applied sciences

MDPI

Article

Time Efficiency Improvement in Quadruped Walking with Supervised Training Joint Model

Chin Ean Yeoh [1], Min Sung Ahn [2], Soomin Choi [1,*] and Hak Yi [1,*]

[1] Department of Mechanical Engineering, Graduate School, Kyungpook National University, Daegu 41566, Republic of Korea
[2] Department of Mechanical and Aerospace Engineering, University of California, 420 Westwood Plaza, Rm 32-117E, Los Angeles, CA 90095, USA
* Correspondence: schoi@knu.ac.kr (S.C.); yihak@knu.ac.kr (H.K.); Tel.: +82-053-950-7541 (H.K.)

Abstract: To generate stable walking of a quadruped, the complexity of the configuration of the robot involves a significant amount of optimization that decreases to its time efficiency. To address this issue, a machine learning method was used to build a simplified control policy using joint models for the supervised training of quadruped robots. This study considered 12 joints for a four-legged robot, and each joint value was determined based on the conventional method of walking simulation and prepossessed, equaling 2508 sets of data. For data training, the multilayer perceptron model was used, and the optimized number of epochs used to train the model was 5000. The trained models were implemented in robot walking simulations, and they improved performance with an average distance error of 0.0719 m and a computational time as low as 91.98 s.

Keywords: supervised learning; quadruped robot; walking locomotion; multilayer perceptron

Citation: Yeoh, C.E.; Ahn, M.S.; Choi, S.; Yi, H. Time Efficiency Improvement in Quadruped Walking with Supervised Training Joint Model. *Appl. Sci.* **2023**, *13*, 2658. https://doi.org/10.3390/app13042658

Academic Editors: Hongbin Ma and Charlie Yang

Received: 26 January 2023
Revised: 7 February 2023
Accepted: 16 February 2023
Published: 18 February 2023

1. Introduction

Since the 1980s [1], the locomotion of quadruped robots has been widely investigated to ascertain a stable and more effective walking motion. Studies have typically involved either the moment of inertia for the entire robot system [2–4] or the zero-moment point (ZMP) [5,6] to optimize the trajectory for the robot's center of mass in order to allow the robot to walk in a stable fashion. To this end, the locomotion control inputs the optimized center-of-mass (COM) trajectory result into the inverse kinematics calculations to obtain the joint solution.

As a promising stable walking approach for a quadruped robot, the robotic dynamic could be modeled as the optimization objective, in which the walking stability, such as the zero-moment point or the inverted pendulum, would be derived as the constraint. However, the resulting calculations are tedious, at best. For example, Jin et al. argued that using an adaptive controller for quadruped locomotion (ACQL) could provide a recursive update. so that the external force of the control law could be incorporated with an inverse-dynamic-based quadratic programming (QP) method to realize an optimized trotting gait [7]. However, although the ACQL provided an optimum and stable walk for the quadruped, it could only do so at the expense of time efficiency. In another study, Chen et al. proposed the implementation of quadratic programming (QP) for the optimization of the model predictive control (MPC) of the inertia of a robot that could effectively generate optimum locomotion [8]. In this case, the robot walking dynamic involved a non-linear model, thus requiring the non-linear programming (NP) optimization method. Among the NP methods for optimizing robot trajectory, the method using mixed-integer quadratic programming (MIQP) has been the most promising for an optimized solution. MIQP has been related to the problem of optimizing a quadratic function over the points in a polyhedral set containing some integers and continuous variables [9]. This optimization was also implemented by Ahn et al. [10] for optimizing the center-of-mass trajectory for quadruped

robot locomotion control. This implementation demonstrated that the MIQP method could provide an optimal solution and offered a time-effective optimization process, as compared to other non-linear programming optimization methods. However, this optimization task required that every walking step be redefined, which affected the time efficiency related to the overall walking performance of the quadruped. To address this issue, this study extended the conventional locomotion manipulation method and implemented a machine-learning algorithm to simplify the control policy while generating the joint solution. By reducing the optimization time, the time efficiency of the walking simulation was noticeably improved. The simplified control policy could effectively generate results for the quadruped in a short time, which enhanced the computational time during locomotion. The advantage of this implementation was that it omitted the optimization time in the conventional method and stabilized the quadruped's walking locomotion.

Machine learning involves two main learning methods: reinforcement (unsupervised) learning and supervised learning. There has been substantial research on locomotion control for legged robots using reinforcement learning, which has provided a practical and general approach in building controllers. These trained controllers have been used to perform a range of sophisticated tasks. However, using the designated behaviors to create the reward functions necessary for reinforcement training involves a tedious task-specific tuning process. In addition, a trained model based on reinforcement learning can adopt unnatural behaviors that are dangerous and infeasible in real-world applications, particularly if the reward functions have not been designed carefully. Therefore, most studies that promote reinforcement learning for locomotion control will require choosing from a large controller family to develop a specific controller suitable to the task at hand. For example, Tan et al. [11] created a dynamic motor model for a "Minitaur" quadruped robot that was used to learn the walking policy for locomotion control. In another study, Jain et al. [12] proposed a hierarchical control approach for reinforcement learning to create a high-level and low-level control scheme for both the quadruped base and the overall locomotion control.

Supervised learning-based models could be trained with a reliable dataset for locomotion. This eliminates the unpredictability of exploring and exploiting an action for locomotion during the training process, such as reinforcement learning. The supervised learning approach would be more efficient than reinforcement learning methods from the standpoint of computational efficiency during the learning process. With a reliable dataset, a joint model training using supervised learning could yield lower inaccuracies in the joint solution prediction, which could then lead to an optimized trained joint model suitable for autonomous walking scenarios.

The benefits of a trained model could provide a one-off solution to support robotic control while maintaining time efficiency. By understanding the process of training models with supervised learning, the accuracy of a trained model could be pre-evaluated for reliability. Reliable control accuracy and time-efficiency in quadruped locomotion control have provided the motivation for this study. Although Hiroki et.al identified the benefits of supervised learning implementation in robotics, a training method for a high degree-of-freedom robot remains unsolved [13]. Therefore, we focused on training the individual joints model of the quadruped and then combining them to perform a walk.

Therefore, this study proposed an offline approach for building a simplified locomotion control policy using a supervised learning method for a 12-degree-of-freedom quadruped robot. The joint solution generated from the conventional method proposed by Ahn et al. [10] was used, which has been proven capable of performing stable quadruped walk motions and whose sample data were, therefore, suitable for generating a joint model using a supervised learning algorithm. Therefore, the dataset used in this study originated from the stable walking control algorithm proposed by Ahn et al.

After obtaining the joint model through the application of a supervised learning algorithm, the robot's leg joints were trained to predict the joint solution for a walking sequence. Furthermore, the data features from the previous study's dataset included the

robot states (robot position before and after each single-cycle walking motion) and the respective sequence of joint solutions. Each walking sequence involved the analysis of multi-input and multi-output data features for joint model training. Among the training methods available for supervised learning, the multilayer perceptron algorithm was ideal for implementing multi-input–multi-output (MIMO) training. Therefore, the multilayer perceptron method was implemented in this study. The trained models were compared with a test set (see Section 6.1) and were subsequently evaluated using a 3D quadruped robot in the GAZEBO simulation environment (see Section 6.2).

The remainder of this paper is organized as follows. The robot configuration is described in Section 2. The concept of ZMP is described in Section 3. The data generation method is explained in Section 4. An overview of the control policy is presented in Section 5. The evaluation results of the proposed control policy are presented in Section 6.

2. Robot Configuration

2.1. Robot Kinematic

The quadruped robot shown in Figure 1 is of a sprawling type with 12 degrees of freedom (DoF). The Denavit–Hartenberg (DH) parameters of the robot leg that referred to the first joint attachment with the robot body, O_i, are listed in Table 1.

