applied sciences

MDPI

Article

Behavior-Based Control Architecture for Legged-and-Climber Robots

Miguel Hernando *, Mercedes Alonso, Carlos Prados and Ernesto Gambao

Centre for Automation and Robotics (UPM-CSIC), Universidad Politécnica de Madrid, 28012 Madrid, Spain; cheripe.caa@gmail.com (M.A.); c.prados@alumnos.upm.es (C.P.); ernesto.gambao@upm.es (E.G.)
* Correspondence: miguel.hernando@upm.es

Featured Application: Control of legged-and-climber robots with at least four legs, under unforeseen failures in one or more legs.

Abstract: In this paper, we present a fully original control architecture for legged-and-climber robots that is level-based, hierarchical, and centralized. The architecture gives the robots the ability to perform self-reconfiguration after unforeseen leg failures, because it can control this kind of robot with different numbers of legs. The results show the capability of performing movements in any direction and inclination planes. The components and functionalities of the developed control architecture for these robots are described, and, the architecture's performance is tested on the ROMHEX robot.

Keywords: behavior-based; climber robot; control; control architecture; fault-tolerant; legged robot; optimization

Citation: Hernando, M.; Alonso, M.; Prados, C.; Gambao, E. Behavior-Based Control Architecture for Legged-and-Climber Robots. *Appl. Sci.* **2021**, *11*, 9547. https://doi.org/10.3390/app11209547

Academic Editors: Alessandro Gasparetto, Stefano Seriani and Lorenzo Scalera

Received: 27 July 2021
Accepted: 9 October 2021
Published: 14 October 2021

1. Introduction

Due to the increased size and complexity of civil construction, using climbing robots in infrastructure inspection is becoming increasingly relevant. Regular maintenance and surveillance of large complexes are extremely important to guarantee their life-cycle. The European consortium SPARC (euRobotics) carried out an analysis, revealing in the "European Robotics & AI workshop Applied to Inspection and Maintenance" report [1] that in the coming years the task of inspection will be increasingly important, and robots will play the main role in maintenance, inspection and dismantling.

Inspection is a particularly structured repetitive task, that requires permanent attention during the operation, and in many cases, it involves placing the human operator in risk situations. That is why robotics is revealed as a technology of direct application. The number of service robots, mainly driven by drones, has increased by 25% in recent years, and the number of autonomous vehicles in a non-manufacturing environment, has increased by 51% in one year [2].

Many solutions are based on remote visual inspection, but this is not valid for many industrial structures. We may not have direct visual access to the areas to be inspected, and in other cases it is not possible to carry a flying drone because of the narrowness or typology of the environment. In addition, an inspection usually requires much more than seeing. Non-destructive techniques require contact or proximity to the surface that is not possible to achieve while flying. Climbing robots have become increasingly attractive for effective infrastructure inspection due to their ability to overcome these limitations.

A crucial part of these robots is their control. Typically, robots are controlled based on control architectures. The lack of existing control architectures for legged-and-climber robots drives a need to design a new architecture. The state-of-the-art of control architectures for this kind of robot reveal that much work remains, because although plenty of legged-and-climber robots are structurally defined, they lack a defined architecture. Only a

few control architectures are found for climber robots. Shen et al. [3] proposed a climber robot for oil tank inspection, whose control architecture only covers the kinematic control of the robot. Several control architectures for legged robots can be found easily, such as [4] or [5], which describe the motion controller of two-legged robots. However, these kinds of legged robots are unable to climb due to their bipedal disposal. Robots with more than two legs, like [6], for example, have a control system that only covers legs management and exclude the high-level control.

A control architecture specifies the organization guidelines of a robot's behavior, establishes the action and movements that the robot must carry out to achieve a goal or a set of goals, according to the robot state. It has the main purpose of establishing a way to organize the system to maintain defined roles, modularize the system components and make the system as fault-tolerant as possible, always trying to keep the system under control over as many situations as possible. The main characteristics of the control architectures are the capacity to (a) face multiple goals simultaneously, (b) integrate data from different sensors, (c) be robust against component failures, (d) be adaptive to new environments, (e) extend and modify its content easily, (f) make its own decisions according to the robot state, and (g) modify the surroundings properly.

Some important definitions should be declared in a control architecture: (a) an agent is a computer system that is capable of autonomous action in its environment to achieve its delegated objectives [7,8], (b) a level is composed of an agent or a group of agents that have the same importance from a determined point of view, (c) hierarchy is how agents of higher levels have a kind of control over the agents of lower levels, (d) a centralized control is characterized by coordinated conduct of the agents, whose decisions depend directly on the state of the other agents, (e) a decentralized control is characterized by freedom in the agent decisions, without directly considering the state of the rest of the agents, and (f) a behavior-based control is a way in which agents are delegated with the main task to achieve a goal through some instructions. Then, it is possible to conclude that a control architecture may be understood as a multi-agent system where the communication rules and protocols are well-defined.

This paper's objective is to design an organic and hierarchical control that allows the safe movement of the legs. Moreover, the control must be generic for a legged robot with any number of legs. In this paper, we test the performance of the control architecture over the legged-and-climber robot ROMHEX. This robot has ben modified; the most remarkable is the change of the initial position of each leg that the robot should adopt to maximize the efficiency of the walking pattern. This maximization has been obtained generically, as explained in Section 3.1. Thus, the initial position directly influences the beginning of the gait, and it is also used as the default position when the robot needs to reconfigure while walking.

We propose a control architecture that fulfills the following requirements: (a) generic for all legged robots, independent of the number of legs, distribution of legs in the body, or the number of joints per leg, (b) agent-based, (c) hierarchical (that is, agents of higher levels should have the control of lower-level agents, and in this way, first agents may disable second agents), and (d) agents of the same level must synchronize their behaviors using synchronization mechanisms.

This paper is organized as follows: In Section 2, we present an overview of the state of the art of control architectures. We describe how different architectures are organized and the levels they use. In Section 3, we describe the hexapod robot used in the tests. Furthermore, we include the developed optimization of the legs' position in the body. In Section 4, we describe the developed control architecture, and we explain its levels, hierarchy and agents. In Section 5, we discuss the results obtained during the tests. Lastly, in Section 6, we detail the obtained conclusions from the results.

2. Related Work

Control architectures are found within autonomous machines, especially in spacecraft. The increasing development of these vehicles has generated generic architectures that may be used in any system. Within the state of the art, we can find Contextual Management of Tasks and Instrumentation (CMTI), which is a mixed architecture between deliberative and reactive architectures [9], originally conceived for an autonomous underwater vehicle (AUV). It is organized into three layers: global supervisory control, local supervisory control and low-level control. CMTI is a well-defined and robust architecture but laborious to implement in short projects. Based on CMTI, Contextual Task Management Architecture (COTAMA) is a control software architecture layered into two levels [10]: decisional level and executive level. The decisional level is in charge of the mission monitoring and decision making according to robot context. The executive level applies these decisions while managing all real-time aspects such as instrumentation conflicts or tasks deadlines. It contemplates the identification of faults to correct them [11]. COTAMA improves the robustness of CMTI while it enlarges the possible applications; however, like CMTI, it is very laborious to implement. Remote Agent architecture (RA) is an autonomous control system capable of closed-loop commanding of spacecraft and other complex systems [12]. It integrates three layers of functionality: a constraint-based planner/scheduler, a reactive executive, and a model identification and recovery system. RA is very useful for spacecraft; however, its usefulness is limited to tasks previously defined in detail. Intelligent Distributed Execution Architecture (IDEA) was created to duplicate RA architecture within a unified agent framework where all the layers have the same structure [12]. IDEA improves the coordination of RA; however, the drawbacks are similar.

