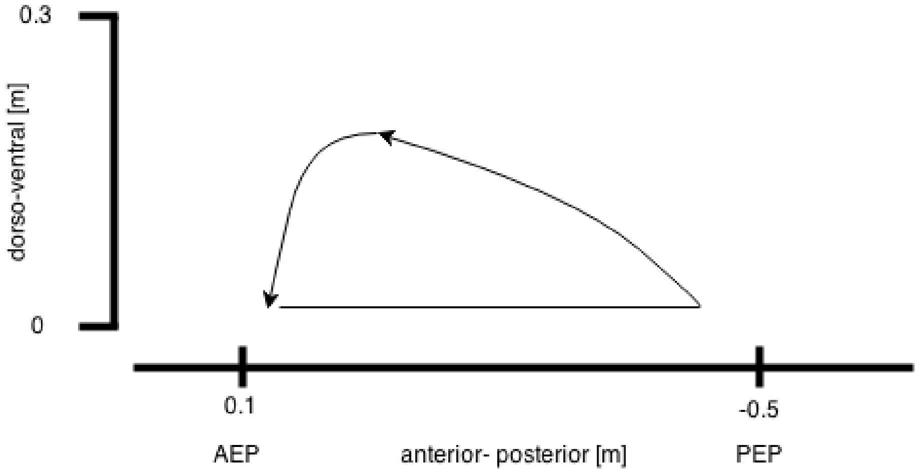


Figure 4.9: Side view hind leg movement with extra weight applied on the leg.

Chapter 5

Discussion

5.1 Feasibility of Neural Network Implementation

The forward walking controllers implemented in this project for the different legs produced satisfactory results, not including the hind leg's controller which could not produce the desired movement without extra weight being applied to it. It might not need that if the leg is installed on the whole robot with all the other controllers since the weight of the robot might provide the necessary force to keep the contact sensor activated all the way through its stance phase. Nevertheless even this behavior seemed biomimetic since a stick insect's behavior in a situation where it loses its grip with the ground might be the same. The model's implementation had other faults. The network parameters had to be hand tuned and even with that could not produce a stable behavior for every test. The working range of the ThC joint was sometimes drifting anterior or posterior when tested with different parameters. Hence in my implementation the tuning of the ThC joint's parameters was not discussed and the final implementation was based on the parameters given in [22], which in term provided better results. The other issue with the neural network was that no exact implementation of the servo modules was given and hence Its implementation needed to be designed and hand tuned to work as the described behavior suggested. Another issue was that no absolute joint velocities were considered for motor activation. Also the neural network model was not described in much detail and the influence of different parameters was not discussed in the original paper. Therefore there was a need to testing all ”free” parameter values in order to implement a properly working model.

5.2 Limitations of Robotic Model

In this section the limitations that a robotic model implementation of the original paper [22] and Yiming Yao's [23] designs are discussed. Alternative solutions and reasoning behind the need for them is presented.

5.2.1 Hind Leg Alternatives

There were several issues that arose when implementing the hind leg's controller. The main difference of the hind legs is that instead of pulling the body they are actually pushing it forward while in a stance phase. Therefore the foot contacts had a larger influence on the FTi's controller. Therefore in the paper [22] and in Yao's [23] implementation of the FTi's controller its actions were only dependent on the foot contact 3.4 part B. I implemented this simplified version of the module but in most cases the leg's behavior was to flex too much in the middle of a stance phase and lose the foot contact for good. The reason behind this behavior was because the foot contact was lost during the stance phase(which in hind legs is usually pushing the body by extending) and hence a it resulted in a swing phase and a flexion action being excited. The tibia's angle with the ground was then too small to detect the foot contact and go into another stance phase again. That was due to the surface of the foot contact being too small and the way it was installed on the leg. Hence an implementation that depended only on the foot contact was not feasible and the alternative solution was taken into consideration. The alternative 3.4 part C considers the FTi's joint angle in addition to the foot contact. This resulted in a behavior that was more stable and this implementation produced better results but still not the perfect results that were expected. By applying additional weight on the leg made the loss of the foot contact happen less but it still happened at some test cases. Another solution that might be considered in the future is using the motor's load to determine if the body is still being pushed in addition to the foot contact and the joint angle. I however did not have enough time to test this implementation and am suggesting it as future improvement of the current implementation.