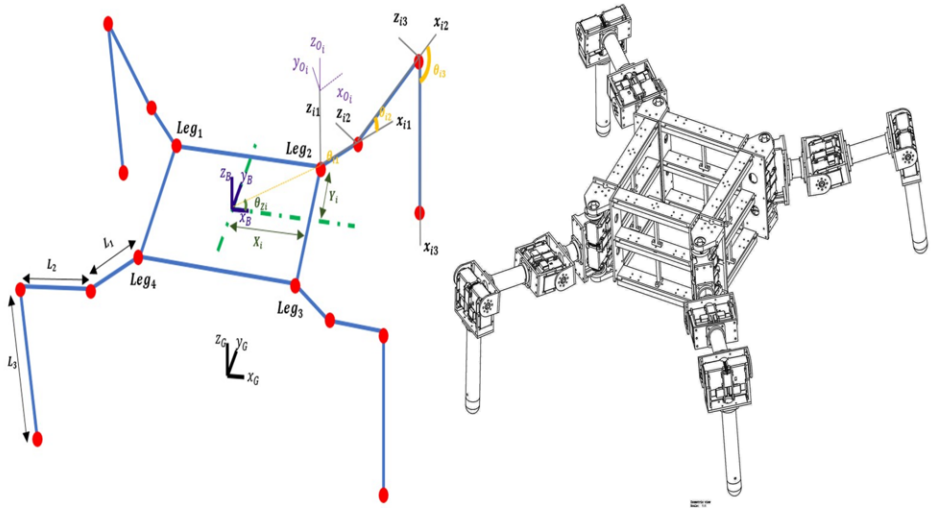

Figure 1. Quadruped robot configuration.

Table 1. DH parameters of each leg.

j	θ_{ij}	α_{ij}	a_{ij}	d_i
1	θ_{i1}	$-90°$	L_1	0
2	θ_{i2}	0	L_2	0
3	θ_{i3}	0	L_3	0

Notation i represents the leg number, and j represents the number of joints. The lengths of the links for each leg's component were $L_1 = 0.114$ m, $L_2 = 0.1389$ m, and $L_3 = 0.2484$ m.

The robot kinematics are described by [14]

$$
\begin{aligned}
{}^{0}A_{i3} &= {}^{0}A_{i1}^{1}A_{i2}^{2}A_{i3} \\
&= \begin{bmatrix} c\theta_{i1}c\theta_{i23} & -c\theta_{i1}s\theta_{i23} & -s\theta_{i1} & L_1c\theta_{i1}+L_2c\theta_{i1}c\theta_{i2}+L_3c\theta_{i1}c\theta_{i23} \\ s\theta_{i1}c\theta_{i23} & -s\theta_{i1}s\theta_{i23} & c\theta_{i1} & L_1s\theta_{i1}+L_2s\theta_{i1}c\theta_{i2}+L_3s\theta_{i1}c\theta_{i23} \\ -s\theta_{i23} & -c\theta_{i23} & 0 & -L_2s\theta_{i2}-L_3s\theta_{i23} \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{1}
$$

With the center of mass as the reference point, the position of the endpoint of a leg could be calculated as

$$\begin{aligned} {}^{B}\mathbf{T}_{i3} &= {}^{B}A_{i0}^{0}A_{i1}^{1}A_{i2}^{2}A_{i3} \\ &= \mathbf{T}_{rans}(X_i, Y_i, Z_i)R_z(\theta_{Zi})^{0}A_{i1}^{1}A_{i2}^{2}A_{i3} \end{aligned} \tag{2}$$

where the definitions for X_i,Y_i,Z_i, and θ_{Zi} are listed in Table 2. With the global coordinate as a reference, the position of the endpoint of a leg could be calculated as follows.

$$ {}^{G}\mathbf{T}_{i3} = \mathbf{T}_{rans}(x_{com}, y_{com}, z_{com})R_z(\theta_{yaw})R_y(\theta_{pitch})R_x(\theta_{roll})^{B}\mathbf{T}_{i3} \tag{3}$$

Table 2. Configuration of the first joint attachment of each leg to the robot's body frame.

i	X_i	Y_i	Z_i	θ_{Zi}
1	−0.1536 m	0.1285 m	0	39.9°
2	0.1536 m	0.1285 m	0	129.9°
3	0.1536 m	−0.1285 m	0	−39.9°
4	−0.1536 m	−0.1285 m	0	−129.9°

With the global coordinate as a reference, the configuration of each leg's first attachment point,O_i, to the robot body could be calculated as

$$ {}^{G}\mathbf{T}_{i0} = \mathbf{T}_{rans}(x_{com}, y_{com}, z_{com})R_z(\theta_{yaw})R_y(\theta_{pitch})R_x(\theta_{roll})^{B}A_{i0} \tag{4}$$

where x_{com}, y_{com}, and z_{com} are the translation distances of the robot's center of mass with respect to the global coordinate system along the x, y, and z axes, respectively. Variables θ_{yaw}, θ_{pitch}, and θ_{roll} denote the orientation information of the robot's center of mass with respect to the yaw, pitch, and roll axes of the global system, respectively.

2.2. Inverse Kinematics

The inverse kinematics could be derived through the analytic method, as follows. Referring to Figure 2, a leg's endpoint coordinates (p_{ix}, p_{iy}, p_{iz}), and the coordinates for the first attachment point of the leg to the robot body (O_{ix}, O_{iy}, O_{iz}) when referring to the global coordinate system are

$$\begin{aligned} {}^{G}\mathbf{T}_{i3}(1:3,4)^{T} &= (p_{ix}, p_{iy}, p_{iz}) \\ {}^{G}\mathbf{T}_{i0}(1:3,4)^{T} &= (O_{ix}, O_{iy}, O_{iz}) \end{aligned} \tag{5}$$

Using trigonometry rules, the joint angles of a leg were solved as follows.

$$\begin{aligned} \theta_{1_{pre}} &= atan2(O_{iy} - p_{iy}, O_{ix} - p_{ix}) \\ \theta_1 &= \theta_{1_{pre}} - \theta_{yaw} - \theta_{Zi} \end{aligned} \tag{6}$$

$$\begin{aligned} \theta_3 &= 180^{\circ} - \arccos\left(\frac{L_2^2 + L_3^2 - L^2}{2L_2L_3}\right) \\ L^2 &= (L_D - L_1)^2 + Z_{offset}^2 \\ Z_{offset} &= Z_{com} \\ L_D &= \sqrt{(O_{iy} - p_{iy})^2 + (O_{ix} - p_{ix})^2} \end{aligned} \tag{7}$$

$$\theta_2 = \arcsin\left(\frac{L_3}{L}\sin(180^{\circ} - \theta_3)\right) - atan2\left(\frac{Z_{offset}}{L_D - L_1}\right) \tag{8}$$

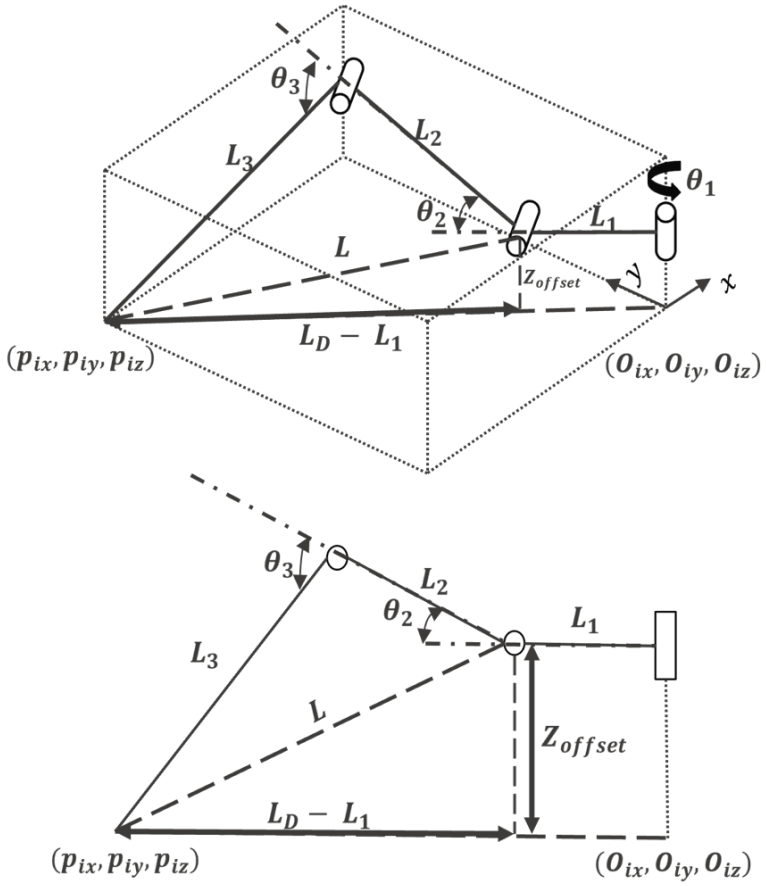

Figure 2. Analytical solution for inverse kinematics.

