robotics

MDPI

Article

Control of a Hexapod Robot Considering Terrain Interaction

Stefano Arrigoni *, Marco Zangrandi, Giovanni Bianchi and Francesco Braghin

Mechanical Engineering Department, Politecnico di Milano, 20156 Milano, Italy; giovanni.bianchi@polimi.it (G.B.); francesco.braghin@polimi.it (F.B.)
* Correspondence: stefano.arrigoni@polimi.it

Abstract: Bioinspired walking hexapod robots are a relatively young branch of robotics. Despite the high degree of flexibility and adaptability derived from their redundant design, open-source implementations do not fully utilize this potential. This paper proposes an exhaustive description of a hexapod robot-specific control architecture based on open-source code that allows for complete control over a robot's speed, body orientation, and walk gait type. Furthermore, terrain interaction is deeply investigated, leading to the development of a terrain-adapting control algorithm that allows the robot to react swiftly to the terrain shape and asperities, such as non-linearities and non-continuity within the workspace. For this purpose, a dynamic model derived from interpreting the hexapod movement is presented and validated through a Matlab SimMechanics™ simulation. Furthermore, a feedback control system is developed, which is able to recognize leg–terrain touch and react accordingly to ensure movement stability. Finally, the results from an experimental campaign based on the PhantomX AX Metal Hexapod Mark II robotic platform by Trossen Robotics™ are reported.

Keywords: hexapod robot; terrain interaction; locomotion control; dynamic model

Citation: Arrigoni, S.; Zangrandi, M.; Bianchi, G.; Braghin, F. Control of a Hexapod Robot Considering Terrain Interaction. *Robotics* **2024**, *13*, 142. https://doi.org/10.3390/robotics13100142

Academic Editor: Chengxu Zhou

Received: 2 August 2024
Revised: 11 September 2024
Accepted: 19 September 2024
Published: 24 September 2024

1. Introduction

Mobile ground robots emerged as an alternative to several tasks originally performed by humans, such as surveillance, reconnaissance, entertainment, personal services, patrolling, and industrial automation, to cite a few [1]. Particularly, nowadays, interest is focused on tasks involving complex and potentially hazardous situations such as planetary exploration, emergency rescue operations, and, more in general, intervention in extreme environments [2].

Mobile ground robots in the literature differ regarding locomotion, surrounding perception, and control methodologies. Focusing on locomotion, they can be grouped into the following:

- Wheeled robots;
- Tracked robots;
- Legged robots;
- Hybrid robots [1].

Wheeled robots are characterized by high speed, high energy efficiency, and a cost-effective design, so they are widespread in industrial, service, and delivery applications on mostly flat surfaces [1]. Conversely, tracked robots are more appropriate for movements on soft and irregular terrains because they have a larger surface in contact with the ground, and they are also able to climb over obstacles [3]. Legged robots show superior mobility compared to wheeled and tracked robots since they can navigate through any terrain, climbing rocks and rough surfaces like humans and animals [1,4,5]. These extraordinary capabilities and their versatility that allow them to operate in different environments make them the preferred choice for search and rescue [6], exploration [7], agriculture [8], or operations in dangerous environments [9–11] despite their high complexity and cost. Finally, some hybrid robots combine these locomotion types, being equipped with legs and

wheels or tracks as paws. These robots have the same adaptability and agility as legged robots because they can easily overcome obstacles and perform different types of gaits on uneven terrains and, at the same time, can achieve high speed with great efficiency on flat surfaces [12]. Nevertheless, the complexity of the design limits the optimality in both aspects [13].

The choice of using bioinspired paws or hybrid locomotion depends on the particular type of terrain in which the robot is supposed to operate. Still, the main parameter that influences the performance of a legged robot is the number of legs [1], and they can be one of the following:

- Bipedal robots;
- Quadrupedal robots;
- Hexapod robots;
- Multilegged robots with more than six legs.

Bipedal robots are mainly humanoid robots designed to replicate many human abilities, such as grasping, handling objects, or human-like sensing, and locomotion is just one of the several tasks these robots are capable of performing. Nevertheless, many humanoid robots can run, jump, climb, and move in any environment with extreme agility [1]. The disadvantage of bipedal locomotion is the intrinsic instability of this gait, which is why humanoid robots require very complex control algorithms to maintain balance even when they do not move.

Quadrupedal robots are the best-performing walking robots in terms of speed and payloads [1]. Moreover, they have the advantages of allowing different gait types and being statically stable when they are not moving. Nevertheless, they still require very complex control algorithms since they possess many degrees of freedom that must be coordinated, and most performing gaits need dynamic walking control since only two limbs are in contact with the ground.

Hexapod robots, instead, always have three limbs in contact with the ground, so their gaits are stable, making walking control much simpler despite the increased number of degrees of freedom to coordinate [14,15]. This stability makes them particularly suitable on uneven or slippery terrains where bipedal or quadrupedal robots struggle to find a stable gait. Furthermore, their redundancy allows them to operate by still using quadrupedal locomotion if one or two legs are not functioning [1], which is a significant advantage for operations in places difficult or unsafe to reach for humans. Robots with more than six legs do not have particular advantages over hexapod robots apart from the increased variety of gaits and the improved redundancy, but these come at the cost of more complex control due to the additional degrees of freedom.

2. State of the Art

In the field of ground robots, one of the crucial benefits of hexapod robots is their outstanding ability to operate on virtually any terrain. An essential aspect is represented by the detection and mapping of terrain shape and unevenness. Common approaches for ground classification can be classified as Visual-Perception-based, Depth-Perception-based, or Tactile-Perception-based, as reported in [16], where all three families of sensors are combined in order to achieve improved accuracy. Visual-Perception-based systems take advantage of image processing for ground classification, starting with cameras. In [17], an online terrain-classification system for hexapod robots capable of adapting energy-efficient gaits is presented, which is based on feature extraction and SVM classification of a monocular camera output, achieving up to 90% accuracy on terrain images. Another example is reported in [18], where an algorithm based on neural networks is used to identify the terrain from an RGB image to define surmountable obstacles and zones. Vision-based terrain mapping is also used in [19], where visual sensors are used to detect the terrain shape using point clouds and unsupervised learning techniques and to control the interaction between two collaborative robots. Depth-Perception-based classification relies on the use of depth data, usually in addition to images from cameras as presented

in [20] with the purpose of characteristics extraction and objects or plane identification. Finally, Tactile Perception relies on using sensors such as IMU, pressure, or torque sensors to perceive the presence of the ground, usually thanks to the measurements (forces and torques) at the tip of the leg as reported in [21]. In this paper, a terrain map is considered as known since it is not the aim of this research.

In order to exploit the potential of hexapod robots in uneven terrain conditions fully, the interaction of their paws with the ground must be taken into account both in the robot's mechanical design and the implementation of the control algorithm [15,22] to properly provide counteraction to terrain-reconstruction errors. For this reason, the most advanced control techniques of hexapod robots not only include a kinematic and dynamic model of the robot itself but also consider the forces that the paws exert on the ground. They use this information to detect contact, consider terrain deformation, and avoid slipping. Among all the possible improvements that are introduced by adding a model of interaction with the ground to the control algorithm, contact detection is the most useful since it allows the robot to maintain balance on uneven terrains and in the presence of obstacles because, by knowing which paws are in contact with the ground, the robot can adjust its posture and adapt its gait to accommodate terrain unevenness, distributing the weight better and improving stability [22].

In addition, real-time feedback on contact detection reduces the time needed for each leg to search for contact with the ground, making the transition between consecutive steps faster and smoother.

As already introduced, contact detection can be achieved by directly measuring the force between each paw and the ground with pressure or force sensors or by estimating this force by measuring the torque or the current of the motors of each joint of the legs [15]. The first approach gives a more accurate measurement of the contact force, but it requires placing sensors under the robot's paws, adding complexity to the geometry of the paw [23–28]. In addition, this strategy relies on components that continuously receive impact forces, and they might easily break. Conversely, estimating the contact force by measuring the motors' torque is a less accurate but more reliable approach. Furthermore, in some scenarios, position feedback can be sufficient to accurately identify terrain presence and maintain horizontal positioning at all times [29–31].

In the field of legged robotics, companies offer proprietary solutions, such as Boston Dynamics [32], or others are more focused on academic research, like Robotis [33] and Trossen Robotics [34]. From a software perspective, academic research tends to favor the use of open-source solutions to provide greater access and control over the robot's implementation. In this context, various frameworks enable the programming of hexapod robots. In particular, NUKE [35] is open-source software designed specifically for the control of hexapod robots. It is optimized to manage the locomotion and kinematics of these robots through an intuitive interface and predefined configurations that facilitate programming and customization. Compared to more general-purpose solutions like ROS2 [36] or ArduPilot [37], NUKE provides predefined and customizable solutions that reduce complexity and accelerate the development of hexapod projects. This makes it a convenient choice, although its support is less extensive and its updates are less frequent compared to other solutions, particularly ROS2.

Based on the previous discussion, the main contribution of this work consists of a deep analysis and extension of the inverse kinematics formulation of NUKE to include the orientation and height of the robot's body from the ground as additional input to the position of the robot's leg's end effectors. Moreover, this paper proposes a comprehensive architecture for handling terrain irregularities that leverages the extended functionalities of NUKE and estimates joint torques in the event of individual leg contact with the ground using a dynamic hexapod model. The standard implementation of NUKE suffers from the main limitation that most multilegged robot-control software has. While perfectly able to move the robot smoothly and adjust stride correctly through a gait engine, they can generally not interact with anything other than flat, continuous, obstacle-free terrain [38].

This paper aims to extend the advantages of the NUKE code in the presence of non-flat, non-continuous, irregular terrain. In particular, the underlying objective is to develop a kinematic model that can account for the terrain shape and location and to improve this analysis further to understand how the robot's physics interacts with the presence of the ground. By utilizing this estimate alongside the one provided by the robot's servomotors, a feedback control system was developed without the need for sensors at the ends of the legs. In order to implement and test the proposed architecture, PhantomX AX Metal Hexapod Mark II from Trossen Robotics™ [39], shown in Figure 1, is considered. It is a versatile, inexpensive, and lightweight robot. Moreover, it is fully programmable, compatible with NUKE, and based on open-source hardware. It has neither force feedback sensors on its legs nor an IMU in its standard configuration, but it mounts 18 Dynamixel™ AX-12A Smart Servomotors [40] that can provide feedback on position and load estimates.

Figure 1. PhantomX AX Metal Hexapod Mark II [39].

The novelty of this work is represented by the development of a control algorithm for hexapod robots that can maintain the robot's body horizontally regardless of the slope and the unevenness of the terrain. This goal can be achieved without the addition of any sensors on the robot's feet because the estimation of the motors' torque allows it to detect contact with the ground, and by knowing the angles of each joint, the robot's pose is reconstructed through inverse kinematics. In addition, this control algorithm extends the functionality of NUKE, an open-source software compatible with different kinds of hexapod robots and widespread in this field, making this work helpful for other studies involving hexapod locomotion.

In Section 3, the kinematic model and the procedure to account for ground shape are described. In Section 4, the dynamic model used to estimate the torque at each motor is proposed and numerically validated. In Section 5, the novel architecture system comprehensive of a compensation algorithm is presented, and finally, in Section 6, experimental results in different scenarios are reported.

