CIKAQ: Using a Custom Inverse Kinematics Model to Animate a Quadruped Robot Dog Gait

MIGUEL NEPOMUCENO, University of California Berkeley, USA
DAVID KIM, University of California Berkeley, USA
TRACY XIA, University of California Berkeley, USA
ADHAM ELARABAWY, University of California Berkeley, USA

An inverse kinematics model of a 2-degree-of-freedom robot leg is presented, including a model for a single foot path of the robot leg as a set of continuous bezier curves. The foot path is interpolated to obtain intermediate foot positions, which will then be passed through the inverse kinematics model in order to compute the angles of each joint in the robot leg. Output kinematics are animated in a custom visualizer to show how the robot leg moves through the foot path.

CCS Concepts: • **Computer systems organization → Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

Additional Key Words and Phrases: robotics, inverse kinematics, quadruped, gait

ACM Reference Format:
Miguel Nepomuceno, David Kim, Tracy Xia, and Adham Elarabawy. 2023. CIKAQ: Using a Custom Inverse Kinematics Model to Animate a Quadruped Robot Dog Gait. *ACM Trans. Graph.* 1, 1 (May 2023), 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Inverse kinematics is a critical concept in robotics, particularly in designing and controlling legged robots like quadruped robots. It is defined as the process of determining how the required joint angles of a robot can achieve a specific end-effector position in the world. It can be essential in modeling how a robot might navigate and interact with its environment through motions like walking, running, and jumping. The application of both inverse kinematics models and dynamics-based forward kinematics using feedback controllers are both active areas of research in the context of legged robotics, and many different techniques and algorithms have been developed to try achieve the highest-performance control and motion planning for robotic models.

Although not as theoretically complex as state-space dynamics, inverse kinematics shares many challenges with that field of research. One of the primary challenges is the complexity of the robot's kinematic structure, which may involve many degrees of freedom and non-linear joint constraints. Additionally, real-world conditions such as sensor noise, mechanical wear, and external disturbances can make it difficult to accurately predict and control the robot's motion. Inverse kinematics also have their own unique challenges, such as how best to point towards control points outside of their reachable domain or how to select from multiple different possible solutions to reach the desired end position. To address these challenges, researchers have developed a wide range of inverse kinematics methods, including analytical solutions, numerical optimization, and machine learning approaches, sometimes drawing from captured motion analysis.

In our paper we propose an animation visualizer for a walking quadruped model. The main deliverables are as follows:

- A mathematical inverse-kinematics model of a two-link, two-dimensional robot leg. For above model, functionality should return the relative angles between limbs required in order to achieve the desired (x,y) position.
- A build environment where we can input joint angles, and see the output leg position.
- Interactive GUI to modulate limb lengths, foot path speed, key foot path parameters, etc.
- The ability to draw out bezier curves to control the walking pathway for the robot dog

2 INVERSE KINEMATICS MODEL

2.1 Common Inverse Kinematics Models

Two commonly used inverse kinematics techniques used to control robots are Forward and Backwards Reaching Inverse Kinematics (FABRIK) and Cyclic Coordinate Descent (CCD).

FABRIK is a relatively new method that is becoming increasing popular due to its computational efficiency and ease of implementation, not requiring the use of any rotation matrices or dynamics. The implementation is based on a series of iterative steps that progressively refines the robot joint angles to hit the desired end position by adjusting angles forwards and backwards along the robot from its end-effector to its base, hence its name. FABRIK is quite robust and is well-suited for controlling the motion of robots with complex kinematic structures or multiple chains of limbs.

CCD is an older and more widely used method that works by iteratively adjusting joint angles for the robot's limbs with a rotation matrix in order to minimize the difference between the current end-effector position and target position. One drawback is that it turns outer joints firstly, which may result in less realistic posing. Additionally, joints close to the end of the limb rotate more than joints close to the root, so the chain of limbs may appear to roll in

Authors' addresses: Miguel Nepomuceno, University of California Berkeley, Berkeley, CA, USA, 94704; David Kim, University of California Berkeley, Berkeley, CA, USA, 94704; Tracy Xia, University of California Berkeley, Berkeley, CA, USA, 94704; Adham Elarabawy, University of California Berkeley, Berkeley, CA, USA, 94704.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Association for Computing Machinery.
0730-0301/2023/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

on itself. However, it tends to be be robust to uncertainty and noise, making it generally applicable in a wide range of robotic systems.

