# Quadruped Dynamic Controller Midterm Update

Alex Tacescu, Shreyash Shantha Kumar, Sinan Morcel and Stephen Crawford

## I. INTRODUCTION & PROJECT REVIEW

Legged robots will perhaps be one of the most important robots in the coming future. Although the software development need is very complex, legged robots have unique terrain traversal capabilities that is hard to emulate with other robot designs. In the past few years, these robots have been making great strides thanks to research institutions like MIT and companies like Boston Dynamics.

However, the software still stands as one of the biggest hurdles for engineers to overcome. This project aims to develop an all-encompassing platform in MATLAB that will dynamically model a 4-DoF quadrupedal robot. Inspired by biology, we will be looking at using Central Pattern Generators (CPGs) and responses/reflexes. We will also look into using new ways to calculate inverse kinematics, including Forward & Backward Reaching Inverse Kinematics (FABRIK) and analytical methods. Finally, we generate a dynamic model for the legs of the robot, and give it a basic walking gait for testing purposes.

## II. LITERATURE REVIEW

### A. Dynamics

Since dynamic modeling was an issue, we dedicated some of our reading to the part of the literature that attempted or talked about the same thing. In [3], they offer something closer to a survey of different methods to model a quadruped. They offer an example of a 2-DOF-legs robot, where they describe the state vector of such systems to have the following variables: 3 Bryant Euler angles for the orientation of the whole system, 3 variables for the position of the system, 3 variables for the linear velocity, 3 for the angular one, and the vectors corresponding to the configurations of all the legs. The control variables can only directly affect the configuration of the joints, and indirectly the others. The dynamic model of such a system has the following familiar form

$$\ddot{q} = M(q)^{-1}\big(B\mathbf{u} - C(q,\dot{q}) - G(q) + J_c(q)^T f_c\big)0 = g_c(q),$$

which is that of a decoupled n-link robot (more than one chain of decoupled joint variables). M is the inertia matrix, B the matrix of friction coefficient of the joints, C is the matrix that corresponds to the Coriolis and centrifugal forces and G is the gravity vector. $J_c$ is the constraint Jacobian that factors into the equation the external ground constraint forces, so that they can be considered as part of the equation. $g_c$ allows us to compute the constraint Jacobian using the equation:

$$J_c = \frac{\partial gc}{\partial q}$$

However, the authors continue the discussion about stability guarantees and algorithms, without dwelling too much on dynamic modeling.

The authors in [2] offer a very good intuition and description about what each of the terms in the typical equation of manipulator dynamics are, which helped us formulate many of our next sub-goals for the project.

### B. FABRIK: An Iterative Solution to Inverse Kinematics

FABRIK, or Forward and Backward Reaching Inverse Kinematics, is an iterative solution to solving an inverse kinematics problem. Most analytical inverse kinematic methods are complex to compute, especially when there are more and more degrees of freedom. In fact, in some cases, it may be impossible to geometrically or algebraically find the inverse kinematic equations for a robot. These methods are often slow to execute, and are susceptible to singularities. Iterative methods approximate an inverse kinematic solution, which lead to faster compute times and less stress on the computer running the algorithm. Although iterative solutions like FABRIK will never give an exact answer, it approaches the best answer faster than other iterative algorithms, as seen in Figure 1

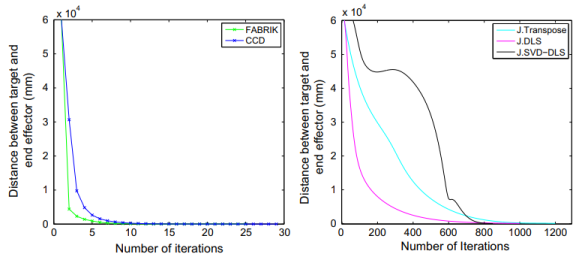


Fig. 1. FABRIK approaches an error of zero faster than other iterative methods [1]

FABRIK is relatively simple to implement. It starts from the last link, and adjusts it so the end effector is at the final goal position, and points it toward the joint it should be connected to. Then, the same thing happens to the next link: it moves so one end connects to the last link, and then points toward the joint it should be connected to. This happens all the way up the link from end to the starting joint, and then back again to the end effector to form one iteration, as shown in Figure 2. Since computation is based on basic geometry, FABRIK is simple to implement and fast to compute.

