Immediate Generation of Jump-and-Hit Motions by a Pneumatic Humanoid Robot Using a Lookup Table of Learned Dynamics

Kazutoshi Tanaka, *Member, IEEE*, Satoshi Nishikawa, *Member, IEEE*, Ryuma Niiyama, *Member, IEEE*, and Yasuo Kuniyoshi, *Member, IEEE*

***Abstract*—This letter focuses on the jump-and-hit motion of a humanoid robot, wherein a robot instantaneously jumps forward and hits a flying ball in the air, similar to how human players behave in volleyball games. We propose a Immediate Motion generation using a Lookup table of learned dynamics (IMoLo) for generating the motions of a pneumatic humanoid robot. To test this method, we developed a humanoid robot called "Liberobot" with eight joints applying structure-integrated pneumatic cable cylinders. Using simulations, the prediction errors of the robot hand positions during the jump-and-hit motions measured via nonlinear interpolation when using IMoLo was smaller than without it in cases having a small number of training trials. In the experiments, the robot jumped and hit the flying ball 16 times out of 20 trials using the proposed motion generation method. The results indicate that a pneumatic humanoid robot using IMoLo can instantaneously perform dynamic whole-body motions, such as jump-and-hit motions, with a changing target within a specified time. Our humanoid robot is the first pneumatic humanoid robot capable of executing such dynamic motions.**

***Index Terms*—Machine learning for robot control, hydraulic/pneumatic actuators.**

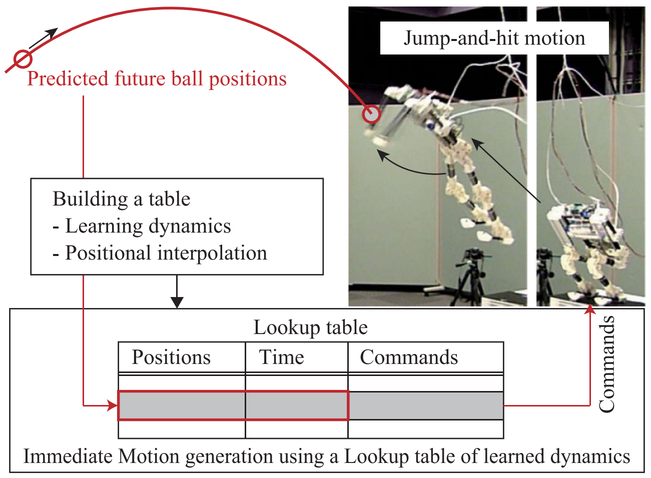
Predicted future ball positions
Jump-and-hit motion
Building a table
- Learning dynamics
- Positional interpolation
Lookup table
Positions
Time
Commands
Commands
Immediate Motion generation using a Lookup table of learned dynamics

Fig. 1. Concept of this study. A pneumatic humanoid robot immediately generates and executes the jump-and-hit motion using a lookup table of learned dynamics.

I. Introduction

We are interested in a humanoid robot that performs dynamic motions. In particular, we focus on the whole-body dynamic motions of a humanoid robot that are instantaneously generated in response to a changing target within a specified time. A jump-and-hit motion (Fig. 1) describes a coordinated whole-body dynamic action in which a humanoid robot must adopt a posture of readiness to jump. It must then detect the ball, predict its trajectory, generate coordinated motion, and strike the ball before it lands. The actuating system of the humanoid robot must be able to accelerate its entire body to jump and swing its arm simultaneously.

Pneumatic actuators have a higher power-to-weight ratio compared with electrical and hydraulic ones, while electrical ones have a higher accuracy of position control, and hydraulic ones generate larger forces. Therefore, pneumatic actuators such as pneumatic cylinders and pneumatic artificial muscles (PAMs) have been used with dynamically moving robots so that they can jump, owing to their lightness and high actuator power. Raibert presented a hopping robot using a pneumatic cylinder [1]. Niiyama *et al.* developed a jumping robot [2] and a running biped robot [3], which were equipped with musculoskeletal body structures using PAMs. Although pneumatic actuators have advantages for dynamic robotic motions, the response to commands is slow, and the accuracy of the position control is low. Thus, it is difficult for a pneumatic humanoid robot to accurately follow the planned trajectory of its end effector via a high-speed motion with regards to a changing target.

The purpose of this research is to develop a method of immediately generating and executing jump-and-hit motions for a pneumatic humanoid robot. Hence, we propose **I**mmediate **Mo**tion generation using a **Lo**okup table of learned dynamics (IMoLo), which enables a pneumatic humanoid robot to instantaneously generate a motion in response to a changing target

Manuscript received December 17, 2020; accepted April 10, 2021. Date of publication April 30, 2021; date of current version May 25, 2021. This letter was recommended for publication by Associate Editor L. Kaul and Editor T. Asfour upon evaluation of the reviewers' comments. This work was supported by JSPS KAKENHI under Grants JP18K18087, JP18H05466, and JP19K14936. *(Corresponding author: Kazutoshi Tanaka.)*

The authors are with the Graduate School of Information Science, and Technology, Mechano-informatics, University of Tokyo, Tokyo 113-0033, Japan (e-mail: tanaka@isi.imi.i.u-tokyo.ac.jp; nisikawa@isi.imi.i.u-tokyo.ac.jp; niiyama@isi.imi.i.u-tokyo.ac.jp; kuniyosh@isi.imi.i.u-tokyo.ac.jp).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2021.3076959, provided by the authors.
Digital Object Identifier 10.1109/LRA.2021.3076959

within a specified time period (Fig. 1). To test this proposed method, we developed a humanoid robot named "Liberobot" using structure-integrated pneumatic cable cylinders (SIP-CCs) that enable the robot to be lightweight while generating sufficiently high power. The main contribution of this letter is IMoLo, which immediately generates a dynamic motion of a pneumatic humanoid robot in response to a changing state and timing target.

