WalkingBot: Modular Interactive Legged Robot with Automated Structure Sensing and Motion Planning

Meng Wang[1,2,3] Yao Su[4] Hangxin Liu[4] Yingqing Xu[1,2,3]

***Abstract*— This paper presents *WalkingBot*, a modular robot system that allows non-expert users to build a multi-legged robot in various morphologies. The system provides a set of building blocks with sensors and actuators embedded. Through the integrated hardware and software designs, the morphology of the built robot is interpreted automatically, and its kinematic model is revealed in a customized GUI in a computer screen, allowing users to understand, control, and program the robot easily. A Model Predictive Control (MPC) scheme is introduced to generate a control policy for various motions (*e.g.* moving forward, turning left) corresponding to the sensed robot structure, affording rich robot motions right after assembling. Targeting different levels of programming skill, two programming methods—visual block programming and events programming are also presented to enable users to create their own interactive legged robot.**

I. Introduction

Modular robots afford a variety of physical structures and models by assembling and connecting a set of building blocks. They result in a large space of robot morphologies with potential benefits in education [1], machine learning [2], disaster response [3] *etc*. Typically the constructed robot has multiple limbs and practical locomotions are also desired to endow it with certain mobility.

The locomotions of multi-legged robots have been studied over the past decades [4]. However, the required kinematic modeling and motion planning framework is still largely relied on manual efforts of engineers [5], which is less practical for modular robots with many different morphologies. Recently, the advance of (deep) reinforcement learning allows generating locomotion for multi-legged robots for users without expert knowledge. But the progresses are either made (i) in a virtual environment where the learned policy suffers when transferring to physical systems [6], [7], or (ii) in actual robots that can be time-consuming and limited in generalization [8].

To alleviate the needs of manual efforts in modeling or excessive training processes, it is desired to have an automated system that can interpret the robot's structure and generate a motion policy accordingly. In this paper, we present *WalkingBot*, a modular robot system that supports creating various legged robot (see Fig. 1), and interprets its corresponding morphology and generates locomotion automatically. Specifically, the system consists of three major components:

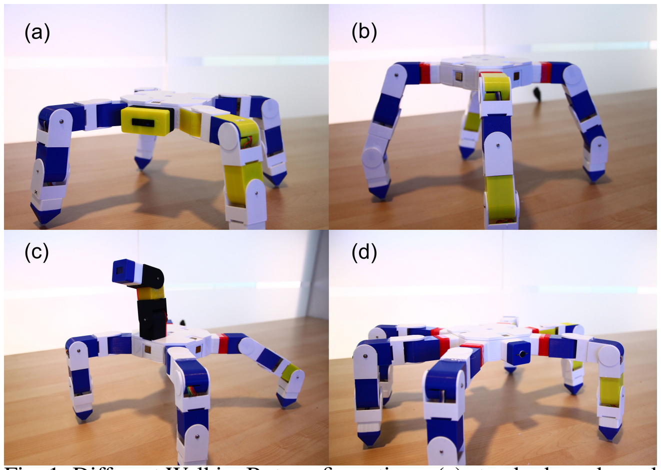


Fig. 1: Different WalkingBot configurations: (a) standard quadruped configuration with an additional range sensor; (b) another quadruped configuration by using rotation blocks; (c) modified quadruped configuration with an additional neck, legs differ in length; (d) standard hexapod configuration with two additional sound sensors.

1) **Hardware Design**. It consists of a set of building blocks with sensors and actuators embedded. A variety of legged robots with different sensing capabilities can be built from it. A local serial connection network detects the blocks connected to interpret the robot's kinematic structure, which later facilitates its motion planning.
2) **Motion Planning**. Given the sensed robot structure, we design a MPC scheme to automate the generation of gait patterns for different walking directions, speed *etc*., by formulating a multi-objective optimization that can be effectively solved online. Finally, position control is implemented to track the generated gait patterns.
3) **Programming Interface** In addition to the software in supporting its structure interpretation and motion planning, two programming interfaces — visual block programming and event programming are introduced to allow non-expert users to interact with the *WalkingBot*.

The remaining of this paper is organized as follows. In Section III, we introduce the overall hardware and software designs of *WalkingBot*, as well as its prototyping. The Methodology of the structure sensing and motion planning are described in Section IV. Section V introduces two programming methods. Finally, we discuss and conclude our work in Section VI.