Because the attachment of the link between the first and second joints of the leg structure was considered for a flat surface, Z_{offset} was always defined to be the same height as Z_{com}.

3. Zero-Moment Point (ZMP)

The ZMP is the point derived from the Newton-–Euler equation of motion applied to a multi-body system. This was the only element used as the verification stability index of the legged robot. If the foot was in contact with the ground along its entire lower surface, then the ZMP was inside the sole. Thus, locomotion was stable if the ZMP remained within the footprint polygons, which could be expressed as follows.

$$\begin{aligned} x_{zmp} &= \frac{\sum_{i=0}^{n} m_i(\ddot{z}_i + G_z)x_i \; - \sum_{i=0}^{n} m_i(\ddot{x}_i + G_x)z_i}{\sum_{i=0}^{n} m_i(\ddot{z}_i + G_z)} \\ y_{zmp} &= \frac{\sum_{i=0}^{n} m_i(\ddot{z}_i + G_z)y_i \; - \sum_{i=0}^{n} m_i\left(\ddot{y}_i + G_y\right)z_i}{\sum_{i=0}^{n} m_i(\ddot{z}_i + G_z)} \end{aligned} \tag{9}$$

In this study, it was assumed that the gravitational acceleration in both the x and y directions was near zero; therefore, the ZMP calculation were modified as follows.

$$\begin{aligned} x_{zmp} &= x - \frac{z\ddot{x}}{(\ddot{z}_i + G_z)} \\ y_{zmp} &= y - \frac{z\ddot{y}}{(\ddot{z}_i + G_z)} \end{aligned} \tag{10}$$

4. Generation of the Dataset

The collection of a reliable dataset that guaranteed walking stability relies on the method for generating the optimized COM of the robot (the conventional method of locomotion manipulation) described in [10]. The implementation details of the dataset used in this study were as follows.

4.1. Data Generation through COM Trajectory Optimization

The pseudo-code shown in Algorithm 1 was used to generate the dataset used in this study. To achieve a stable walking motion after inputting the feature vector (robot COM coordinates specifying the start and goal positions), a footstep could be predefined after checking the target direction. The robot COM trajectory was optimized by referring to a previous research method [10] that implemented the MIQP method and ZMP theory. Then, the vector of the control parameter (joint solutions to be sent to the robot in GAZEBO) was obtained from the inverse kinematics solution. Each optimization provided a single dynamically feasible path over one or more steps, from which the feature and control parameter vectors were extracted at each step. The dataset could be built from a few to as many as a few thousand optimizations. The COM optimization rule was established according to the constraint specified in Table 3 to minimize the sum of the second derivative of the robot's states.

Algorithm 1: Pseudo-Code for Data Collection.

Input goal, $G = (g_x, g_y)$
Robot pose, $S = getPos$
for $(i : 4)$:
  Footprint polygons, $fp = Footstepredefine(S, G)$
  COM trajectory, $Y = Op(fp)$
  Joint solutions, $\theta = IK(Y)$
  $save(\theta)$
$save(S, G)$
Target distance = $|G - S|$
if Target distance $< 0.15m$:
  END
  else: return 2

Table 3. COM Optimization Constraint.

Equality Constraint	The continuous of two trajectories: $x_i(T_{end}) = x_{i+1}(T_{initial})$ $\dot{x}_i(T_{end}) = \dot{x}_{i+1}(T_{initial})$ $\ddot{x}_i(T_{end}) = \ddot{x}_{i+1}(T_{initial})$
Inequality Constraint	To guarantee that the ZMP is always inside the support polygon: $px_{zmp} + qy_{zmp} + r \leq 0$

In quadratic form, the cost function was codified from the integration of the square of the acceleration of the trajectory.

$$cost\ function = \begin{bmatrix} \frac{400}{7}t^7 & 40\ t^6 & 24\ t^5 & 10\ t^4 & 0 & 0 \\ 40\ t^6 & \frac{144}{5}t^5 & 18\ t^4 & 8\ t^3 & 0 & 0 \\ 24\ t^5 & 18\ t^4 & 12\ t^3 & 6\ t^2 & 0 & 0 \\ 10\ t^4 & 8\ t^3 & 6\ t^2 & 4\ t & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (11)$$

Figure 3 shows the joint trajectory in time-series form, collected from a change of COM (from 0.0 m to −0.25 m) in quadruped locomotion.

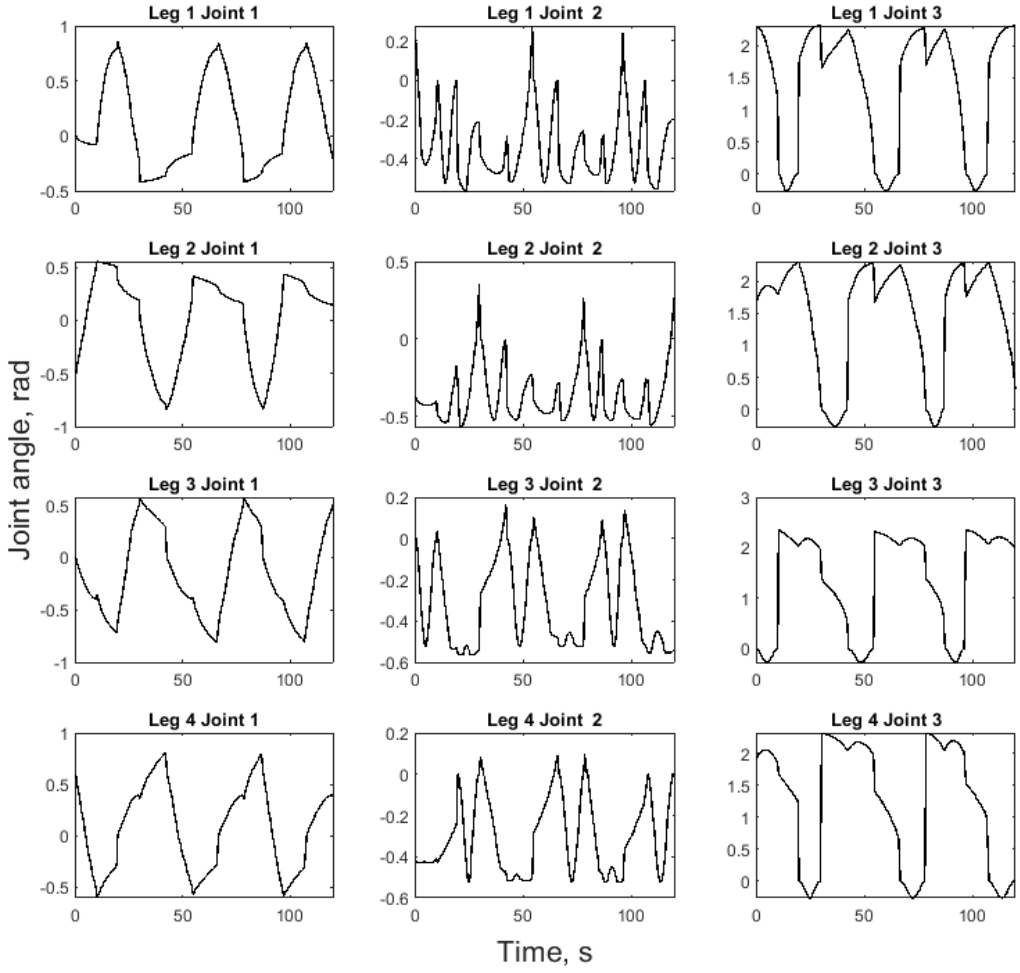

Figure 3. Data of robot joint trajectories.