II. Related Works

Control methods for catching and hitting flying objects have been proposed. Birbach *et al.* introduced a method that enables a wheeled humanoid robot to perceive two flying balls in real time and to catch them both [4]. The catching motion was generated by solving a nonlinear optimization problem having nonlinear constraints [5]. Kim *et al.* presented a method of catching a flying object using machine-learning techniques [6]. The robots in the study learned the flying dynamics of the objects and the reachable and graspable spaces and generated a trajectory for their arm before performing the catching action.

Senoo *et al.* proposed a batting method for a robotic arm using high-speed cameras [7]. The position and timing of hitting were optimized to allow a ball to reach a target position after hitting it. Their robotic arm trajectory was planned using a fifth-polynomial function of time based on the position and timing, and the arm followed this trajectory using proportional-integral-derivative control. Mülling *et al.* adopted a machine-learning approach that enabled a robotic arm to play table tennis [8]. The robot in their study learned a gating network for various templates of swing motions to generate a paddle trajectory prior to playing [9]. Huang *et al.* presented a method of generating a trajectory of a paddle for robotic table-tennis tasks [10]. A robot arm using this method generates a trajectory using a database constructed beforehand and the function learned using reinforcement learning. In their method, database of ball trajectories were applied using regression with experience data. Jia *et al.* presented a method for hitting a flying object toward a target [11]. The feasible arm state for the batting action was determined from an iterative computation using analytical models of flying objects and those of the impact between the objects and the robot.

Note that by using the aforementioned methods, robots planned the trajectory of their arm, and the arm followed this trajectory to complete the task. However, it is difficult for a pneumatic robot to follow a planned trajectory using feedback control, owing to the slow responses of pneumatic actuators. Thus, selecting a high-speed motion parameter and executing this motion in a feedforward manner in IMoLo is suitable for generating motion by a pneumatic robot.

Büchler *et al.* demonstrate the safe learning of table tennis by robot arms driven by PAMs using model-free reinforcement learning [12]. A robot arm using this method can directly decide optimal output commands to pneumatic valves from the observed state. The softness and back-drivability properties of PAMs in their method allowed the arm to safely learn paddle operations within 14 h. A biped humanoid robot cannot safely learn jump-and-hit motions, owing to the mechanical property of PAMs: they will break.

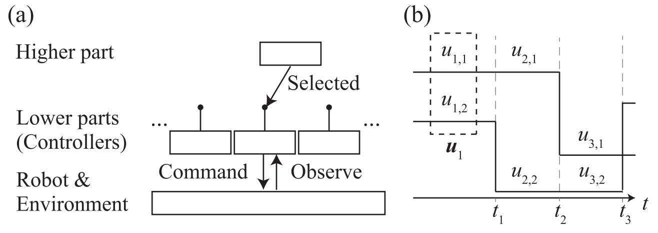

Fig. 2. (a) Hierarchical system of robots. (b) Control target switching.

A humanoid robot can immediately generate a reaching-motion trajectory by computing the positions needed prior to executing the reaching motion. Guilamo *et al.* introduced a method that performed the advanced computation using a database containing the reachable positions of a humanoid robot equipped with redundant joints, which allowed fast querying of the joint configurations with high manipulability [13]. Guan and Yokoi used a database of reachable positions to generate a method based on the Monte Carlo algorithm [14]. These advanced computations enable robots to instantaneously generate motions. IMoLo follows this line of implementation employing a database.

III. Motion Generation With Target Switching

The behavior of a robot is often generated using a hierarchical system that contains higher and lower parts (Fig. 2(a)). The higher part selects a lower part for concerted actions. These lower parts are controllers. Each controller has a target. The selected lower part decides the commands to converge the state of the robot to the target of the control part.

Several variables are used for the control target, such as the position and posture of the end effector in point-to-point control, the velocity of the center of gravity of a humanoid robot, and the zero-moment point [15] of a legged robot to control its balance. Variables of a pneumatic robot are also used for the target, such as the open/closed states of a pneumatic valve, flow rates through the valve, and air pressure of a chamber. Liberobot is equipped with pressure-proportional valves. Thus, we used the target pressure of the valves as the control target, $\mathbf{u}$.

The higher part in a kind of hierarchical system observes the state of the robot and switches the lower parts as options in the reinforcement-learning framework [16]. The higher part in another hierarchical system switches the lower parts after a certain time has elapsed. In the latter system, the state of the robot is not required to converge to the control target before switching the lower parts. Thus, this type of system is suitable for and has actually been used for generating dynamic motions of pneumatic robots, which has difficulty following a position trajectory [3], [17], [18]. We also focus on this type of system and use it to generate the jump-and-hit motion of a pneumatic humanoid robot.

In this system, the controllers follow the control target, $\mathbf{u}$. The higher part switches $\mathbf{u}$ at every switching time, t_i, and $\mathbf{u}$ does not change prior to the switching (Fig. 2(b)). $\mathbf{u}$ is represented by a vector, $\mathbf{u}_i = [u_{i,1}, u_{i,2}, \dots]^\top$, which is $\mathbf{u}$ in the i^{th} phase,