II. Related Work

Building Blocks have been demonstrated as a type of promising modular Tangible User Interface (TUI) for interactive construction. Anderson *et al.* introduces a method

[1] Academy of Arts and Design, Tsinghua University, China. Emails: mengwangthu@tsinghua.edu.cn, yqxu@tsinghua.edu.cn
[2] The Future Laboratory, Tsinghua University, China
[3] The Lab for Lifelong Learning, Tsinghua University, China
[4] DMAI Research, Los Angeles, USA. Emails: yaosu@dm.ai, liuhangxin@dm.ai

to build virtual structures with LEGO-like blocks as early as 1999 [9]. Kitamura *et al.* presents ActiveCube, which allows for real-time 3D interaction with tangible cubes [10]. Jacobson and Glauser develop a tangible input device as an armature to rig an animation [11]. TwistBlocks provides building blocks for children to create armature-based animation between multiple characters [12]. However, the building blocks involved above did not incorporate actuators, whereas those in *WalkingBot* embed actuation for a more interactive experience.

Modular Reconfigurable Robot is a popular topic in robotics. Versatile functions can be implemented by the combination of limited robotic modules. Yim *et al.* discusses a lot about modular self-reconfigurable robot systems [13]. PolyBot and PolyKinetic [14] present a modular reconfigurable robot, a robotic scripting language, and a programming environment. Kalouche *et al.* integrates modularity into legged robots and presents many different configurations [15]. Snapbot [16] introduces a reconfigurable legged robot, and presents an automated learning environment for developing control policies directly on the hardware [2]. Despite their good design and performance, developing an automated way to interpret robot morphology and generate motions has not been the focus. *WalkingBot* addresses this problem to encourage more natural interactions between the modular robot and non-expert users.

Programming Education for teaching young children programming is a classic research topic in the field of HCI. A notable early work is the LOGO language by Papert [17]. Nowadays children can use Scratch [18] to program the behavior of a virtual cat. Recently, a larger number of creative construction kits have been introduced. Notable product series designed for children are LEGO Mindstorms and WeDo [19]. Tangible programming is another framework proposed that aims to encourage young children in learning physical programming and robotics. Topobo [20] introduces a 3D constructive assembly system embedded with kinetic memory and the ability to record and playback physical motion. roBlocks (now Cubelets) [21] provides tangible logic cubes for programming between sensors and actuators. Following this trend, we develop two programming interfaces, visual block programming, and events programming to allow non-expert users to interact with the system.

III. Design and Modeling

WalkingBot aims to promote interactions in building a mobile multi-legged robot. Thus, we introduce the hardware and software designs for its building blocks and connections in the subsequent subsections to achieve this goal.

A. Hardware Design

WalkingBot consists of a main body and various modular blocks, as shown in Fig. 2. The connections between blocks and the body are done through a unified design of dovetail joint with 5-pin male/female connectors embedded to allow easy attach and detach the blocks while to transfer mechanical forces, electrical power, and communication among them. The blocks' sizes and weights are tabulated in Table I.

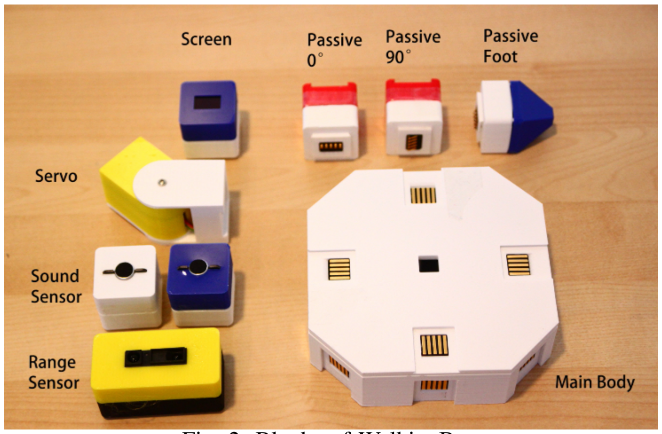


Fig. 2: Blocks of WalkingBot.

Main Body contains the necessary components for programming and running the robot, including a battery, a voltage regulator, a wireless module, a Micro Control Unit (MCU) *etc*. The main body is designed into an octagonal prism shape, and blocks can be connected to each of the eight side faces. The octagonal prism offers a convenient way of constructing a robot with up to 8 legs. In addition, there are multiple connectors on the upper surface, enabling users to add other functional structures such as sensors and display screen or kinematic structures, *e.g.* a neck.