Laboratory for Analysis and Architecture of Systems (LAAS) architecture is presented in [13] as an architecture for reflexive autonomous vehicle control. It decomposes the robot software into three main levels, having different temporal constraints and manipulating different data representations. LAAS is thought to improve robustness; however, it is poorly-defined and very open to the developer. Coupled Layer Architecture for Robotic Autonomy (CLARAty) is designed for improving the modularity of system software while more tightly coupling the interaction of autonomy controls [14]. According to the CLARAty developers, typical robot and autonomy architectures comprise three levels: functional, executive and planner. To correct the shortfalls in the three-level architecture, they propose a two-tiered architecture, in which the executive and planner layers are combined. As well as LAAS, CLARAty is very open to the user because it only describes the two main levels. Cooperative Intelligent Real-Time Control Architecture (CIRCA) was designed in [15] to guarantee the control-level goals, but not necessarily the task-level goals. They divide the architecture into three main parts: the real-time subsystem (RTS) that is responsible for implementing the responses, the AI subsystem (AIS) that decomposes task-level goals into plans consisting of several phases, and the scheduler. CIRCA has the limitations that it only works for well-known and defined problems. Open Robot Controller Computer Aided Design (ORCCAD) architecture is an open architecture where qualified users have access to every control level: the application layer is accessed to by the end-user of the system, the control layer is programmed by an automatic control expert, and the lowest one, the system layer, is overseen by a system engineer [16]. ORCCAD has the problem that the system's complexity may increase exponentially with new fault tolerance techniques, while the organization structure may become untenable. The described architectures are generic for any system; however, their application in legged-and-climber robots may be laborious and complex. Thus, the decision to develope a new architecture has been considered a better option than augmenting an existing one.

Control leaders in multiple-legged robots, such as Boston Dynamics, hide their control architectures while other researcher groups show their work. For example, in [17], Jakimovski et al. present an Organic Self-Configuration and Adapting Robot (OSCAR), a hexapod robot that is described through the Organic Robot Control Architecture (ORCA). ORCA [18] proposes creating an entire system out of subsystems, where each of the subsys-

tems is designed for a determined task [19]. More complex subsystems can be generated by combining and cascading smaller subsystems [20]. Each subsystem may be supervised by another subsystem that evaluates its performance and may even change its behavior to optimize the performance of the whole system. As another example, in [21], Pack et al. present Robotic Inspector (ROBIN), a robot designed for climbing infrastructures that uses a behavior-based control architecture arbitration by subsumption [22]. This robot is composed of two vacuum fixtures, so its architecture is completely dependant on the performance of both devices. Lastly, in [23], Ronnau et al. describe LAURON V, a legged robot controlled by its own control architecture, which is a modular and behavior-based design approach. It subdivides the system into understandable hierarchical layers and small individual behaviors. The layers are the hardware architecture, the hardware abstraction layer, and the behavior-based control system. Finally, Fankhauser et al. present Free Gait in [24] a software framework for the task-oriented control of legged robots, which they check over ANYmal [25]. Free Gait consists of a whole-body abstraction layer and several tools designed to interface higher-level motion goals with the lower-level tracking and stabilizing controllers.

Architectures for legged robots exist, but none exist for legged-and-climber robots. Furthermore, these architectures are usually conceived for a defined and not modifiable number of legs. Leg problems are possible, especially in climber robots, due to the harsh conditions they are involved in. OSCAR robots contemplate the situation of leg amputation; however, the visible face of its architecture does not allow to define clearly the behavior of a new robot.

3. The Climber Hexapod Robot ROMHEX

The ROMHEX robot is a commercial platform called *xyzrobot bolide crawler Y-01* with some modifications. The robot is a hexapod with three degrees of freedom in each leg. The reference systems of each leg according to the robot body are referred to as shown in Figure 1a, while the axes of the leg joints are illustrated in Figure 1b. Mainly, the robot is composed of an electronic board called *MCU board Y-01* and motors called *xyzrobot smart servo A1-16*.

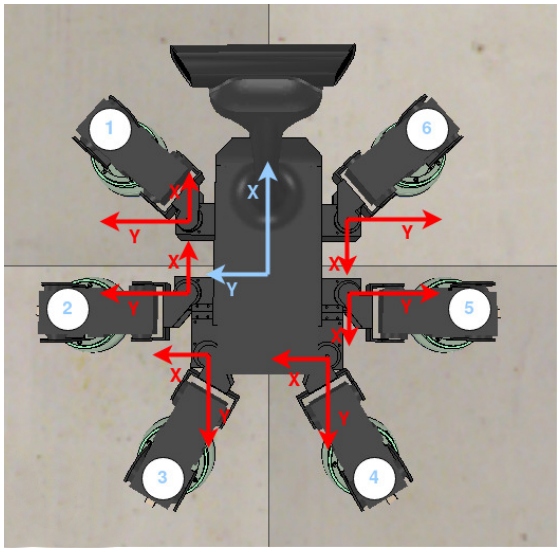
The development kit *Intel Euclid* has been added to the robot through a plastic piece that locates it in a proper position to take advantage of all its features. This device provides a motion camera (not used, so external obstacles are not considered), a computer processing unit and a depth camera. Furthermore, suction cups have been added to the legs extremes in order to hold on to any surface and allow the robot to climb. Every suction cup is equipped with its own centrifugal impeller and motor that creates and maintains the vacuum even on porous surfaces, extracting the internal air [26]. The complete griping system consists of (a) an electronic circuit inside the cup that sensorizes the system and measures the pressure and the distance to the support surface, (b) a turbine motor with its variator, (c) an electronic board that acts as a link between sensorization circuits and the control system of the suction cups and the microcontroller, and (d) a mechanical system with three rotary degrees of freedom to properly align with the surface.

Lastly, to increase the work-space of the legs, the configuration of the motors has been modified, changing the position of the second motor. In this way, the center of mass is lowered, increasing the robot stability.

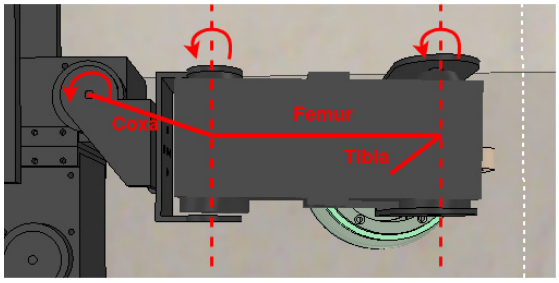
A critical aspect of controlling the robot while climbing is to ensure the normal and shear forces at the suction cup do not exceed certain limits during movement, creating the risk of loss of grip [27]. The hexapod robot is a hyper-static system whose elastic model is too complex to be included in a control loop. Given the hyperstatic nature of the problem, a simplified dynamic model is calculated in [28] and included in the control.

Climber robots are deployed in dangerous situations, where the power consumption must be optimized to guarantee the finalization of the task. Possible solutions are found in weight reduction, calculation of the best path, or optimization of the walking patterns.

The leg position is a critical aspect that determines the distance traveled in a given period of time. For this reason, it is highly desired to optimize the robot's leg positions.



(**a**)



(**b**)

Figure 1. Joint axis and reference systems of the legs. (**a**) ROMHEX with the reference system of the body and legs, and leg identifier. (**b**) Axis of ROMHEX joints.