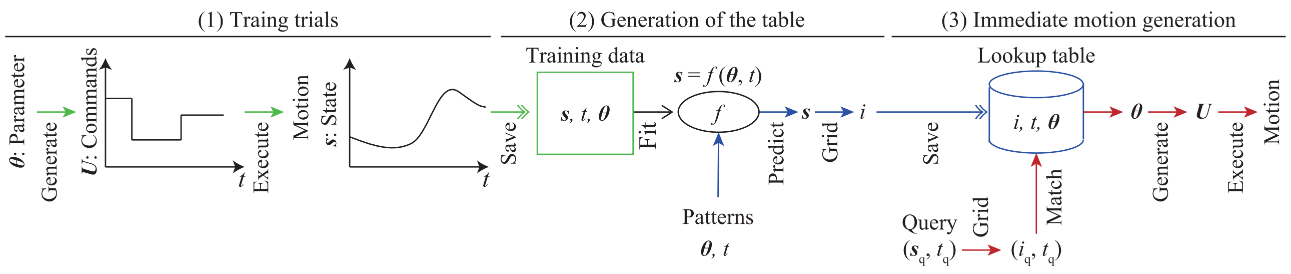

Fig. 3. Schematic of IMoLo.

where $u_{i,j}$ is the j^{th} variable of the control target in the i^{th} phase. The time-series of the control target is represented by a vector, $\mathbf{U} = [t_1, t_2, \ldots, \mathbf{u}_1^\top, \mathbf{u}_2^\top, \ldots]^\top$.

The dynamic motion of a pneumatic robot to the changing target can be generated from $\mathbf{U}$. A part of motion, such as the position of the robot hand, can be changed by changing a part of $\mathbf{U}$. As one of the simplest methods of changing a part of $\mathbf{U}$, the target in the c^{th} phase is changed as

$$\mathbf{u}_c = \mathbf{u}_{\text{m}} + k\mathbf{u}_k, \tag{1}$$

where $\mathbf{u}_{\text{m}}$ and $\mathbf{u}_k$ are vectors, and k is a variable. This method enables a robot to change positions, wherein the hand of this robot reaches the target in the jump-and-hit motions by only changing k. Thus, we used this method to generate the motions.

IV. IMoLo

In this study, we aimed to let a pneumatic humanoid robot immediately start the motion, jump forward, swing its arm, and reach its hand to the sriking position at which a flying ball will be hit. We assumed that the robot anticipates the ball and takes the same ready posture prior to launch. We developed a method for generating this motion: IMoLo. See Fig. 3).

A robot using IMoLo can immediately generate motion using a lookup table (Fig. 3(3)), which shows the time series of the command using the hand of the robot to reach the target position at the desired time. Thus, the robot can also start this motion at the appropriate time and reach the hand to the position at the target timing. The robot is not required to follow any trajectory; it only needs to execute the commands.

A. Main Procedure

A robot using IMoLo constructs a lookup table and generates motions as follows. First, this robot creates the time series of commands, $\mathbf{U}$, from a motion parameter, $\boldsymbol{\theta}$. It executes $\mathbf{U}$, performs the motion, and records the tuples $(\mathrm{s}, t, \boldsymbol{\theta})$, where t is the time elapsed since the initiation of the motion, and s is the state of the robot at t (Fig. 3(1)). The robot repeats this changing $\boldsymbol{\theta}$. Then, it constructs a lookup table using these tuples (Fig. 3(2)).

Finally, it generates a motion referring to this table (Fig. 3(3)). The robot obtains the query of the state, s_{q}, and the time, t_{q}. The query requires that the state of the robot, s, reaches s_{q} after t_{q}. The robot inputs the query to the table, and the table outputs whether the table contains a tuple that includes both s and t close enough to s_{q} and t_{q}, respectively. In this study, we defined the two times, t_{q} and t, and the two states, s_{q} and s, to be enough close to each other when $|t_{\text{q}} - t| < \Delta t/2$ and when $||\mathrm{s}_{\text{q}} - \mathrm{s}|| < d_{\text{grid}}/2$, respectively, where Δt is the time step, and d_{grid} is the grid interval described in the next subsection, respectively. If the table outputs *true*, the table provides the robot with $\boldsymbol{\theta}$ in this tuple, and the robot creates $\mathbf{U}$ from this $\boldsymbol{\theta}$ and executes $\mathbf{U}$.

B. Basics of IMoLo

Here, we provide some details information about IMoLo, which help improve efficiency.

1) Nonlinear Interpolation Using Forward Models: Generating the lookup table (Fig. 3(2)) only from samples acquired from trials of the robot results in a trade-off. That is, conducting many trials to obtain dense s values increases the risk of the robot being damaged. This risk decreases as the number of trials decreases, but the distribution of s could be sparser. The robot thus estimates s generated from $\boldsymbol{\theta}$, which it has not yet tried using nonlinear interpolation, and adds the sets to the table. Thus, using the training trials data, the robot approximates the nonlinear functions, f, predicting s as

$$\mathrm{s} = f(\boldsymbol{\theta}, t). \tag{2}$$

Any nonparametric regression methods, such as a computational neural network [19], Gaussian process regression [20], and a ν-support vector regression [21] can be used as a nonlinear approximation.

Using the inverse model used to calculate $\boldsymbol{\theta}$ and t from a given s is another method for nonlinear interpolation. However, inverse models can introduce errors, as with the case shown in Fig. 4(a). In this case, similar samples in the input space may result in different samples in the output space and a large prediction error. Thus, we adopt forward models (Fig. 4(b)).

Additionally, with IMoLo, the control target is switched as described in Section III, and $\mathbf{U}$ is changed using Eq. (1) to decrease the number of variables in $\boldsymbol{\theta}$.