3. Locomotion Control

Locomotion control of the hexapod robot is mainly accomplished through the direct command of its legs' endpoints. Leg endpoints are defined as the 3D coordinate points located at the extremities of each leg, as shown in Figure 2.

Since legs are 3-DOF systems that lead to the endpoints, it is possible to attain full control of a leg by imposing its endpoint position and employing an inverse kinematics engine to calculate the associated servomotor angles.

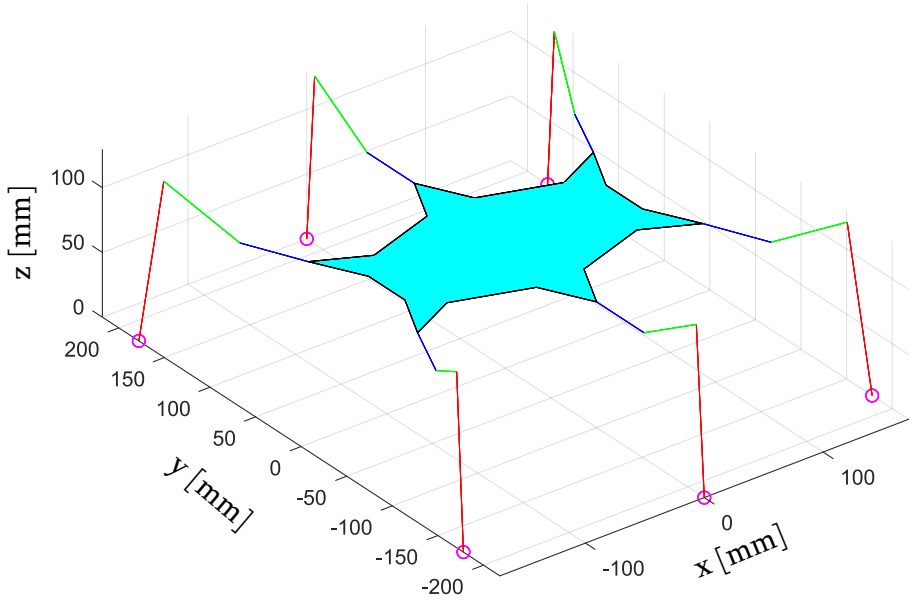


Figure 2. Robot endpoints in neutral position. The blue segment represents the coxa, the green one the femur, and the red one the tibia.

Such control architecture offers the advantage of collapsing the description of each leg configuration to a single point position in space, leading to much simpler and clearer handling of the robot's movement.

The position that the robot assumes at the deployment state is called the neutral position, and the associated endpoint coordinates are hardcoded into the robot controller software. This position represents a neutral state that the kinematic engine will use as a reference to build its walking gait.

3.1. Endpoints Handling

The coordinate systems employed in the kinematic analysis of the robot are presented in Figure 3. The *global coordinate system* is fixed at terrain level at 0,0,0 coordinates and will be used to account for the robot's position with regard to the terrain and the environment. Therefore, for the first iteration, the transformation matrix for the global-to-body coordinate system $T_{globalbody,0}$ (alias: $T_{gb,0}$) is as shown in (1):

$$T_{gb,0} = \begin{bmatrix} 1 & 0 & 0 & \mathrm{SP}_x \\ 0 & 1 & 0 & \mathrm{SP}_y \\ 0 & 0 & 1 & \mathrm{SP}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

where 'SP' is the *starting position*. Note that SP_z should be the initial robot height and must be consistent with the neutral position endpoint coordinates.

The *body coordinate system* and the *legs coordinate systems* instead move alongside the robot's body and are used to both describe the robot's orientation and solve the inverse kinematics for endpoint position and servomotor angles.

The transformation matrix that binds the leg c.s. and the body c.s. $T_{bodyleg}$ (as: T_{bl}) is defined as (2):

$$T_{bl} = \begin{bmatrix} 1 & 0 & 0 & x_j \\ 0 & -1 & 0 & y_j \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where in (2), x_j and y_j are the joint positions in the body c.s., from which it is inferred that T_{bl} is unique for each leg as each leg has a different joint position.

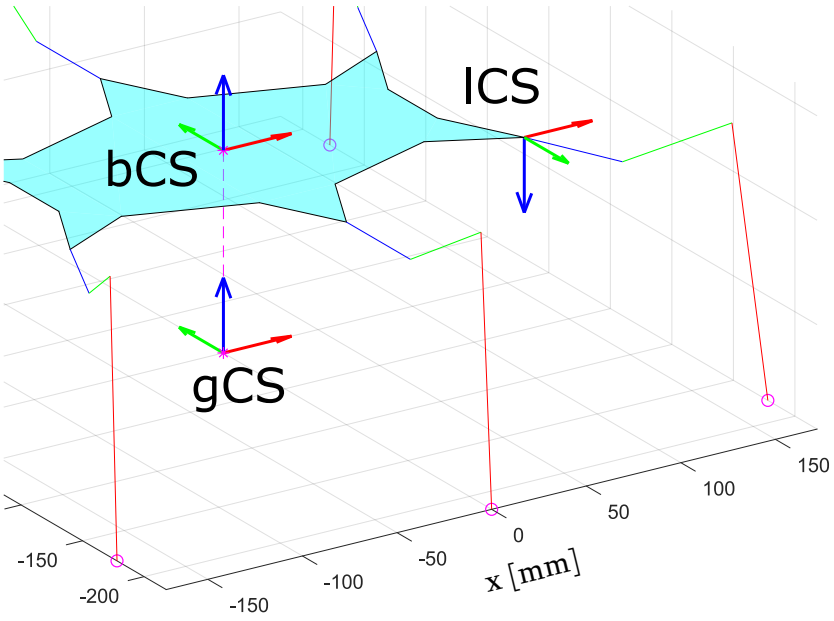


Figure 3. The body, leg, and global coordinate systems.

Once the desired endpoint position has been defined, servomotor angles can be assessed. The convention used in the kinematic analysis is shown in Figure 4, where ϵ represents the servomotor orientation and θ, ϕ, and ψ represent the coxa, femur, and tibia servomotor angles, respectively.

By expressing the desired endpoint position in the leg coordinate system, eventually, through (2), the servomotor angles can be calculated through (3)–(7):

$$trueX = \sqrt{x^2 + y^2} - l_c \tag{3}$$

$$im = \sqrt{trueX^2 + z^2} \tag{4}$$

$$\theta = -\arctan\left(\frac{y}{x}\right) - \epsilon \tag{5}$$

$$\phi = \frac{\pi}{2} - \arctan\left(\frac{trueX}{z}\right) - \arccos\left(\frac{l_f^2 + im^2 - l_t^2}{2\,im\,l_f}\right) \tag{6}$$

$$\psi = \pi - \arccos\left(\frac{l_f^2 + l_t^2 - im^2}{2\,l_f\,l_t}\right) \tag{7}$$

where in (3)–(7), (x, y, z) is the position of the desired leg endpoint in the leg coordinate system, and l_c, l_f, and l_t are the lengths of the coxa, femur, and tibia, respectively. The endpoint positions are defined in the body coordinate system; therefore, their significance is fully derivative of the position and orientation of the robot's body. This means that every movement that the robot's body is instructed to make can be immediately transposed to a relative displacement of the endpoints; the complete kinematic model can be built based on that assumption.

A gait engine is a subroutine of the locomotion algorithm that handles leg synchronization: in general, any pedal locomotion divides each leg in either a *pushing* or *swinging* state in a fixed order in such a way to allow for repetitive, continuous movement. The gait engine is responsible for assigning the *swing* and *push* roles as well as the direction and amount of space to cover for each iteration tick. The gait engine employed is not fundamental to describe the desired formulation and will not be discussed as descriptions are already

available in the literature (the gait engine employed in this treatment is the one provided by NUKE). Therefore, from the user input comprising the x-speed, y-speed, and z-axis rotation speed, the gait engine provides the movement data that the robot is supposed to follow, and the kinematic problem is to find the related endpoint displacement.

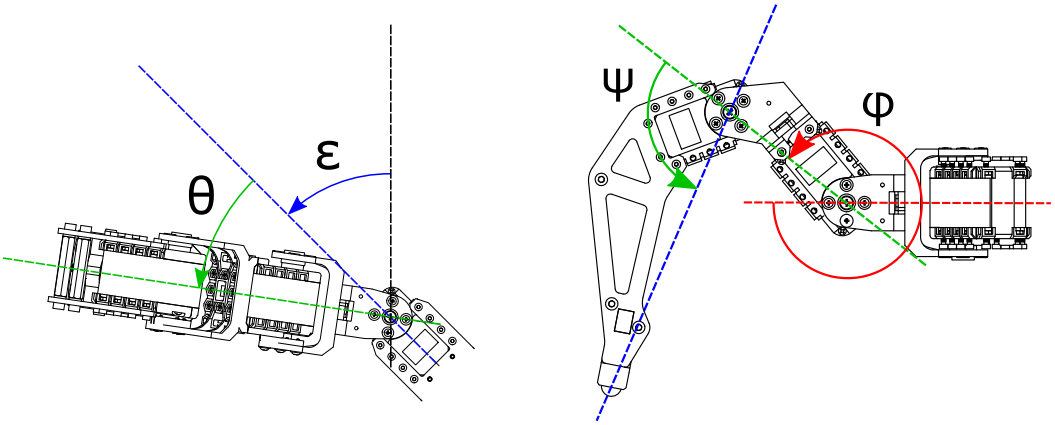


Figure 4. Definition of θ, ϕ, and ψ.

In order to solve the kinematic problem, it is imperative to be able to describe the robot's position and orientation at each iteration step by taking into consideration all the movement data provided by the gait engine and by direct command of the user (the user is supposed to be able to bypass the gait engine instructions to apply direct control of the robot's pose at each iteration step).

That is performed by building the $T_{globalbody}$ transformation matrix at each ith iteration step (as $T_{gb,i}$) as shown in (8):

$$T_{gb,i} = T_{tr,i}\, Q_{ter,i}\, Mov_i\, RotZ_{b,i}\, RotY_i\, RotX_i \tag{8}$$

where in (8),

- $T_{tr,i}$ is the translational transformation matrix holding data about the movement instructions coming from the gait engine. It is built as shown in (9) and reconstructs the robot's position as if it were just under the influence of the gait engine alone:

$$T_{tr,i} = T_{tr,i-1}\, Q_{tr,i} = T_{gb,0} \cdot Q_{tr,1} \cdot \ldots \cdot Q_{tr,i} \tag{9}$$

  In which $Q_{tr,i}$ is the transformation matrix representing the ith step movement due to the gait engine instructions. It is made of two contributions, as shown in (10):

$$Q_{tr,i} = \delta T_{mov,i}\, \delta RotZ_{g,i} \tag{10}$$

  In which $\delta T_{mov,i}$ is the component related to the translational movement and $\delta RotZ_{g,i}$ is the one related to the z-axis rotation. Those are defined in (11) and (12), respectively:

$$\delta T_{mov,i} = \begin{bmatrix} 1 & 0 & 0 & \delta x_{gait,i} \\ 0 & 1 & 0 & \delta y_{gait,i} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

$$\delta RotZ_{g,i} = \begin{bmatrix} \cos\left(\delta rot_{z_{gait,i}}\right) & -\sin\left(\delta rot_{z_{gait,i}}\right) & 0 & 0 \\ \sin\left(\delta rot_{z_{gait,i}}\right) & \cos\left(\delta rot_{z_{gait,i}}\right) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

where in (11) and (12), $\delta x_{gait,i}$, $\delta y_{gait,i}$, and $\delta rot_{z_{gait,i}}$ are the gait engine movement instructions for the i^{th} step x-axis displacement, y-axis displacement, and z-axis rotation, respectively.