3.1. Optimization of the Leg Positions

Making use of genetic algorithms (in this case, MatLab's *ga* function), it is possible to find the best position of the robot's legs' initial configuration according to a criterion. The algorithm uses a combination of the distance the robot can walk and the forces produced at the legs as a cost function. Because the objective is to obtain the optimal initial position of the legs to walk, the "genes" or decision variables are the initial positions of the legs (X and Y for each leg and a global Z with respect to the center of the robot, this is, 13 positions in total).

An analysis has been carried out on the center of mass and how its position affects the forces produced in the different legs to improve the results. The objective of the cost function is to obtain a genetic individual (legs position configuration) that achieves the greatest distance while walking with a given number of movements following a predefined pattern, keeping the robot safe. It considers the distance that the robot moves, as well as the maximum permissible forces in the legs, as indicated in (1), where C is the cost function, D is the distance traveled, and F is a matrix where each row corresponds to a moment in the execution of the walking movement and each element of the row corresponds to each leg. The cost function is negative because it is required to minimize the value. Both $f(x)$ and

D are positive values, being a single S calculated in (2), where S represents the vector of forces applied over a leg, and $S(2)$ is the force applied over the z axis.

$$C = -min(f(max(F)), f(min(F))) \cdot D \quad (1)$$

$$S = norm(S) \cdot sign(S(2)) \quad (2)$$

Function $f(x)$ is distance scaling and piecewise defined, as shown in Figure 2. The objective of scaling the distance obtained by the factor produced by the function is to penalize the individuals that produce the highest forces, even if they manage to move a greater distance. The function takes into account the sign on the z axis, to differentiate the dangerous forces. Both signs are considered to eliminate individuals that make the suction cups detach and reduce extreme forces on a single leg to reduce unequal wearing.

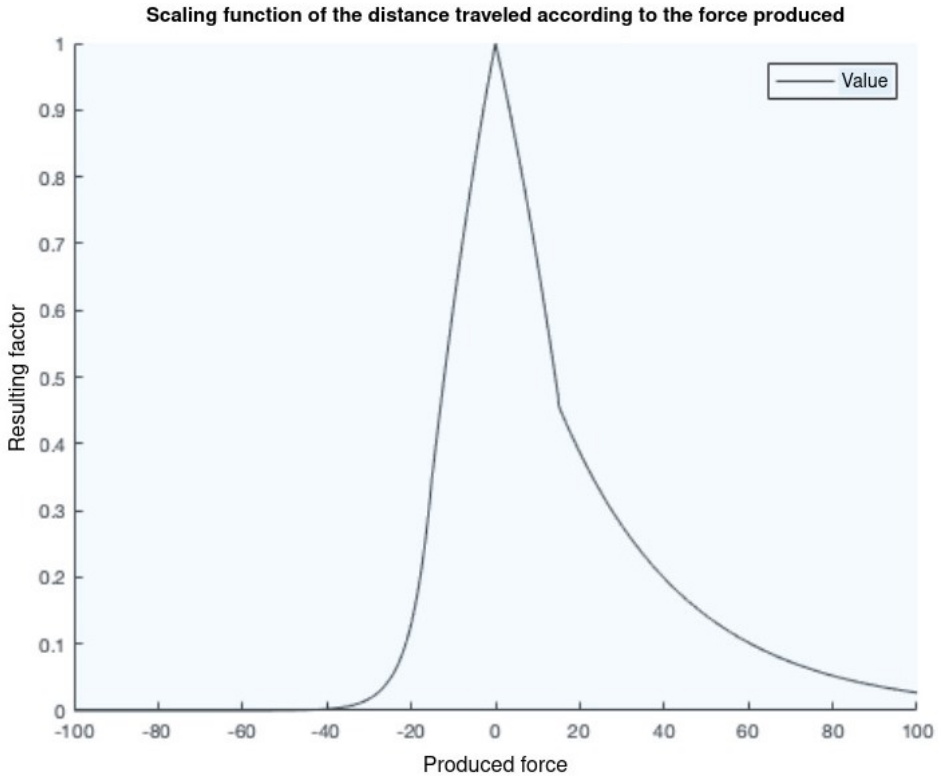


Figure 2. Distance scaling function.

To obtain the value of *dist*, the developed walking movement has been simulated in the following way: First, it is checked that the individual is valid, this is, (a) the position of all the legs is reachable with the inverse kinematics, (b) the position of the motors is within the specified ranges, and (c) there is no collision between legs. Second, the cost function value is obtained.

The results of the genetic algorithm are an increase of 107% in the distance traveled (from 355 mm to 735 mm) and a decrease of 10% in the force. Figure 3 shows a representation of the optimized version over the previous one. As illustrated in that picture, the position of the legs has undergone a slight variation to achieve an initial position that optimizes the evaluation criteria. Table 1 denotes the joint initial position increment between before and after the optimization, with the references in the motor encoder origins. Furthermore, both tables show the end-effector positions (feet) when the motors are in the given initial position.

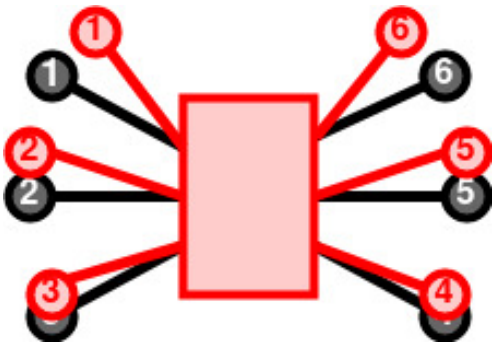


Figure 3. Comparison between the position of the legs before (gray) and after (red) the optimization through the genetic algorithm. Positions specified in Table 1.

Table 1. Variation of the position of each joint and suction cup after the optimization.

Leg	Joint Angles (rad)			Feet Position (mm)		
	q_0	q_1	q_2	x	y	z
1	−0.1	−0.13	0.33	28	6	−3
2	−0.1	−0.18	0.49	22	35	−3
3	0.36	−0.36	−1.15	79	−129	−3
4	−0.66	0.15	−0.75	−17	127	−3
5	−0.11	−0.08	0.19	−21	−11	−3
6	0.11	−0.19	0.49	36	−11	−3

4. Control Architecture

A new control architecture that considers safety under unforeseen circumstances is needed to guide legged-and-climber robots. The proposed control architecture is characterized as a behavior-based control, hierarchical and centralized. As shown in Figure 4, the architecture is split in the Executive, the Planner and the User Interface. The Planner is divided into three main levels, which make use of complementary modules located in the Executive. The architecture includes a User interface, with which the user may control the behavior of the robot and observe the state of the robot and the legs. Each level of the Planner has a set of critical and given objectives:

1. **Level 1**: Corresponds to the nominal and continuous behavior without checking the safety at any moment. This level is responsible for the body movement in the desired direction, through the performance of the robot legs.
2. **Level 2**: Corresponds to behaviors about movements under expected situations, having considered basic safety issues. It is responsible for determining if a movement may still be developed.
3. **Level 3**: Corresponds to the critical safety checks to ensure that the robot is not in a hazardous situation. This level is vitally important in robots like the one in question here, where the goal is to allow it to walk safely on the wall and ceiling.

There is a hierarchical relationship between the different levels in that the higher level is able to disable the lower level. Dependencies occur from top to bottom; in other words, what happens at the upper level is unknown by lower levels. The agents of the same level are in a situation of equality, so they need a synchronization mechanism in case two behaviors are mutually exclusive. A token synchronization has been used to do this: the agent with the token is the one that can be executed. When it stops executing, it will drop the token and other behavior will be free to catch it.