2) Voxelization: The number of s can be large and biased in a case where the robot saves all estimated s in the table. Hence, one area might include many s values, whereas another area might consist of only a few s values. In this case, searching for the nearest s among all s values would be time-consuming. Additionally, the table contains unused s values that are almost the same. Thus, it has considerable computational costs and memory consumption. Therefore, the robot uses grid points

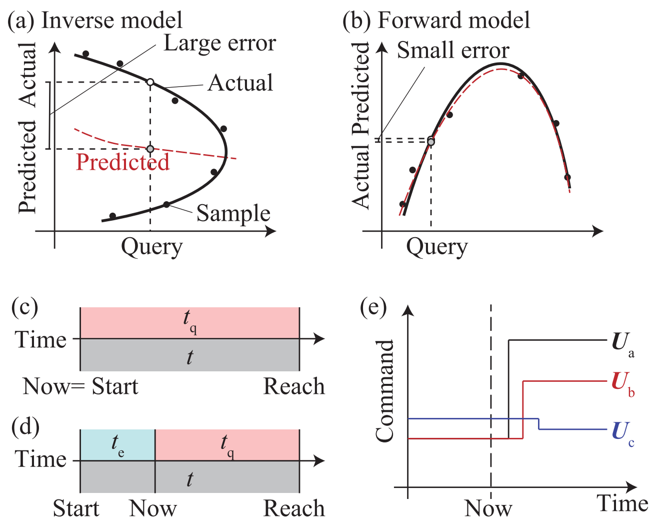

Fig. 4. (a) Inverse model and (b) Forward model. (c–e) Update of commands. (c) Comparison of query t_q with the time in table t before starting a motion. (d) Comparison after starting a motion. (e) Changeable time series of commands. The robot can change $\mathbf{U}$ from $\mathbf{U}_a$ to $\mathbf{U}_b$, but it cannot change $\mathbf{U}$ from $\mathbf{U}_a$ to $\mathbf{U}_c$.

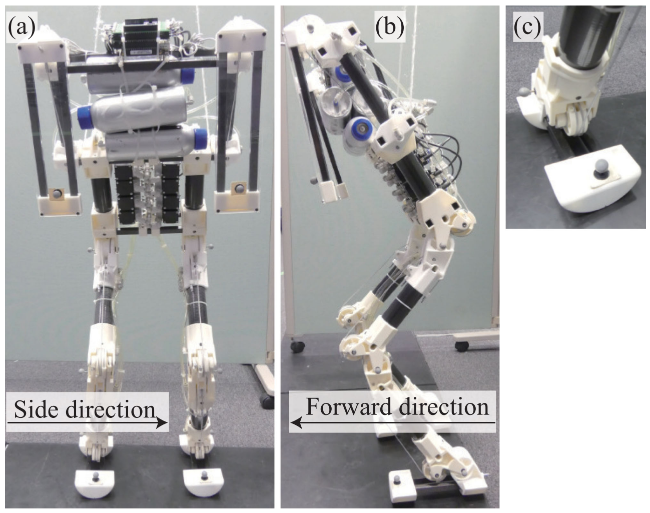

Fig. 5. Liberobot.

TABLE I
SPECIFICATIONS OF THE ROBOT

Actuators	Structure integrated pneumatic cable cylinder
Valves	Pressure-proportional valves (Tecno-basic; Hoerbiger)
Air compressor	SLP-221EBD; ANEST IWATA Corp. (external)
Air tank	550-ml aluminum tank
Sensors	Pressure sensor (PSE560; SMC), Motion-capture camera (Prime13W; NaturalPoint, Inc.)
CPU	BeagleBone Black, BeagleBoard.org
Materials	Nylon, CFRP pipes

and indices to increase the speed at which $\boldsymbol{\theta}$ is selected as the *voxelization*.

The robot arranges $\mathbf{s}_{\mathrm{grid}}$ grid points evenly at intervals of d_{grid} in the area where $\mathbf{s}$ is distributed. Each grid point, $\mathbf{s}_{\mathrm{grid}}$, sets its index as i. The robot searches for the nearest $\mathbf{s}_{\mathrm{grid}}$ from all $\mathbf{s}$, calculates i, and obtains tuples $(i, t, \boldsymbol{\theta})$. Unique sets of these tuples are stored in a table (Fig. 3 (2)).

Note that some $\mathbf{s}$ have the same nearest $\mathbf{s}_{\mathrm{grid}}$ and give the same i, but no $\mathbf{s}$ gives an i. Therefore, the number of i in this table is smaller than that of all predicted $\mathbf{s}$. When selecting the unique sets, if several $\mathbf{s}$ results in the same nearest $\mathbf{s}_{\mathrm{grid}}$ and i, one $\mathbf{s}$ is randomly selected.

3) Update of Commands: The robot should update $\mathbf{U}$ after initiating a motion in some cases. For example, the predicted future positions of a flying ball $\mathbf{p}_{\mathrm{ball}}$ are used as $\mathbf{s}_{\mathrm{q}}$ in the jump-and-hit motions in this study. In such a case, the robot should repeat predicting the positions, obtain new $\mathbf{s}_{\mathrm{q}}$, and update $\mathbf{U}$ to compensate for prediction errors.

The tuple $(i, t, \boldsymbol{\theta})$ indicates that the state reaches the i^{th} grid point at time t after the onset of a motion if the robot generates and executes this motion using $\boldsymbol{\theta}$. Thus, the robot selects a tuple such that t is sufficiently close to the future time, t_{q}, before the onset of motion (Fig. 4(c)). After a time, t_{e}, from initiating a motion, the robot selects another tuple such that t is sufficiently close to $t_{\mathrm{q}} + t_{\mathrm{e}}$ (Fig. 4(d)). The robot should select a tuple having a new motion parameter, $\boldsymbol{\theta}$, to generate a new $\mathbf{U}$ such that the already executed parts of an old $\mathbf{U}$ match parts prior to time t_{e} of this new $\mathbf{U}$ (Fig. 4(e)).

C. Limitation

IMoLo can be used for other tasks in which the control targets are switched to correspond with a movement of an object, such as the pick and place task of the robot arm while avoiding a moving object, catching a thrown object, and fast walking while avoiding humans. IMoLo works without any additional learning when conditions change, such as a moving ball during a hitting task as long as the robot can manage this changing by changing control target and achieve its goal.