- Mov_i is the transformation matrix holding the user-imposed translational movement data of the robot's body, defined as in (13):

$$Mov_i = \begin{bmatrix} 1 & 0 & 0 & bodyPos_{x_i} \\ 0 & 1 & 0 & bodyPos_{y_i} \\ 0 & 0 & 1 & bodyPos_{z_i} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

- $RotZ_{b,i}$, $RotY_i$, and $RotX_i$ are the transformation matrices holding the user-imposed orientation of the robot's body, being the z-axis rotational matrix, y-axis rotational matrix, and x-axis rotational matrix, respectively.
- $Q_{ter,i}$ is the terrain-compensated reorientation matrix addressing additional body displacement due to terrain shape. It depends on the position of the robot's body in the terrain environment. It can either be given by user input or computed in real-time based on the terrain shape in the robot's surroundings. Our objective is to assemble this matrix automatically by employing a terrain-tuning algorithm that takes as its only input the terrain elevation function $h(x, y)$, which can again either be given as user input (assuming perfect knowledge of the terrain shape) or constructed by an estimation architecture. $Q_{ter,i}$ is an identity matrix for completely flat terrains.

Therefore, the entire movement comprising all contributions that the robot's body goes through at the ith iteration step is represented by the transformation matrix calculated as in (14):

$$Q_{body,i} = T_{gb,i-1}^{-1} T_{gb,i} \tag{14}$$

Due to friction, the *pushing legs* endpoints are fixed to the terrain; therefore, their global position should not change between iterations. This means that if the body moves as described by (14), then the relative position of the endpoints should change as (15):

$$\begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = Q_{body,i}^{-1} \begin{bmatrix} x_{i-1} \\ y_{i-1} \\ z_{i-1} \\ 1 \end{bmatrix} \tag{15}$$

This way, it is possible to solve the kinematic problem for the *pushing legs* endpoints. Note that since (15) is built recursively, it needs starting values. These are the neutral position endpoints, which is why their coordinates are hardcoded into the control software. The *swinging legs* endpoint positions are defined as (16):

$$\begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}_{gCS} = T_{tr,i}\, Q_{ter,i}\, RotZ_{g,i}\, T_{mov,i} \begin{bmatrix} x_{neu} \\ y_{neu} \\ z_{neu} \\ 1 \end{bmatrix}_{bCS} \tag{16}$$

where x_{neu}, y_{neu}, and z_{neu} are the endpoint coordinates of the neutral position. The reason why (16) refers to the neutral position endpoint coordinates and addresses $RotZ_{g,i}$ and $T_{mov,i}$ instead of $\delta RotZ_{g,i}$ and $\delta T_{mov,i}$ is because of the core difference between (8) and (14): instead of building the endpoint transformation from the previous endpoint position,

the gait engine now needs to displace the leg in such a way that the swinging stride motion is obtained. This means that the movement data coming from $RotZ_{g,i}$ and $T_{mov,i}$ are no longer about increments of movement but effective displacements from the neutral position. The gait engine should return these values when assessing the *swinging* legs endpoints.

To account for terrain presence and ensure that the leg will always be over the terrain height while swinging, the correction shown in (17) should be applied:

$$z_{i,gCS} = z_{gait,i} + h\left(x_{i,gCS}, y_{i,gCS}\right) \tag{17}$$

where $h(x, y)$ is the *elevation function* of the terrain.

Once the global coordinates of the swinging legs endpoints are found, the body-centered coordinates are found as in (18):

$$\begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}_{bCS} = T_{gb,i}^{-1} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}_{gCS} \tag{18}$$

Completing the kinematic problem assessment.

3.2. Terrain Compensation Algorithm

The objective of this section is to develop a way to orientate the body pose of the robot in such a way that while moving the robot freely on the ground, its body becomes 'isolated' with respect to the ground itself.

Assuming perfect knowledge of terrain geometry in terms of angular and discontinuity interface position would not be realistic nor aligned with our objective of developing an adaptive algorithm. Our purpose is the design of an algorithm able to deal with any terrain just relying on the elevation function, which could be measured by sensors mounted on the robot itself, but this is not the purpose of this work [16,41].

First of all, it is important to unequivocally define the 'body isolation' condition: this can be achieved by defining a set of points in the robot's body and then considering the height of these points with respect to the ground as a way to evaluate the body's relative position to the terrain. A possible approach is to ask for those points to maintain a distance from the ground as close as possible to the one defined by SP_z in (1). If these points of interest are correctly chosen in order to represent the vertices of the robot's body, the entire base should follow the terrain profile and prevent unwanted situations like the ones discussed beforehand.

With the proposed algorithm, the robot's body will be able to position itself in such a way that it complies with the terrain geometry no matter the harshness.

In the algorithm presented, six points are selected in the locations of the robot's shoulders, as shown in Figure 5.

The main assumption of the compensation model is that the robot will be able to position itself in the optimal pose by moving from its horizontal pose defined by T_{tr} with three degrees of freedom only: a first displacement in the z-direction followed by a rotation around its body c.s. y-axis and a final relative rotation around its body c.s. x-axis. In transformation matrices, this is described by (19):

$$Q_{ter,i} = D_{ter,i}\, RotY_{ter,i}\, RotX_{ter,i} \tag{19}$$

where

- $D_{ter,i}$ is the z-axis rigid translation matrix defined as in (20):

$$Disp_{ter,i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \delta z_{ter,i} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{20}$$

- $RotY_{ter,i}$ and $RotX_{ter,i}$ are the rotation matrices defined as in (21) and (22):

$$RotY_{ter,i} = Rot(\alpha_{ter,i}, Y) \tag{21}$$
$$RotX_{ter,i} = Rot(\beta_{ter,i}, X) \tag{22}$$

where in (21) and (22), $\alpha_{ter,i}$ is the angular rotation around the body c.s. y-axis and $\beta_{ter,i}$ is the angular rotation around the body c.s. x-axis.

From the global coordinate system, the joint positions after the terrain reorientation are (23):

$$\begin{bmatrix} x_{j_{ter,i}} \\ y_{j_{ter,i}} \\ z_{j_{ter,i}} \\ 1 \end{bmatrix}_{gCS} = T_{tr,i}\, Disp_{ter_i}\, RotY_{ter_i}\, RotX_{ter_i} \begin{bmatrix} x_j \\ y_j \\ z_j \\ 1 \end{bmatrix}_{bCS} \tag{23}$$

Since we want our points to have relative heights as close to SP_z as possible, we can build an algorithm that minimizes the total relative height quadratic error. It is accomplished by accounting for each point, shown in (24) with reference to (25):

$$\begin{aligned} f &= \sum \left(z_{j_{ter}} - Z_{terrain} - \mathrm{SP}_z\right)^2 = \\ &= \sum \epsilon_j^2(a, b, c, d, e) = \\ &= \sum (a - x_j\, b + z_j\, c\, d + y_j\, d\, e + \\ &\qquad - h\left(x_{j_{ter}}, y_{j_{ter}}\right) - \mathrm{SP}_z\big)^2 \end{aligned} \tag{24}$$

$$\begin{cases} a = \delta z_{ter,i} \\ b = \sin \alpha_{ter,i} \\ c = \cos \beta_{ter,i} \\ d = \cos \alpha_{ter,i} \\ e = \sin \beta_{ter,i} \end{cases} \tag{25}$$

The problem is a multivariable optimization problem with equality constraints coming from trigonometric function consistency, solvable by writing the Newton–Euler equations with Lagrangian multipliers [42]. The formulation obtained and reported in Equation (26) comes from adjoining the equality constraints in Equation (25) to the cost function in Equation (24):

$$L = f + \begin{bmatrix} \lambda_1 & \lambda_2 \end{bmatrix} \begin{bmatrix} c^2 + e^2 - 1 \\ b^2 + d^2 - 1 \end{bmatrix} \tag{26}$$

The necessary conditions hold for the minimum [42] as $\nabla L = 0$.

The Lagrangian function is differentiable, and its gradient is calculated by taking the partial derivatives of Equation (26).

In order to find the solution to $\nabla L = 0$, M-V-O-P numerical methods need to be employed. In this situation, given that the cost function is quadratic, it is possible to effectively employ the steepest descent algorithm to iterate and find the solution very quickly and with a relatively low computational cost.

Having already defined the gradient of the Lagrangian, the steepest descent algorithm is implemented as in (27):

$$X_{it+1} = X_{it} - \nabla L(X_{it}) \cdot Weight \tag{27}$$

With the stopping condition being $||X_{it} - X_{it-1}|| \leq tol$. In (27), the *Weight* parameter represents the adjustment index of the increment to the next-iteration solution. Generally, a *Weight* of 1×10^{-5} is good enough to obtain convergent solutions in a few iterations in most general applications.

This method ensures good performances in most terrain situations, only experiencing non-convergence problems in extreme scenarios. As shown in Figure 6, with reasonable weight (*Weight*) and tolerance (*tol*), the solution is generally found very quickly.

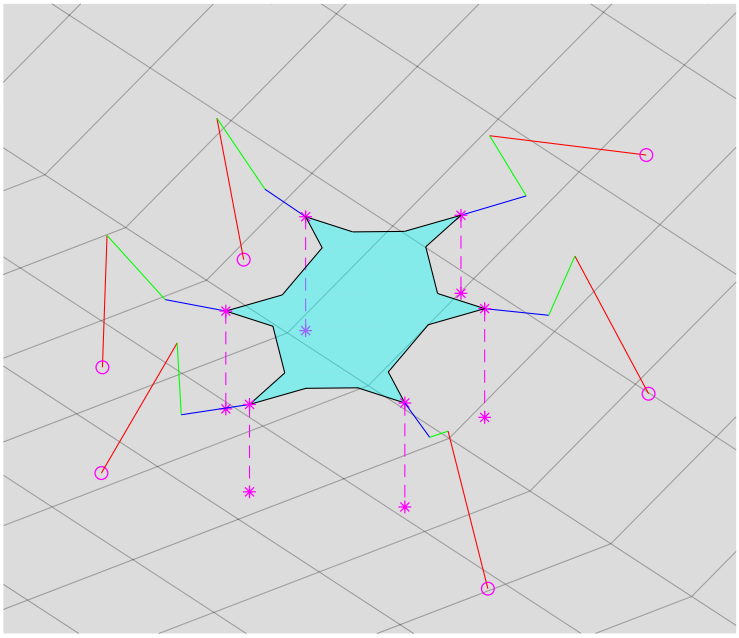


Figure 5. Interest points positioned with robot's shoulder joints.