5.2.2 Height Control Servo

The height control servo had several issues that needed to be addressed. First in the original paper [22] its input came from neurons 22 and 23 which represented the relative femur and tibia heights and was independent on the angle of the ThC joint nor the angle at which this joint was installed. This implementation is feasible if the model of the robot had the ThC joint installed on the Thorax (body) at a 0 degree angle as they mention in the paper. This influence is not taken into account in their robot implementation and hence in my design where it is at a 40 degree angle produced inaccurate results. The height of the femur and tibia change with the change of the ThC angle in this case so the implementation suggested in the original paper could not produce accurate results in my robot's implementation. Even in Yao's implementation [23] it was taking this offset as input but the output produced was not used properly. I therefore changed the height control servo to return 1 or -1 whether or not the desired CTr angle is different than the current one (with respect to the desired height that is set as a constant in the beginning of the code). In Yao's implementation it was returning the angle as output and this was afterwards sent to a neuron that was not described neither in Yao's nor in the original paper's implementation. The inStepLevation method represented this neuron and it took the resulting angle from the height servo + 50 (I could not follow the logic behind this and in fact Yao's CTr joint controller produced peculiar results). Also his implementation does not capture the idea of parallel neural pathways described in von Twickel's paper but instead introduces a new element and does not consider the case of in step depression. Also in the original paper [22] they did not consider the foot contact as input to this servo, whereas no adequate height computations can be done if the leg is in mid air. Therefore I implemented a new height servo that took the three joints' angles and the foot contact as input.

5.3 Previous Work Comparison

During the evaluation and implementation of Yiming Yao's middle leg controller there were several issues that arose and they are discussed in detail in this section. Also issues with the original neural network implementation [22] are analyzed and reasoning why alternatives were chosen is presented.

5.3.1 Angle Limits

The angle limits in Yao's implementation were not based on any logical data but were instead each joints motor maximum achievable angle. If his controller is then used in a the whole robot's control network the different legs might hit each other. And also there are proper angle limits for each legs' joint in [22]. I set them as stated in the implementation section as motor limits according to these angles.

5.3.2 Sensor Inputs

In Yimings design there was a simple sigmoid method that was used to determine the outputs of the sigmoid neurons. But since there are angular thresholds that need to be taken into account for different joint actions this implementation provided unstable results with the change of bias and weight values because the joint controller was independent on the angle thresholds described in Figure 3.1 and these parameters seemed to be hard-coded to his design. In order to fully implement the biomimetic controller discussed in the paper [22] I needed an alternative sigmoid function implementation. Therefore in order to take these angles into account I wrote the "Push_s_sensor" method in the hidden class. In the testing phase of the project the use of this function for the neurons that had a sensor input provided the exact behavior described in 2.3.

5.3.3 ThC Servo

The thorax-coxa module for the front and middle legs' neural networks was given by two alternating solutions in the original paper [22] one with neuron 14 being a simple sigmoid neuron with input pathway coming fron neuron 17 and the other it being a servo neuron. The version of it being a sigmoid neuron proved to be inefficient when