Servo Blocks are the key to assemble legs and enable motions for the robot. Every servo block can perform 180-degree rotation and have a pair of male and female connectors at the two ends to extend the structures by connecting different parts. Some examples are shown in Fig. 3.

Sensor Blocks include various kinds of sensors supported by our hardware protocol. In this work, we present a range sensor block, sound sensor block, and screen block (see Fig. 2) for the interaction with the environment and users.

Passive Blocks only serve as mechanical parts. Normal passive blocks can be used to adjust the length or rotate the orientation of a leg structure. The foot block is the special block to be put at the end of a leg.

B. Prototyping

The hardware design follows the principle of modularity in both connection and functionality. Fig. 4 shows the major components in prototyping to support the desired modularity. The main body contains a battery (7.4V) with a voltage regulator to power the whole robot, a MCU (ATMEGA 328P) to memory the control policy and coordinate every block, and a BLE wireless module to communicate with control panels.

A servo block has a servo motor (DYNAMIXEL XL-320) connecting to an embedded MCU that stores a designated ID

TABLE I: Hardware parameters of the robot

	Size L×W×H (*mm*)	Weight (*g*)
Main body	120×120×32	185
Servo block	68×36×30	44
Passive block	34×30×30	15
Foot block	46×30×30	11
Sound sensor	34×30×30	18
Range sensor	34×60×30	23

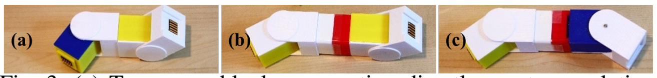


Fig. 3: (a) Two servo blocks connecting directly creates a relative rotation of 90-degree; (b) Extension block can be used to adjust the length of legs; (c) Rotation block can be used to adjust the rotation axis of legs.

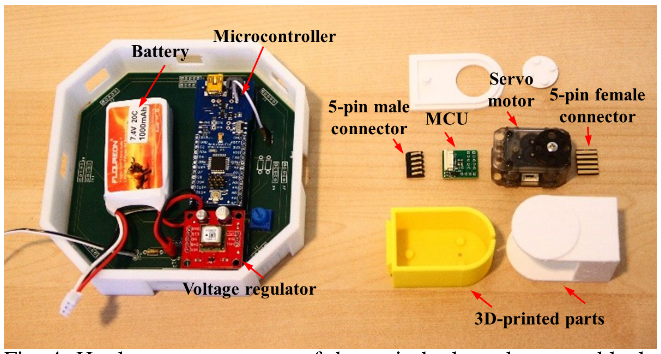


Fig. 4: Hardware components of the main body and a servo block.

to the block. A pair of male and female connectors are also connected to the MCU, enabling the block to join the IIC data bus and meanwhile receive/send topology information from/to nearby blocks. Sensor blocks have a similar design with different sensors embedded instead of servo motors, while passive blocks have nothing embedded but vary in the direction of the dovetail joints. All the blocks communicate through a generate protocol, such that various sensors or actuators can be compatible with the system. All the blocks are 3D-printed with PLA material in the prototype.

To construct the robot's legs, multiple blocks can be used. The rotation axes of two servo blocks are offset by 90-degree if they are directly connected (Fig. 3a). Passive blocks, with or without rotation, can be used to adjust the length of a leg as well as the rotation axes (Fig. 3bc). In addition to the legs connected to the main body, other structures, such as neck, face, and sensors can also be connected, resulting in various robot configurations, as shown in Fig. 1.

C. Software Design

Network Architecture: Our data network (Fig. 5) consists of three physical layers: an IIC bus for data transmission within the robot, a local serial connection network between blocks for topology detection and reconstruction, and a BLE connection between the robot and the control panel.

The IIC bus works at 400kHz and can hold at most 127 blocks in theory. The main body works as a master device, consolidating sensing data from blocks and sending commands to them. The local serial connection is a unidirectional serial interface between each two connected blocks, by which they communicate to know who they are connected to. The BLE connection is used by the control panel to receive sensing data, control the robot, or upload scripts.

Protocol: The system's data packet is carefully designed with a size less than 10Bytes, which can be completed within 0.2ms in our 400kHz data bus, to avoid system latency. Therefore, roughly $1/5$ MCU operational time can be reserved for consolidating data and at least 50 blocks can be working simultaneously.