The robot's stability in the current formulation is ensured through NUKE's original gait engine, which arithmetically enforces the robot COG to always stay within bounds. Indeed, the only gait types accepted by the algorithm are those that allow the hexapod robot to maintain static stability in all intermediate positions during movement, with the "less-redundant" one being the tripod gait, where three legs are moved at each time, but the algorithm always ensures that the support triangle bounds the COG projection.

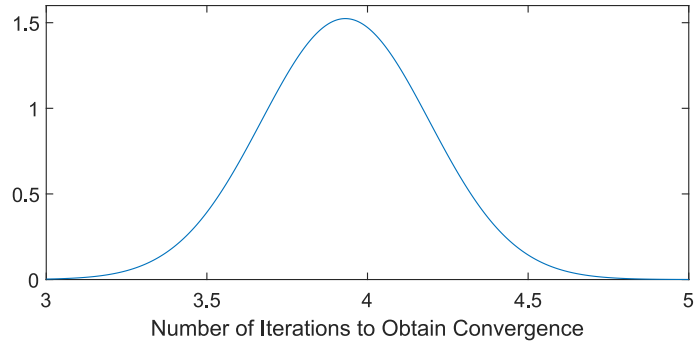


Figure 6. Gaussian distribution of iterations required for convergence with $Weight = 1 \times 10^{-5}$ and $tol = 0.01$ for a typical walk task in the terrain $h(x, y) = 50\,\text{mm} \times \left(\sin\left(\frac{x}{100\,\text{mm}}\right) + \cos\left(\frac{y}{100\,\text{mm}}\right)\right)$.

Since the problem was defined in general terms, the algorithm can run even in situations where rough interfaces are present, moving the robot in such a way as to allow for a smooth transition between angular interfaces and navigation over non-continuous terrains. An important assumption considered in this analysis is the simplification of the terrain behavior: an ideal undamped and rigid terrain is assumed, and the potential slipping of the

leg upon exceeding the static friction limit is neglected since a purely geometrical approach is proposed.

In the scenario of a steep ramp given a full horizontal speed input, the robot can smoothly go from being completely horizontal to adapting to the terrain inclination, as shown in Figures 7 and 8.

Since the algorithm runs on the elevation function only, realistic terrains defined through conventional discrete models [43] can be used without particular adjustments.

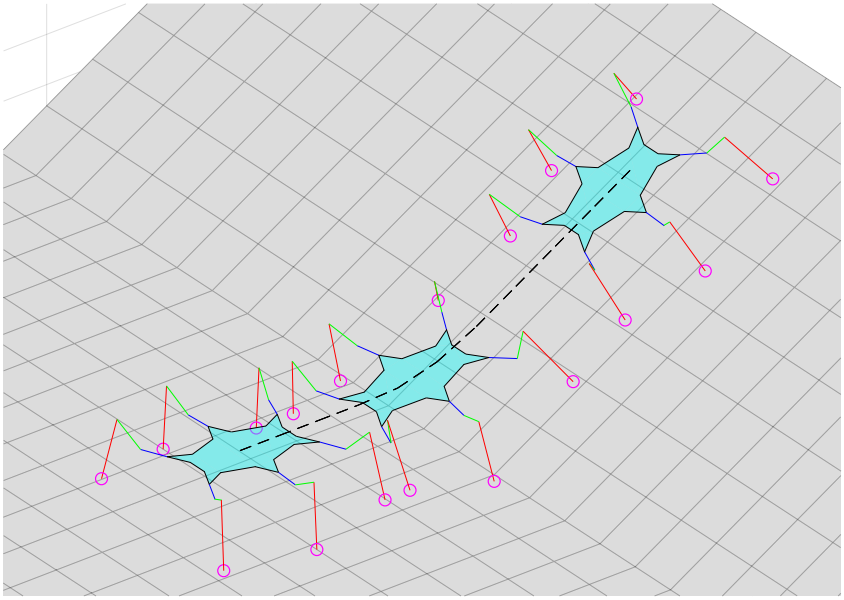


Figure 7. Robot continuous adaptation to interface and ramp.

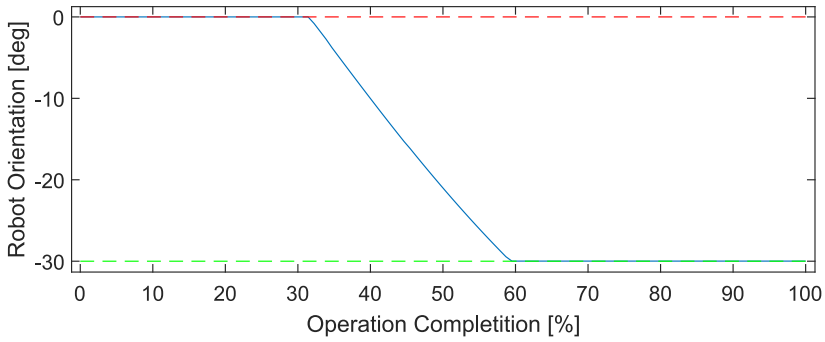


Figure 8. Angle variation in robot's inclination during ramp interface. Note how it is a continuous distribution until it matches the ramp's angulation.

4. Torque Estimation

4.1. Dynamic Model

Smart servos generally employed in physical hexapods can provide feedback in both position and speed; however, these signals are almost completely insensitive to terrain interaction effects. In particular, they are blind to sliding and instability issues. Moreover, they might not be affected by trips or grip losses. Even in overstepping cases, the robot will rigidly fall while reporting to be working perfectly.

For this reason, to intercept terrain contact, another signal must be taken into account. Torque is a value that is highly dependent on which legs support the body or not, as well as reacting swiftly to terrain interaction. Servomotor torque can, therefore, become our way to make the robot inspect its surroundings, and by compensating unstable-pose scenarios by repositioning the legs correctly, we can ensure robust movement through the entire control operation.

However, to correctly interpret the torque values coming from servos, developing a full dynamic model of the robot's movement is necessary, as presented in the following section.

The degrees of freedom of the robot's body are defined as in (28):

$$\mathcal{X}_b = \begin{bmatrix} x_b \\ y_b \\ z_b \\ rot_x \\ rot_y \\ rot_z \end{bmatrix} \tag{28}$$

where the DOFs follow the order set in (8).

The degrees of freedom of a single leg are defined as the state vector shown in (29) with consistency with the previous definitions of θ, ϕ, and ψ:

$$\mathcal{X}_{a,l} = \begin{bmatrix} \theta_i \\ \phi_i \\ \psi_i \end{bmatrix} \tag{29}$$

The state vector comprising all legs' degrees of freedom is defined as $\mathcal{X}_a$, shown in (30):

$$\mathcal{X}_a = \begin{bmatrix} \mathcal{X}_{a,1} \\ \vdots \\ \mathcal{X}_{a,6} \end{bmatrix} \tag{30}$$

The main problem with the modeling of the hexapod robot and legged locomotion, in general, is that to use lightweight dynamic algorithms like Newton–Euler equations, it is an absolute necessity to have one and only one grounded joint at all times [44–46]. Legged locomotion usually does not fall under these requirements, and multiple ground constraints must be addressed, adding great complexity to the model.

In order to avoid this kind of problem, it is possible to take an alternate route for modeling: instead of considering the robot as positionally constrained at the ground at the pushing legs endpoints, we consider the robot as not having any constraints at all and adjoining the kinematic constraints to the Lagrangian dynamic equations.

Since the ground contact points act as hinges for the robot, the kinematic constraints will be the nullity of these points' linear velocities.

The coxa, femur, and tibia coordinate systems with reference to Figure 9 are identified with the matrices (31)–(33). Those are used to find the relative positions of either the coxa, femur, and tibia joints or the feet endpoint, e.g., r_{f-c} being the position of the femur joint in the coxa coordinate system:

$$T_{bodycoxa} = \begin{bmatrix} \cos(\epsilon+\theta) & -\sin(\epsilon+\theta) & 0 & x_j \\ \sin(\epsilon+\theta) & \cos(\epsilon+\theta) & 0 & y_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{31}$$

$$T_{coxafemur} = \begin{bmatrix} \cos\phi & 0 & \sin\phi & l_c \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{32}$$

$$T_{femurtibia} = \begin{bmatrix} \cos\psi & 0 & \sin\psi & l_f \\ 0 & 1 & 0 & 0 \\ -\sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{33}$$

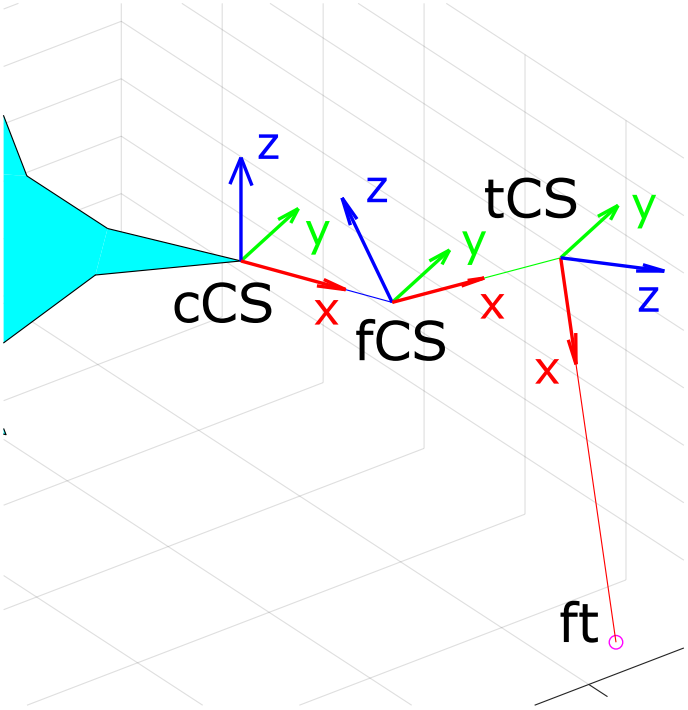


Figure 9. Zoom on one leg of the robot highlighting coxa, femur, and tibia coordinate systems.

By defining T_{trbody} as in (34), the transformation matrix that describes the effects of the body DOF on its pose, the axes of rotation of the coxa, femur, and tibia joints can be found under a fixed coordinate system as shown in (35)–(37), and are those represented in Figure 10:

$$T_{trbody} = Mov\ RotZ\ RotY\ RotX \tag{34}$$

$$A_c = T_{trbody}\ T_{bodycoxa}\ A_z \tag{35}$$

$$A_f = T_{trbody}\ T_{bodyfemur}\ A_y \tag{36}$$

$$A_t = T_{trbody}\ T_{bodytibia}\ A_y \tag{37}$$

where A_z is the $[0, 0, 1, 0]$ axis and A_y is the $[0, 1, 0, 0]$ axis.