evaluating the behavior of the legs. Most of the test cases the foot was starting its swing phase prematurely in the middle of a stance. The reason behind this behavior was because the foot contact was lost during the stance phase and the protraction and retraction action only depending on the it as sensor input produced a premature excitory signal on the protraction neuron.
Another issue with the ThC servo was that in the original paper it takes as inputs the foot contact and the ThC joint angle and velocity. I implemented a case when it was using these three sensory inputs but the resulting behavior was unstable and unpredictable. The usage of the ThC velocity provided oscillations of the joint and in most cases resulting in an endless loop between a protraction and retraction action at small ranges (dorsal view). This was partially due to the fact that the tests were done on a leg that was attached to a spring but nonetheless when stabilizing the spring it still had similar unstable behavior. In my implementation the joint velocity is not taken into account and this implementation provided a smoother movement of the legs. Even in my implementation there was still the problem with the servo module because of the angle at which the foot contact switch was installed and its sensitivity. Its contact switch is smaller than the tip of the leg and when the angle between the tibia and the ground was smaller it lost the foot contact. This behavior was not that distinguishable with the front and middle legs but when testing the hind leg's controllers it was observable in most cases. The reason for this is because the hind leg is pushing the body in the stance phase its angle with the ground was changed more often and when losing the foot contact a new swing phase was instantiated hence its posterior extreme position was in most cases hardly reached.This however might not be the case when implementing the the controllers on the whole robot since the weight of the robot itself is enough to hold the foot contact switch on during the stance phase. On the other hand the behavior that was observed although looking inaccurate seemed to be more likely to be considered similar to a real stick insect's movement. This is because if the insect's grip with the ground is lost then it starts a new swing phase to reach a non-slippery or a surface that it would have a better grip with.

5.3.4 Synapse Weight and Bias Parameters

The weight and bias parameter values that were given by Yao [23] were working just for his middle leg controller and since his design was actually not based on the paper I could not use them. Therefore I used the given parameter values in the original paper and for the ”Free” ones I conducted thorough experimentation in order to understand their influence and choose a proper value.

5.3.5 Motor Layer Implementation

The motor layer that Yao implemented was not feasible for parameter testing since its activation conditions were hard coded and were not doing the behavior described in 2.3. I implemented simple and robust motor neurons methods that simulated the behavior Ekeberg described.

5.4 Outlook and Conclusion

As future work a more extensive testing needs to be done for all the legs in different perturbing conditions. The problems with the foot contact switch being lost can be addressed by either implementing a more sensitive foot contact sensor or use the joint load as sensory input.
I did not have enough time to implement the walknet controller discussed in the background section and can be considered as future expansion of the project. The walknet decribed in figure 2.8 can be implemented by using the controllers derived in this project but with in order to get the state (whether it is in a protraction or a retraction state) of each leg a new method that holds the value of the state of a leg needs to be implemented in each legs' controller. After implementing such a method then with the use of the rules one and two described in 2.6.2 a simple neural network can be designed for the brain module.
The design of Yao's [23] middle leg controller was tested and evaluated and based on his work and von Twickel's [22] paper neural network controllers have been implemented for the front, middle and hind left legs and their contra-lateral parts respectively. The influence of neural network parameters has been discussed in detail and optimal values for these parameters have been derived. Different problems of robotic representation of the leg controllers have been addressed and some for some solutions have been found and suggested. The biomimetic control model of the legs implemented has a modular structure that allows further modifications to be easily made. Thorough explanation of the inner workings of the model implemented, its setbacks and advantages compared to other designs and the influences of different joints in different legs is presented in this paper. This in term would help for easier future improvements and development of this project.

Bibliography

[1] Christine Azevedo, Bernard Espiau, Bernard Amblard, and Christine Assaiante. Bipedal locomotion: toward unified concepts in robotics and neuroscience. *Biological Cybernetics*, 96(2):209–228, 2007.

[2] Ulrich Bässler and Ansgar Büschges. Pattern generation for stick insect walking movementsmultisensory control of a locomotor program. *Brain Research Reviews*, 27(1):65–88, 1998.

[3] Alan Calvitti and Randall D Beer. Analysis of a distributed model of leg coordination. *Biological Cybernetics*, 82(3):197–206, 2000.

[4] Holk Cruse. What mechanisms coordinate leg movement in walking arthropods? *Trends in neurosciences*, 13(1):15–21, 1990.

[5] Holk Cruse, Volker Dürr, and Josef Schmitz. Insect walking is based on a decentralized architecture revealing a simple and robust controller. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1850):221–250, 2007.

[6] Holk Cruse, Thomas Kindermann, Michael Schumm, Jeffrey Dean, and Josef Schmitz. Walkneta biologically inspired network to control six-legged walking. *Neural networks*, 11(7):1435–1447, 1998.

