Kinematics and Control of Quadruped A1

Fatih Özgan, Karsten Appenzeller

November 25, 2024

Contents

1 Denavit-Hartenberg Convention

The Denavit-Hartenberg-Convention is a systematic, general Method to derive transformations between reference frames, each of which represents a robot joint. The joints can be of types fixed, revolute, or prismatic [1]. It takes four parameters $\theta_i, \alpha_i, a_i, d_i$ to describe the transformation between two links in a kinematic chain, where i is the reference frame index:

- θ_i : Angle between axes x_{i-1} and x_i about axis z_{i-1}.
- α_i : Angle between axes z_{i-1} and z_i about axis X_i.
- a_i : Length of a common normal between z_{i-1} and z_i, measured along x_i.
- d_i : Length of a common normal between x_{i-1} and x_i, measured along z_{i-1}.

The convention includes a set of rules to derive how the frame coordinates are chosen and oriented

1. Use right-hand frames for each joint.
2. Set z_i -axis along the revolution or translation axis for the joint $i+1$.
3. Select the origin o_i at the intersection point of axes z_i and the common normal with axis z_{i-1}.
4. Select the x_i -axes in the same direction of common normal from z_{i-1} and z_i .

Link	a_i	α_i	d_i	ϑ_i
0	a_0	$\pi/2$	0	ϑ_0
1	0	$-\pi/2$	0	$\pi/2$
2	a_2	0	0	$\vartheta_2 - \pi/2$
3	a_3	0	0	ϑ_3

Table 1: Conventional DH-parameters for a quadruped leg

2 Direct Kinematics

Direct kinematics establish a functional relationship between joint configurations and end-effector pose. The resulting matrix contains the position and orientation of the end-effector based on the current θ_i values.

$$A_i^{i-1}(q_i) = A_{i'}^{i-1} A_i^{i'} = \begin{bmatrix} c_{\vartheta_i} & -s_{\vartheta_i} c_{\alpha_i} & s_{\vartheta_i} s_{\alpha_i} & a_i c_{\vartheta_i} \\ s_{\vartheta_i} & c_{\vartheta_i} c_{\alpha_i} & -c_{\vartheta_i} s_{\alpha_i} & a_i s_{\vartheta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$A_1^0(\vartheta_0) = \begin{bmatrix} c_0 & 0 & s_0 & a_0 c_0 \\ s_0 & 0 & -c_0 & a_0 s_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$A_2^1(\vartheta_1 = \pi/2) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$A_i^{i-1}(\vartheta_i) = \begin{bmatrix} c_i & -s_i & 0 & a_i c_i \\ s_i & ci & 0 & a_i s_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad i = 3, 4 \tag{4}$$

With our DH-parameters and by applying the transformations using (1) - (4) we get

$$T_4^0(q) = A_1^0 A_2^1 A_3^2 A_4^3 = \begin{bmatrix} s_0 s_{23} & s_0 c_{23} & c_0 & a_0 c_0 + s_0(a_2 s_2 + a_3 s_{23}) \\ -c_0 s_{23} & -c_0 c_{23} & s_0 & a_0 s_0 - c_0(a_2 s_2 + a_3 s_{23}) \\ c_{23} & -s_{23} & 0 & a_2 c_2 + a_3 c_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

as our matrix represents the end-effector pose.

The matrix

$$O_{EE} = \begin{bmatrix} s_0 s_{23} & s_0 c_{23} & c_0 \\ -c_0 s_{23} & -c_0 c_{23} & s_0 \\ c_{23} & -s_{23} & 0 \end{bmatrix} \tag{6}$$

corresponds to the end-effector's orientation while the foot coordinates are given by the equations:

$$P_x = a_0 c_0 + s_0(a_2 s_2 + a_3 s_{23}) \tag{7}$$
$$P_y = a_0 s_0 - c_0(a_2 s_2 + a_3 s_{23}) \tag{8}$$
$$P_z = a_2 c_2 + a_3 c_{23} \tag{9}$$

$$T_4^0(q) = \begin{bmatrix} O_{EE} & P \\ 0 & 1 \end{bmatrix} \tag{10}$$

3 Inverse Kinematics

While the direct kinematic equations of a robot solve the end-effectors' position and orientation with respect to a given joint configuration, the inverse kinematic problem is the determination of possible joint configurations for a given end-effector position.