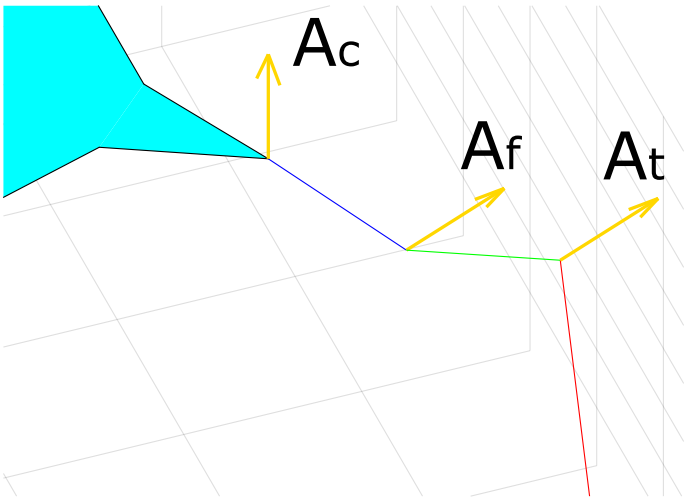


Figure 10. Axes of rotations of actuated joints.

From this definition, we can define the kinematic constraint for pushing the legs' grounded endpoints as (38), where the derivation of (39) is trivial, which shows the relation between the velocities of the actuators and the robot's body DOF:

$$A_{c,l}\,\dot{\mathcal{X}}_b + B_{c,l}\,\dot{\mathcal{X}}_{a,l} = 0 \tag{38}$$

$$\dot{\mathcal{X}}_{a,l} = -B_{c,l}^{-1}\,A_{c,l}\,\dot{\mathcal{X}}_b \tag{39}$$

The $B_{c,l}$ and $B_{c,l}$ Jacobian constraint matrices' definition is reported in (40) and (41):

$$B_{c,l} = \left[\ -\hat{r}_{ft-c}\,A_c \ \middle|\ -\hat{r}_{ft-f}\,A_f \ \middle|\ -\hat{r}_{ft-t}\,A_t\ \right] \tag{40}$$

$$A_{c,l} = \left[\ (RotZ\,RotY\,RotX) \ \middle|\ -\hat{r}_{f-b}(RotZ\,RotY\,RotX)\ \right]\Psi \tag{41}$$

In which Ψ is the matrix that transforms the time derivatives of the robot's body DOF into the robot's body twist ('twist' being the name given by [46] of the kinematic screw, and $\hat{r}$ is the skew matrix representation of the r position vector. The kinematic screw is the velocity vector field of dimension 6×1 composed of the linear velocities and the angular velocities).

A single constraint equation for all degrees of freedom of the robot requires applying the definitions provided by (42) and (43), which results in the expression (44):

$$A_c = \begin{bmatrix} ip_1\,A_{c,1} \\ \vdots \\ ip_6\,A_{c,6} \end{bmatrix} \tag{42}$$

$$B_c = \begin{bmatrix} ip_1\,B_{c,1} & & \\ & \ddots & \\ & & ip_6\,B_{c,6} \end{bmatrix} \tag{43}$$

$$A_c\,\dot{\mathcal{X}}_b + B_c\,\dot{\mathcal{X}}_a = 0 \tag{44}$$

where ip_i is a boolean value that accounts for whether the ith leg is in a pushing, constrained state (and therefore its endpoint is grounded) or in a swinging state.

Since the kinematic constraint Equation (44) was written with reference to both $\mathcal{X}_a$ and $\mathcal{X}_b$, the Lagrange equations will consider the full state vector as in (45):

$$\mathcal{X} = \left[\frac{\mathcal{X}_a}{\mathcal{X}_b}\right] \tag{45}$$

By defining the kinematic constraint equation, the dynamic model is expressed as (46):

$$\tau = \tau_a + \left(-A_c^{+}\,B_c\right)^T \tau_b \tag{46}$$

where τ_a is the contributions coming from the real actuated joints $\mathcal{X}_a$ and τ_b is the ones coming from the body DOF $\mathcal{X}_b$. A_c^{+} is the left pseudoinverse of the A_c matrix.

By defining the inertia matrix as in (47) and the potential energies as (48) [46], the two contributions can be calculated as (49) and (50):

$$M(\mathcal{X}) = \sum_{l=1}^{6}\left(J_{c,l}^{T}\, M_c\, J_{c,l} + J_{f,l}^{T}\, M_f\, J_{f,l} + J_{t,l}^{T}\, M_t\, J_{t,l}\right) + \\ + J_b^T\, M_b\, J_b = \begin{bmatrix} M_a(\mathcal{X}) & \\ & M_b(\mathcal{X}) \end{bmatrix} \tag{47}$$

$$U = -\begin{bmatrix} g^T & 0 \end{bmatrix} \sum_{l=1}^{6} (m_c\, T_{trbody}\, T_{bodycoxa,l}\, r_{g-c} + \\ + m_f\, T_{trbody}\, T_{bodyfemur,l}\, r_{g-f} + \\ + m_t\, T_{trbody}\, T_{bodytibia,l}\, r_{g-t}) + \\ - \begin{bmatrix} g^T & 0 \end{bmatrix} m_b\, T_{trbody}\, r_{g-b} \tag{48}$$

$$\tau_a = M_a \ddot{\mathcal{X}} + \left(\left(\mathcal{X}^T \otimes I_{18}\right)\frac{\partial M_a}{\partial \mathcal{X}} + \\ -\frac{1}{2}\left(I_{18} \otimes \dot{\mathcal{X}}^T\right)\frac{\partial M}{\partial \mathcal{X}_a}\right)\dot{\mathcal{X}} + \left(\frac{\partial U}{\partial \mathcal{X}_a}\right)^T \tag{49}$$

$$\tau_b = M_b \ddot{\mathcal{X}} + \left(\left(\mathcal{X}^T \otimes I_6\right)\frac{\partial M_b}{\partial \mathcal{X}} + \\ -\frac{1}{2}\left(I_6 \otimes \dot{\mathcal{X}}^T\right)\frac{\partial M}{\partial \mathcal{X}_b}\right)\dot{\mathcal{X}} + \left(\frac{\partial U}{\partial \mathcal{X}_b}\right)^T \tag{50}$$

In which $\otimes$ is the Kronecker product following notation by [47]. Note that since $\mathcal{X}_a$ is measurable and $\mathcal{X}_b$ can be estimated through $Q_{body,i}$ calculated through (14), the full-state-vector knowledge is assumed at all times.

4.2. Dynamic Model Validation

The dynamic model was developed through various simplifying assumptions. Hence, its validity is subordinate to the fact that those assumptions and simplifications are small enough to be neglected from a real application perspective. Following other works that deal with crawler robot physics [48], the dynamic model validation is carried out through a direct comparison of results with a Matlab SimMechanics™ simulation.

The Matlab SimMechanics™ simulation employs a physics engine and uses a full 3D model of the hexapod robot, as well as fully simulating friction and terrain physics through the Simscape Multibody Contact Forces Library [49] (the foot–terrain interaction system is shown in Figure 11).

The comparisons shown in Figure 12 show that the dynamic model closely matches the result from the SimMechanics™ physics engine. While there are differences in the torque spikes coming from the simulated friction model physics, the overall torque behavior perceived by the SimMechanics™ physics engine closely matches the one generated by our dynamic algorithm. This not only means that our algorithm is indeed accurate enough to provide good results, but the assumptions we made while developing the dynamic model, such as the way we modeled the grounded endpoints through kinematic constraints, mean that the PKM interpretation of the hexapod movement and the simplification of the hexapod feet providing single-point contact between the leg and terrain is small enough not to cause mismodeling errors in the final calculations.

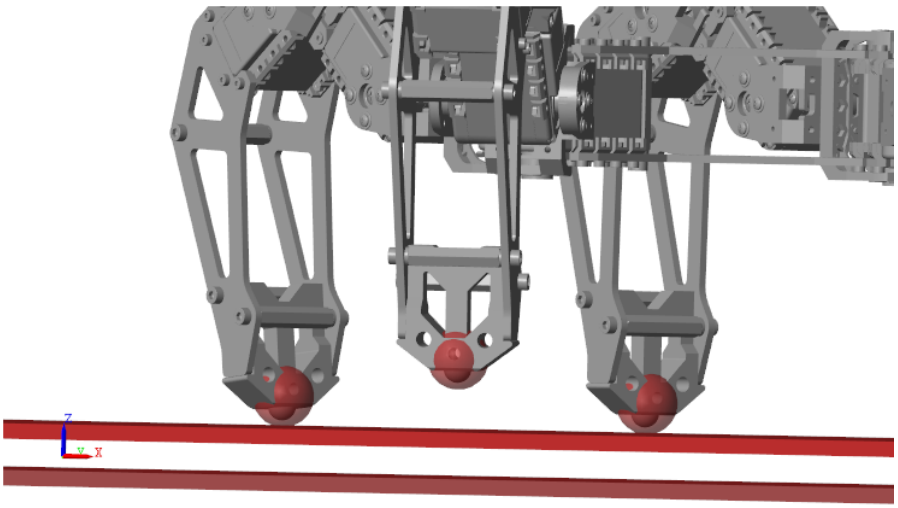


Figure 11. Graphical rendition of the plane–spheres interaction through the Simscape Multibody Contact Forces Library.

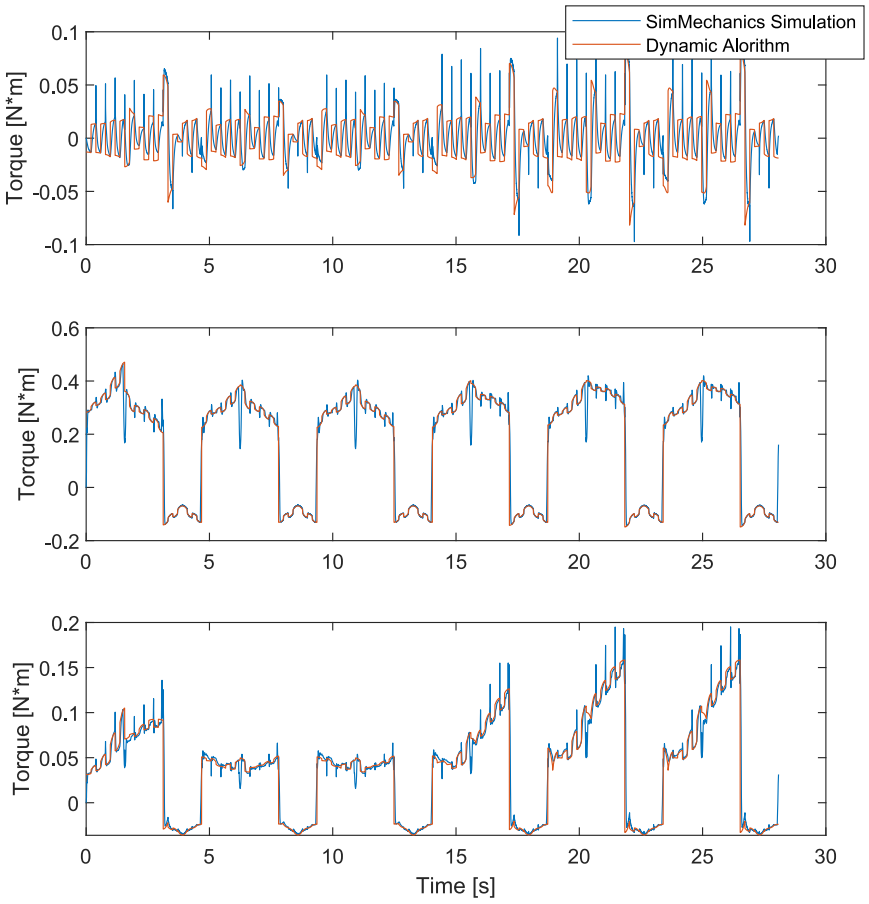


Figure 12. Comparisonsof dynamic results of a simple walk task from SimMechanics^{TM} simulation and the dynamic model. Graphs refer to the right front leg's *coxa*, *femur*, and *tibia* actuators, respectively.