[7] Silvia Daun-Gruhn. A mathematical modeling study of inter-segmental coordination during stick insect walking. *Journal of computational neuroscience*, 30(2):255–278, 2011.

[8] Volker Dürr. Context-dependent changes in strength and efficacy of leg coordination mechanisms. *Journal of experimental biology*, 208(12):2253–2267, 2005.

[9] Volker Dürr, Josef Schmitz, and Holk Cruse. Behaviour-based modelling of hexapod locomotion: linking biology and technical application. *Arthropod structure & development*, 33(3):237–250, 2004.

[10] Örjan Ekeberg, Marcus Blümel, and Ansgar Büschges. Dynamic simulation of insect walking. *Arthropod structure & development*, 33(3):287–300, 2004.

[11] geoffoddie. Stick insect. http://www.geoffoddie.com/stick-insect.

[12] Sten Grillner, Peter Wallén, Kazuya Saitoh, Alexander Kozlov, and Brita Robertson. Neural bases of goal-directed locomotion in vertebratesan overview. *Brain research reviews*, 57(1):2–12, 2008.

[13] William A Lewinger, H Martin Reekie, and Barbara Webb. A hexapod robot modeled on the stick insect, carausius morosus. In *Advanced Robotics (ICAR), 2011 15th International Conference on*, pages 541–548. IEEE, 2011.

[14] William A Lewinger, Brandon L Rutter, Marcus Blümel, Ansgar Büschges, and Roger D Quinn. Sensory coupled action switching modules (scasm) generate robust, adaptive stepping in legged robots. In *Proceedings of the 9th International Conference on Climbing and Walking Robots (CLAWAR'06), Brussels, Belgium, September 12-14*, pages 661–71. Professional Engineering Publishing, 2006.

[15] mbed org. mbed compiler. https://mbed.org/compiler/.

[16] mbed.org. mbed LPC1768. http://mbed.org/platforms/mbed-LPC1768/.

[17] KG Pearson and JF Iles. Nervous mechanisms underlying intersegmental coordination of leg movements during walking in the cockroach. *Journal of Experimental Biology*, 58(3):725–744, 1973.

[18] Georgios Petrou. hexapod video. https://www.youtube.com/watch?v=jQZfs4vVuUA.

[19] Roger D Quinn and Roy E Ritzmann. Construction of a hexapod robot with cockroach kinematics benefits both robotics and biology. *Connection Science*, 10(3-4):239–254, 1998.

[20] Robotis. dynamixel mx-28. http://support.robotics.com.

[21] Malte Schilling, Holk Cruse, and Paolo Arena. Hexapod walking: an expansion to walknet dealing with leg amputations and force oscillations. *Biological cybernetics*, 96(3):323–340, 2007.

[22] Arndt von Twickel, Ansgar Büschges, and Frank Pasemann. Deriving neural network controllers from neuro-biological data: implementation of a single-leg stick insect controller. *Biological cybernetics*, 104(1-2):95–119, 2011.

[23] Yiming Yao. Biomimetic control of a hexapod robot. Master's thesis, University of Edinburgh.