5. Control Policy

Figure 4 presents an overview of the control policy design. The control policy proposed in this study related a vector consisting of features to a set of joint control parameters. In this supervised learning, the policy was built using the supervised learning technique, together with the joint solutions collected, and its input and output labels were the robot state positions and the joint solutions of the legged robot, respectively. The control policy was used to generate the joint position through the position controller and send it to the robot's actuator when walking. In this study, there were a total of 12 joints model being trained, as the robot used had of 12 actuators, or 3 for each leg. To proceed with supervised learning, the following step should be taken into account:

- Selection of vector features and control parameters
- Preprocessing the dataset prior to model training
- Architecture of training model
- Model evaluation

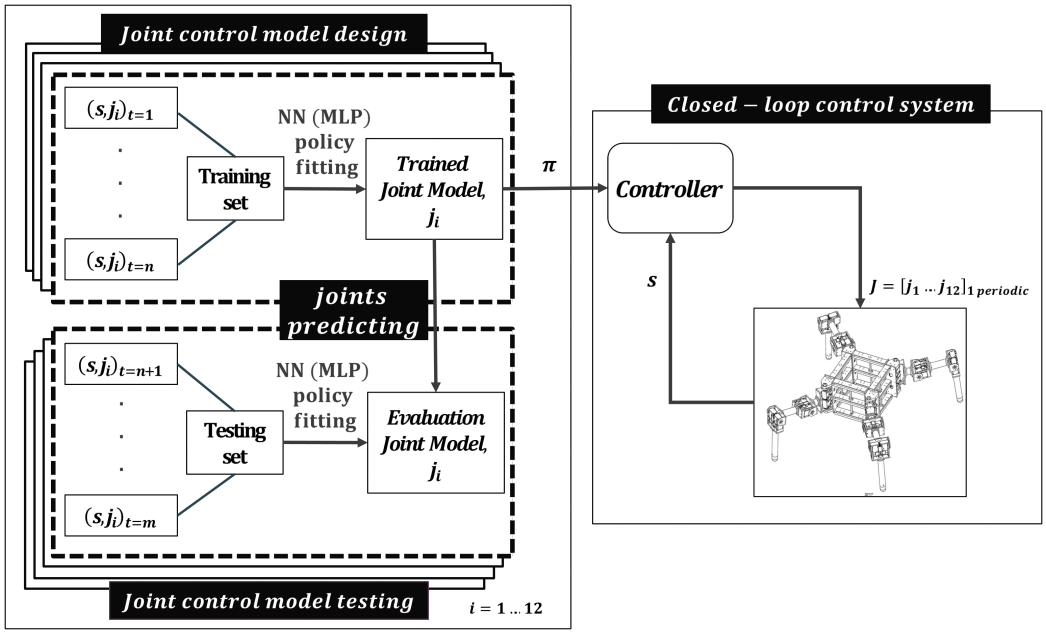

Figure 4. Control policy design and implementation.

5.1. Selection of Vector Features and Control Parameters

The feature vector specified the robot's center-of-mass position before and after one periodic walking motion. The joint control parameters were the joint solutions for one walking period. The control policy in this study was a function that mapped the feature vector to a vector of control parameters. To accomplish this, the collected feature vector and the respective vector of the control parameter had to be dimensionally consistent to prevent training failure. In this study, the feature vector included the information of the robot's current position and the goal position. However, the respective vector of the control parameter was the output of the individual joint of the robot. The dimension of these vectors are described in Section 5.2.

5.2. Preprocessing the Dataset Prior to Training

The simulation of the walking robot was performed 400 times to extract as much information as possible from the dataset. It was then necessary to rearrange the shape of the collected dataset vector prior to its use in machine learning. Every iteration of the simulation in Section 4.1 involved only one stepping motion for a single leg. In contrast, every four iterations defined one walking cycle. However, the movement of the robot from one point to the target position could involve different walking cycles, depending on the distance traveled. Therefore, the joint model had to be trained based on walking cycles to ensure a consistent shape for the dataset. Therefore, every four consecutive iterations of the dataset collected from the simulation were considered one output dataset, and the input data were extracted from every four iterations of the input dataset recorded during the simulations. In this case, every iteration of the simulation of the walking motion consisted of 60 time steps, which resulted in 60 values for each joint for each iteration. Therefore, the vector of each set of the dataset (encompassing four iterations) for a single joint was 240 × 12, where 12 was the DoF. Furthermore, the input dataset was 1 × 4, which specified the x and y coordinates of the robot's current and goal positions. When cleaning the collected data, fewer than 240 time steps were eliminated. Because 12 joints must be trained for the 4-legged robot evaluated in this study, the joint model training was performed separately. Each joint model was individually trained, resulting in 12 trained models using input and output datasets prepared in one-dimensional form, such that a total of 2508 datasets were obtained for each joint model.

5.3. Architecture of Training Model

The construction of the neural network structure is shown in Figure 5, and the details of the joint model used for training purposes are described in the following subsections.

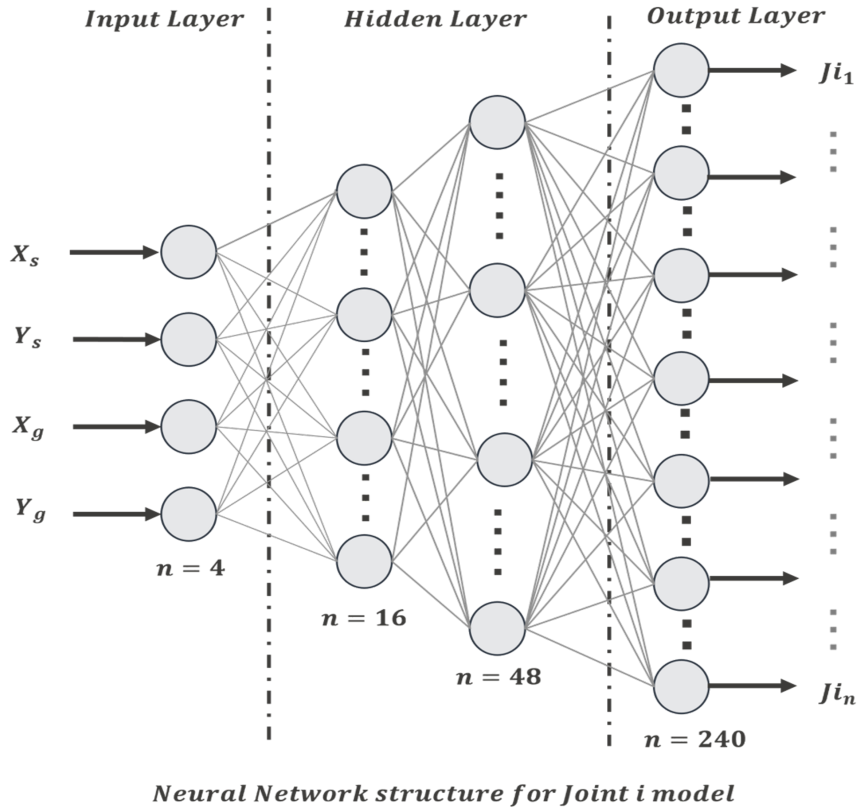

Figure 5. Neural network model structure.