Table II shows an example of the designed packet format.

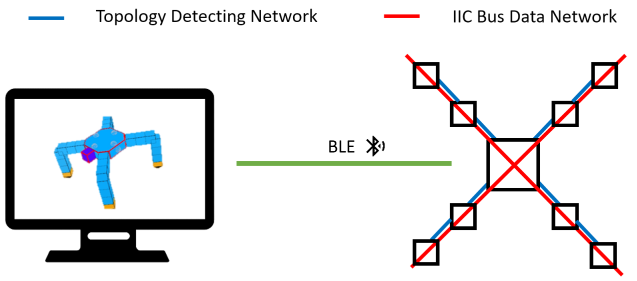


Fig. 5: Network Architecture. IIC Bus is responsible for the data transmission between body and blocks. The detection network senses the connectivity between blocks to interpret the robot's kinematic structure. Information between the robot and a PC is transmitted through Bluetooth.

In particular, Header indicates the start of the packet. Type indicates the type of block or command. ID indicates the ID of a block. Event indicates different events if used Data contains a 2Byte parameter. CRC checks if the packet has been damaged during communication. This format is compact enough to meet the size constraint while ensuring all necessary communications between the body and blocks; some examples are listed as follows:

- Send a packet to a sensor to register an event.
- Receive sensing data from a sensor.
- Send command to an actuator.
- Send or receive topological information.
- Send ACK to control panel.
- Send debug information to control panel.
- Set PID parameters.

The specific values in Table II used during events programming stand for sending an event packet (Type=0×F9) from the main body to senor (ID=0×10) and telling it to respond when value >512. The range sensor (Type=0×04) returns a response when its value=767. Then the main body sends an event packet to a servo block (ID=0×40) telling it to rotate to 90 degree.

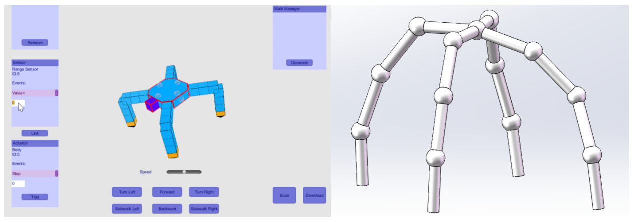
IV. Structure Interpretation and Motion Planning

A. Structure Interpretation and GUI

We developed a software control panel (Fig. 6a) to visualize the motion planning process. To begin with, the robot checks its topological structure automatically on startup and forwards it wirelessly to the screen on the control panel. Any physical structure change after startup can be applied by clicking the *rescan* button. Blocks are displayed in different colors according to their type. One can drag, rotate, or scale

TABLE II: Example of packets of different usage.

	Register	**Sensing**	**Command**
Header	0×FC	0×FC	0×FC
Type	0×F9	0×04	0×F9
ID	0×10	0x×0	0×40
Event	0×01	0×01	0×00
Data H	0×02	0×02	0×02
Data L	0×00	0×FF	0×00
CRC	0×08	0×12	0×37



(a) (b)

Fig. 6: (a) Control panel showing the structure of quadruped robot in Fig. 1a. (b) Abstracted model of a quadruped robot.

the model to examine every part of the robot, and even choose any block to see its status or test its functions.

ID Generation The system utilizes dynamic IDs, such that a control policy will still work after any blocks are replaced. The main body can have up to 16 branches, of which 8 are legs. Every branch can have up to 8 blocks, such that every possible topological position has a unique ID from 0 to 127, which also serves as the IIC address of the device. The main body keeps sending topological packets via its female connectors, indicating the ID from which the branch starts. Any block connected to the main body waits for the packet, initializes the device, and sends the next ID via its female connector. Finally, every block can get its ID according to its topological position.

Scan The main body scans all 128 addresses to see if there are a response and records all valid addresses in a list. Then the main body will keep consolidating sensing data from blocks. The received data packet containing block type, ID, and sensing value will be forwarded to the control panel to be reconstructed.

Reconstruction The control panel records properties of every block, including length, width, height, weight, connector position, and so on. After receiving a packet, the type of data can be used to get these properties, the ID data indicates how these blocks are connected, and the sensing value of servo blocks presents its current rotation. Then we have enough information to easily reconstruct the robot structure and present it on the screen as shown in Fig. 6a.

