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

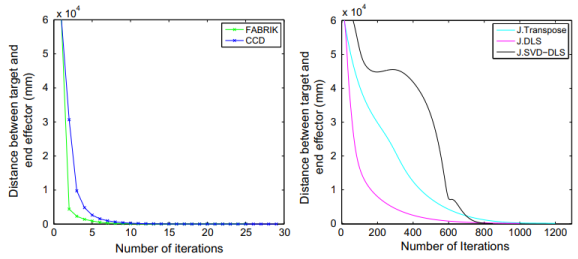

Fig. 1. FABRIK approaches an error of zero faster than other iterative methods [1]

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

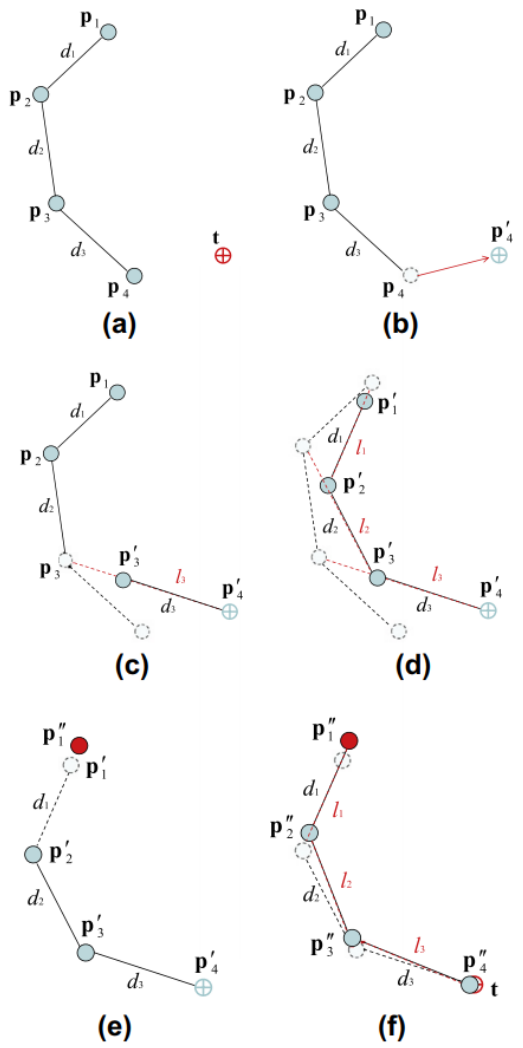

Fig. 2. Steps to Approximate Inverse Kinematics using FABRIK [1]

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]

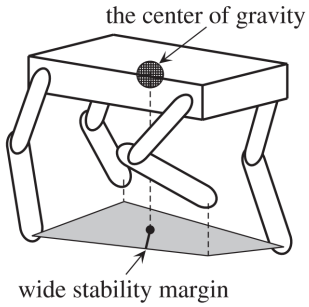

Fig. 3. Wide Stability Margin on a quadruped [4]

III. PROJECT IMPLEMENTATION

A. CAD Model Design

To simplify development of the project, we developed a very simple model of a quadruped in Dassault SolidWorks, as seen in figure 4. This model has the 4-DoF legs we want, and allows us to simplify the computation needed to more efficiently develop a parametric model of a quadruped.

4-DoF Planar Leg Design: Most companies and research institutions will look toward a 3-DoF leg when designing a quadrupedal robot, and with good reason. Adding unnecessary links and joints make the robot more prone to failure, while complicating dynamic modeling and increasing