IMoLo has some limitations. A robot using IMoLo learns motions from the same initial state. Thus, it will need to reach its initial state each time before performing the learned motion. Additoinally, a robot using IMoLo will perform many different patterns of motions generated from different motion parameters to learn new motions. Conducting trials to learn models in IMoLo could damage the robot.

V. LIBEROBOT: A PNEUMATIC HUMANOID ROBOT EXECUTING JUMP-AND-HIT MOTIONS

Experiments to test IMoLo require a pneumatic humanoid robot that performs jump-and-hit motions. Such a robot should be light enough to decrease the damage of a dynamic motion while generating high-power. However, such robots are currently unavailable on the market. Thus, we developed a pneumatic humanoid robot called "Liberobot" (Fig. 5).

A. Mechanical System

Fig. 5 presents the appearance of Liberobot. Table I lists its specifications. The robot has a height of 1163 mm and width of

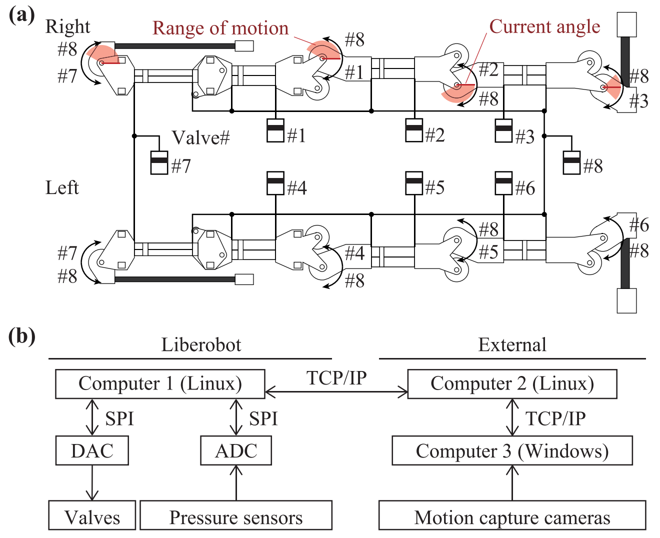

Fig. 6. (a) Connections between cylinder chambers and valves. (b) Software communication for Liberobot.

400 mm, and it weighs 7.4 kg without its power cable, local area network (LAN) cable, and external air tube. The robot has two legs and two arms. We attached an acrylic plate to each arm to enable it to hit a ball. The size of this plate is 300 × 80 mm, and its thickness is 3 mm. The shorter side of the plate corresponds to the side direction in Fig. 5(a). The robot has eight joints: one shoulder joint for each arm, and hip, knee, and ankle joints in both legs. The range of motion of the joints is from 0 to 150° for the shoulder, hip, and knee, and from −45 to +45° for the ankle relative to the posture in Fig. 6(a). The lengths between the shoulder and hip, hip, and knee, and knee and ankle are 446, 290, and 310 mm, respectively. The length of the feet is 211 mm.

We arranged the axes of all joints in parallel as pitch joints orthogonal to the sagittal plane. Liberobots with flat soles can only generate motions along the sagittal plane; it cannot reach different hand positions in the side direction (Fig. 5(a)) during the jump-and-hit motions. Attaching other actuated joints with axes along directions allow the hand to reach different positions in the side direction while increasing the weight of the robot. Thus, inspired by research on the shapes of the soles of biped robots [22], we equipped ours with non-flat soles instead of attaching extra joints (Fig. 5(c)). The soles have a semicircular shape with a radius of 48 mm and an axis aligned in the forward direction (Fig. 5(b)). We selected this semicircular shape as a simple curve of the non-flat sole. The sole is rolled to the side by the force biased toward the side by only the joint torque on the pitch axes, and the rolling shifts the jump-and-hit motions in the roll direction. The roll motion enables the robot with the joints of the same axis to reach the hand to different lateral positions.

Two cylinders having an inner diameter of 25 mm drive the arms of the robot, and cylinders having a diameter of 40 mm actuate the legs. The thickness of the cylinder tube of these cylinders is 1 mm. The cylinder tubes are made of carbon fiber reinforced plastic (CFRP). The length of the joint moment arm is 26 mm. Speed-reduction pulleys having a speed reduction ratio of two are attached to the leg joints to generate a large torque. These parameters were determined as described in our previous work [23].

B. Control System

Electrical power is externally supplied to the robot via electric cables. The robot computer was provided with a power of 5 V, and the air-pressure sensors and valves were provided with a power of 24 V. An external compressor (SLP-221EBD, ANEST IWATA Corp.) supplied compressed air to the robot. This compressor sends compressed air to the cylinder through a polyurethane tube with inner and outer diameters of 4 and 6 mm, respectively, and four aluminum tanks on board with a volume of 550 ml each.

The robot was equipped with eight pressure-proportional valves (Tecno-basic, Hoerbiger). Fig. 6(a) shows the connection between these valves with the chambers and valve numbers. Using one valve to control one chamber of a cylinder (i.e., using twice the number of valves as cylinders) increases the weight of the robot. Therefore, we decreased the number of valves so that one valve (#7) controlled the two chambers that elevate the arms, and another valve (#8) controlled the two chambers that depress the arms and the six chambers that bend the legs.

The robot has eight pressure sensors (PSE560, SMC) onboard and receives motion data from 11 external motion-capture cameras (Prime13 W, NaturalPoint, Inc.) installed in the room. We connected these pressure sensors to the tubes between the valves and chambers to debug the robot motion data and analyze the results. These cameras were installed to measure the positions of the ball and body parts of the robot.