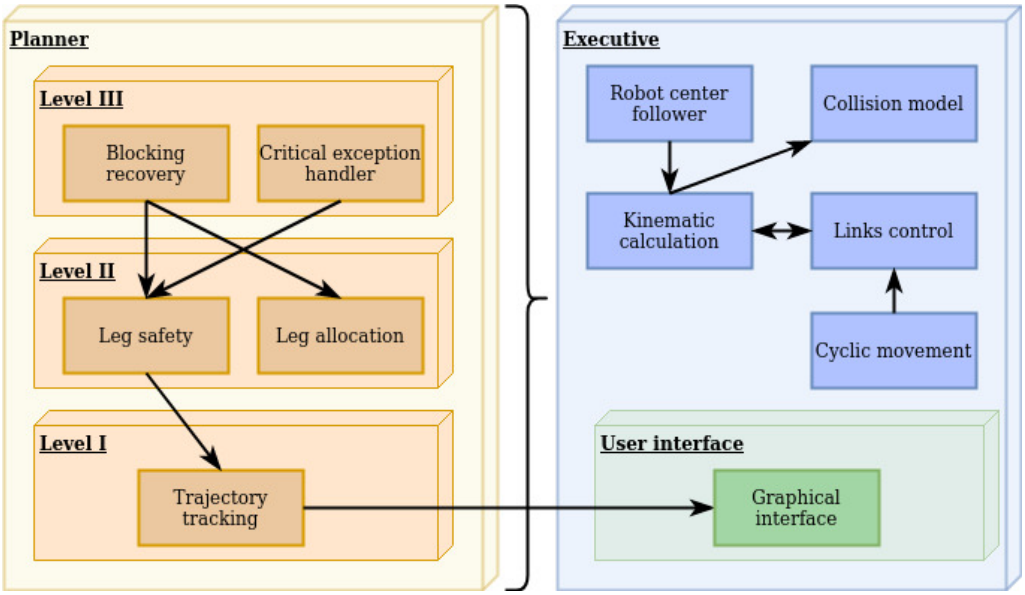


Figure 4. Control architecture.

Each behavior has its own functionalities, inputs, outputs, and implementation features. Architecture modularity allows developers to add more, increasing control capabilities.

4.1. Level 1: Nominal Movement of the Body

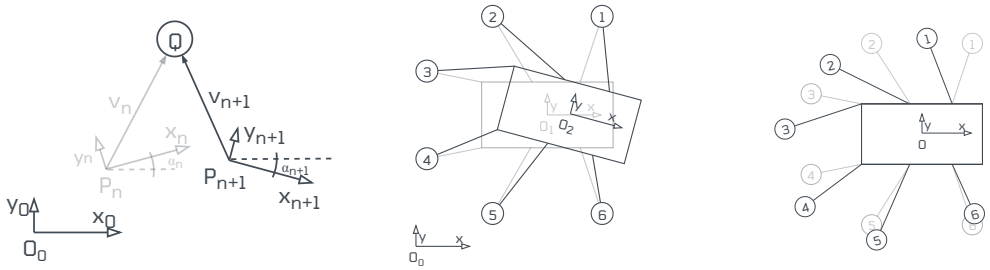
Trajectory Tracking

The objective of this behavior is that the robot center follows a trajectory in terms of different global positions and orientations, without explicit information about velocities. Basically, it carries out the inverse kinematics of the robot, where the input is the robot center trajectory, and the output is the position of the leg extremity. The output is generated dynamically through a close chain. However, this agent neither checks the stability nor sends commands if the inverse kinematics can not obtain a required point. Interpolation is needed if two consecutive poses are too far apart to obtain as many intermediate ones as needed. In this case, the point is divided into position and orientation, where spherical linear interpolation (SLERP) [29] is used for the orientation interpolation, to obtain the maximum precision, while the position interpolation is linearly done.

Because several legs are attached to the ground to move the center of the body in the world coordinates, the legs will move opposite to the body robot coordinates. When P_n is the position of the center of the robot, R_n is its orientation, v_n is the vector that describes the position of one of the leg extremities, v_{n+1} is the vector in the position to be achieved, (P_n, R_n) denotes the robot pose, and (P_{n+1}, R_{n+1}) is the pose to be achieved. Then, the position of the leg extremity in both references (3) is obtained, while v_{n+1} is calculated in (4). For better understanding, Figure 5 shows a comparison of the movement in the robot's and world coordinates.

$$P_n + R_n \cdot v_n = P_{n+1} + R_{n+1} \cdot v_{n+1} \tag{3}$$

$$v_{n+1} = (R_{n+1})^t \cdot (P_n - P_{n+1} + R_n \cdot v_n) \tag{4}$$



(**a**) (**b**) (**c**)

Figure 5. Comparison of the movement in the robot's and world coordinates. (**a**) Representation of the reference change of the point Q between (P_n, R_n) and (P_{n+1}, R_{n+1}), where α_k represent the angle for the rotation matrix R_k. (**b**) Comparison between the initial (light color) and final position in global coordinates. (**c**) Comparison between the initial (light color) and final position in the robot's coordinates.

4.2. Level 2: Expected Situations and Leg Allocation

4.2.1. Leg Safety

The objective of this behavior is to predict when a leg will find an instability or blocking situation and move it to avoid this state. It takes as input the current pose of each leg and the current motion tendency, among others. The main output is which leg is required to relocate to where to ensure the robot's stability; that is, move a leg to a new position.

To implement this behavior, a metric about how urgently each leg should be relocated is obtained, in this case: how close the joints are to their limits, from 0.6 rad (not urgent) to 0 rad (critical); and how close each foot is to the center of mass (COM) of the robot, from 20 cm (not urgent) to 5 cm (critical). Furthermore, it checks that, in the future position, each leg's kinematics will allow lifting them in case a reallocation is needed in that state.

Two limits have been set, a motion limit and a danger limit. They represent the limit within which a leg can move and the limit within which the leg can move without entering a dangerous situation, respectively. If the robot enters a dangerous situation, the behavior is blocked until the legs in hazard are reallocated.

Whenever a leg is expected to move, it moves the maximum possible distance without colliding with other legs in the direction of movement, minimizing the number of movements needed. To move a leg, it must have enough space to move and lift without creating an unstable situation, considering how the rest of the legs are positioned. For that, the force model generated in [28] is needed.

During the behavior execution, it is possible to enter in a blocked state, in which there is no leg to move. In this case, it informs the upper level to take control to return the robot to a safe state. When this behavior is disabled, the trajectory control is disabled to avoid hazardous situations. Whenever it is enabled again, it enables the trajectory control to continue with the previous execution.

4.2.2. Leg Allocation

This behavior is responsible for safely moving all the legs, or a subset of them, to a given position. The input is the desired position, and the output is a movement of each leg, in the proper order and moment. The order is defined as a function of the safety in which the movement may be done (state while a leg is lifted). Leg allocation checks that the forces of lifting the legs do not produce any hazard and the possibility of reaching the desired position without colliding with other legs exits.

This behavior can not work at the same time as the "leg safety" behavior due to the possibility of conflicts. Thus, a timing mechanism of a *token* is used, because there is no hierarchy of any kind between them.

4.3. Level 3: Critical Exceptions and Blocking Situations

4.3.1. Blocking Recovery

This behavior is thought of as an external observer. It is responsible for detecting when there is a blocking state to unlock it. The input is the system state, and the output is the position in which the legs should move to solve the blocking state.

The "leg safety" behavior informs when it detects that no legs can move because it is not possible to lift a leg or move enough in the required direction. When the advice is received, this behavior blocks the advice emitter to avoid any leg movement during the reallocation. It asks the "leg allocation" behavior to move all legs to a default safety position. When this process finishes, it enables the "leg safety" behavior to continue with the nominal execution.