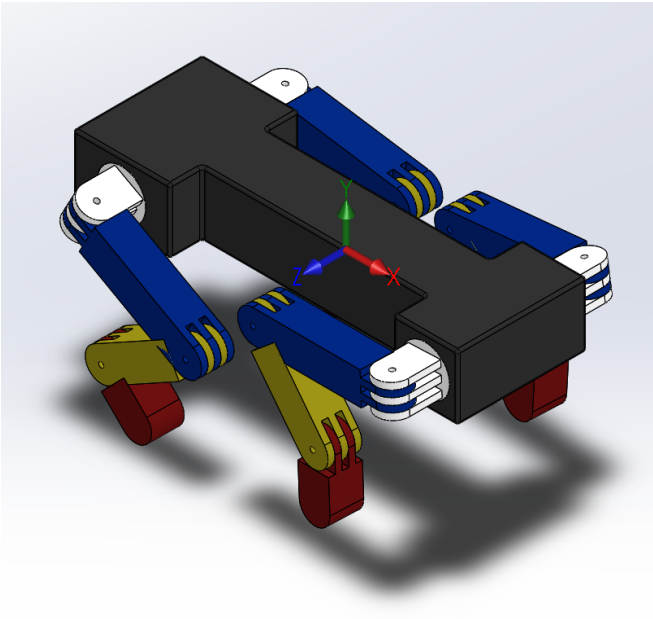

Fig. 4. The CAD model of the simplified robot.

computational power required. However, we propose that a 4-DoF planar leg, as depicted in Figure 5, is a good option for off-road quadrupedal robots. The redundant last link gives the engineer the capability to set the contact angle to the ground, potentially increasing performance in rough terrain. This design is also capable of emulating a generic 3-DoF quadruped, due to its similar design. Therefore we believed that this leg design is best for research purposes.

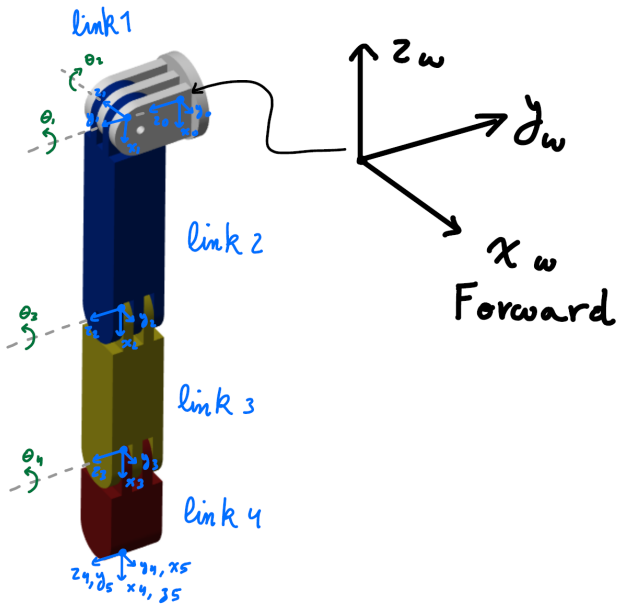

Fig. 5. A depiction of the chosen DH frames

Importing the Model into Simscape: We then ported the model into MATLAB using exporting software offered by SolidWorks and we used the Simscape Multibody Simulink library in MATLAB to base the simulation of the robot. For the purpose of simplicity, all legs are equivalent, but given different poses to make it look more like a cat.

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 6 shows an older model colliding with the floor.

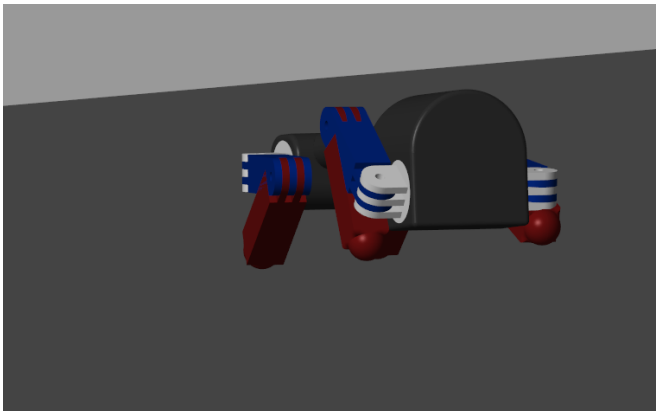

Fig. 6. The cat crashing into the floor after falling from a height, proving collision modeling is working between paws and ground