5. Closed-Loop Control

A closed-loop-control approach is essential to obtain stable, robust movement throughout the entire range of movement tasks that the robot will be subjected to. After a few steps, a completely feed-forward approach may lead to positioning errors related mainly to robot–interface mismatching between reality and expected values, which would translate to grip losses and other unwanted behaviors to avoid in the presence of uneven terrain.

The two major situations to avoid are the cases where the feed-forward locomotion-control system expects the terrain to be higher and, therefore, leaves the leg hanging. The case where it expects it to be lower and, therefore, pushes the body back in an overstepping action destabilizes the entire robot pose. These situations are bound to happen due to either terrain mismodeling or desynchronization between the expected behavior from the simulation environment and the actual robot motion, which is mainly due to the increasing drift between the real and expected position of the robot as it accumulates positioning errors due to all non-ideal behaviors and performances coming from its hardware and components. Note how both situations happen in the leg-lowering part of the swing phase *only*, meaning a feedback control algorithm acting on real ground touching just needs to be called during that movement section.

The feedback control system works by comparing the expected value of the actuators' torques coming from the dynamic model with the actual sensed torques coming from the sensors system.

By reading the actuator position and speed and estimating the body movement from $Q_{body,i}$, it is possible at each iteration to build the estimated $\mathcal{X}_a$ and $\mathcal{X}_b$ state vectors, to which the algorithm immediately calculates the τ_a and tau_b components as defined in (49) and (50), with these being the only actual dependencies.

Once these two values are stored in memory, the lowering legs' sudden terrain touch can be simulated in the dynamic model by simply interacting with the ip_i values defined earlier in (42) and (43). The ip_i boolean represents the grounded state of the ith leg's endpoint; therefore, by temporarily setting it to one and building the A_c and B_c constraint matrices, it is possible to obtain the hypothetical terrain-reached torques for all actuators through (46). This is an easy, relatively CPU-light way to obtain full references for the leg–terrain touch situations.

If there is a sensible correspondence between the expected torques and the sensed ones, then the terrain is considered reached for that particular leg, and its movements are stopped. This accounts for the overstepping part of the problem, while it may still be possible for the legs to reach their desired end positions when no terrain has been sensed.

In order to solve the 'leg-hanging' problem, it is only necessary to lower the expected altitude of the desired leg endpoint position (in the employed algorithm, the gait altitude instruction is lowered by 1 cm each time), $z_{gait,i}$, and recompute the needed servomotor angles. This can be obtained by lowering the legs until the terrain is touched and the previous condition is met, effectively assuring that the legs reach the ground and stop right before moving to the *push* phase.

Finally, since the servomotor stopping condition is changed to sensed terrain touch, the lowering leg endpoints will probably differ from the ones expected by the locomotion system instructions. Therefore, the new positions need to be updated by employing forward kinematics tools such as the $T_{bodycoxa}$, $T_{bodyfemur}$, and $T_{bodytibia}$ transformation matrices derived through (31)–(33).

A scheme summarizing the leg-handling system is shown in Figure 13. An implementation example of the closed-loop-control solution is presented in Algorithm 1.

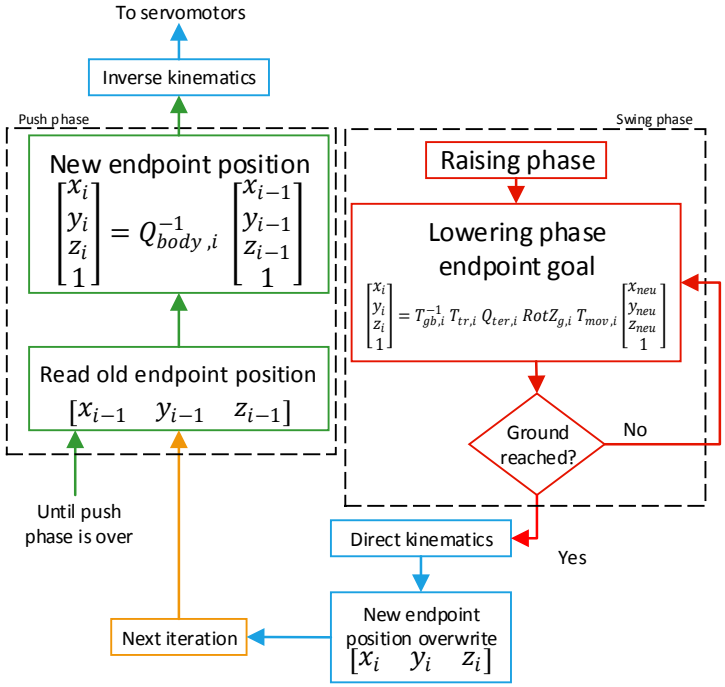


Figure 13. Scheme of the feedback control algorithm interaction with *push* and *swing* phases.

Algorithm 1: Lowering-leg-control algorithm pseudocode.

```
tau_a, tau_b = dynamic_model(X_a, X_b, DynValues, TrMatrices);
torques = read_motor_torques();
while lowering_legs is not empty do
    for leg in lowering_legs do
        ip[leg] = 1;
        A_c, B_c = constr_matr(TrMatrices, ip);
        tau = tau_a + (-pinv(A_c)*B_c).T * tau_b;
        if 0.9*corrispondence(tau, torques) == 1 then
            stop_motors(leg);
            theta, phi, psi = read_motor_angles(leg);
            endp_pos[leg] = FK(theta, phi, psi, leg);
            del leg in lowering_legs
        else if is_motors_stop(leg) then
            endp_pos[leg].z = endp_pos[leg].z - 10;
        else
            pass;
    end
end
```

6. Experimental Results

The developed motion-control architecture was experimentally verified by accomplishing a series of testing scenarios. The experimental tests aim to stress two essential aspects of the architecture presented and highlight the improvements with respect to a stock controller. They also aim to analyze the effect of feed-forward locomotion control and the feedback-stabilization contribution. In particular, the two scenarios reported are as follows:

- *Obstacle recognition*: Where the robot is fed with flat terrain data and it needs to adapt to the presence of software 'invisible' terrain. The task will be considered completed if the robot can remain stable due to overstepping into obstacles and tripping, completing the traversal correctly.

- *Ramp climbing*: Where the robot is fed with data of the ramp terrain while the body is asked to remain horizontal throughout the full movement. The task will be considered completed if the robot can handle the angular interface, climb the ramp effectively, and maintain the body horizontally at all times. The difficulty comes from the fact that even a tiny deviation in the trajectory direction, which is highly possible since no positional feedback is enforced in the robotic platform, can cause significant errors between the actual and expected position of the terrain.

Moreover, in order to analyze the behavior of the overall architecture further, two additional scenarios were introduced to highlight the system's behavior. These can be considered as variations of the two previous scenarios and are presented to further demonstrate the system's ability to adjust the orientation of its base according to the terrain's slope in the first case and its capability to handle the complete lack of reliable terrain mapping in the second case (simulated by the presence of multiple obstacles).

6.1. Obstacle Recognition

The feedback control algorithm adds to the robot's walking-motion robustness by allowing it to sense and react to unexpected obstacles in the workspace. This means that when facing a mismodeled obstacle, like a section of the terrain wrongly modeled as flat and instead being at a higher altitude, the feedback logic is expected to compensate for the feed-forward error and update the gait to maintain movement stability.

In order to reproduce this condition, a solid object was placed within the robot's walking path. The obstacle considered has a parallelepiped shape (a rigid book). In our tests, the obstacle is approached vertically, and so the static coefficient between rubber and cardboard (that could be estimated at approximately [0.5–0.8] [50]) is not considered. Moreover, its surface is approximated to be undamped and rigid. The obstacle and robot setup schematics are shown in Figure 14.

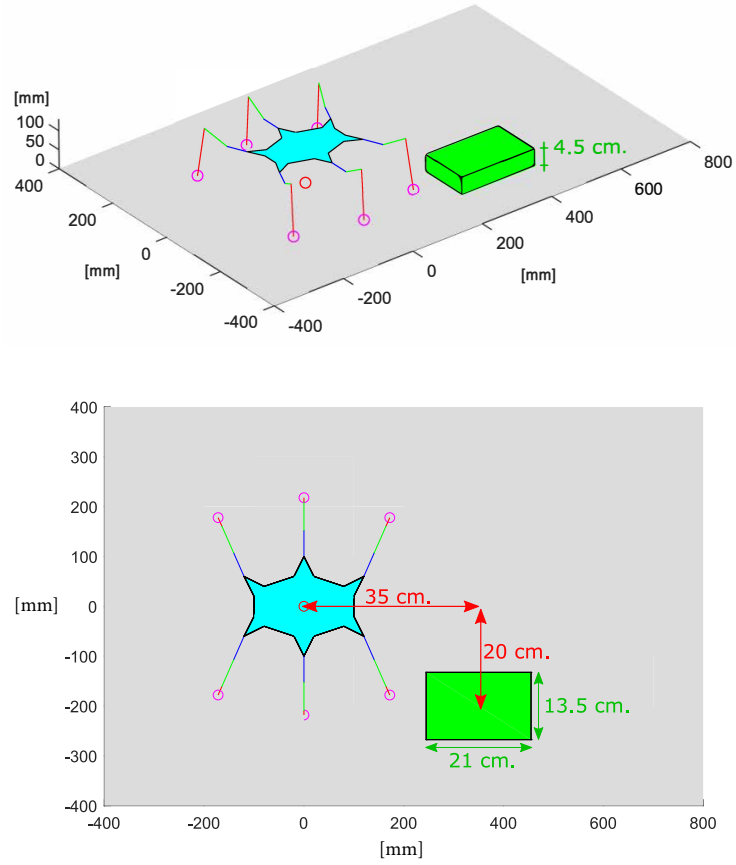
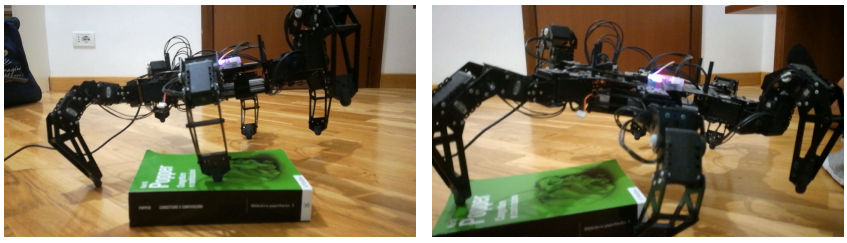


Figure 14. The obstacle course setup dimensions and position.