4.3.2. Critical Exception Handler

This behavior is responsible for ensuring that the robot is not in danger. The input is the system state with the information from all sensors. The output is the fact of blocking the motion system, keeping the robot completely still, and informing the user about the critical error. Then, the user should solve the problem using the graphical user interface to check forces, leg positions, and other factors.

4.4. Complementary Modules

These modules are needed to make the behaviors work. They include different geometric calculations and control over the links at a low level.

4.4.1. Robot Center Follower

This module obtains the movement of the robot center. Its input is the position of the legs, and its output is the position of the robot center in the world coordinates. To observe how the robot follows the trajectory from an initial state (P_n, R_n) to a target state (P^*_{n+1}, R^*_{n+1}), the real state of the center of the robot (P, R) is calculated in an instant between t_n and t_{n+1} in absolute coordinates, making use of the relative position vectors of each of the different legs attached to the ground. u^k are the vectors of the relative position of the leg k-th respecting the robot center in the current time (with reference (P, R)), u^k_n are the vectors of the relative position at the t_n time respecting the reference (P_n, R_n), and K is the number of legs attached to the ground. A system of $3 \cdot K$ equations is obtained (5), where the unknown values of P and R are obtained through numeric solvers due to the non-linearity of the problem. For that, the gradient descent method has been used.

$$P + Ru_k = P_n + R_n u^k_n, \quad k = 1, \ldots, K \tag{5}$$

4.4.2. Collision Model

This module describes a simplified collision model with which is possible to calculate if a leg collides with another leg in a given configuration. The simplification consists of a 2D model of the ROMHEX robot, where each leg is represented as a linear segment, and each suction cup is represented as a circle. The module checks if there is a collision of the type (a) between two circles, (b) between a circle and a segment, or (c) between two segments.

4.4.3. Kinematics Calculation

This module obtains the direct and inverse kinematic of a leg, with the reference system in the robot center. It is completely dependent on the robot, so this module must be changed if another robot is used. Using the tests with ROMHEX, we present the forward and inverse kinematics of this robot, obtaining the algebraic solution.

Following Figure 1, forward kinematics is calculated in (6)–(8), where A_{coxa} is the angle between the first motor origin and the femur. P_x, P_y and P_z denote the position of the end-effector with respect to the leg coordinate system, L_{coxa}, L_{femur}, and L_{tibia} denote the link lengths, while q_1, q_2 and q_3 denote the joint angles. Furthermore, thanks to Figure 1a it

is possible to obtain the transformation matrix between the body center and each leg origin. These transformation matrices must be applied over the body and legs reference systems.

$$P_x = L_{coxa} \cdot cos(A_{coxa} + q_1) + L_{femur} \cdot cos(q_2) \cdot sin(q_1) + L_{tibia} \cdot cos(q_2 + q_3) \cdot sin(q_1) \quad (6)$$

$$P_y = L_{coxa} \cdot sin(A_{coxa} + q_1) + L_{femur} \cdot cos(q_2) \cdot cos(q_1) + L_{tibia} \cdot cos(q_2 + q_3) \cdot cos(q_1) \quad (7)$$

$$P_z = L_{coxa,z} + L_{femur} \cdot sin(q_2) + L_{tibia} \cdot sin(q_2 + q_3) \quad (8)$$

With respect the inverse kinematic, the solution is found in (9)–(11), where variables B, A_1, etc. are calculated in (12)–(16).

$$q_3 = \begin{cases} B - \pi & if\ third\ link\ up \\ \pi - B & if\ third\ link\ down \end{cases} \quad (9)$$

$$q_2 = \begin{cases} (A_1 + A_2) - \frac{\pi}{2} & if\ third\ link\ up \\ \frac{\pi}{2} - (A_1 + A_2) & if\ third\ link\ down \end{cases} \quad (10)$$

$$q_1 = \frac{P_x}{P_y} - \frac{L_{coxa,x}}{LP + L_{coxa,y}} \quad (11)$$

$$B = arccos(\frac{HF^2 - L_{femur}^2 - L_{tibia}^2}{-2 \cdot L_{femur} \cdot L_{tibia}}) \quad (12)$$

$$HF = \sqrt{LP^2 + (P_z - L_{coxa,z})^2} \quad (13)$$

$$LP = \sqrt{P_x^2 + P_y^2 - L_{coxa,x}^2} - L_{coxa,y} \quad (14)$$

$$A_1 = arctan(\frac{LP}{|P_z - P_{coxa,z}|}) \quad (15)$$

$$A_2 = arccos(\frac{L_{tibia}^2 - L_{femur}^2 - HF^2}{-2 \cdot L_{femur} \cdot HF}) \quad (16)$$

4.4.4. Center of Mass Calculation

This module calculates the center of mass with respect to the robot center. It is implemented with the knowledge of the joints' state, the links' mass and the links' shape.

4.4.5. Links Control

This module is responsible for unifying all movements for the different legs, which are sent from the different behaviors to be executed at the link level. Whenever a behavior desires to move a leg, it sends a command with the leg identifier, where to move it (in cartesian or articular coordinates), and the priority of the movement. Behaviors that send movements must be aware that some movements may be overwritten and partially executed because a higher priority message is received.

4.4.6. Cyclic Movement

This module aims to generate a default cyclic movement that allows the robot to walk forward. Without inputs, the output is the positions that the legs should reach in each moment. The module checks the performance of the control architecture, so a simple walking process has been developed. It consists of moving the legs individually from the back to the front after moving the whole body forward. Its behavior may be replaced by changing and complex patterns.

4.5. Graphical User Interface

A graphical user interface (GUI) for a generic legged robot has been developed to make interacting with the robot easier. The GUI is shown in Figure 6, and it includes (a) information about the robot status, (b) plots of the positions of the motors along a given period of time, and (c) a graphical representation of the position of the legs and the robot trajectory. It is also possible to create trajectories and send them to the robot.