2.2 Trigonometric Model

For this paper, we implement a much simpler and less robust trigonometric method for calculating joint angles. Knowing the limb lengths, the law of cosines can be used to custom inverse kinematics that makes use of trigonometry to solve the angles [Corke 2017]. If the target position of the leg is reachable by the limbs in the kinematics model (i.e. distance to the desired target foot point is less than or equal to the sum of the length of the leg segments) then we can take advantage of the law of cosines to calculate the relative angles required to make the leg locate directly to the target point. Two solutions should arise from this method, one where the knee bends forwards, and one where the knee bends back. Since we are working with a quadruped model, we can safely assume the knee will bend backwards and come to a singular solution.

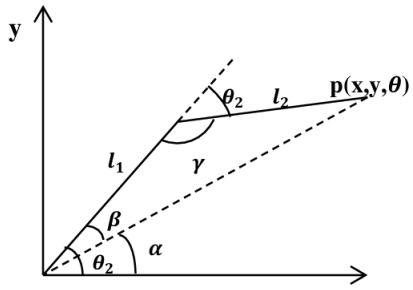

Fig. 1. Angle Relationships with desired (X,Y) Position

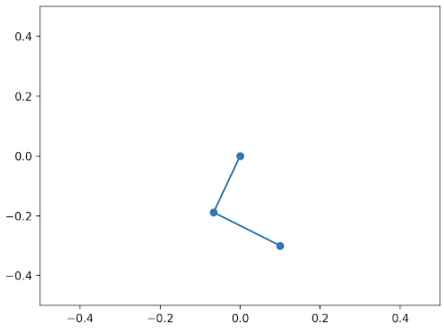

Fig. 2. Visualization of controlled limbs towards desired position using trigonometric solver. Hip placed at (X,Y) = (0,0)

A clear shortcoming of this method is that it cannot handle desired target positions that lay outside of the circular domain of radius equal to the sum of the limb lengths $l1$ and $l2$, and the trigonometry will error out.

2.3 Gait Selection

Many different methods exist for gathering realistic gaits of legged robots. Quadrupeds also have a wide variety of modes of locomotion, from simple walking/crawling to galloping, pronking, or bounding. Recent studies have made use of video capture from wildlife to gather motion data on leg joints, while other studies have utilized reinforcement learning models in order to learn a unique locomotion gait from scratch.

In our implementation, the gait is coded as a parametric function with the time t as an input. This method is used to calculate the pathway of the foot positions as the robot dog moves, and each foot will follow this path, while being offset by one another. Opposite legs will move at the same time, following the idea of a trotting gait. By interpolating over a set amount of time steps we achieve a set of discrete positions for the foot during the duration of the step. At each time step, we apply our trigonometric model to gather the angles required to move the foot to the desired target position. For simplicity, the gait is defined here using a sinusoidal curve for the swing phase and a linear path for the stance phase. In reality, this parametric function might have different, smoother shapes to reduce sudden jerks and would target positions slightly underneath the ground plane (x = 0) to account for compliance in the actuated joints. The gait is customizable and can accept different parameters for desired COM height of the robot, as well as desired step height above the ground, desired length of stance phase in order to control ground velocity of the foot, and speed that the foot will take to traverse either swing or stance phases.