However, the best part of FABRIK is its versatility. It has proven itself to be resistant to singularities, as well as work with multiple end effectors. [1]. It also works with joint

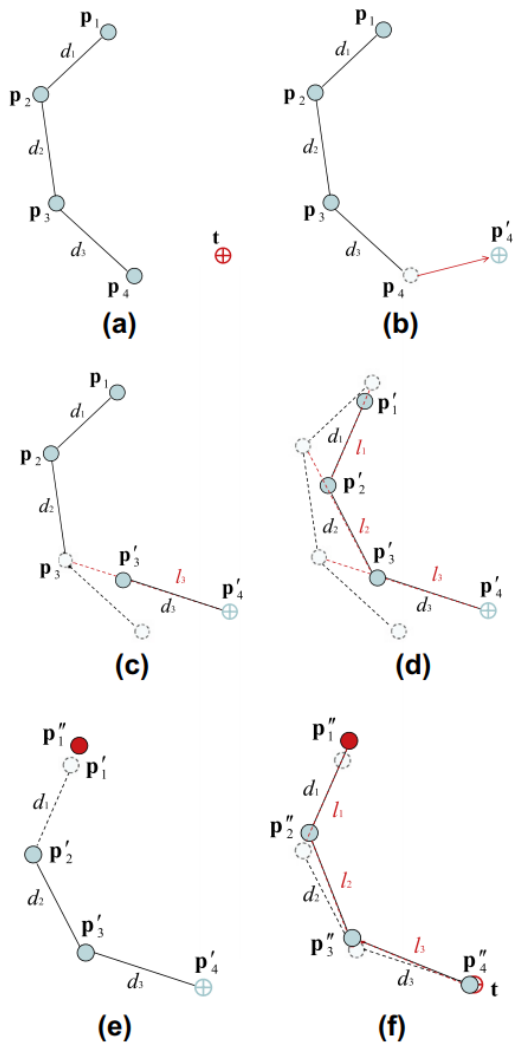


Fig. 2. Steps to Approximate Inverse Kinematics using FABRIK [1]

limits, and can even act as a closed loop controller when implemented correctly. Better yet, there hasn't been a case where FABRIK has shown no weakness, and is faster than all other known methods.

### C. Dynamic Walking

One very effective method to designing a walking gait is to look toward nature for inspiration. There are two concepts that can be used on legged robots that come from animals: central pattern generators and reflexes/responses

*Central Pattern Generators:* A Central Pattern Generator (CPG) is a control method that generates trajectories based on loosed-coupled oscillators, similar to the neural circuits found in animals. Essentially, CPGs are a biological neural network that creates a rhythmic output patter without any necessary sensory input. This allows animals to perform basic tasks, like breathe, walk, and chew, and the same basic concept can be used to generate trajectories for robots. There are many ways to create this output, but two of the main methods include the Rulkov map type neuron - which models natural CPGs - and the Kuramoto oscillators - which are more predictable due to their consistent wave. These signals can input into a walking gait to control speed and position of limbs, similar to how a musician would be signaled by different notes in an orchestra. Although CPGs are, by definition, open loop, sensory feedback can be incorporated to modify parameters. This can be used to make changes to the main gait in response to an environment change or possibly a gradual change into another gait.

*Reflexes & Responses:* The concept of reflexes and responses is similar to animals. Reflexes are immediate actions to emergency situations. For example, if an unexpected force would push a robot sideways, a reflex would kick out the legs to one side to catch its fall and ensure it doesn't fall over. On the other hand, responses are a change to a gait caused by a change in the environment. An example of a response would be changing the walking gait by some factor when walking on an incline.

*The Wide Stability Margin:* One way to measure stability of a legged robot is to use the Wide Stability Margin (WSM). It uses the basic physics concept of fulcrums, which states that balance is based on keeping the center of gravity between your points of contact. Finding the WSM is rather straight forward: project the points of contact on the ground. These 4 points, in the case of a quadruped, will draw out a 4-sided polygon. If the projected center of gravity is outside this polygon, then the robot is going to tip over. The Wide Stability Margin is the distance to the closest side of the polygon, and measures how close the robot is to tipping over, as seen in Figure 3. [4]

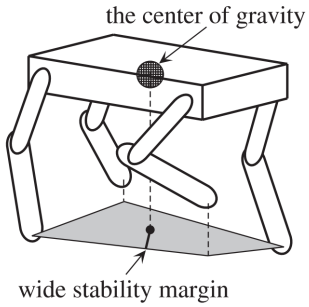


Fig. 3. Wide Stability Margin on a quadruped [4]