Fig. 6(b) shows the software communication channels used in Liberobot. The robot has a single onboard computer (i.e., computer 1, BeagleBone Black, BeagleBoard.org), which controls the communication of the robot with the external computers (i.e., computer 2 and computer 3) via a LAN cable, pressure sensors via an analog-to-digital converter (ADC) board, and valves via a digital-to-analog converter (DAC) board. Computer 1 receives the commands for the output of the robot from an external computer (computer 2, Intel Xeon, 3.10 GHz) and sends these to the valves via a DAC board. Computer 1 also receives data from the pressure sensors via an ADC board and sends these to computer 2. The software running on computer 3 (Intel Core i7-3770, 3.40 GHz) receives the positions of the motion-capture markers from the software of the motion-capture system (MOTIVE; NaturalPoint, Inc.) running on the same machine via NatNetSDK, which serves as the application programming interface. It sends these positions to computer 2, which then computes the commands. These three computers communicate with each other via TCP/IP protocol.

C. Ball-Trajectory Prediction

Liberobot observes the position of the ball using the positions of the markers measured by the motion-capture cameras. When the robot detects a marker located farther from the robot than 2.2 m, the robot recognizes this marker as the ball. The robot can also detect the position of the ball by directly processing

TABLE II
MOTOR COMMANDS [0.1 MPA] FOR JUMP-AND-HIT MOTIONS

t_i [s]	$\mathbf{u}$	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8
0.10	$\mathbf{u}_1$	$\mathbf{u}_\mathrm{m} + k\mathbf{u}_k$							
t_w	$\mathbf{u}_2$	6.0	4.0	1.0	6.0	4.0	1.0	0.0	0.5
1.20	$\mathbf{u}_3$	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0
	$\mathbf{u}_\mathrm{m}$	0.5	2.5	0.2	0.5	2.5	0.2	0.0	0.5
	$\mathbf{u}_k$	0.0	1.5	0.0	0.0	-1.5	0.0	0.0	0.0

a camera image, as in a previous work [4]. In this study, we adopted the method of detection using a marker, because it is a simpler than camera-image processing.

The state of the ball is represented as a vector, $\mathbf{x}(t) = [\mathbf{p}_\mathrm{ball}(t)^\top, \mathbf{v}_\mathrm{ball}(t)^\top]^\top$, where $\mathbf{p}_\mathrm{ball}(t)$ and $\mathbf{v}_\mathrm{ball}(t)$ are vectors that represent the position and velocity of the ball at time t, respectively, after the current time. We assume that $\mathbf{x}(t)$ changes as a system according to the linear equation, $\mathbf{x}(t + \Delta t) = A\mathbf{x}(t) + \mathbf{B}$, where A is a matrix, $\mathbf{B}$ is a vector, and Δt is the time step as in [24]. We set the parameters in A and $\mathbf{B}$ by throwing the ball from and to various positions 10 times, measuring the ball, and evaluating the parameters using the leave-one-out method. The robot estimated $\mathbf{p}_\mathrm{ball}(t)$ and $\mathbf{v}_\mathrm{ball}(t)$ using a Kalman filter [25] from the position of the ball measured by the motion-capture cameras and predicted the future $\mathbf{p}_\mathrm{ball}$ by iteratively calculating this linear equation. These $\mathbf{p}_\mathrm{ball}$ and t were used as $\mathbf{s}_\mathrm{q}$ and t_q in IMoLo, respectively.

D. Generation of the Jump-and-Hit Motions

We set the vector of the motion parameter, $\boldsymbol{\theta}$, described in Section IV and generated $\mathbf{U}$ as

$$\boldsymbol{\theta} = [k,\, t_w]^\top, \tag{3}$$

where k is a variable in Eq. 1, and t_w is the w^th timing of the switching control target (Fig. 2(b)). The Liberobot can change the positions where the hand reaches by changing only k and t_w.

Table II presents the center of command changes, $\mathbf{u}_\mathrm{m}$, the series of commands, $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$, the unit commands of the command changes, $\mathbf{u}_k$, and switching time, t_i. The valve numbers in this table are the same as in Fig. 6. To generate the jump-and-hit motions by Liberobot, we divided the jump-and-hit motions into three phases: extending the knees to lean forward, extending the legs to jump, and swinging the arms to hit the ball. To generate these motions, we set $c = 1\,(\mathbf{u}_1 = \mathbf{u}_\mathrm{m} + k\mathbf{u}_k)$ and the changed value of the left knee (#5) in $\mathbf{u}_k$ to the opposite value of that of the right knee (#2). This was intended to change the balance on the ground reaction force from kicking by using the left and right legs of the robot and the position of its hand in the side direction when hitting in the air. We also set $w = 2\,(t_w = t_2)$ to adjust the timing at which the robot starts to swing its arm. We determined the commands in the table using a trial-and-error approach. We determined the phase number as the minimum number needed for the robot to jump and hit a ball, which then flies to various positions. We added the phase to lean forward and decided the balance of the output of the joint in the leg in order to jump high and far.

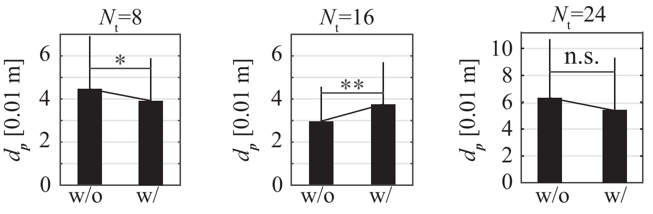

Fig. 7. Errors when using interpolation were smaller than those without interpolation for eight training trials ($N_\mathrm{t} = 8$). Bars and lines represent the mean values and standard deviations. $*$ and $**$, respectively, represent $p < 0.05$ and $p < 0.01$ in a t-test. $N = 100$.

VI. JUMP-AND-HIT SIMULATIONS

We simulated the jump-and-hit motions using a pneumatic humanoid robot to validate the nonlinear interpolation in IMoLo. We used the Open Dynamics Engine [26] as a dynamic simulator. We set the error reduction parameter, the constraint force mixing parameter, the time step, and the number of positions at which we calculate the collision to 10^{-6}, 10^{-6}, 10^{-4} s, and 3, respectively. We set the joint composition and the size, mass, and inertia of the links of the model to be the same as those of Liberobot. We simulated actuation of air cylinders in the robot using air dynamics models from [27].