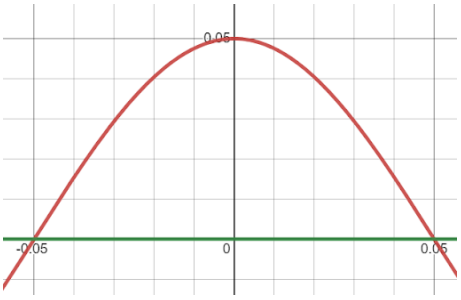

Fig. 3. swing and stance

3 VISUALIZATION ENVIRONMENT

In order to visualize the inverse kinematics model, we built a visualizer with PyBullet [PyBullet 2023] [Erwin Coumans 2022], a real-time physics simulator. Pybullet is a fast and easily implementable Python module with specific tools for robotic simulation and machine learning. Multi-link articulated bodies are loaded in using the URDF format, which defines constraints and self-collision properties between each link. The environment also comes with many helpful robotic examples to help guide users in how to control their robots. It features a CPU renderer and OpenGL 3.x rendering and visualization, and is compatible with TensorFlow and OpenAI Gym, which is useful for machine learning approaches to robotic locomotion.

ALGORITHM 1: Trigonometric IK Calculation

Input: Desired (X,Y) position of the foot relative to the shoulder joint, as well as lengths of each leg segment l_1 and l_2

Output: Tuple $(theta_1, theta_2)$ containing the relative angles of the upper and lower leg joints in radians.

$l3 = 0$;;

$alpha = math.atan2(y, x)$;

$beta = math.acos(\frac{l_1^2+l_3^2-l_2^2}{2*l_1*l_3})$;

$gamma = math.acos(\frac{l_1^2+l_2^2-l_3^2}{2*l_1*l_2})$;

$theta_1 = alpha = beta$

$theta_1 = 2 * (math.pi - theta_1) + theta_1$

$theta_2 = math.pi - gamma$

$theta_2 = 2 * (math.pi - theta_2) + theta_2$

return $(theta_1, theta_2)$

3.1 Environment

Our instance of PyBullet featured a ground plane for the robot to walk on, three cubes to act as control points for our bezier curve path, and a robot model. For our model, we used the URDF for the Unitree A1 robot dog [Robotics 2022]. This URDF came with 21 total joints, 12 of which were actuatable. There are 3 for each leg that control the hip rotation, upper leg, and the lower leg joint. For simplicity in this project, we ignore the hip joint and only consider movement surrounding the upper leg and lower leg joints.

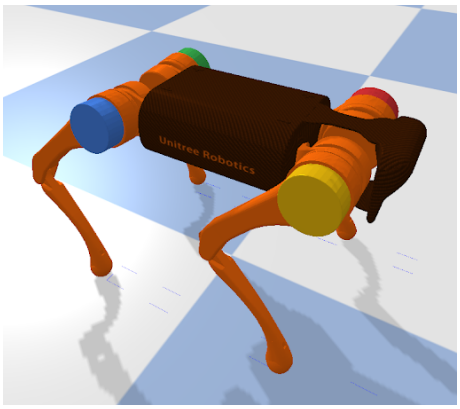

Fig. 4. Unitree A1 robot dog model

In addition to the preset key bindings, we added some of our own to customize camera views and simulation properties:

- Select a control point on the bezier curve for adjustment
- Move the control point freely around the 2D plane
- Switch between top-down view for user-defined bezier curve control and a close-up view to see details on robot leg behavior.
- Start the simulation with the space bar command.
- Reset the position of the robot dog.
- Trace out the defined bezier curve path and place down stepping stones

One visualization that our environment features is an option to see the intended foot landing positions at the beginning of each gait period. These stepping stones show where each offset leg will land as the dog traverses the bezier curve, and places a small blue square at that location. The stepping stones are mostly aesthetic, however this set of (x,y) positions can also be used as an input to the built-in PyBullet inverse kinematics model, targeting the robot to hit each square and then starting a new gait cycle that lasts until the dog approaches the next upcoming stepping stone.

When the simulation first begins or when the position of the robot dog is reset, a Bezier curve based on the control points will be drawn on the surface of the ground in the simulation environment. This helps to visualize the path of the robot dog.