## III. PROJECT IMPLEMENTATION

### A. CAD Model Design

When starting the project, we had a model of a quadruped from another project. This quadruped had 4 DoF (degrees of freedom) per leg like we wanted, but it had one major problem: the model complexity made modeling it very difficult. Therefore, our team decided to design a simple model in Dassault Solidworks, then import it into MATLAB Simulink. The latest model can be seen in Figure 1. We then ported the model into MATLAB using exporting software offered by Dassault Solidworks and we used the Simscape Multibody Simulink library in MATLAB to base the simulation of the

robot. Figure 4 shows the cad model. The legs are equivalent, but are given this specific pose to project our vision of how we see the cat stand in the future.

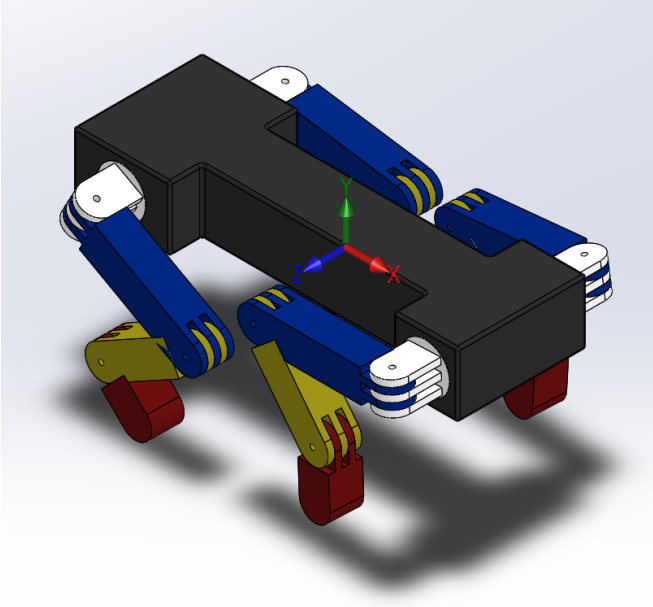

Fig. 4. The CAD model of the simplified robot.

### B. Collision in Simulink 3D Model

We used the Contact Forces Library to implement collision between parts of the robot and a plane we had placed underneath the robot. we used primitives called Sphere-to-plane collision blocks, which allow us to model a sphere to plane collision in SimuLink. However, without any control inputs on the legs we can't keep the legs straight to demonstrate a standing behavior of the quadruped. We did achieve a dangling pose and a crashing one for the robot, though.

Aiming to fully constraining the model (adding self-collision and body-plane collision) would send us off on a path that may derail us from reaching our reach goal. Figure 5 shows an older model colliding with the floor.

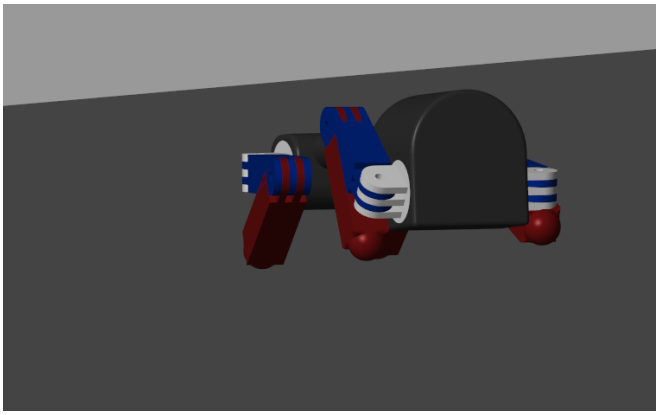

Fig. 5. The cat crashing into the floor after falling from a height.

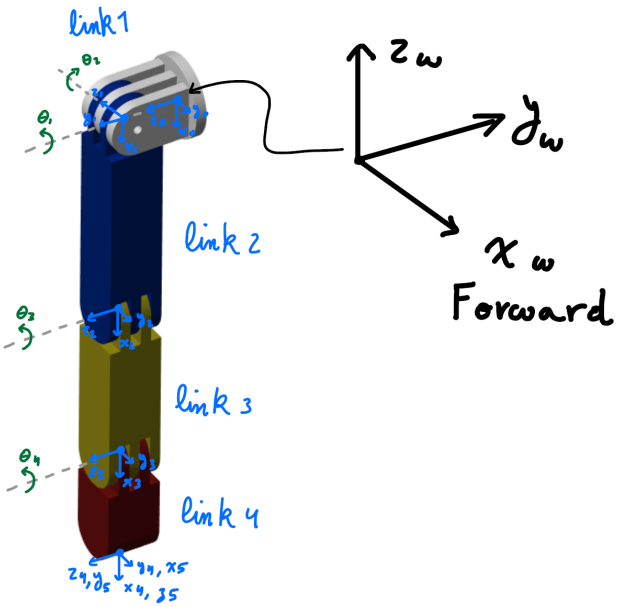


Fig. 6. The chosen DH frames.