5.3.1. Training Method

To train a model to predict a continuous sequence of joint angle values, a multi-layer perceptron (MLP), which is a typical example of a feed-forward artificial neural network (ANN), was used. Provided that ANN had the benefit of learning a continuous function, the model could demonstrate a smoother relationship between changes in input and output [15–17]. In addition, this study involved 4 input states and 240 outputs in a model that could handle the multiple input–output training tasks to learn the continuous sequence of the joint solution [18–20].

5.3.2. Activation Function

Because the input data in this study had to represent a branch of an output, the rectifier linear unit (ReLU) activation function was used to process the input weight values of the hidden layers in this study. ReLU could overcome numerical problems related to the sigmoid activation function [21,22].

5.3.3. Layers

The layers in a deep-learning model defines the structure or network topology of the model. Each layer usually takes information from a previous layer, processes the information, and then passes it to the next layer. In this study, the type of layer used was a dense layer. A dense layer, also known as a fully connected layer, is a layer in which all inside nodes connect to each node in the preceding layer through weighted lines. As there were four inputs and 240 outputs for each dataset, the input layer consisted of four nodes, and the output layer consisted of 240 nodes. Our architecture used two hidden layers: the first hidden layer consisted of 16 nodes, and the second hidden layer consisted of 48 nodes,

which were chosen after some trial and error. The kernel initializer was set to $he\_uniform$ and was an initializer that drew samples from a uniform distribution within (−limit, limit), where the limit was $(6/n_{input})^{1/2}$ [23].

5.3.4. Learning Method

After receiving a value processed through the activation function of a layer, the learning process in the MLP changed the connection weight based on the error between the expected result and the actual result in the output for each input data vector. The weight changing process was performed through backpropagation. This study was generalized using the least mean square algorithm to minimize the error between the expected value and the actual value by representing the degree of error for the jth output node in the n th data point as $e_j(n) = d_j(n) - y_j(n)$, where d is the target value and y is the current value produced by the perceptron. Each weight could then be adjusted to minimize the error in the entire output layer. In general, gradient descent was used to change the weights connected to each node using $\Delta w_{ji}(n) = -\eta \partial \xi(n) \partial v_j(n) y_i(n)$ where y_j is the output obtained by the previous learning step and n is the learning rate, which was selected to ensure that the network quickly converged to the desired response without oscillations [24,25]. In this study, the adaptive moment estimation (Adam) was used to implement the backpropagation algorithm [26]. After setting the parameters for the neural network model, it was compiled with the loss function mean squared error (MSE) [27], and Adam was used as the optimizer. In this study, the epochs used for model learning were 100, 500, 1000, 5000, 7500, and 10,000. By investigating the simulation performance of a trained model, which varied with the number of epochs, the optimized number of epochs was identified and is discussed in the Results and Discussion sections.

5.4. Model Evaluation

To ensure the prediction accuracy of a trained model, each model was evaluated by comparing the prediction solution of the trained model to the expected outcome using the test dataset. In this study, the number of training and test samples were 1946 and 562, respectively. The k-fold [28] data splitting of the training and test sets was applied to validate the algorithm (parameters $n\_split = 4$, $n\_repeat = 240$, and $random\_state = 1$). The prediction accuracy of the model was measured according to the mean squared error, where a lower value indicated a higher model prediction accuracy.

The language used to train the model was Python, using the tensorflow [29] and keras [30] libraries. Table 4 lists the model training parameters used in this study.

Table 4. Model Training Parameters.

Training Parameter	Values
Neural network used	MLP
Node number in input layer	4
Activation function input layer	ReLu
Node number in output layer	240
Number of hidden layers	2
Number nodes in hidden layer	1st layer: 16; 2nd layer: 48
Activation function for hidden layers	ReLu
Kernel initializer limit	he_uniform
Batch size	1
Loss function for model training	Mean square error (MSE)
Optimizer	Adam

6. Results and Discussion

To evaluate any enhancement in time efficiency using the proposed method for robot locomotion, the proposed method was compared to the conventional method proposed by Ahn et al. [10]. To compare the performance of both methods, they were tested 5 times

using the same parameters, including the same starting and goal positions of (0, 0) and (0, −0.25) as well as for the distance allowance. Table 5 shows all results for the distance error (D, units in centimeters (cm) and time T, units in seconds (s)) after completing the walking simulations, with the respective average value (Ave.) shown for each measure. The result showed that the average time spent to complete the walk was half the time, as compared to the conventional method, with a result of 91.9862 s. These results proved that the proposed method significantly improved the locomotion time efficiency of the quadruped robot and enhanced its accuracy to arrive at the desired target coordinate.

Table 5. Comparison of walking performance between the conventional method (CM) and the proposed method (PM).

	CM		PM	
	D	**T**	**D**	**T**
1	9.2522	169.756	9.6576	88.8268
2	1.0878	273.166	2.9283	93.3085
3	10.4994	173.572	4.3526	94.2199
4	10.6994	160.542	9.8244	91.2268
5	8.3304	171.874	9.2229	92.349
Ave.	7.9738	189.7820	7.1972	91.9862

There were two evaluation methods that could be used to investigate the joint model performance: comparing the predicted parameters from the trained models with the test dataset, and testing the trained model in a walking simulation. The results are presented in Sections 6.1 and 6.2, respectively.

6.1. Trained Model Evaluation

In this study, the joint models were trained using different numbers of epochs (100, 500, 1000, 2500, 5000, 7500 and 10,000) to identify the best prediction model. The results are shown in Figure 6.

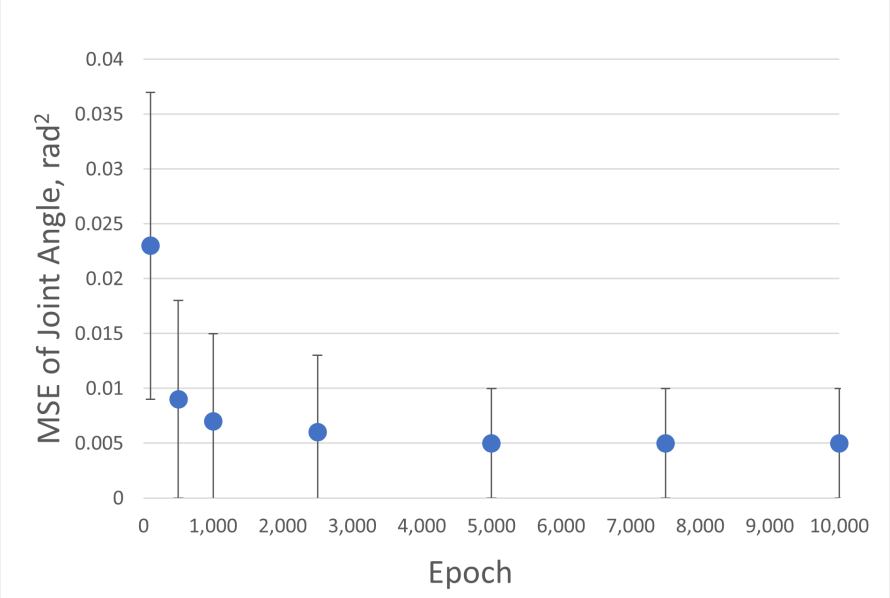

Figure 6. Model evaluation results.

The results indicated that joint models trained using more than 5000 epochs began showing a saturated result for the root mean squared error at 0.005 radians^2 with a standard deviation value of 0.005 radians. This result indicated that the accuracy of the joint prediction was comparatively higher than that of the models trained using fewer than

5000 epochs, indicating that the latter exhibited an under-fitting performance. That is, the number of training epochs was insufficient to enable the trained model to provide accurate predictions. In contrast, joint models trained using more than 5000 epochs yielded similar evaluation results as when 5000 epochs were used; therefore, the optimized number of training epochs was 5000 for the model architecture used in this study. To verify the performance of the trained models, a walking simulation was conducted to evaluate the distance error and the total running time, as discussed in the following section.