Upload After the robot structure is reconstructed in the control panel, one can click the *generate* button to produce a corresponding motion policy, including gaits for walking forward, sideways, and turning in place (if able), within a few seconds. The policy is subsequently loaded into the simulator to help users verify the generated motions. Users can also send the policy to the robot by clicking *upload*.

B. Motion Planning

WalkingBot can be configured into various morphologies, it is impossible to pre-design all control policies for its motions. Inspired by the idea of Megaro *et al.* [22], who developed an algorithm to generate a control policy after non-expert users edited the robot's structure (add/remove joints, adjust bone length) or the footfall pattern in the software, our system integrates a similar function such that its structure interpretation and motion planning can be completed automatically

Structure Abstraction We firstly abstract the robot's kinematic to a joint-link structure. The center of each servo block becomes a 1-DOF joint and a link is modeled by the length and weight of corresponding blocks. We also use a matrix to present the relative direction of every joint.

Footfall Pattern Generation In order to speed up the motion planning process and enable a more responsive interaction, we design some general footfall patterns based on the number of legs. After recognizing the robot's structure, these pre-designed footfall patterns can be utilized to generate control policies.

Motion Generation With different parameters such as the walking direction, speed, or turning rate, gaits for different motions (walking forward, sideways, and turning) can also be generated.

We implemented a MPC scheme which is commonly used in robotics to decouple the generation of the Center of Mass (CoM) trajectories, the motion of feet, and full-body joint angles [23]. In particular, a Linear Inverted Pendulum Model (LIPM) is utilized to simplify the relationship between the built robot's Center of Pressure (CoP) and CoM [24]. The discretized dynamics of the LIPM is described as

$$\begin{aligned}\hat{x}_{k+1} &= \begin{bmatrix}1 & T & T^2/2\\ 0 & 1 & T\\ 0 & 0 & 1\end{bmatrix}\hat{x}_k + \begin{bmatrix}T^3/6\\ T^2/2\\ T\end{bmatrix}u_k\\ p_{x,k} &= [1 \quad 0 \quad -z_c/g]\hat{x}_k\end{aligned} \tag{1}$$

where $u_k = \dddot{x}_k$ and $\hat{x}_k = [x_k, \dot{x}_k, \ddot{x}_k]^T$. x_k, $\dot{x}_k$, and $\ddot{x}_k$ are the horizontal position, velocity, and acceleration of the CoM at k-th sample time step, respectively. $p_{x,k}$ is the CoP position planned prior to k, and z_c is the desired control height of the CoM.

According to [24], we can augment this model through N steps' iteration and get an augmented linear dynamic system as shown in Equation 2:

$$\begin{cases} P_{x,k+1} = M_x\hat{x}_k + M_u U_k\\ M_x = \begin{bmatrix}1 & T & T^2/2 - z_c/g\\ \vdots & \vdots & \vdots\\ 1 & NT & N^2T^2/2 - z_c/g\end{bmatrix}_{N\times 3}\\ M_u = \begin{bmatrix}T^3/6 - Tz_c/g & 0 & 0\\ \vdots & \ddots & \vdots\\ (1+3N+3N^2)T^3/6 - Tz_c/g & \dots & T^3/6 - Tz_c/g\end{bmatrix}_{N\times N}\end{cases} \tag{2}$$

where N is the preview horizon, $P_{x,k+1} = [p_{x,k+1}, \ \dots \ , p_{x,k+N}]^T$, $U_k = [\dddot{x}_k, \dddot{x}_{k+1}, \ \dots \ , \dddot{x}_{k+N}]^T$.

Given a reference CoP trajectory $P^{ref}_{x,k+1} = [p^{ref}_{x,k+1}, p^{ref}_{x,k+2}, \ \dots \ , p^{ref}_{x,k+N}]^T$, Equation 2 can be formulated as a *Quadratic Programming* problem, and the objective function is chosen as:

$$\min_{U_k} \frac{1}{2}Q(P_{x,i+1} - P^{ref}_{x,i+1})^2 + \frac{1}{2}RU_k^2 \tag{3}$$

where R and Q are the weight for reference tracking errors and the minimum $\dddot{x}_k$, respectively. The optimal solution to the jerk of x_k is given as [25]:

$$\dddot{x}_k = -e^T((M_u^T M_u + \frac{R}{Q}I)^{-1}M_u^T(M_x\hat{x}_k - P^{ref}_{x,k+1})) \tag{4}$$

where $e = [1, 0, \ \dots \ , 0]^T_{1\times N}$ and I is an identity matrix of N order. This optimal input $u_k = \dddot{x}_k$ yields the desired CoM