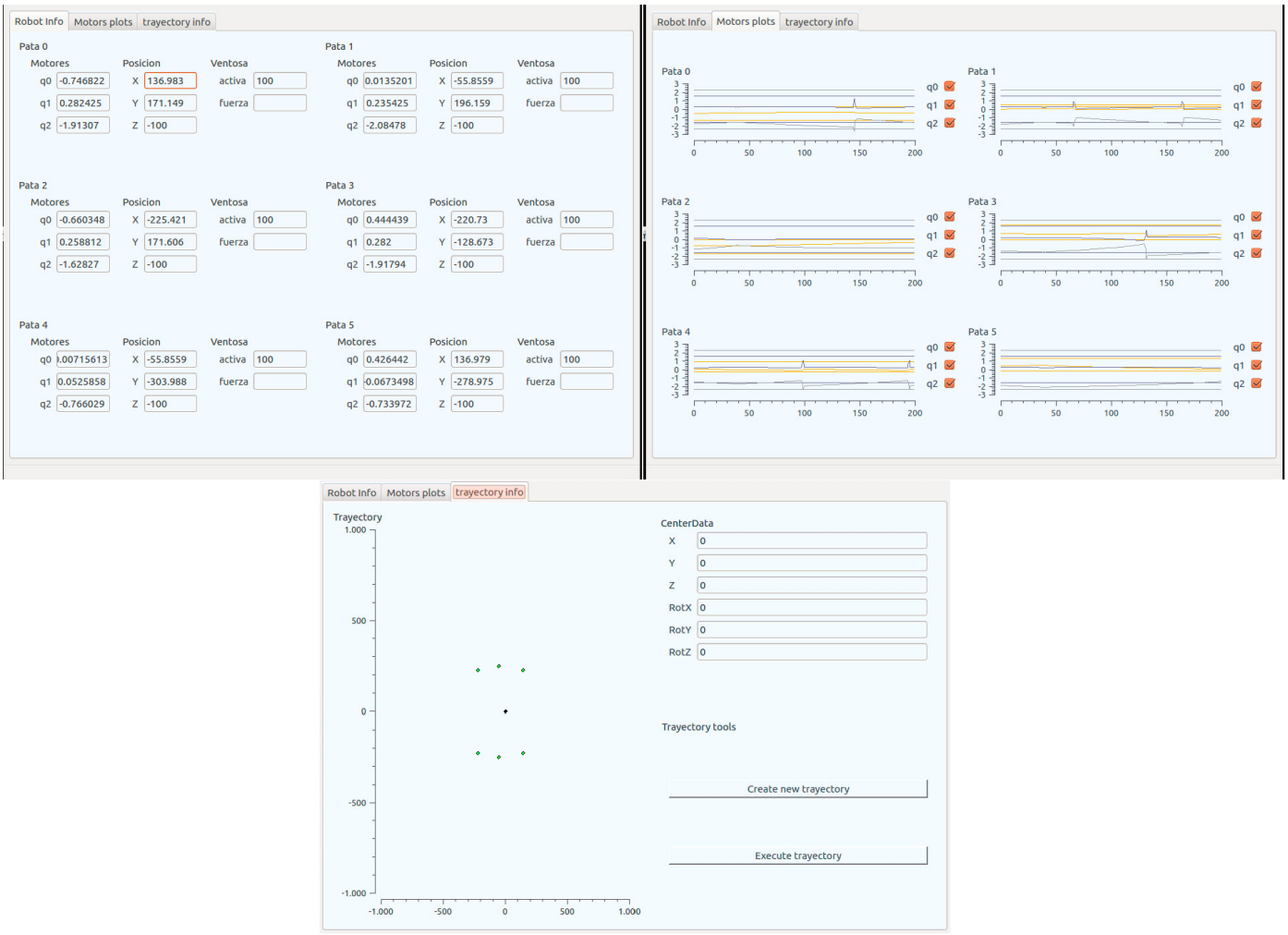


Figure 6. Graphical User Interface. The first tab shows information about the robot, including the motors and suction cup. The second tab shows the motors position during a given period. The third tab shows the whole robot trajectory, and it allows setting the goal position to execute a new trajectory.

5. Experimental Results

The performance of the simulated robot while walking on a flat plane was tested in CoppeliaSim. The system was studied while moving forward, laterally, diagonally, rotatory, and walking with a combination of movements. After validating the performance in these conditions, the system was tested on sloping (45°) and vertical walls with successful results. However, the presence of gravity in different axes required adjusting the controller gains.

To test the generalization of the control, the performance was checked when the number of legs was changed. The behavior when two legs are removed is also valid, even though the control code is not modified. Figure 7 shows the generated walking pattern when the robot detects six legs and when it detects a malfunction in two of its legs. As it is observed, the walking pattern when the robot has six legs is periodic, because the tolerances specified for a leg to move were adjusted with a hexapod robot. In the case of four legs,

the walking pattern is not periodic. In this case, legs are moved a non-defined distance when the space in front of them is higher than a threshold.

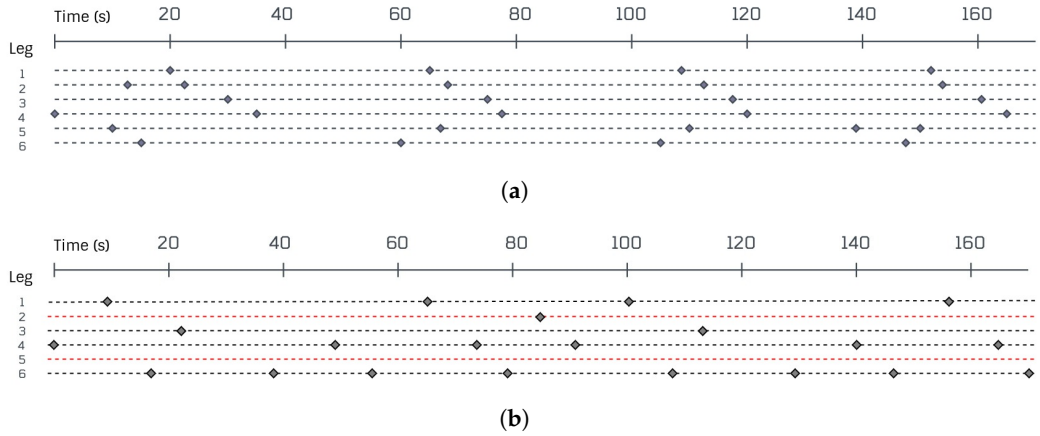


Figure 7. Walking patterns automatically generated for different numbers of legs detected. Each point represents the reallocation of a leg, that is, the turn of a leg to move. For example, in the upper walking pattern, first the fourth leg moves, second the fifth leg, third the second leg, etc. (**a**) Six legs detected. (**b**) Four legs detected. Red lines represent the disabling of a leg.

Figure 8 shows the motion sequences that the robot follows during a walking pattern. Furthermore, we tested how capable the robot is of moving its center to desired positions and orientations. A video summary of the robot's movements during this simulation is found in https://youtu.be/ex1Dj-uwluE, accessed on 12 October 2021.

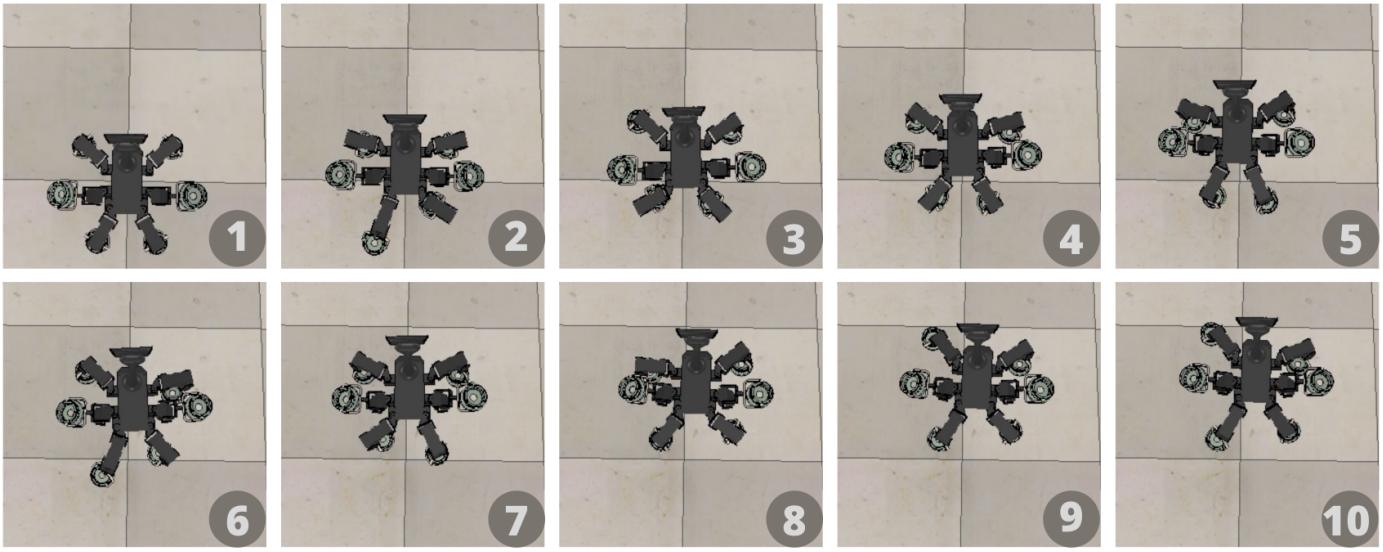


Figure 8. Motion sequences during a walking pattern. The two center legs are disabled, pointing the suction cup up.