6.2. Walking Simulation

The joint models trained using different epoch numbers were used to predict the joint solution in the walking simulation five times for each model. Using the conventional walking simulation based on the COM optimization as the benchmark for this study, the running time of the simulation was limited to 250 s for performing a walk from the origin (0 m, 0 m) to the goal (0 m, −0.25 m). The average results for the distance error and running time were calculated and are plotted in Figures 7 and 8, respectively.

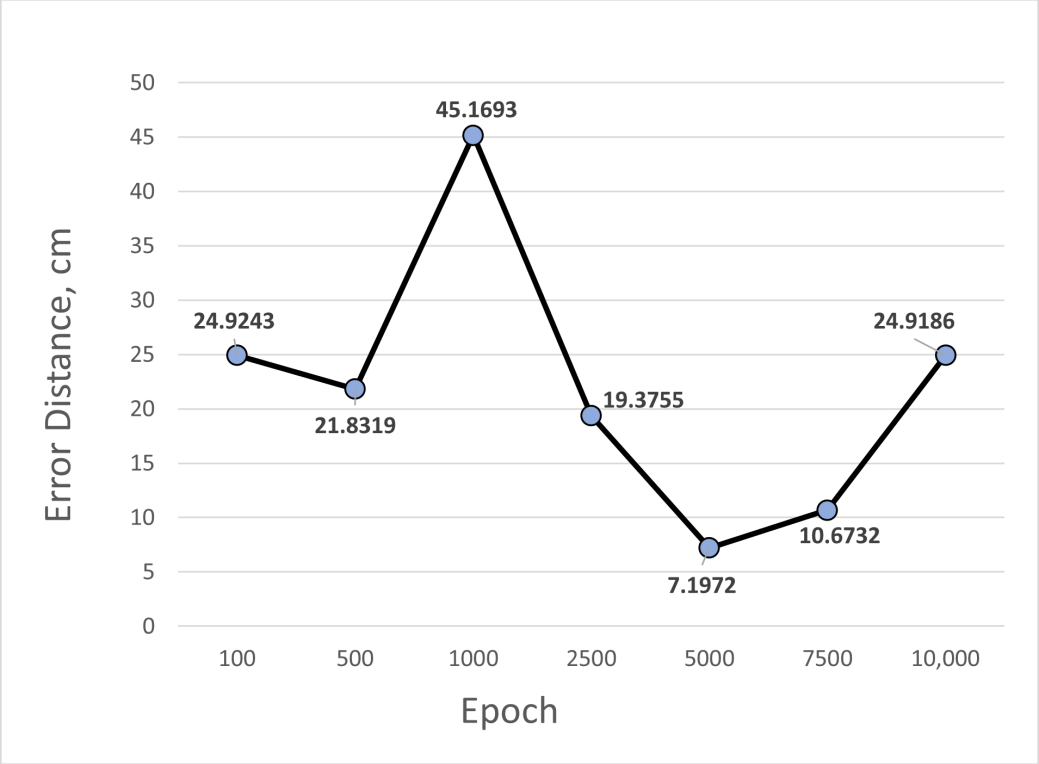

Figure 7. Average distance error versus the number of epochs.

Referring to Figure 7, although there was some fluctuation in the distance error as the number of epochs increased, the distance error for the joint model trained using 5000 epochs achieved the lowest distance error of 7.197 cm. This value was even lower than that of the conventional walking simulation (7.9738 cm) using COM optimization. The trained model, therefore, enhanced the walking performance by reducing the distance error.

Figure 8 shows the curve of the simulation running time versus the number of epochs. The shortest running time of 91.986 s for a successful walk was achieved using the model trained for 5000 epochs. This was consumed by the conventional walking method simulation (189.7820 s) approximately half of the time.

The model evaluation and simulation results showed that an epoch number of 5000 was the optimum number for training the joint models used in this study. These results also proved that the supervised trained joint model enhanced walking simulation performance by reducing both the distance error and the computational time.

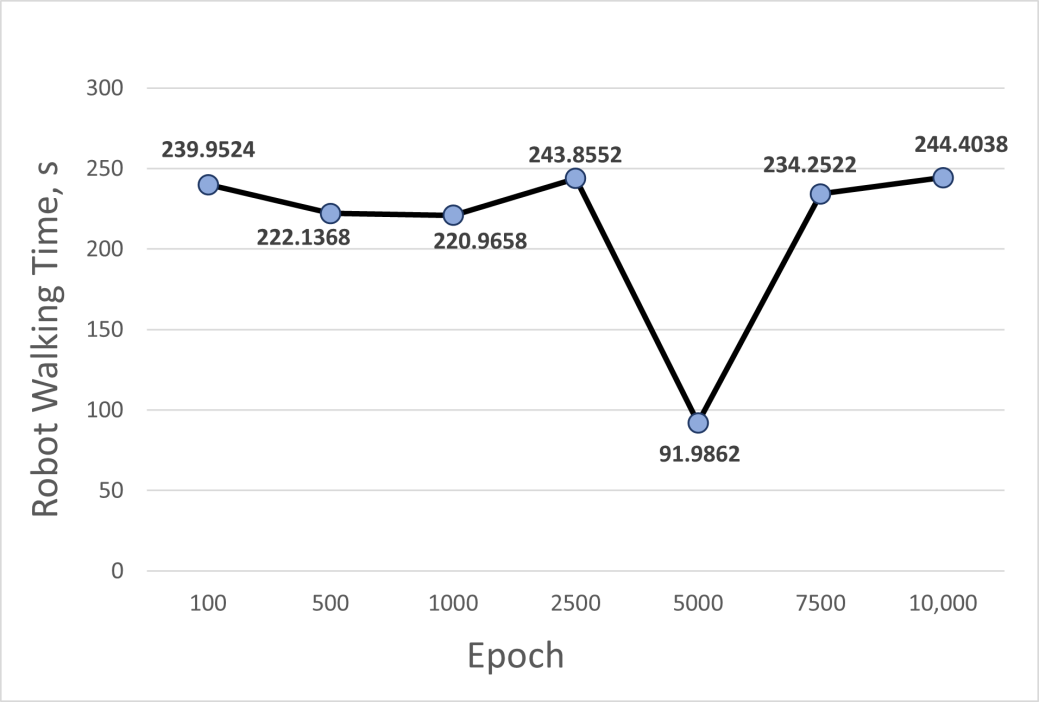

Figure 8. Average running time versus the number of epochs.

Figure 9 shows each trajectory of the robot's joints with respect to a change in the robot's COM in the Gazebo simulation (Figure 10). Trajectories of each joint in Figure 9 proved to have a similar pattern as the data that were collected in Section 4.1 and the data generated in Figure 3. The trained joint model in Section 6.1 evaluated the accuracy of the robot locomotion as reliable for stable walking.

Furthermore, to verify the robot's walking stability, the rotational angle of the robot (in both the pitch and roll directions) was recorded during the walking simulations. Table 6 indicates that the minimum and maximum values for the angle in the pitch and roll directions ranged from −0.0308 to 0.0554 rad and from −0.0433 to 0.0242 rad, respectively. The angle variation gap for the pitch direction was 0.0987 rad, and that for the roll direction was 0.0675 rad, which was much lower than the angle values that could cause an unrecoverable tilting action. Therefore, the walking simulation indicated that a stable walk could be achieved by implementing the trained joint models using the method proposed in this study.

Table 6. Rotational angle of robot's center of mass in pitch and roll directions (units in radians).