### C. DH-Parameters and Validation

The most important set of calculations is the DH-parameters. This computation is the precursor of all the work that follows in terms of forward kinematics and dynamic modeling. Since, in our model, all legs are designed to be similar, we only need to specify the DH-parameters for one leg. Figure 6 shows the chosen frames. The rationale behind choosing the robot's frame $F_w$ as having the z-axis pointing upward as opposed to having it point in the direction of the axis of the first joint is to make subsequent computations of the dynamic model intuitive, especially when gravity is to be considered. This design choice gives us some sort of intuitive assumption that gravity points down by default (which will also be made to change with the orientation of the robot). Table I shows the DH-table that situates the frames with respect to the robot's world frame as shown in figure 6. The first two rows correspond to the transformation from the world frame to the first joint's frame, the next four are those that propagate the frame orientations based on the values of the joint angles, and the last one orients the approach vector along the final link's axis, which is customary to do, in the literature, and useful, in practice.

Figure 7 shows how we validated those DH-parameters.

### D. Forward Kinematics

Given those DH-parameters shown in III-C, it is almost trivial to compute the forward kinematics. A MATLAB function is setup for reuse across this project. The MATLAB function returns all the intermediate frames in a 3D matrix, for convenience and efficiency. This function will be used once in the Jacobian computation and the 3D matrix is used more than once, which removes the overhead due to calling this function and computing the forward kinematics.

TABLE I
DH-PARAMETERS

| link | $\theta_i$ | $d_i$ | $a_i$ | $\alpha_i$ |
|---|---|---|---|---|
| Rot1 | 0 | 0 | 0 | $\frac{\pi}{2}$ |
| Rot2 | $-\frac{\pi}{2}$ | 0 | 0 | 0 |
| 1 | $\theta_1$ | 30 | 0 | $\frac{\pi}{2}$ |
| 2 | $\theta_2$ | 0 | 100 | $-\frac{\pi}{2}$ |
| 3 | $\theta_3$ | 0 | 75 | 0 |
| 4 | $\theta_4$ | 0 | 50 | 0 |
| 5 | $\frac{\pi}{2}$ | 0 | 0 | $\frac{\pi}{2}$ |

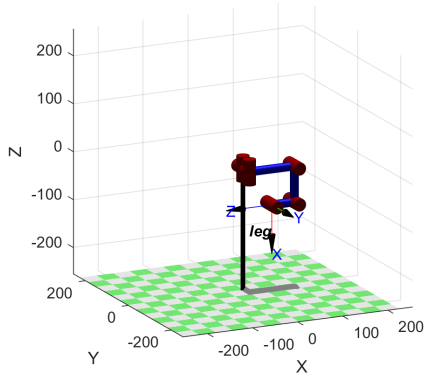


Fig. 7. The plot that validates our DH-parameters. The configuration $\theta$ was set to $[\frac{\pi}{2}\ 0\ -\frac{\pi}{2}\ -\frac{\pi}{2}]$ to give the expected configuration in a plot. The Robotics Toolbox for MATLAB was used to generate this plot.

### E. Inverse Kinematics

At first we had decided to calculate the inverse kinematics using a geometric approach, and we actually did manage to do so. However, after doing some research, we decided to use a different approach. By calculating the inverse velocity kinematics, we can represent a trajectory in task-space, and interpolate the intermediate joint angles from there, as explained in [5], and in Section IV-A Trajectories and Controller.

### F. The Jacobian

The Jacobian of an individual leg is the single most important aspect in the process of dynamically modeling the quadruped in question. In fact, we need to be able to compute more than one Jacobian for each configuration, one for each link in the manipulator/leg. For this purpose, we referred to the book "Robot Modeling and Control" [5], which gives a cookbook method of computing the Jacobian for a point along one of the links of an n-link manipulator. Based on that, we created a function that takes the index of the joint and returns a corresponding 6x4 Jacobian matrix. For the purpose of dynamically modeling the robot, we do not need the matrix to be square. However, for the purpose of task-space trajectory tracking we need the pseudo-inverse.