3.2 Bezier Curve Path Generation

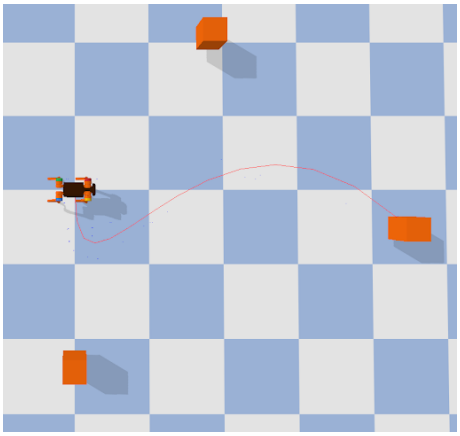

Fig. 5. A bezier path, with the robot as the starting point, and three additional control points

In the visualizer, we are initializing three additional target points, which combined with the origin will form a cubic Bezier curve path for the dog to follow. Thus, given four control points, the position of a point along the path at timestep t would be given by the formula:

$$P = (1-t)^3P_1 + 3(1-t)^2P_2 + 3(1-t)t^2P_3 + t^3P_4$$

The returned value is the expected location of the COM at timestep t. Using this point, we can apply some offsets defined by the relative leg positions of the dog from the COM. Applying our gait centered at these offset points allows us to calculate each foots expected location for that time step, which should hit its respective stepping stone if it is starting its stance phase.

3.3 Final Walking Integration

After running the simulation, we see that the dog visually has a realistic walking gait, and can drag itself in a straight line across the simulation plane. In order to get it to follow the bezier curve, we save the dog's last position in x and y every 500 steps of the simulation, and take the inverse tangent of the difference between the current position and last position in order to define the heading of the dog as a rotation (yaw) about the z-axis. Our simulation uses PyBullet commands to force the model of the dog to face in its heading direction every time its updated, and so it follows the bezier curve smoothly. With this, we have successfully integrated

the inverse kinematics model with the PyBullet visualizer and the A1 quadruped model. Checking the error between the desired target position of each foot a

4 CONCLUSIONS

In this project, we developed a simulation for moving a robotic dog using inverse kinematics. There are several limitations of our model that we can improve upon in the future. The first is the ability to handle infeasible points in the inverse kinematics model, as currently our model will error if it is given points outside of its range. This is a normal feature of the common inverse kinematics models mentioned previous (FABRIK and CCD) and would be a reasonable next step for our algorithm. Next is to expand the model to include hip and body rotation. Since we made simplifications to 2D planar movement for each leg, we were not able to include the ability to make turns dynamically, which would require the use of the hip joint. Implementing this would allow our dog to make more realistic movements and control more accurately around turns. Another improvement is to improve the path-building process and GUI, possibly utilizing Quintic Hermite interpolation such that the intended control path will be defined as the curve going through the environment's control points instead of the path being abstracted away as a function of the location of the control points. We can also increase the number of control points to create more complex paths. Finally, we can improve upon our sinusoidal gait model and utilize other models that prove more realistic gait construction. Numerous other pieces of research have been done to characterize realistic leg gaits and we have access to a wide variety of locomotion modes to try in order to make the movement of our quadruped more realistic. Despite the numerous improvements needed, our work done in this paper gives us a good foundation to build upon and a great launchpad to start building these features on.

REFERENCES

Peter Corke. 2017. Inverse Kinematics for a 2-Joint Robot Arm Using Trigonometry. https://robotacademy.net.au/lesson/inverse-kinematics-for-a-2-joint-robot-arm-using-geometry/

Yunfei Bai Erwin Coumans. 2016-2022. PyBullet Quickstart Guide. https://docs.google.com/document/d/10sXEhzFRSnvFcl3XxNGhnD4N2SedqwdAvK3dsihxVUA/edit#heading=h.2ye70wns7io3

PyBullet. 2023. Bullet 3. https://github.com/bulletphysics/bullet3

Unitree Robotics. 2022. Unitree Robotics. https://github.com/unitreerobotics/unitree_ros/tree/master/robots/a1_description/urdf