C. DH-Parameters and Validation

We decided to follow the Denavit-Hartenberg convention when modeling DynaKat, since it is the chosen standard in the robotics industry. Since all of our legs are the same, we specify one set of DH parameters, seen in Table I. Figure 5 also shows a visual representation of the DH parameters we chose. The rationale behind choosing the robot's frame F_w having its z axis pointing upward is to make the dynamic model more intuitive, both during development and for future application, given the assumption that gravity is considered to be pointing downward.

As stated previously, Table I displays the DH parameters based on the robot's world frame. The first two rows correspond to the transformation from the world frame to the first joint's frame, while the next four are those that propagate the frame orientations based on the values of the joint angles. The last row orients the approach vector along the final link's axis.

Figure 7 shows the validation of our DH parameters. We used the Robotics Toolbox to ensure our DH parameters were accurate to how we wanted the legs to be described

link	θ_i	d_i	a_i	α_i
Rot1	0	0	0	$\frac{\pi}{2}$
Rot2	$-\frac{\pi}{2}$	0	0	0
1	θ_1	30	0	$\frac{\pi}{2}$
2	θ_2	0	100	$-\frac{\pi}{2}$
3	θ_3	0	75	0
4	θ_4	0	50	0
5	$\frac{\pi}{2}$	0	0	$\frac{\pi}{2}$

TABLE I
DH-PARAMETERS

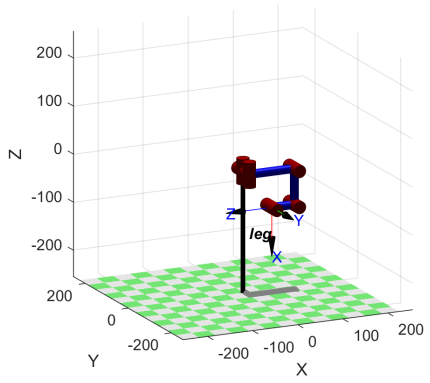

Fig. 7. The plot that validates our DH-parameters. The configuration θ was set to [$\frac{\pi}{2}$ 0 -$\frac{\pi}{2}$ -$\frac{\pi}{2}$] to give the expected configuration in a plot. The Robotics Toolbox for MATLAB was used to generate this plot.

D. Forward Kinematics

Given those DH-parameters shown in III-C, it is almost trivial to compute the forward kinematics. We developed a MATLAB function to return the transformation matrix from each leg's base frame to its end effector. This function is designed to run once, and outputs a symbolic matrix with all variables with DH parameters applied, and symbols replacing the q variables. Therefore, we can reduce computation speed when calculating forward kinematics while running. The matrix also aids in Jacobian computation, reducing overhead.

E. Inverse Kinematics

There are several methods to create an inverse kinematic model of a manipulator, including analytical and heuristic methods. At first, we tried calculating the inverse kinematics using a geometric approach. However, throughout our research, we discovered many other methods to calculating the inverse kinematics of a manipulator. As talked about in Section III-G, we calculated the inverse velocity kinematics, and represented a trajectory in task-space. Finally, we interpolated the intermediate joint angles to get our final positions. This method was used in the dynamic model demonstration.

We also looked at using a heuristic method of computing our inverse kinematics. We chose FABRIK, since it is fast, resistant to singularities, and capable of modeling multiple end effectors. More about this algorithm is talked about in II-B. In the end, we created a demonstration showing the FABRIK algorithm outputting joint angles for a leg going in a pre-defined trajectory.