We validated nonlinear interpolation in IMoLo as follows. First, we randomly selected $\boldsymbol{\theta}$, simulated a motion using this $\boldsymbol{\theta}$, recorded the hand position of the model, $\mathbf{p}_\mathrm{hand}$, randomly selected t from the time when swinging the arm in this motion as t_q, and obtained $\mathbf{p}_\mathrm{q} = \mathbf{p}_\mathrm{hand}(t)$ as $\mathbf{s}_\mathrm{q}$. Second, we randomly selected $\boldsymbol{\theta}$, simulated a motion using this $\boldsymbol{\theta}$, and recorded $\mathbf{p}_\mathrm{hand}$. We repeated this N_t times as training trials. Third, we created a lookup table using the data of these training trials. When creating this table, we did not voxelize the hand positions to omit the effects of voxelization. Finally, we calculated $\boldsymbol{\theta}$ from $\mathbf{s}_\mathrm{q}$ and t_q using the table, simulated a motion using this $\boldsymbol{\theta}$, recorded $\mathbf{p}_\mathrm{hand}$, and calculated the distance, $d_p = |\mathbf{p}_\mathrm{q} - \mathbf{p}_\mathrm{hand}(t_\mathrm{q})|$. If the lookup table gives the optimal θ, then d_p is close to zero. We repeated this process 100 times and recorded d_p with interpolation (w/) and d_p without interpolation (w/o). We compared d_p (w/) with d_p (w/o). We changed the number of training trials, N_t, to 8, 16, and 24. We generated models designed with a radial basis function (RBF) kernel setting the regularization constant, insensitive loss parameter, and kernel width to 0.1427, 0.0143, and 1.0, respectively.

Fig. 7 shows d_p. The figure indicates that the mean values when adopting nonlinear interpolation were smaller than those in the case in which $N_\mathrm{t} = 8$ and larger than those in the cases in which $N_\mathrm{t} = 16$. The p values for $N_\mathrm{t} = 8$, $N_\mathrm{t} = 16$, and $N_\mathrm{t} = 24$ in the T-test were 0.02, 0.0001, and 0.07, respectively. We supposed that the model could estimate commands more accurately through nonlinear interpolation when there were few training trials, but the model could be estimated by using a sufficient number of valid commands and times by only using samples with many training trials. These results suggest that the nonlinear interpolation would enable a robot to learn motions with few training trials, as with a real robot.

TABLE III
MOTION PARAMETERS

	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9	# 10	# 11	# 12	# 13	# 14	# 15
t_w [s]	0.210	0.210	0.410	0.410	0.373	0.259	0.396	0.280	0.249	0.260	0.333	0.305	0.280	0.376	0.327
k	1.000	-1.000	1.000	-1.000	0.833	-0.427	0.513	0.507	-0.240	0.133	-0.847	-0.893	0.060	0.560	0.867

VII. Jump-and-Hit Experiments Using Liberobot

We conducted jump-and-hit experiments using Liberobot to confirm that a pneumatic humanoid robot can execute jump-and-hit motions using IMoLo.

A. Setup

To learn the motions and evaluate the accuracy with which they are performed, we measured the motions of the Liberobot using a motion-capture system. To measure the motions, we attached spherical markers covered with reflective materials to the joint axis, hands, and feet of the robot. The motion-capture system measured the positions of the markers at 120 fps.

We used a soft polyurethane foam ball (diameter: 90 mm; weight: 10 g) to avoid damage or injury during the experiments. We attached reflective seals to the surface of the ball to measure its position using motion-capture cameras.

To enable us to control the experimental conditions, we developed and used a machine to throw the ball to the robot during the experiments. The dimensions of this machine were 0.4 m (W) $\times$ 0.6 m (D) $\times$ 1.1 m (H). The machine was equipped with one pneumatic rod cylinder to move its arm and the same two pressure-proportional valves and computer as those used in Liberobot. We decided that the ball would be thrown from about 5-m away from the robot to land in front of it after 1 s. The robot would then jump forward and hit the ball in the air according to the motion setting. We prepared the machine to throw a ball to satisfy these features.

We placed the robot on a box of 400-mm height. The robot sat by stopping its movement at the joints in its legs as they were at the ends of their range of motion. We marked the feet of the robot and the box to ensure the same initial posture and position of the robot at each run.

We placed a soft athletics mat in front of the robot to reduce damage. We also connected the robot to a rope attached to a crane in the room. This rope was long and was adequately supported to avoid disrupting the motions of the robot. Subsequent to the hitting motion, the rope pulled the robot upward, ensuring that the legs of the robot landed on the mat softly.

B. Experimental Procedure

The simulation results in Section VI indicates that the prediction errors using nonlinear interpolation in IMoLo were smaller than those without it when the trial number was eight. We supposed that the dynamics of the real robot are more complex than those in simulations, and a large number of trials was required to accurately approximate the dynamics. Thus, during training, the robot executed 15 motions with different commands.

Table III also presents k and t_w used in this training. The valvenumbers in this table are the same as in Fig. 6. We set the supplied air pressure at 0.9 MPa.

We extracted the hand position, $\mathbf{p}_{\text{hand}}$, as $\mathbf{s}$ at $0.13 < t - t_w < 0.22$ for the training sample as the positions where the hand is placed at the moment of hitting, where t is the elapsed time from the onset of the motion. As a result, we sampled 161 training data points. Using support vector regression [21], we generated models designed with the RBF kernel setting the regularization constant, an insensitive loss parameter, and the kernel width to the same parameters used in Section VI.