	Pitch		**Roll**	
	max	**min**	**max**	**min**
1	0.0468	−0.0335	0.0246	−0.0433
2	0.0466	−0.0320	0.0260	−0.0336
3	0.0554	−0.0326	0.0242	−0.0344
4	0.0485	−0.0356	0.0260	−0.0406
5	0.0408	−0.0308	0.0250	−0.0320

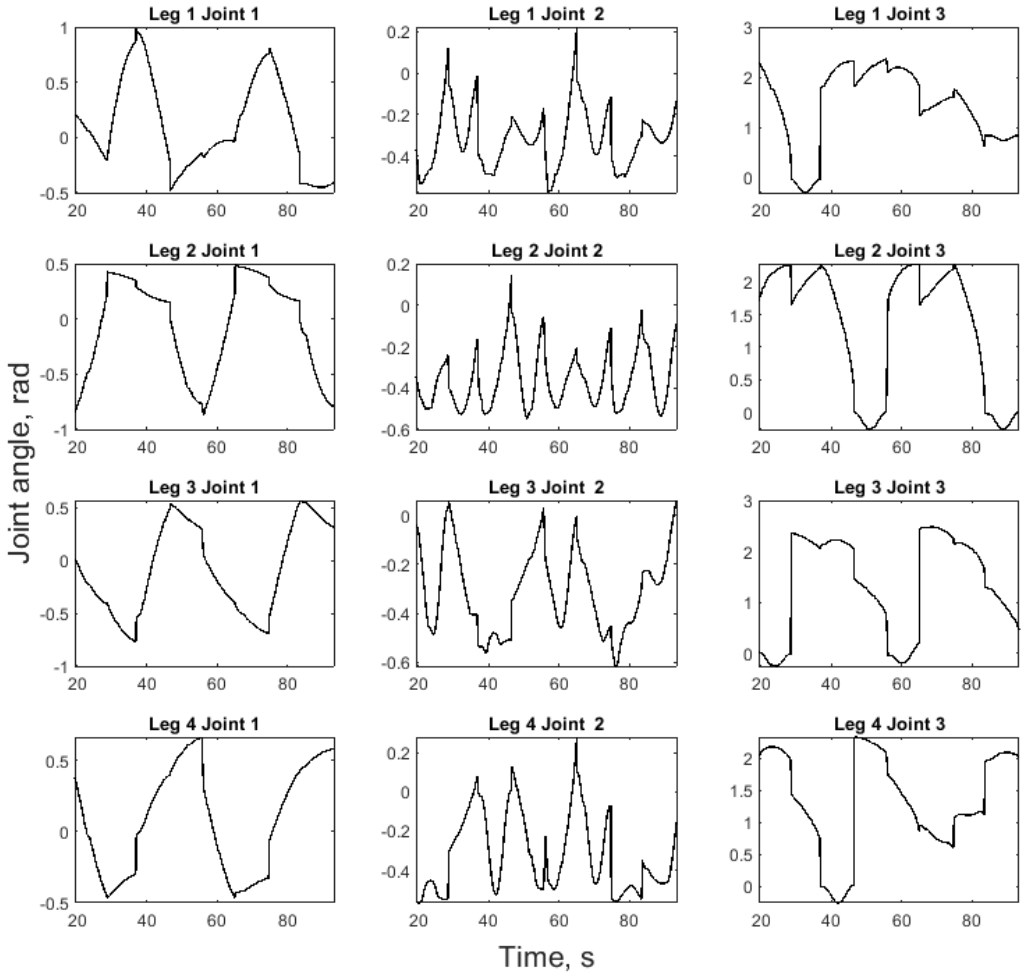

Figure 9. Robot joints trajectories.

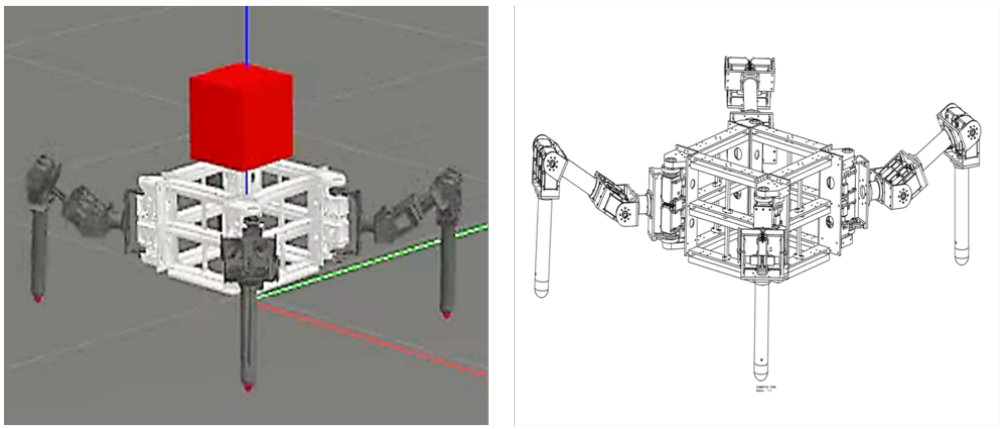

Figure 10. Robot in simulation (Gazebo) and CAD model.

7. Conclusions

In this study, the employment of supervised learning could assist in improving the conventional method of robotic control. Implementing the dataset generated from the COM optimization solution based on MIQP and ZMP theory to train the joint models of a four-legged robot was a promising method for achieving suitable walking stability. In general, quadruped robot locomotion involves 12 degrees of freedom (DoF) for locomotion control, which has proved to be difficult when defining the reward rules that are needed in reinforcement learning. In deep reinforcement learning, to achieve higher rewards,

12 variables must be manipulated through trial and error, which may require a long or unpredictable running time and may cause unpredictable robot behavior. In contrast, by providing the collected data from the simulations, using both the joint output and robot state information, as the input could be sufficient to train the joint models for successful quadruped walking tasks. In addition, training quadruped locomotion requires high computational performance to manipulate and alter the memory of the learning process [11,12]. Moreover, the proposed method of this study provided a light computational process to train a high DoF locomotion robot, resulting in a time-efficient model.

Therefore, employing a multilayer perceptron architecture for supervised training of the multi-joint models of the quadruped was more time-effective and lower in cost than deep reinforcement learning. For validation of the trained joint models, each model was shown to have a low mean squared error value, indicating that the model prediction could perform a joint solution prediction close to the joint solution generated by the COM-optimized algorithm. Meanwhile, the reliability analysis of this trained model for an autonomous walking simulation showed that the walking robot could reduce the distance error tolerance, as compared to the simulation using the conventional method, as well as the total computational time required. Therefore, implementing the dataset from the COM optimization study and using a multilayer perceptron for each joint model under supervised learning was an effective method for simplifying the control policy for autonomous walking of a quadruped robot. Though the outcome of this study, in terms of the time efficiency and accuracy enhancement, met the desired objectives, it could be further extended by combining the perception module for acquiring the environment state information.

To stress the significant contribution of this study's proposals, it should be noted that we built a correct schematic for joint model training. This should help to speed up computations in robotics applications. In particular, the contribution of this work in artificially intelligent robots may extend its applicability to robotic research.

Author Contributions: Conceptualization, C.E.Y. and M.S.A.; methodology, C.E.Y. and M.S.A.; software, C.E.Y. and M.S.A.; validation, C.E.Y. and M.S.A. and H.Y.; formal analysis, C.E.Y.; investigation, C.E.Y.; resources, C.E.Y.; data curation, C.E.Y.; writing—original draft preparation, C.E.Y. and M.S.A.; writing—review and editing, C.E.Y., M.S.A., S.C. and H.Y.; visualization, C.E.Y. and M.S.A.; supervision, S.C. and H.Y.; project administration, S.C. and H.Y.; funding acquisition, S.C. and H.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by the BK21 funded by the Ministry of Education, Republic of Korea (4199990314305).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The related research data are available in the respective repository: https://github.com/ycean/time-efficient-improve-for-walking-robot.git (accessed on 12 October 2022). The joint model training code (Train_Joint_Model5000.py) is available in the respective repository: https://github.com/ycean/time-efficient-improve-for-walking-robot/blob/main/Train_Joint_model5000.py (accessed on 11 October 2022).