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" [6], 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. The Dynamics

In order to control the robot using torque, we need to provide a signal that takes into account the dynamics of the system, hence the need for a dynamical model. To acquire one, we followed the cookbook method shown in the "Robot Modeling and Control" [6] and [5]. We used the first resource to get the general equations and the second to resolve some ambiguity about whether the **Inertia Tensors** used in the equations were defined with respect to the center of the link or its hinge point. Both resources are complementary and discuss examples that highlight different nuances. From [6], we got the general equation of the Inertia Matrix:

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

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. In the rotational part of the equation, I_i is the constant inertia tensor defined with respect to the i^{th} link, and R_i the rotation matrix that aligns the inertia tensor to the link with respect to the base-frame. $R_i(q) I_i R_i(q)^T$ is thus the inertia tensor of the i^{th} link with respect to the base frame, which is a function of q. We acquired the inertia tensors from the SolidWorks model of the leg that we had designed and the rotation was computed by combining the rotation that transforms coordinates from the base to the end of the i^{th} link and another which we designed by hand so that the inertia tensor is aligned with the one represented in SolidWorks. The process was iterative, but we could not validate whether we had the correct configuration until we were able to test it on the model itself. The other terms are easy to acquire from the properties of the robot, as well as the Jacobian computations. Then, the elements of the Coriolis and Centrifugal forces matrix were computed by using M's

elements and the following equation:

$$c_{kj} = \sum_{i=1}^{n} \{ \frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{ki}}{\partial q_j} - \frac{\partial d_{ij}}{\partial q_k} \} \dot{q}_i$$

where c_{kj} is the intersection of M's k^{th} row and M's j^{th} column. Then, to get the symbolic G vector, we get the system's potential energy as follows:

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

where $g^T r_{c_i}$ is the project of the distance to the base-frame of the i^{th} link. We leave the variable g parametric, that is, we let be a symbolic vector and based on the situation in which the arm/leg find itself in, we would then set this g accordingly to counter the effect of gravity on the arm. Differentiating this equation with respect to the vector of joint values, we obtain the G vector, which is the gravity vector of a manipulator. The following shows the equation we referred to:

$$g_k = \frac{\partial P}{\partial q_k}$$

The resulting matrices can be used to express the dynamic model of the robot which goes like this:

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

Implementation wise, the M, C and G matrices were computed in a file called generatingDynamics.m, which initializes the parameters like weights and inertia tensors and then calls another function which returns the Jacobians for the chosen center of mass, in addition to the rotation that aligns the corresponding inertia matrix and r_{c_i} vectors (the vector that indicates the location of the center of mass of the i^{th} link with respect to the base). Once the matrices are computed, they are turned from symbolic matrices, which are very slow to evaluate, into matlab functions using the matlabFunction routine which is part of the symbolic toolbox. The result is a set of functions, computeM, computeC and computeG which combined run at an average of 25 milliseconds, which is good enough for simulation. Figure 5 also shows the frame in which the dynamic model is shown. Note that in this frame, the gravity can sometimes be in another orientation than the negative z_world vector.

IV. PROJECT MANAGEMENT

A. Updated Schedule

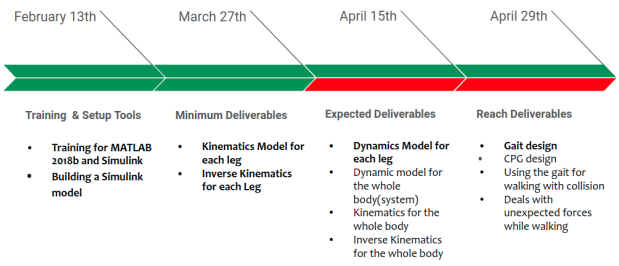

Fig. 8. Our Final Deliverables, bolded tasks have been completed

B. 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.

Another challenge we faced was getting the inertia tensors aligned. This is very important to ensuring inertia is accounted for when calculating the dynamics for the robot. Although it was tedious, we ended up figuring it out after a lot of research and time.

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.

V. 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 endeavors 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] Richard M Murray. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

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