In the tests with the real robot, it is crucial to determine the time the suction cups spend to be attached to the ground or wall and the amount of time they spend to be detached. These times are 0.5 s and 1.5 s, respectively. The tests were carried out in the ROMHEX robot, to check the feasibility of our approach for its implementation in the ROMERIN robot (a modular climber robot for infrastructure inspection) [28]. The tests reveal a good performance during the movements in the horizontal plane. However, the tests on the sloping wall reveal hardware problems. The first problem is related to the suction cups, which have three free joints. These joints make the suction cup focus down instead of against the wall, spoiling the correct pulling force. As a result, one of the free joints has been removed, while another has been limited in movement. Once the first problem was

solved, the second problem involved the grip force of the suction cups. The maximum inclination that the robot can manage to hold by itself is 60°. However, in this situation, a small perturbation may make the robot fall. The walking pattern during the tests with ROMHEX is shown in Figure 9 with successful results. The video of the robot moving can be found in https://youtu.be/-ASO8B4THEU, accessed on 12 October 2021.

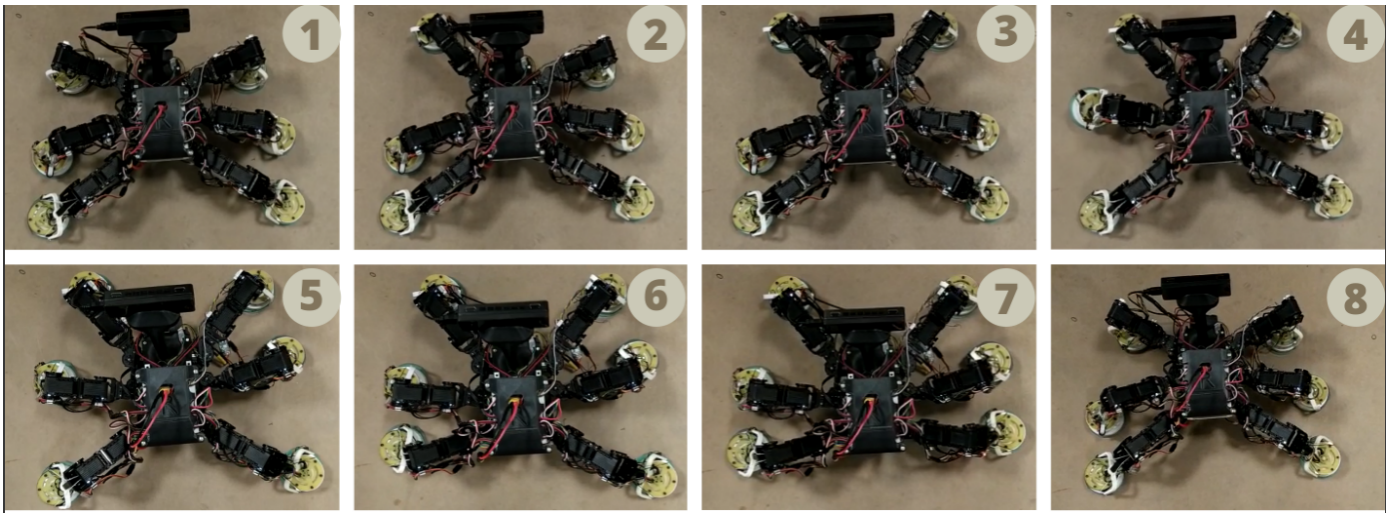


Figure 9. Motion sequences during a walking pattern with ROMHEX.

Finally, the control architecture has been tested and found to work when the robot loses more legs than allowed. For example, if the hexapod robot loses three legs, it is statically unstable, but it can stay still with three legs supporting its weight.

6. Conclusions

First of all, implementing the described control has completed the task of making the robot capable of walking in any direction while maintaining safety. Thanks to behavior-based control, it has been possible to divide the global problem into smaller and more encompassing parts, obtaining a more modular control. This structure also allows adding new functionality in a simple way, by adding layers in the control without changing the current control. The generality of the system allows using a large part of the control with any legged robot typically between four to eight legs, because the majority of legged-and-climber robots dispose of these number of legs. However, the control architecture could be used for a legged robot of more than eight legs, because there is no upper limit.

We achieve a generic control for a robot with an unpredefined number of legs. A cyclic walking pattern has been tested in the hexapod ROMHEX robot with successful results, even when the robot suffers a malfunction of two legs. Taking advantage of the agent-based structure, the system may be improved with the easy addition of new agents over the used standard framework ROS.

Optimizing the initial position of the legs allows increasing the mobility of the robot and obtaining a better understanding of how the forces are distributed when walking. As the movement is generated dynamically, it sometimes reaches a configuration where it cannot easily move. In this case, all legs are reconfigured to this optimized initial position, which allows the robot to continue moving easily. The tests carried out with the real robot demonstrate its potential for climbing, although the hardware may undergo some modifications. Each iteration carried out on the robot has improved its ability to walk, and increase knowledge about the effects of gravity.

All results and changes made with the current robot, as well as improving its ability to move and climb correctly, serve as inspiration for designing future robots. It is important to consider all the details in which ROMHEX fails to obtain a more complete and robust platform in these designs.

Contrasting with state of art, this paper presents a new architecture especially created for legged-and-climber robots, where the number of layers is reduced from the typical three-layer architecture [30] to only two layers, as done previously in CLARAty and COTAMA. Unlike CLARAty, where the internal behaviors are open to the developer, we define specific behaviors for legged-and-climber. Unlike COTAMA architecture, we dispense with the supervisors and scheduler, to particularize our problem.

Author Contributions: Conceptualization, M.H., M.A., C.P. and E.G.; methodology, M.H. and M.A.; software, M.A.; validation, M.A.; formal analysis, M.H. and M.A.; investigation, M.H. and M.A.; resources, M.H.; data curation, M.A.; writing—original draft preparation, C.P.; writing—review and editing, C.P. and E.G.; visualization, M.A. and C.P; supervision, M.H.; project administration, M.H. and E.G.; funding acquisition, M.H. and E.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research is part of The ROMERIN project (DPI2017-85738-R) funded by the Spanish Ministry of Science and Innovation (RETOS research and innovation program).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

Abbreviations

The following abbreviations are used in this manuscript:

ROMHEX	Romerin Hexapod
SLERP	Spherical linear interpolation
COM	Center of mass
GUI	Graphical user interface
ROMERIN	Modular Climber Robot for Infrastructure Inspection
ROS	Robot Operating System