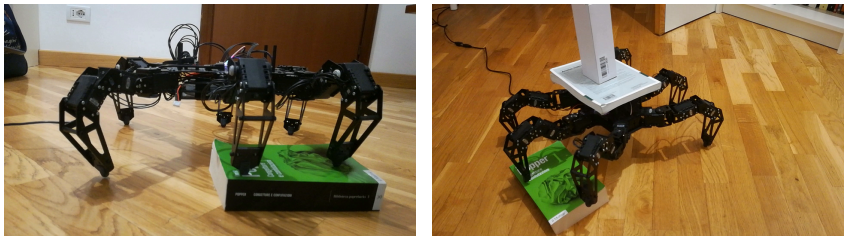
As shown in Figure 15a, while the stock controller puts the robot in an unstable stride, the novel closed-loop solution is indeed able to maintain the body horizontally and, by not overstepping onto the solid object, all legs hold to the ground, assuring equilibrium (Figure 15b). These aspects are further highlighted when a payload is mounted on top of it, showing how the stability of movement transfers to the stability of the robot's body. The robustness of the walking stride made it possible to completely disregard the presence of the obstacle in the robot's path.

A comparison of the front right and middle right femur servo angle variation through the entire operation between NUKE and the feedback control logic is shown in Figure 16. It is evident how, thanks to the feedback control algorithm, in the middle of the stride cycle, the legs found the obstacle, which resulted in the respective femur actuator stopping in its tracks.

In Figure 15a, it is also possible to check the 'blindness' of the robot while being moved by the stock controller: despite being in an unstable position, the angular feedback of the servomotors still reports no extraordinary values, and therefore the robot still attempts to walk as if no obstacle was in its path in the first place.



(**a**)



(**b**)

Figure 15. Physical applications of obstacle-recognition capability. On top are shown performances of the NUKE controller; at the bottom, the closed-loop control logic is used instead. (**a**) Stock NUKE controller instabilities; (**b**) feedback-control-logic-aided walking.

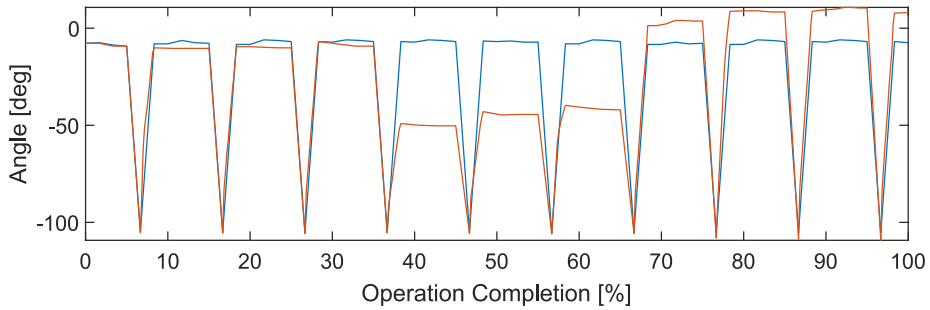


Figure 16. Right middlefemur actuator angle comparison between operations. In blue, the operation employs the NUKE controller; in red, the feedback control algorithm is used instead.

6.2. Ramp Climbing

In the ramp-climbing task, the robot is positioned right in front of the angular interface, with the instruction of a forward constant movement. The setup scheme is presented in

Figure 17. In order to request the robot's body to remain horizontal, the Q_{ter} matrix is built via (51) rather than (19):

$$Q_{ter,i} = D_{ter,i} \tag{51}$$

As shown in Figure 17, the ramp inclination angle considered is small so that the static friction coefficient between rubber and plexiglass (that could be estimated approximately around [0.8] [50]) is neglected and so that the slipping effect between the leg and terrain is not affecting the results. Moreover, the surface of the ramp is approximated to be undamped and rigid. The most challenging part of the task is the traversal of the interface between the flat plane and the ramp, as desynchronization effects are present the most in that location.

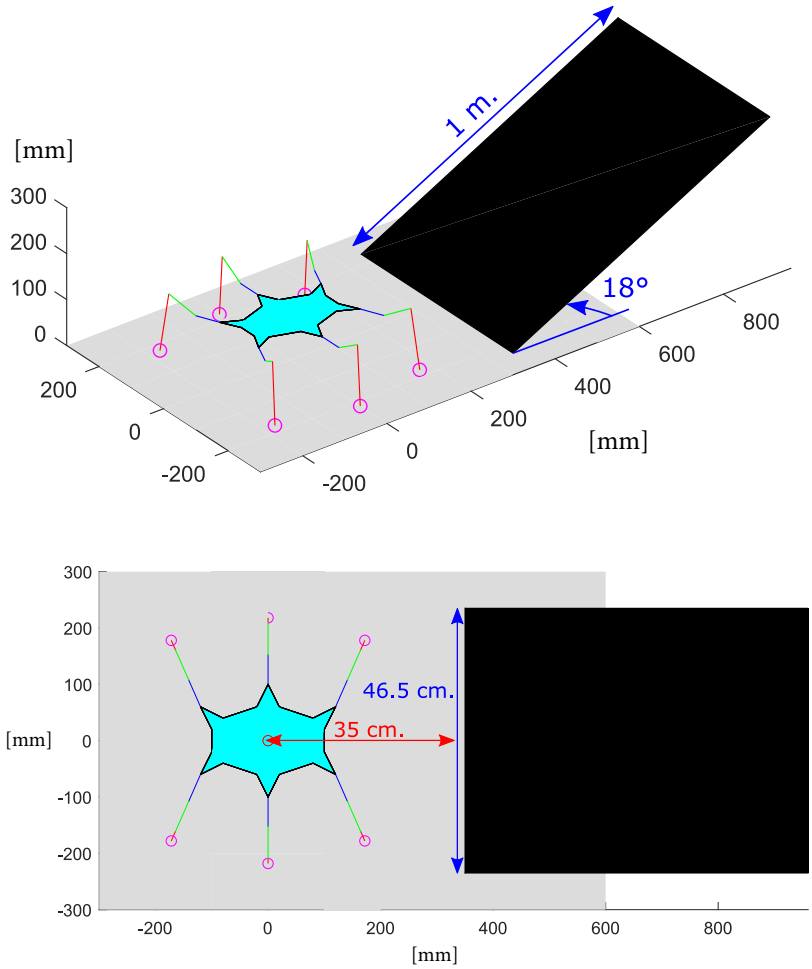


Figure 17. The ramp-climbing task setup dimensions, angulation, and position.

This is visible in Figure 18a, where when the control feedback is absent, the legs are left hanging often, leading to movement unsteadiness and ultimately increased traversal difficulty. By inducing the novel control architecture, as shown in Figure 18b, the robot is instead able to climb the ramp correctly because feedback control allows for correct contact between the legs and terrain and because the control of the body pose introduced in the proposed algorithm allows the robot to maintain a horizontal body pose regardless of the incline angle of the terrain.

(**a**)

(**b**)

Figure 18. Physical applications of the closed-loop-control architecture. On top, the feedback algorithm is absent; at the bottom, the full control logic is used instead. (**a**) Feed-forward-only controller-stability problems; (**b**) feedback control logic performance.

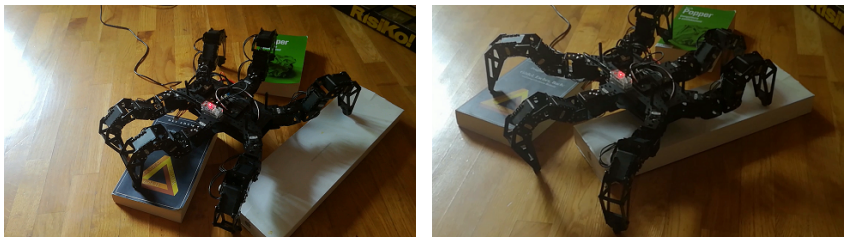
6.3. Additional Tests

In order to investigate the performance of the system better, two specific additional tests are reported. The first test is an evolution of the ramp-climbing test previously described and schematized in Figure 17. In this case, an object with predominant vertical development (a paper box) is placed on the robot. Due to its geometry, it represents an unstable equilibrium configuration that can easily fall in case of a robot misstep or an excessive tilting of its base.

The second test is developed to extend further the terrain-mapping error previously presented in the obstacle-recognition scenario. The test was carried out by instructing the robot to move forward, expecting a fully flat terrain but then littering the entire path with unknown and unforeseen obstacles of different sizes, resulting in a completely unknown terrain map. Figure 19 collects some photos representing the robot while correctly completing the mission. In detail, for the first additional scenario, the frames show the horizontal position of the robot's body and the obstacle in balance before approaching and on the ramp. The frames of the second additional scenario show the robot during its motion on an unknown terrain. It is possible to notice that all legs are properly touching the terrain as expected.

(**a**)

Figure 19. *Cont.*



(**b**)

Figure 19. Tests designed to show the control algorithm's capabilities and performance. On top, an oblong object is put on an unstable base on top of the robot, ready to fall should the robot misstep or fail to achieve base flatness; at the bottom, the robot is sent moving in an unknown environment (algorithm expects full flatness) and it needs to avoid missteps by reacting swiftly to unexpected torque changes. (**a**) Inclination control stress test to check for base flatness during ramp climbing; (**b**) unknown, obstacle-filled environment stress test to check for responsiveness in adapting to unknown terrain with varying features.

7. Assumptions and Limitations

In the presented study, the terrain was assumed to be rigid and non-deformable. Additionally, we neglected friction effects during walking, thus neglecting potential slippages that may occur. Furthermore, the current architecture lacks a localization and feedback control that considers the robot's position on the terrain map, resulting in an open-loop trajectory execution. Additionally, the map definition is simplified at this stage, being represented merely as a set of elevations (z) in the 2D (x, y) plane, treated as known data.

8. Conclusions

This paper presents a simple but effective kinematic model derived from a standard open-source formulation (NUKE), a widely recognized and utilized model in the field of robotics. The original formulation is further improved through the manipulation of hexapod leg endpoints: through the use of transformation and pose matrices, the legs' position in space and the robot's pose can be accessed at any time. Furthermore, a comprehensive architecture is developed to handle terrain irregularities. This architecture is designed to adjust the robot's body pose, ensuring its isolation from rough terrain imperfections without the addition of any sensors to the original hardware configuration. For this purpose, a dynamic but computationally lightweight model is presented with the aim of accurately estimating servomotor torques. Moreover, a terrain-sensing feedback control algorithm is introduced to correct instability situations resulting from measurement and actuation errors. The algorithm guarantees leg–ground reach based on a feedback architecture based on estimated torque (from the model) and the torque provided by servomotors as inputs. The proposed architecture, composed of a terrain-adapting, torque estimation, and terrain-sensing algorithm, was evaluated in terms of applicability and performance through experimental tests conducted on the PhantomX Mark II robotic platform. The experimental results presented confirm the expected behavior of the system in different scenarios of interest, such as ramp climbing and the presence of unknown obstacles on the ground. The proposed control architecture is based on several assumptions (such as the knowledge of the terrain shape by other systems or the assumption that the terrain is undeformable). Relaxing these assumptions could be an interesting research direction for future developments.

Author Contributions: Conceptualization, M.Z. and S.A.; methodology, M.Z.; software, M.Z.; resources, F.B. and G.B.; writing—original draft preparation, M.Z.; writing—review and editing, S.A. and G.B.; supervision, S.A.; project administration, F.B.; funding acquisition, F.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data that could be shared for research purposes were created.