3.1 Calculating θ_3

Let P_x, P_y, P_z be in as 2.

$$P_x^2 + P_y^2 + P_z^2 = a_0^2 + a_2^2 + a_3^2 + 2a_2a_3c_3$$

$$c_3 = \frac{P_x^2 + P_y^2 + P_z^2 - a_0^2 - a_2^2 - a_3^2}{2a_2a_3} \tag{11}$$

$$s_3 = \pm\sqrt{1 - c_3^2} \tag{12}$$

$$\tag{13}$$

$$\theta_3 = \arctan^2(s_3, c_3)$$

Yielding two possible solutions for either sign of s_3,

$$\theta_{3,1} \in [-\pi, \pi]$$
$$\theta_{3,2} = -\theta_{3,1}.$$

3.2 Calculating θ_2

$$P_x^2 + P_y^2 = a_0^2 + (a_2s_2 + a_3s_{23})$$
$$P_z = a_2c_2 + a_3c_{23}$$

$$s_2 = \frac{\pm\sqrt{P_x^2 + P_y^2 - a_0^2}(a_2 + a_3c_3) - a_3s_3P_z}{a_2^2 + a_3^2 + 2a_2a_3c_3} \tag{14}$$

$$c_2 = \frac{\pm\sqrt{P_x^2 + P_y^2 - a_0^2}(a_3s_3) + (a_2 - a_3c_3)P_z}{a_2^2 + a_3^2 + 2a_2a_3c_3} \tag{15}$$

From (13), (14) follows that

$$\theta_2 = \arctan^2(s_2, c_2)$$

Which yields four possible Solutions depending on the signs of s_3 and $\sqrt{P_x^2 + P_y^2 - a_0^2}$:

$$\theta_{2,1} = \arctan^2(\sqrt{P_x^2 + P_y^2 - a_0^2}(a_2 + a_3c_3) - a_3s_3^+P_z, \sqrt{P_x^2 + P_y^2 - a_0^2}(a_3s_3^+) + (a_2 - a_3c_3)P_z)$$
$$\theta_{2,2} = \arctan^2(-\sqrt{P_x^2 + P_y^2 - a_0^2}(a_2 + a_3c_3) - a_3s_3^+P_z, (a_2 - a_3c_3)P_z - \sqrt{P_x^2 + P_y^2 - a_0^2}(a_3s_3^+))$$
$$\theta_{2,3} = \arctan^2(\sqrt{P_x^2 + P_y^2 - a_0^2}(a_2 + a_3c_3) - a_3s_3^-P_z, \sqrt{P_x^2 + P_y^2 - a_0^2}(a_3s_3^-) + (a_2 - a_3c_3)P_z)$$
$$\theta_{2,4} = \arctan^2(-\sqrt{P_x^2 + P_y^2 - a_0^2}(a_2 + a_3c_3) - a_3s_3^-P_z, (a_2 - a_3c_3)P_z - \sqrt{P_x^2 + P_y^2 - a_0^2}(a_3s_3^-))$$

3.3 Calculating θ_0

Using P_x and P_y. We can calculate θ_0 as follows:
Out of

$$P_x^2 + P_y^2 = a_0^2 + (a_2 s_2 + a_3 s_{23})^2$$

follows that

$$a_2 + a_3 s_{23} = \pm\sqrt{P_x^2 + P_y^2 - a_0^2} \tag{16}$$

now substitute (16) in P_x and solve it for s_0

$$s_0 = \frac{P_x - a_0 c_0}{\pm\sqrt{P_x^2 + P_y^2 - a_0^2}} \tag{17}$$

substituting (17) in P_y leaves us with only constants and the variables we have previously determined. Now we can solve for c_0, which leaves us with:

$$c_0 = \frac{a_0 P_x - P_y(\pm\sqrt{P_x^2 + P_y^2 - a_0^2})}{P_x^2 + P_y^2} \tag{18}$$