Acknowledgments: The authors would like to extend their appreciation for the technical support by the laboratory members from the Robotics and Automation and Control of Kyungpook National University.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ZMP	Zero-moment point
COM	Center of mass
ACQL	Adaptive controller for quadruped locomotion
QP	Quadratic programming
MPC	Model predictive control
NP	Non linear programming
MIQP	Mixed-integer quadratic programming
MIMO	multi-input multi-output
DoF	Degrees of freedom
DH	Denavit-Hartenberg
MLP	Multi-layer preceptron
ANN	Artificial neural network
ReLU	Rectifier linear unit
Adam	adaptive moment estimation
MSE	Mean square error
D	distance error
Ave.	Average value
CM	Conventional method
PM	Proposed method
T	Time
CPU	Central processing unit

References

1. Raibert, M.H.; Tello, E.R. The stable walk locomotion. In *Legged Robots That Balance*; Patrick, W., Michael, B., Eds.; MIT Press: Cambridge, MA, USA, 1986; pp. 10–15.
2. Liu, M.M.; Qu, D.K.; Xu, F.; Zou, F.S.; Di, P.; Tang, C. Quadrupedal robots whole-body motion control based on centroidal momentum dynamics. *Appl. Sci.* **2019**, *9*, 1335. [CrossRef]
3. Lee, Y.H.; Lee, Y.H.; Lee, H.; Kang, H.; Lee, J.H.; Park, J.M.; Kim, Y.B.; Moon, H.; Koo, J.C.; Choi, H.R. Whole-Body Control and Angular Momentum Regulation using Torque Sensors for Quadrupedal Robots. *J. Intell. Robot. Syst.* **2021**, *102*, 1–15. [CrossRef]
4. Mastalli, C.; Havoutis, I.; Focchi, M.; Caldwell, D.G.; Semini, C. Motion Planning for Quadrupedal Locomotion: Coupled Planning, Terrain Mapping, and Whole-Body Control. *IEEE Trans. Robot.* **2020**, *36*, 1635–1648. [CrossRef]
5. Fahmi, S.; Mastalli, C.; Focchi, M.; Semini, C. Passive Whole-Body Control for Quadruped Robots: Experimental Validation Over Challenging Terrain. *IEEE Robot. Autom. Lett.* **2019**, *4*, 2553–2560. [CrossRef]
6. de Viragh, Y.; Bjelonic, M.; Bellicoso, C.D.; Jenelten, F.; Hutter, M. Trajectory optimization for wheeled-legged quadrupedal robots using linearized ZMP constraints. *IEEE Robot. Autom. Lett.* **2019**, *4*, 1633–1640. [CrossRef]
7. Jin, B.; Zhou, Y.; Zhao, Y.; Liu, M.; Song, C.; Luo, J. An Adaptive Control Algorithm for Quadruped Locomotion with Proprioceptive Linear Legs. *arXiv* **2022**, arXiv:2107.12482.
8. Chen, H.; Hong, Z.; Yang, S.; Patrick, M.W.; Zhang, W. Quadruped Capturability and Push Recovery via a Switched-Systems Characterization of Dynamic Balance. *arXiv* **2022**, arXiv:2201.11928.
9. Alberto, D.P.; Santanu, S.D.; Marco, M. Mixed-integer Quadratic Programming is in NP. *arXiv* **2014**, arXiv:1407.4798.
10. Ahn, M.S.; Chae, H.; Hong, D.W. Unknown Terrain Locomotion for Quadrupeds based on Visual Feedback and Mixed-Integer Convex Optimization. In Proceeding of 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 3791–3798. [CrossRef]
11. Kohl, N.; Stone, P. Policy gradient reinforcement learning for fast quadrupedal locomotion. In Proceeding of 2004 IEEE International Conferrence Robotic Automation (ICRA), New Orleans, LA, USA, 26 April–1 May 2004; pp. 2619–2624. [CrossRef]
12. Jain, D.; Iscen, A.; Caluwaerts, K. Hierarchical reinforcement learning for quadruped locomotion. In Proceeding of 2019 IEEE International Confonference Intelligent Robots Systems (IROS), Macau, China, 3–8 November 2019; pp. 7551–7557. [CrossRef]
13. Hiroki, Y.; Sungi, K.; Yuichiro, I.; Yusuke, I. Generalization of movements in quadruped robot locomotion by learning specialized motion data. *Robomech J.* **2020**, *7*, 29. [CrossRef]
14. Robert, J.S. Direct Kinematics: The Arm Equation. In *Fundamentals of Robotics: Analysis and Control*; Asoke, K.G., Ed.; Prentice-Hall of India Private Limited: New Delhi, India, 2004; pp. 41–47, ISBN-81-203-1047-0.
15. Pereira, M.A.; Fan, D.D.; An, G.N.; Theodorou, E.A. Mpc-inspired neural network policies for sequential decision making. *arXiv* **2018**, arXiv:1802.05803.
16. Ali, A.; Aliakbary, S. Predicting citation counts based on deep neural network learning techniques. *J. Informetr.* **2019**, *13*, 485–499.

17. Chattopadhyay, A.; Manupriya, P.; Sarkar, A.; Balasubramanian, V.N. Neural network attributions: A causal perspective, In Proceeding of the 36th International Conference Machine Learning (ICML), Long Beach, CA, USA, 10–15 June 2019; pp. 1–10.
18. Calik, N.; Belen, M.A.; Mahouti, P. Deep learning base modified MLP model for precise scattering parameter prediction of capacitive feed antenna. *Int. J. Numer. Model.* **2019**, *33*, e2682. [CrossRef]
19. Putra, M.A.P.; Hermawan, A.P.; Kim, D.S.; Lee, J.M. Energy Efficient-based Sensor Data Prediction using Deep Concatenate MLP. In Proceeding of the 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vasteras, Sweden, 7–10 September 2021; pp. 1–4. [CrossRef]
20. Chen, S.; Xie, E.; Ge, C.; Liang, D.; Chen, R.; Luo, P. Cyclemlp: A mlp-like architecture for dense prediction. *arXiv* **2022**, arXiv:2107.10224.
21. Konstantin, E.; Johannes, S. A comparison of deep networks with ReLU activation function and linear spline-type methods. *Neural Netw.* **2019**, *110*, 232–242. [CrossRef]
22. Hayou. S.; Doucet, A.; Rousseau, J. On the impact of the activation function on deep neural networks training. *arXiv* **2019**, arXiv:1902.06853.
23. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks In Proceeding of the 13th International Conference Artificial Intelligent Statistics, PMLR 9, Sardinia, Italy, 13–15 May 2010; pp. 249–256.
24. Suriya, G.; Lee, J.; Soudry, D.; Srebro, N. Implicit bias of gradient descent on linear convolutional networks. *arXiv* **2019**, arXiv:1806.00468.
25. Giovanni, C.; Perret, B. Ultrametric fitting by gradient descent. *J. Stat. Mech. Theory Exp.* **2020**, *2020*, 1–21.
26. Zhou, J.; Wang, H.; Wei, J.; Liu, L.; Huang, X.; Gao, S.; Liu, W.; Li, J.; Yu, C.; Li, Z. Adaptive moment estimation for polynomial nonlinear equalizer in PAM8-based optical interconnects. *Opt. Express* **2019**, *27*, 32210–32216. [CrossRef]
27. Hocine, M.; Balakrishnan, N.; Colton, T.; Everitt, B.; Piegorsch, W.; Ruggeri, F.; Teugels, J. The Mean Square Error Equation. In *Wiley Statsref: Statistics Reference Online*; Balakrishnan, N., Colton, T., Everitt, B., Piegorsch, W., Ruggeri, F., Teugels, J., Eds.; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2014; ISBN 9781118445112.
28. Wong, T.; Yeh, P. Reliable accuracy estimates from k-fold cross validation. *IEEE Trans. Knowl. Data Eng.* **2019**, *32*, 1586–1594. [CrossRef]
29. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. Tensorflow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16), Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
30. Gulli, A.; Pal, S. The python library Keras. In *Deep Learning with Keras*; Packt Publishing Ltd.: Birmingham, UK, 2017.