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

References

1. Rubio, F.; Valero, F.; Llopis-Albert, C. A review of mobile robots: Concepts, methods, theoretical framework, and applications. *Int. J. Adv. Robot. Syst.* **2019**, *16*, 1729881419839596. [CrossRef]
2. Coelho, J.; Ribeiro, F.; Dias, B.; Lopes, G.; Flores, P. Trends in the Control of Hexapod Robots: A Survey. *Robotics* **2021**, *10*, 100. [CrossRef]
3. Bruzzone, L.; Nodehi, S.E.; Fanghella, P. Tracked Locomotion Systems for Ground Mobile Robots: A Review. *Machines* **2022**, *10* 648. [CrossRef]
4. Swaminathan, A.; Surendran, R.; Joel Benjamin, J.; Jaswant, V. Design and Development of Light Weight and Low-Cost Quadruped Robot for Spying and Surveillance. In Proceedings of the 2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), Sakheer, Bahrain, 20–21 November 2022; pp. 500–504. [CrossRef]
5. Luneckas, M.; Luneckas, T.; Udris, D.; Plonis, D.; Maskeliunas, R.; Damasevicius, R. Energy-efficient walking over irregular terrain: A case of hexapod robot. *Metrol. Meas. Syst.* **2019**, *26*, 645–660. [CrossRef]
6. Deng, H.; Xin, G.; Zhong, G.; Mistry, M. Gait and trajectory rolling planning and control of hexapod robots for disaster rescue applications. *Robot. Auton. Syst.* **2017**, *95*, 13–24. [CrossRef]
7. Chen, G.; Han, Y.; Li, Y.; Shen, J.; Tu, J.; Yu, Z.; Zhang, J.; Cheng, H.; Zhu, L.; Dong, F. Autonomous gait switching method and experiments of a hexapod walking robot for Mars environment with multiple terrains. *Intell. Serv. Robot.* **2024**, *17*, 533–553. [CrossRef]
8. Yang, K.; Liu, X.; Liu, C.; Wang, Z. Motion-Control Strategy for a Heavy-Duty Transport Hexapod Robot on Rugged Agricultural Terrains. *Agriculture* **2023**, *13*, 2131. [CrossRef]
9. Sanz-Merodio, D.; Garcia, E.; Gonzalez-de-Santos, P. Analyzing energy-efficient configurations in hexapod robots for demining applications. *Ind. Robot* **2012**, *39*, 357–364. [CrossRef]
10. Agheli, M.; Qu, L.; Nestinger, S.S. SHeRo: Scalable hexapod robot for maintenance, repair, and operations. *Robot. Comput.-Integr. Manuf.* **2014**, *30*, 478–488. [CrossRef]
11. Sato, Y.; Kakuto, T.; Tanaka, T.; Shimano, H.; Morohashi, Y.; Hatakeyama, T.; Nakajima, J.; Ishiyama, M. Development of a radioactive substance detection system integrating a Compton camera and a LiDAR camera with a hexapod robot. *Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrometers Detect. Assoc. Equip.* **2024**, *1063*, 169300. [CrossRef]
12. Olivier A., K.Y.; Biradar, R.C.; Karthik, R.; Devanagavi, G.D. Design of a Robot for carrying out research on hybrid robot's mobility: Case of a mecanum wheel-legged robot. In Proceedings of the 2023 Fifth International Conference on Electrical, Computer and Communication Technologies (ICECCT), Erode, India, 22–24 February 2023; pp. 1–8. [CrossRef]
13. Fu, Z.; Xu, H.; Li, Y.; Guo, W. Design of A Novel Wheel-Legged Robot with Rim Shape Changeable Wheels. *Chin. J. Mech. Eng.* **2023**, *36*, 153. [CrossRef]
14. Žák, M.; Rozman, J.; Zbořil, F.V. Energy Efficiency of a Wheeled Bio-Inspired Hexapod Walking Robot in Sloping Terrain. *Robotics* **2023**, *12*, 42. [CrossRef]
15. Čížek, P.; Zoula, M.; Faigl, J. Design, Construction, and Rough-Terrain Locomotion Control of Novel Hexapod Walking Robot with Four Degrees of Freedom Per Leg. *IEEE Access* **2021**, *9*, 17866–17881. [CrossRef]
16. Walas, K. Terrain Classification and Negotiation with a Walking Robot. *J. Intell. Robot Syst.* **2015**, *78*, 401–423. [CrossRef]
17. Zenker, S.; Aksoy, E.E.; Goldschmidt, D.; Wörgötter, F.; Manoonpong, P. Visual terrain classification for selecting energy efficient gaits of a hexapod robot. In Proceedings of the 2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Wollongong, NSW, Australia, 9–12 July 2013; pp. 577–584. [CrossRef]
18. Cruz Ulloa, C.; Sánchez, L.; Del Cerro, J.; Barrientos, A. Deep Learning Vision System for Quadruped Robot Gait Pattern Regulation. *Biomimetics* **2023**, *8*, 289. [CrossRef] [PubMed]
19. Cruz Ulloa, C.; Álvarez, J.; del Cerro, J.; Barrientos, A. Vision-based collaborative robots for exploration in uneven terrains. *Mechatronics* **2024**, *100*, 103184. [CrossRef]
20. Cruz Ulloa, C.; del Cerro, J.; Barrientos, A. Mixed-reality for quadruped-robotic guidance in SAR tasks. *J. Comput. Des. Eng.* **2023**, *10*, 1479–1489. [CrossRef]
21. Walas, K. Tactile Sensing for Ground Classification. *J. Autom. Mob. Robot. Intell. Syst.* **2013**, *7*, 18–23.
22. Hu, N.; Li, S.; Zhu, Y.; Gao, F. Constrained Model Predictive Control for a Hexapod Robot Walking on Irregular Terrain. *J. Intell. Robot. Syst.* **2018**, *94*, 179–201. [CrossRef]
23. Maiti, T.; Ochi, Y.; Navarro, D.; Miura-Mattausch, M.; Mattausch, H. Walking robot movement on non-smooth surface controlled by pressure sensor. *Adv. Mater. Lett.* **2018**, *9*, 123–127. [CrossRef]

24. Mattausch, H.J.; Luo, A.; Bhattacharya, S.; Dutta, S.; Maiti, T.K.; Miura-Mattausch, M. Force-Sensor-Based Walking-Environment Recognition of Biped Robots. In Proceedings of the 2020 International Symposium on Devices, Circuits and Systems (ISDCS), Howrah, India, 4–6 March 2020; pp. 1–4. [CrossRef]
25. Coelho, J.; Dias, B.; Lopes, G.; Ribeiro, F.; Flores, P. Development and implementation of a new approach for posture control of a hexapod robot to walk in irregular terrains. *Robotica* **2024**, *42*, 792–816. [CrossRef]
26. López-Osorio, P.; Domínguez-Morales, J.P.; Perez-Peña, F. A Neuromorphic Vision and Feedback Sensor Fusion Based on Spiking Neural Networks for Real-Time Robot Adaption. *Adv. Intell. Syst.* **2024**, *6*, 2300646. [CrossRef]
27. Cao, M.; Yamashita, K.; Kiyozumi, T.; Tada, Y. Hexapod Robot with Ground Reaction Force Sensor on Rough Terrain. In Proceedings of the 2021 IEEE International Conference on Mechatronics and Automation (ICMA), Takamatsu, Japan, 8–11 August 2021; pp. 1088–1093. [CrossRef]
28. Čížek, P.; Kubík, J.; Faigl, J. Online Foot-Strike Detection Using Inertial Measurements for Multi-Legged Walking Robots. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 7622–7627. [CrossRef]
29. Faigl, J.; Čížek, P. Adaptive locomotion control of hexapod walking robot for traversing rough terrains with position feedback only. *Robot. Auton. Syst.* **2019**, *116*, 136–147. [CrossRef]
30. Li, D.; Wei, W.; Qiu, Z. Combined Reinforcement Learning and CPG Algorithm to Generate Terrain-Adaptive Gait of Hexapod Robots. *Actuators* **2023**, *12*, 157. [CrossRef]
31. Wang, L.; Li, R.; Huangfu, Z.; Feng, Y.; Chen, Y. A Soft Actor-Critic Approach for a Blind Walking Hexapod Robot with Obstacle Avoidance. *Actuators* **2023**, *12*, 393. [CrossRef]
32. Boston Dynamics Website. Available online: https://bostondynamics.com (accessed on 22 May 2024).
33. Robotis Website. Available online: https://www.robotis.us (accessed on 22 May 2024).
34. TrossenRobotics Website. Available online: https://www.trossenrobotics.com (accessed on 22 May 2024).
35. Arbotix NUKE Code. Available online: https://github.com/vanadiumlabs/pypose (accessed on 22 May 2024).
36. Macenski, S.; Foote, T.; Gerkey, B.; Lalancette, C.; Woodall, W. Robot Operating System 2: Design, architecture, and uses in the wild. *Sci. Robot.* **2022**, *7*, eabm6074. [CrossRef]
37. Ardupilot Website. Available online: https://ardupilot.org/ (accessed on 22 May 2024).
38. Tedeschi, F.; Carbone, G. Design Issues for Hexapod Walking Robots. *Robotics* **2014**, *3*, 181–206. [CrossRef]
39. Hexapod Webpage. Available online: https://www.interbotix.com/Robotic-Hexapod (accessed on 22 May 2024).
40. Dynamixel Webpage. Available online: https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/ (accessed on 22 May 2024).
41. Belter, D.; Skrzypczynski, P. Rough Terrain Mapping and Classification for Foothold Selection in a Walking Robot. *J. Field Robot.* **2011**, *28*, 497–528. [CrossRef]
42. Stengel, R. *Optimal Control and Estimation*; Dover: New York, NY, USA, 1994.
43. Hirt, C. Digital Terrain Models. In *Encyclopedia of Geodesy*; Springer: Cham, Switzerland, 2016. [CrossRef]
44. Featherstone, R. *Rigid Body Dynamics Algorithms*; Springer: New York, NY, USA, 2008. [CrossRef]
45. Legnani, G. *Robotica Industriale*; Casa Editrice Ambrosiana: Milan, Italy, 2003.
46. Briot, S.; Khalil, W. *Dynamics of Parallel Robots*; Springer: Cham, Switzerland, 2015; Volume 35. [CrossRef]
47. Taghirad, H. *Parallel Robots: Mechanics and Control*; CRC Press: Boca Raton, FL, USA, 2020.
48. Beaber, S.; Zaghloul, A.; Kamel, M.; Hussein, W. Dynamic Modeling and Control of the Hexapod Robot Using Matlab SimMechanics. In Proceedings of the ASME 2018 International Mechanical Engineering Congress and Exposition, Pittsburgh, PA, USA, 9–15 November 2018; p. V04AT06A036. [CrossRef]
49. Miller, S. Simscape Multibody Contact Forces Library. 2022. Available online: https://github.com/mathworks/Simscape-Multibody-Contact-Forces-Library/releases/tag/20.1.5.1 (accessed on 25 May 2024).
50. The Engineering ToolBox (2004). Friction—Friction Coefficients and Calculator. Available online: https://www.engineeringtoolbox.com/friction-coefficients-d_778.html (accessed on 5 September 2024).