Doing the according steps to solve for s_0 we get:

$$s_0 = \frac{a_0 P_y + P_x(\pm\sqrt{P_x^2 + P_y^2 - a_0^2})}{P_x^2 + P_y^2} \tag{19}$$

Our θ_0 is only dependent on constants and the end-effector position. This means we can determine two possible solutions for either sign of (16)

$$\theta_{0,1} = \arctan^2(a_0 P_y + P_x\sqrt{P_x^2 + P_y^2 - a_0^2}, a_0 P_x - P_y\sqrt{P_x^2 + P_y^2 - a_0^2}) \tag{20}$$

$$\theta_{0,2} = \arctan^2(a_0 P_y - P_x\sqrt{P_x^2 + P_y^2 - a_0^2}, a_0 P_x + P_y\sqrt{P_x^2 + P_y^2 - a_0^2}) \tag{21}$$

3.4 Possible solutions for given end-effector position

For any reachable position, there are four distinct solutions to our inverse kinematic problem, which can be verified by applying the direct kinematic formula to these solutions:

1. $\theta_{0,1}, \theta_{2,1}, \theta_{3,1}$
2. $\theta_{0,2}, \theta_{2,2}, \theta_{3,1}$
3. $\theta_{0,1}, \theta_{2,3}, \theta_{3,2}$
4. $\theta_{0,2}, \theta_{2,4}, \theta_{3,2}$

4 Operator workspace and redundancies

The workspace is limited by every possible coordinate in 3D space the end-effector (in this case the robot's foot) can reach. As previously derived, there are four unique solutions to every position (assuming every joint has a range $R = [-\pi, \pi]$). Considering all four solutions would result in a sphere-shaped workspace with a non-reachable area around the origin in a cylindrical shape. The non-reachable area is a product of shifting the thigh joint off center by a_0.

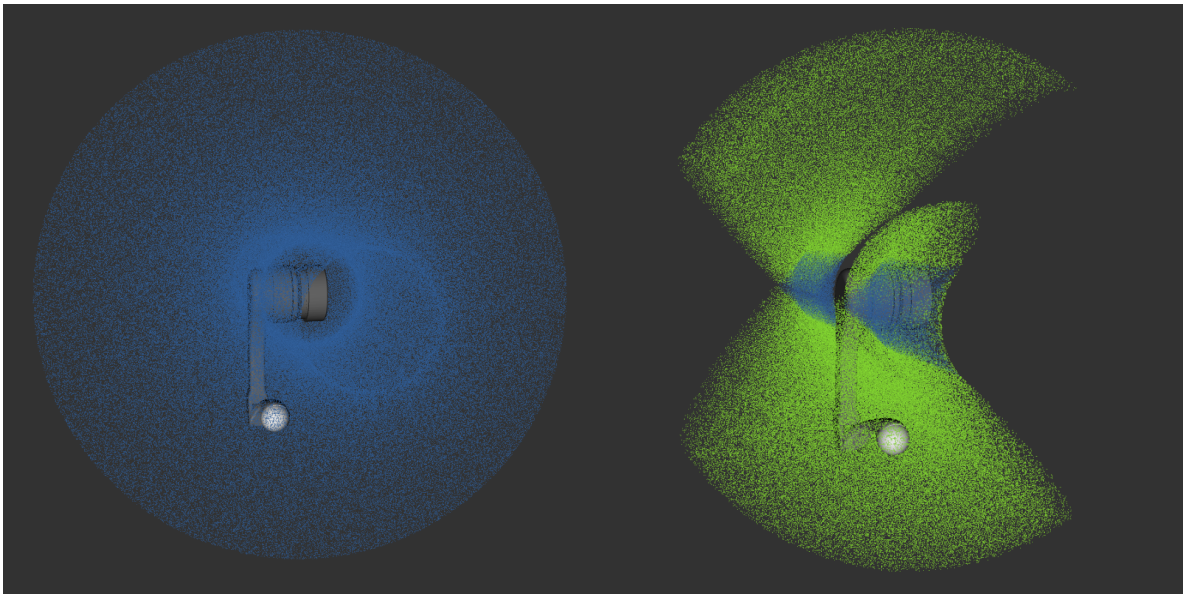
In reality the joints do not have a full 360° of freedom, which means that some of our solutions are not valid as joint configurations as our robot is physically incapable of achieving the desired position. Utilizing the manufacturers' joint ranges we can already rule out the first two solutions to the inverse kinematic problem as valid configurations, as θ_3 is positive in these cases. The other two solutions can be classified in 3 cases:

Joint	Minimum Angle ($\theta_{\min}$)	Maximum Angle ($\theta_{\max}$)
θ_0 (Hip)	-0.803	0.803
θ_2 (Thigh)	-1.047	4.189
θ_3 (Calf)	-2.693	-0.916

Table 2: Joint angle limits in radians

If there is only one correct solution below the xy-plane ($P_y < 0$) it is always the third solution. Correspondingly the fourth solution is what leads to the reachable points (with only one solution) above the xy-plane. The third case is when θ_0 and P_y get close to ± 0 where both remaining solutions hold.

This redundancy of solutions has to be taken into consideration when planning a trajectory for the robots' foot because using any "correct" solution for a given pose will cause erratic movement around the xy plane, i.e when in the robots' current position $\theta_0 = 0$ and the first solution for the next position would include $\theta_{0,max/min}$ the robot would have to move half of its joint range within split seconds. Using the solution that deviates the least from the current joint configuration as input for the robots' motors is generally a good idea.



(a) without joint limits (b) with actual joint limits

Figure 1: Visualizing the workspace in RViz with point clouds. Mind that every point in (b) is also in (a). Unique positions (only one solution exists) are colored green, whereas points with ≥ 2 solutions are in blue.

5 Controlling the Robot

The Git repository with the source code can be found on Github[1]. This section provides a brief elaboration on how we used our kinematics model to control the position and orientation of the robot on the spot as well as locomotion tasks (such as gait and trajectory generation). To get a general sense of the Package Structure, check out the ros_gz_project_template[2].

5.1 The Simulation

We used Gazebo Sim (Harmonic), accompanying ROS2 Jazzy to simulate the A1 Quadruped robot. Earlier versions of Gazebo Sim do not fully support all necessary Sensors and Controllers the physical A1 uses, for example, the contact sensors in older Gazebo Sim distributions don't provide a force vector (only a true or false if a collision occurred). In Gazebo Classic with ROS1, these features were provided by the manufacturer[3]. But for newer versions of ROS and Gazebo the support is declining and most quadruped simulations still take place in Gazebo Classic.

To simulate the Robot in ROS2 with the newer Gazebo Sim, we generated our own SDF-File from Unitree's provided URDF-File and integrated all necessary plugins (imu, contact, joint controller, ...). The Simulation consists of two ROS2 Packages:

ros_gz_a1_description: New SDF-File of the Quadruped Robot (includes imu and contact sensors)

ros_gz_a1_gazebo: SDF-Files of the Gazebo Worlds:

- **empty_a1.sdf:** Empty World file for joint position control
- **empty_a1.sdf:** Empty World file for joint torque control

5.2 Launch

The package ros_gz_a1_bringup consists of launch and config files that automatically run all necessary nodes in ROS2 and bridge the topics to the Gazebo Sim equivalents. To launch the simulation you can use:

- ros2 launch ros_gz_a1_bringup a1_gazebo_sim.launch.py use_force_ctrl:=True [Default: False]

The argument use_force_ctrl lets you switch between Joint Position and Joint Force controller.