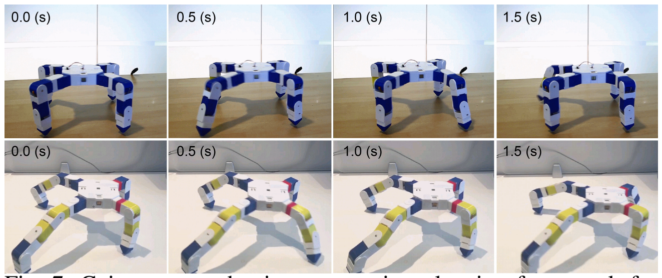


Fig. 7: Gaits generated using our motion planning framework for two robot morphologies.

and CoP trajectories according to Equation 1.

We modified the motion generation framework in [22] to generate a joint angle command for each motor by treating the errors between desired and actual CoM and CoP, and the end-effector positions as the objective of a constrained multi-objective optimization. In particular, [22] treats the CoM as an optimization variable and uses finite difference method to calculate the CoP reference in LIPM model. We instead use the CoM and CoP obtained from the above MPC problem as the reference trajectory to accelerate the optimization process with fewer variables.

Our modified framework can effectively generate a reasonable walking gait, and the different CoM trajectories can be used to guide the robot walk to different directions or change walking speed. Finally, we implement a position control to track the generated gait and output a table of servo positions to be uploaded into the robot. With well-tuned PID parameters of motors, the robot is able to walk stably; examples are shown in Fig. 7.

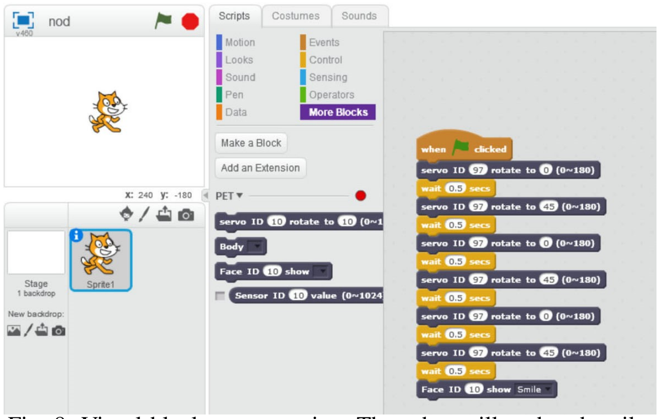
Noted that not every robot configuration supports walking. Although the algorithm sometimes does produce a solution, the robot may not be able to execute that properly and some basic instructions to users of creating a walkable robot are needed. We will discuss this in Section VI.

V. Programming Interface

Two programming methods, targeting different levels of programming skill, are further developed for non-expert users, *e.g.* children, to create their interactive robots.

Visual Block Programming is a framework utilized by many education software such as the MIT Scratch [18]. We developed a plug-in for the Scratch software, such that users can easily program the robot in Scratch (Fig. 8). The plug-in works together with the control panel. One can look up the ID of every block in the control panel. In Scratch, they drag sensor blocks to read a value or drag actuator blocks to execute some actions. We also integrate the visual block programming into the control panel, such that available blocks will turn up automatically without users' efforts in retrieving the IDs.

Events Programming While the Scratch utilizes intuitive visual blocks, the programming framework is still the same as traditional programming, which requires the understanding of concepts, such as If, Loop, *etc*.. Considering users with less experience in programming, we provide a more intuitive way to interpret how a robot really works. This utilizes a simplified subsumption architecture [26] in robotics. Every



```
when [flag] clicked
servo ID 97 rotate to 0 (0~180)
wait 0.5 secs
servo ID 97 rotate to 45 (0~180)
wait 0.5 secs
servo ID 97 rotate to 0 (0~180)
wait 0.5 secs
servo ID 97 rotate to 45 (0~180)
wait 0.5 secs
servo ID 97 rotate to 0 (0~180)
wait 0.5 secs
servo ID 97 rotate to 45 (0~180)
wait 0.5 secs
Face ID 10 show Smile
```

Fig. 8: Visual block programming. The robot will nod and smile.

sensor has a list of input events, and every actuator (including the main body) has a list of output events. An input event and an output event can be associated together with some parameters, such that an event-response control rule is created (Fig. 9). Multiple rules can be sorted in a stack and work together. If any conflict occurs, the actuator will work according to the one with higher priority in the stack. The robot can follow these rules to interact with the environment, and act like real living creatures.