### G. Dynamic Modeling

From the process of dynamically modeling the robot, we computed the Lagrangian, as well as the inertia matrix and the potential energy matrix. Thus far, we have a MATLAB script that outputs the inertia matrix and the potential energy vector of a specific configuration, what remains to be done is making the set of equations symbolic and doing the differentiations that the Euler Lagrange method requires. However, we are expecting to get a slower version of the current code, which currently runs at 9 milliseconds (very good for control), because the symbolic library in MATLAB is known to be slow when used to compute the actual values through substitution. However, we still have to validate whether it does that or not. After we do the differentiation, we get the equations of motion that should allow us to figure out the torques required to fight off gravity, for example.

*1) The System's Lagrangian:* We referred to [5] for the process of computing the total energy of the system, kinetic and potential. This allowed us to use the Euler-Lagrangian equations of motion that are essential to perform gravity compensation and other dynamic terms cancellation that are needed when we use computed torque control, as a way to control the robot and have it follow trajectories. We were able to compute or acquire the matrices that define the following, so called inertia matrix, as defined in [5].

$$M(q) = \sum_{i=1}^{n} \{m_i J_{vi}(q)^T J_{vi}(q) + J_{\omega i}(q)^T R_i(q) I_i R_i(q) J_{\omega i}(q)\}$$

The total kinetic energy of the system can be computed using the following equation:

$$k = \frac{1}{2} \dot{q}^T M(q) \dot{q}$$

where i is in the index of the center of mass of the $i^{th}$ link, $J_{vi}$ the linear velocity part of the Jacobian, and $J_{\omega i}$ the rotational part of it for the center of mass of the $i^{th}$ link. The total potential energy in the system is defined as follows.

$$P = \sum_{i=1}^{n} m_i g^T r_{ci}$$

where $r_{ci}$ is the location of the center of mass of the $i^{th}$ link and $m_{ci}$ is the mass thereof. g in here is the orientation of the gravity vector. This is what we are planning to change to accommodate the model to different base orientations when the robot is walking.

## IV. ENVISIONED APPROACH

The process that we envision is the following. After getting the dynamic model of a leg of the robot

$$\tau = M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q),$$

we can extend that model in two ways. First, we either maintain world to leg frame transformation that we update using our measurements (prone to error in the real world), or recompute from scratch the equations and change the gravity vector to reflect the new orientation of the body which we can

readily measure using an IMU assumed to be on-board (like proprioception that humans have). We may also parameterize the gravity vector to include those measurements. Independent from this model we may maintain position, and velocity (linear and rotational) variables for the whole system, as prescribed in [3]. However, our aim is to make the robot walk a few steps as opposed to being provably stable, so whatever gets us to make the robot walk as a reach goal, we will undergo.

### A. Trajectories and Controller

This model will then be used in a computed torque controller to make the leg faithfully follow the trajectories we specify. While MATLAB allows us to do position-based control, we will refrain from using that feature, as this is not the case with real-robots.

As for trajectory generation, we will generate these in the task-space as opposed to the joint-space. For that, we do not need any inverse kinematics, only inverse velocity kinematics. From the reference book [5], we can represent the trajectory-to-follow in the task-space and get its equivalent trajectory in the joint-space. Given the following equations, we can do the translation.

$$\dot{X} = J_a(q)\dot{q}$$

$$\ddot{X} = J_a(q)\ddot{q} + \dot{J}_a(q)\dot{q}$$

$$\ddot{q} = a_q$$

By defining $a_q$ as $J^{-1}\{a_X - \dot{J}\dot{q}\}$, we can can then directly command the acceleration, as can be seen in the resulting equation $\ddot{X} = a_X$. Then, we can track the trajectory using a PD controller with feed-forward like so:

$$a_X = \ddot{X}^d - k_p(X - X^d) - k_d(\dot{X} - \dot{X}^d)$$

Thus, by computing the desired acceleration at the joint level using $a_q = J^{-1}\{a_X - \dot{J}\dot{q}\}$, we can track a trajectory in task-space. In here, the task space is considered with respect to the frame of the leg, as shown in figure 6.