Appendix A

Appendix A - Control Class Code (Left Middle Leg)

```
void Control::Control_all(uint16_t speed)
{
    ctrlTimer.reset();
    ctrlTimer.start();
    int handler = 10 ;
    while(isWalking&&(handler>0))
    {
        int timerValue = ctrlTimer.read_us();
        if(timerValue >= ctrlInterval){
            //....................Read Inputs..................
            pb.mode(PullUp);
            double step_speed = (unsigned)speed;
            double tcAngle = (unsigned)jointTC.GetAngle();
            double tcVel = (unsigned)jointTC.GetVelocity();
            double ctAngle = (unsigned)jointCT.GetAngle();
            double ftAngle = (unsigned)(jointFT.GetAngle()-0x0400);
double f_Contact = !pb?1.0:0.0;

// Adjusting the angles to be the same as in Von Twickel's implementation.
tcAngle = tcAngle - 2048;
tcAngle = -tcAngle *0.088;
ctAngle = 270-ctAngle*0.088;
ftAngle = ftAngle*0.088;

//....................ThC Joint Control...........//
            // 7->17
            output17 = sigmoid17.Push_s_sensor(f_Contact,20,0.5);
// 1,2,17->14
            output14 = servo14.ThCServo(tcAngle,tcVel,output17);
// 14->8
```

```
Protraction.tcJointCtrl(output14,32,step_speed);
            // 14->9
            Retraction.tcJointCtrl(output14,-32,step_speed);


            //....................CTr Joint Control...........//
double dep_to_lev_ft = ftAngle-120;
double dep_to_lev_tc = tcAngle -(-25);
            double lev_to_dep_ft = ftAngle-70;

// 1->18
output18 = sigmoid18.Push_s_sensor(dep_to_lev_tc,-32,0.361);
// 5->19
            output19 = sigmoid19.Push_s_sensor(dep_to_lev_ft,32,0.750);
// 5->21
            output21 = sigmoid21.Push_s_sensor(lev_to_dep_ft,-32,0.472);
// 18,19 -> 20
output20 = orNeuron20.Push_or(output18,output19,32,32);
// 20,21->15
            output15 = sigmoid15.Push_s(output20,output21,32,-30);

   // HEIGHT CONTROLLER
output24 = servo24.CTrHeightServo(ftAngle,ctAngle,tcAngle,-4.2);
if (f_Contact>0.5){
if (output24>0){
Levation.ctJointCtrl(output24,32,step_speed);
} else {
Depression.ctJointCtrl(output24,32,step_speed);
}
}
// 15->10
            Levation.ctJointCtrl(output15,32,step_speed);
            // 15->11
            Depression.ctJointCtrl(output15,-32,step_speed);


            //....................FTi Joint Control...........

double flx_to_ext_ft = ftAngle - 105;
double ext_to_flx_ft = ftAngle - 105;

//5->25
            output25 = sigmoid25.Push_s_sensor(flx_to_ext_ft,32,0.667);
//7->26
            output26 = sigmoid26.Push_s(0,f_Contact,0,-20);
//25,26->27
```

```
            output27 = orNeuron27.Push_or(output25,output26,32,32);
//5->28
            output28 = sigmoid28.Push_s_sensor(ext_to_flx_ft,-32,0.667);
// 7->29
            output29 = sigmoid29.Push_s(0,f_Contact,0,20);
//28,29->30
            output30 = andNeuron30.Push_and(output28,output29,32,32);
            // 27,30->16
            output16 = sigmoid16.Push_s(output27,output30,-30,32);
// 16->12
Flexion.ftJointCtrl(output16,32,step_speed);
            // 16->13
Extension.ftJointCtrl(output16,-32,step_speed);

ctrlTimer.reset();
            handler = handler - 1;

        }

    }
}

Control::Control(uint8_t id1, uint8_t id2, uint8_t id3)
{
    jointTC.SetId(id1);
    jointCT.SetId(id2);
    jointFT.SetId(id3);

}
```

Appendix B

Appendix B - How to Run the code

How to run the code was the first thing that I needed to do in my implementation. There were several dependencies that needed to be installed in order to run it. I ran it on windows 8 and the first part was to register at the mbed.org website in order to be able to use their online compiler that is crucial to running code on the mbed. After registering in the compiler you can select the type of mbed that the code will be implemented for inside the mbed compiler workspace management.
After selecting the proper model of the mbed then the relevant libraries need to be included (the mbed library, the MX28 library and the SerialHalfDuplex). Each library can be imported in the online compiler either from your computer or from the website's repository.after having the relevant libraries the code can be compiled. If it is compiled successfully then a .bin file will be downloaded which can be put on the mbed via a usb cable. The current implementation I have provided is activited via the USB port with the help of a terminal application. One such application is the Terra term appliction that I was using.
In order to communicate with the mbed via a USB port the relevant drivers need to be installed on your computer (more information in the mbed's website)Keep in mind that in order for the communication to be established the baud rate inside the code that was compiled needs to be the same as the baud rate set in the terminal application. If that is the case then establishing a new connection via port3.0 and the command to start the simulation is the '1' key being pressed. To stop it you can press '2'.