Comparison A pilot study is conducted to compare the two programming interfaces, and the results are shown in Fig. 10. A total of 17 participants are invited to evaluate the two programming interfaces based on score 1 to 10 for four criteria: easy-to-use, flexibility, efficiency, and personal preference. The result (two-tailed t-test)indicates that Events Programming is easier to use ($p = 0.041$) but less flexible ($p = 0.003$) compared to Visual Block Programming.

VI. Discussion and Conclusion

A. *Discussion*

Gait Pattern In the current stage, we use pre-designed footfall patterns to generate gait's control policy. Meanwhile, some configurations of *WalkingBot*, *e.g.* those with an odd number of legs, cannot walk stably; the system produces an awkward or cumbersome walking gait.

Indeed, we can hardly find natural creatures with 3 or 5 legs, which creates a challenge for footfall pattern design. A future direction of *WalkingBot* would be developing an automated approach to generate the footfall pattern and couple that with higher-level features such as personality and emotion. In addition, the system can be used for *education* proposes by providing knowledge of animal (insect) locomotion to children through creating robots with higher mobility.

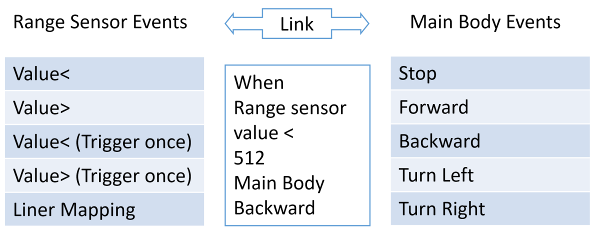


Fig. 9: Events programming. The robot will go backward if anything approaches it.

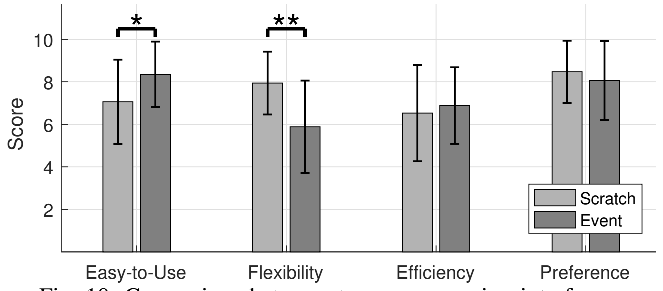


Fig. 10: Comparison between two programming interfaces.

Programming Interface Visual block programming and events programming have different trade-offs. Visual block programming is similar to a traditional programming method that requires a higher level of skill. Events programming is easier to use but could be over complicated when there are too many rules working together. The visual block programming helps naive users (*e.g.* children) to improve their programming skills, while the events programming is aimed to help them to learn about the logic relationship in an interactive robot system. Both methods can provide simple access to create an interactive legged robot.

Limitations Some configurations of the robot cannot walk well in the real-world even though a desired gait is generated. The friction coefficient of the surface may be an important factor for this issue, and a footpad for foot blocks can be introduced to adapt to the ground of different materials.

In addition, we will develop more sensor and actuator blocks to expand its functionality and capability, which can encourage childrens imagination, creativity, and support creative learning.

B. Conclusion

This paper presented *WalkingBot*, a modular multi-legged robot system that can automatically interpret the robot's kinematic structure through an integrated hardware and software design, and generate a walking gait using a MPC scheme. Visual block programming and event programming are introduced for non-expect users, showing the potential of the proposed system in *e.g.* education, human-robot interaction.

Acknowledgement

We thank Ms. Jiasi GAO and Dr. Yixin Zhu at DMAI for the creative discussion. This work is supported by National Key Research and Development Plan of China under Grant No. 2016YFB1001402, Tsinghua Shuimu Scholarship, and Tsinghua University Initiative Scientific Research Program (20197010003).

References

[1] J. Leong, F. Perteneder, H.-C. Jetter, and M. Haller, "What a life!: Building a framework for constructive assemblies," in *Proceedings of the Eleventh International Conference on Tangible, Embedded, and Embodied Interaction*, pp. 57–66, ACM, 2017.