### B. Central Pattern Generator

As per the work in [4], we will use the concept of a central pattern generator. Once we have the trajectories, which can be systematically generated or specified manually (engineered), we can create a simple central pattern generator, a higher-level controller that regulates the phase between the trajectories that the legs follows (the same trajectory for all legs), possibly with some minor changes to angle signs.

A simple approach to the central pattern generator would simply regulate the difference between $t_1$, $t_2$, $t_3$, and $t_4$, such that $0 < t_1 < t_2 < t_3 < t_4 < T$ where T is the final time of the trajectory, and each of those time variables corresponds to the progress of the respective leg along its trajectory.

### C. Assumptions Taken

One important consequence of our modeling approach is that we are treating the weight of the body as a disturbance, by virtue of the choice of modeling each leg independently. This means that we are also assuming that we can measure the forces at the leg that are generated by the weight of the body very accurately and separate it out from the weight of the leg (which we can do since we are modeling it's weight). We are also assuming that we can measure the legs configurations in terms of position and velocity.

## V. PROJECT MANAGEMENT

### A. Updated Schedule

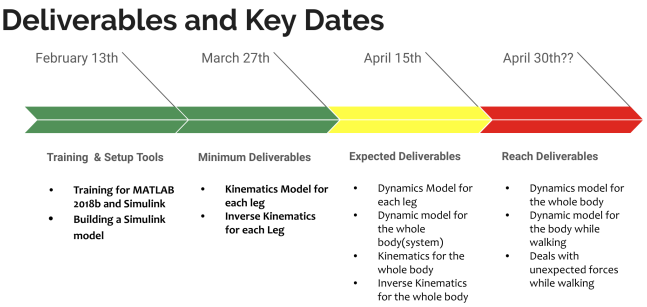


Fig. 8. Our updated schedule. Bolded tasks have been completed

### B. Dependencies

We have two major dependencies coming into the second half of our project: our dynamic model for our leg and our dynamic model for our body. As seen in the schedule, we want to not only model our quadruped, but also give it basic control over its body. This will require a bullet proof model for each leg, which will be used for a dynamic body model. Although this is our reach goal, we hope to finish it within the next few weeks, and to have a demo ready.

### C. Challenges

Like any project, we ran into a few major issues that slowed our progress more than expected. The first of these problems were our 4 DoF leg. Unlike most quadrupeds, our leg design has a single roll joint, with a planar 3DoF manipulator. This configuration allows us to have nearly unlimited attack angles to the ground, but also introduces a lot of complexity when calculating inverse kinematics and modeling the dynamics of the leg.

We also ran into problems when dynamically modeling the legs themselves. No group member had experience in dynamic modeling, so we studied literature and asked professors to learn how dynamic modeling worked in a case like ours. We also simplified the model (see Section III-A), to make the modeling process easier.

Our final major problem we faced was our inexperience with MATLAB SimMechanics modeling. This inexperience led to many refactors of our code base, and although another refactor may happen in the near future, we think that our software is at a point where we can continue development with less distractions.

## VI. CONCLUSION

Although we are on track with our original estimation, our team would like to be further ahead than we are. However, the steps we have taken to learn key concepts and develop good code has put us in a position that will hopefully expedite our future endeavours in our project. We are excited to see if our modeling approaches will allow us to achieve a torque controlled dynamic walking robot. If successful, this may provide a decent method to model and control commercial grade walking quadrupeds, and allow for more advanced stability studies and improvements to the controller.

## ACKNOWLEDGMENT

## REFERENCES


[1] Andreas Aristidou and Joan Lasenby. Fabrik: A fast, iterative solver for the inverse kinematics problem. *Graphical Models*, 73(5):243 – 260, 2011.

[2] Farid Ferguene and Redouane Toumi. Dynamic external force feedback loop control of a robot manipulator using a neural compensatorapplication to the trajectory following in an unknown environment. *International Journal of Applied Mathematics and Computer Science*, 19(1):113–126, 2009.

[3] Michael Hardt and Oskar von Stryk. Dynamic modeling in the simulation, optimization, and control of bipedal and quadrupedal robots. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik: Applied Mathematics and Mechanics*, 83(10):648–662, 2003.

[4] Hiroshi Kimura, Yasuhiro Fukuoka, and Avis H Cohen. Adaptive dynamic walking of a quadruped robot on natural ground based on biological concepts. *The International Journal of Robotics Research*, 26(5):475–490, 2007.

[5] Mark W Spong, Seth Hutchinson, Mathukumalli Vidyasagar, et al. *Robot modeling and control*. Wiley, 2006.