5.3 Controller

The package ros_gz_a1_controller consists of ROS2 Nodes to calculate control inputs for the simulation. The nodes that make up the controller package are:

joint_state_publisher: receives a pose message to move/rotate the robot's center of mass (COM) along a certain offset/angle. The commands can be sent using the pose_pub_gui node from the ros_gz_a1_ui package. (Simulation has to be in Joint Position Controller-Mode)

The translation in space of the robot COM (movement in x,y, and z direction) is simply done by adding the control input offset to the robot's current foot position, calculating the inverse kinematics for the new position, and sending the new joint configuration topic to the Gazebo Position Controller.

The rotation on the other hand can not be done from the hip frame alone. We have to rotate each foot position around the COM reference frame to change the robot's $roll, pitch$, and yaw. Rotating the body in any direction is the same as rotating the rectangle made up of the foot positions about the center of mass in the opposite direction. This means we have to first translate the hip frame coordinates into body frame coordinates, then rotate those coordinates around the origin, translate the result back into the hip reference frame, and finally calculate the inverse kinematics for the new goal position. For a more intuitive and visual explanation of kinematics

[1] https://github.com/K-d4wg/QuadrupedA1Controller
[2] https://github.com/gazebosim/ros_gz_project_template
[3] https://github.com/unitreerobotics/unitree_ros

(though not using the DH Convention) consider watching James Bruton's openDog robot series on YouTube. This video, in particular, might be helpful: https://www.youtube.com/watch?v=4MGZvcd0xxc.

low_level_controller: receives the current joint states of the robot model and publishes force commands to the simulation according to a PD-Control Law. (Simulation has to be in Apply Joint Force-Mode).

a1_controller: The implemented Gait is a trotting Gait that can move the robot along its body x-axis and rotate along the z-axis. The controller receives a Twist message for example from the cmdvel_pub_gui node (in ros_gz_a1_ui package). (low_level_controller node has to be running).

The implemented trajectory generation is a variation of the curve generation method using Bezier Curves from [2]. Instead of 12 control points, we only used 10 and scaled the control point coordinates by the commanded velocity value. This results in a distinct curve for each velocity input and thus allows the robot to switch between standing and running smoothly.

The scripts for our kinematics model, curve, and trajectory planning can be found in the 'lib' directory within the ros_gz_a1_controller package.

- A1_kinematic.py: a collection of the previously inferred forward and inverse kinematics equations, mainly used by the controller scripts
- Bezier.py: script to calculate a bezier curve according to our control points and commanded velocity, running the script itself will plot a default curve using matplotlib
- Tajectory_planner.py: Here we implemented the functions to calculate the next foot positions for the following time steps. There are also a few helper functions like conversion between body and hip frames and rotation of points along all axes.

5.4 Example Usage

- To only run the simulation with all topics published via ros_gz_bridge you can launch the simulation with either use_force_ctrl:=True/False and provide your controller package to manipulate the robot using position or force commands.
- To use the whole-body kinematics demo first launch the simulation in joint position controller mode (use_force_ctrl:=False) and then the nodes for robot control and user input.
  1. ros2 launch ros_gz_a1_bringup a1_gazebo_sim.launch.py
  2. ros2 run ros_gz_a1_controller joint_state_publisher
  3. ros2 run ros_gz_a1_ui pose_pub_gui
- To use the locomotion example first launch the simulation in joint force mode and then the low_level_controller, a1_controller and cmdvel_pub_gui nodes.
  1. ros2 launch ros_gz_a1_bringup a1_gazebo_sim.launch.py use_force_ctrl:=True
  2. Click play in Gazebo Sim to let the robot lay on the ground before running the low-level controller
  3. ros2 run ros_gz_a1_controller low_level_controller
  4. ros2 run ros_gz_a1_controller a1_controller
  5. ros2 run ros_gz_a1_ui cmdvel_pub_gui

References

[1] B. Siciliano, *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing, London: Springer London, 2009.

[2] X. Zeng, S. Zhang, H. Zhang, X. Li, H. Zhou, and Y. Fu, "Leg trajectory planning for quadruped robots with high-speed trot gait," *Applied Sciences*, vol. 9, no. 7, 2019.