[2] S. Ha, J. Kim, and K. Yamane, "Automated deep reinforcement learning environment for hardware of a modular legged robot," in *2018 15th International Conference on Ubiquitous Robots (UR)*, pp. 348–354, IEEE, 2018.

[3] J. Daudelin, G. Jing, T. Tosun, M. Yim, H. Kress-Gazit, and M. Campbell, "An integrated system for perception-driven autonomy with modular robots," *Science Robotics*, vol. 3, no. 23, p. eaat4983, 2018.

[4] X. Lin, H. Krishnan, Y. Su, and D. W. Hong, "Multi-limbed robot vertical two wall climbing based on static indeterminacy modeling and feasibility region analysis," in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018.

[5] J. J. Craig, *Introduction to robotics: mechanics and control, 3/E*. Pearson Education India, 2009.

[6] X. B. Peng, G. Berseth, and M. Van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 81, 2016.

[7] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Robotics: Science and Systems (RSS)*, 2018.

[8] T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," in *Robotics: Science and Systems (RSS)*, 2018.

[9] D. Anderson, J. L. Frankel, J. Marks, D. Leigh, E. Sullivan, J. Yedidia, and K. Ryall, "Building virtual structures with physical blocks," in *Proceedings of the 12th annual ACM symposium on User interface software and technology*, pp. 71–72, ACM, 1999.

[10] R. Watanabe, Y. Itoh, M. Asai, Y. Kitamura, F. Kishino, and H. Kikuchi, "The soul of activecube: implementing a flexible, multimodal, three-dimensional spatial tangible interface," *Computers in Entertainment (CIE)*, vol. 2, no. 4, pp. 15–15, 2004.

[11] A. Jacobson, D. Panozzo, O. Glauser, C. Pradalier, O. Hilliges, and O. Sorkine-Hornung, "Tangible and modular input device for character articulation," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 82, 2014.

[12] M. Wang, K. Lei, Z. Li, H. Mi, and Y. Xu, "Twistblocks: Pluggable and twistable modular tui for armature interaction in 3d design," in *Proceedings of the Twelfth International Conference on Tangible, Embedded, and Embodied Interaction*, pp. 19–26, ACM, 2018.

[13] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[14] A. Golovinsky, M. Yim, Y. Zhang, C. Eldershaw, and D. Duff, "Polybot and polykinetic/spl trade/system: a modular robotic platform for education," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2004.

[15] S. Kalouche, D. Rollinson, and H. Choset, "Modularity for maximum mobility and manipulation: Control of a reconfigurable legged robot with series-elastic actuators," in *Safety, Security, and Rescue Robotics (SSRR), 2015 IEEE International Symposium on*, pp. 1–8, IEEE, 2015.

[16] J. Kim, A. Alspach, and K. Yamane, "Snapbot: A reconfigurable legged robot," in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017.

[17] S. Papert, *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., 1980.

[18] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, p. 16, 2010.

[19] "Lego education." Accessed: 2018-08-31.

[20] H. S. Raffle, A. J. Parkes, and H. Ishii, "Topobo: a constructive assembly system with kinetic memory," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 647–654, ACM, 2004.

[21] E. Schweikardt and M. D. Gross, "roblocks: a robotic construction kit for mathematics and science education," in *Proceedings of the 8th international conference on Multimodal interfaces*, pp. 72–75, ACM, 2006.

[22] V. Megaro, B. Thomaszewski, M. Nitti, O. Hilliges, M. Gross, and S. Coros, "Interactive design of 3d-printable robotic creatures," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, p. 216, 2015.

[23] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, "Biped walking pattern generation by using preview control of zero-moment point," in *International Conference on Robotics and Automation (ICRA)*, 2003.

[24] J.-w. Luo, Y.-l. Fu, and S.-g. Wang, "3d stable biped walking control and implementation on real robot," *Advanced Robotics*, vol. 31, no. 12, pp. 634–649, 2017.

[25] P.-b. Wieber, "Trajectory free linear model predictive control for stable walking in the presence of strong perturbations," in *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pp. 137–142, IEEE, 2006.

[26] R. Brooks, "A robust layered control system for a mobile robot," *IEEE journal on robotics and automation*, vol. 2, no. 1, pp. 14–23, 1986.