References

1. SPARC. The Partnership for Robotics in Europe. Robotics 2020 Multi-Annual Roadmap for Robotics in Europe. Horizon 2020. Available online: https://www.eu-robotics.net/sparc/about/roadmap (accessed on 27 August 2021).
2. Sostero, M. *Automation and Robots in Services: Review of Data and Taxonomy*; European Commission, Joint Research Centre (JRC): Seville, Spain, 2020.
3. Shen, W.; Gu, J.; Shen, Y. Proposed wall climbing robot with permanent magnetic tracks for inspecting oil tanks. In Proceedings of the IEEE International Conference Mechatronics and Automation, Niagara Falls, ON, Canada, 29 July–1 August 2005; Volume 4, pp. 2072–2077.
4. Yi, S.J.; Lee, D.D. Dynamic heel-strike toe-off walking controller for full-size modular humanoid robots. In Proceedings of the 2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids), Cancun, Mexico, 15–17 November 2016; pp. 395–400.
5. Park, I.W.; Kim, J.Y.; Oh, J.H. Online biped walking pattern generation for humanoid robot khr-3 (kaist humanoid robot-3: Hubo). In Proceedings of the 2006 6th IEEE-RAS International Conference on Humanoid Robots, Genova, Italy, 4–6 December 2006; pp. 398–403.
6. Vladareanu, V.; Boscoianu, C.; Munteanu, R.I.; Yu, H.; Vladareanu, L. Dynamic control of a walking robot using the versatile intelligent portable robot platform. In Proceedings of the IEEE 2015 20th International Conference on Control Systems and Computer Science, Bucharest, Romania, 27–29 May 2015; pp. 38–45.
7. Wooldridge, M. Intelligent agents. *Multiagent Syst.* **1999**, *6*, 3–50.
8. Jamroga, W.; Ågotnes, T. What agents can achieve under incomplete information. In Proceedings of the Fifth International Joint Conference On Autonomous Agents and Multiagent Systems, Hakodate, Japan, 8–12 May 2006; pp. 232–234.
9. El Jalaoui, A.; Andreu, D.; Jouvencel, B. Contextual Management of Tasks and Instrumentation within an AUV control software architecture. In Proceedings of the IEEE International Workshop on Intelligent Robots and Systems (IROS), Beijing, China, 9–15 October 2006; pp. 3761–3766.
10. El Jalaoui, A.; Andreu, D.; Jouvencel, B. A control architecture for contextual tasks management: Application to the AUV Taipan. In Proceedings of the IEEE Europe Oceans 2005, Brest, France, 20–23 June 2005; Volume 2, pp. 752–757.

11. Durand, B.; Godary-Dejean, K.; Lapierre, L.; Passama, R.; Crestani, D. Fault tolerance enhancement using autonomy adaptation for autonomous mobile robots. In Proceedings of the IEEE 2010 Conference on Control and Fault-Tolerant Systems (SysTol), Nice, France, 6–8 October 2010; pp. 24–29.
12. Muscettola, N.; Dorais, G.A.; Fry, C.; Levinson, R.; Plaunt, C.; Clancy, D. Idea: Planning at the Core of Autonomous Reactive Agents. In Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space, Houston, TX, USA, 27–29 October 2002.
13. Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; Ingrand, F. An architecture for autonomy. *Int. J. Robot. Res.* **1998**, *17*, 315–337. [CrossRef]
14. Volpe, R.; Nesnas, I.; Estlin, T.; Mutz, D.; Petras, R.; Das, H. The CLARAty architecture for robotic autonomy. In Proceedings of the 2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542), Big Sky, MT, USA, 10–17 March 2001; Volume 1, pp. 1–121.
15. Musliner, D.J.; Durfee, E.H.; Shin, K.G. CIRCA: A cooperative intelligent real-time control architecture. *IEEE Trans. Syst. Man Cybern.* **1993**, *23*, 1561–1574. [CrossRef]
16. Simon, D.; Espiau, B.; Castillo, E.; Kapellos, K. Computer-aided design of a generic robot controller handling reactivity and real-time control issues. *IEEE Trans. Control Syst. Technol.* **1993**, *1*, 213–229. [CrossRef]
17. Jakimovski, B.; Meyer, B.; Maehle, E. Self-reconfiguring hexapod robot OSCAR using organically inspired approaches and innovative robot leg amputation mechanism. In Proceedings of the International Conference on Automation, Robotics and Control Systems (ARCS-09), Orlando, FL, USA, 13–16 July 2009.
18. Maehle, E.; Brockmann, W.; Grosspietsch, K.E.; Auf, A.E.S.; Jakimovski, B.; Krannich, S.; Litza, M.; Maas, R.; Al-Homsy, A. Application of the organic robot control architecture ORCA to the six-legged walking robot OSCAR. In *Organic Computing—A Paradigm Shift for Complex Systems*; Springer: Basel, Switzerland, 2011; pp. 517–530.
19. Brockmann, W.; Maehle, E.; Grosspietsch, K.E.; Rosemann, N.; Jakimovski, B. ORCA: An organic robot control architecture. In *Organic Computing—A Paradigm Shift for Complex Systems*; Springer: Basel, Switzerland, 2011; pp. 385–398.
20. Maas, R.; Maehle, E.; Großpietsch, K.E. Applying the organic robot control architecture ORCA to cyber-physical systems. In Proceedings of the IEEE 2012 38th Euromicro Conference on Software Engineering and Advanced Applications, Cesme, Turkey, 5–8 September 2012; pp. 250–257.
21. Pack, R.T.; Christopher, J.L.; Kawamura, K. A rubbertuator-based structure-climbing inspection robot. In Proceedings of the IEEE International Conference on Robotics and Automation, Albuquerque, NM, USA, 25 April 1997; Volume 3, pp. 1869–1874.
22. Brooks, R. A robust layered control system for a mobile robot. *IEEE J. Robot. Autom.* **1986**, *2*, 14–23. [CrossRef]
23. Rönnau, A.; Heppner, G.; Nowicki, M.; Dillmann, R. LAURON V: A versatile six-legged walking robot with advanced maneuverability. In Proceedings of the 2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Besacon, France, 8–11 July 2014; pp. 82–87.
24. Fankhauser, P.; Bellicoso, C.D.; Gehring, C.; Dubé, R.; Gawel, A.; Hutter, M. Free gait—An architecture for the versatile control of legged robots. In Proceedings of the 2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids), Cancun, Mexico, 15–17 November 2016; pp. 1052–1058.
25. Hutter, M.; Gehring, C.; Jud, D.; Lauber, A.; Bellicoso, C.D.; Tsounis, V.; Hwangbo, J.; Bodie, K.; Fankhauser, P.; Bloesch, M.; et al. Anymal—A highly mobile and dynamic quadrupedal robot. In Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea, 9–14 October 2016; pp. 38–44.
26. Hernando, M.; Gómez, V.; Brunete, A.; Gambao, E. CFD Modelling and Optimization Procedure of an Adhesive System for a Modular Climbing Robot. *Sensors* **2021**, *21*,1117. [CrossRef] [PubMed]
27. Ko, H.; Yi, H.; Jeong, H.E. Wall and ceiling climbing quadruped robot with superior water repellency manufactured using 3D printing (UNIclimb). *Int. J. Precis. Eng. Manuf.-Green Technol.* **2017**, *4*, 273–280. [CrossRef]
28. Hernando, M.; Brunete, A.; Gambao, E. ROMERIN: A Modular Climber Robot for Infrastructure Inspection. *IFAC-PapersOnLine* **2019**, *52*, 424–429. [CrossRef]
29. Barrera, T.; Hast, A.; Bengtsson, E. Incremental spherical linear interpolation. In *The Annual SIGRAD Conference. Special Theme-Environmental Visualization*; Linköping University Electronic Press: Linköping, Sweden, 2004; pp. 7–10.
30. Gat, E.; Bonnasso, R.P.; Murphy, R. On three-layer architectures. *Artif. Intell. Mob. Robot.* **1998**, *195*, 210.