We estimated $\mathbf{p}_{\text{hand}}$ by varying t_w from 0.21 to 0.41 s in intervals of 0.002 s, k from -1 to +1 in intervals of 0.0133, and $t - t_w$ from 0.14 to 0.21 s at intervals of 0.002 s. We set $d_{\text{grid}} = 10\,\text{mm}$ and the time step of the state transition of the ball to predict its position $\Delta t = 0.0083\,\text{s}$, respectively.

In the test trials, the machine threw the ball 20 times from two positions (i.e., 10 times from each position) in different directions. Trials in which the arm of the robot hit the ball were considered successful. We calculated the success ratio and accuracy of the positions reached by the robotic hand.

C. Results

Fig. 8(a) shows the snapshots of the jump-and-hit motions of the robot in the experiments. This figure indicates that the robot generated different motions without being pulled by the rope and jumped forward and hit the ball in the air. The robot hit the ball 16 times in 20 trials. The success rate was 0.8.

Fig. 8(b) shows the trajectories of the ball and the hand positions of the robot before hitting in successful cases. The figure indicates that the ball was thrown from two positions, the robot generated different motions, and the ball was hit at different positions.

The success rate and the distance between the hand and ball when hitting was determined by the repeatability of the robot, the errors of the regression, the ones of using the grid, and the prediction error of the ball. The mean and the standard deviation of the distance between the selected grid positions and the positions where the hand reached were 12.9 and 9.4 mm, respectively. The mean and the standard deviation of the distance between the predicted ball positions before 0.25, 0.50, and 0.75 s and the positions where the ball reached were 46.1 and 34.2 mm, 155.9 and 32.0 mm, and 338.2 and 35.7 mm, respectively. The prediction errors of the ball were larger than the errors of the hand. Thus, these results suggest that the success ratio was mainly determined the prediction errors of the ball.

VIII. Conclusion

This study proposed the IMoLo method for immediately generating dynamic motions of a pneumatic humanoid robot

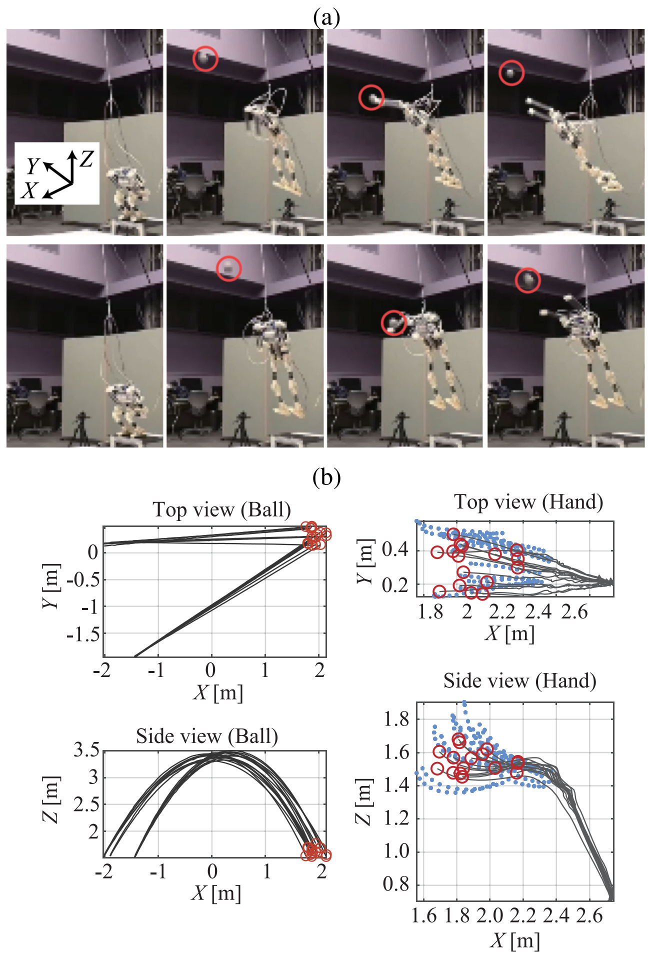

Fig. 8. (a) Snapshots of two jump-and-hit motions and the coordinates. The red circles show the positions of the ball. (b) Trajectories of the ball (left) and hand (right) before hitting. The red circles show the positions of the ball and hand at the instance of hitting. The blue dots shows the positions of the hand in the training data.

to support jump-and-hit motions. In simulations, the prediction errors of the positions where the hand of a humanoid robot reached in its jump-and-hit motions with the nonlinear interpolation in IMoLo were smaller than those without it. A pneumatic humanoid robot called "Liberobot," equipped with SIP-CCs, performed different jump-and-hit motions using IMoLo and hit a ball originating from different positions at a success rate of 0.8. The results indicate that a pneumatic humanoid robot can instantaneously generate motion in response to a moving object within the target timing and perform these motions using IMoLo. The jump-and-hit motions of the Liberobot provide the first case of such motions for a pneumatic humanoid robot.

References

[1] M. H. Raibert, *Legged Robots That Balance*. MIT press, 1986.

[2] R. Niiyama, A. Nagakubo, and Y. Kuniyoshi, "Mowgli: A bipedal jumping and landing robot with an artificial musculoskeletal system," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2007, pp. 2546–2551.

[3] R. Niiyama, S. Nishikawa, and Y. Kuniyoshi, "Biomechanical approach to open-loop bipedal running with a musculoskeletal athlete robot," *Adv. Robot.*, vol. 26, no. 3/4, pp. 383–398, 2012.

[4] O. Birbach, U. Frese, and B. Bäuml, "Realtime perception for catching a flying ball with a mobile humanoid," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 5955–5962.

[5] B. Bäuml, T. Wimböck, and G. Hirzinger, "Kinematically optimal catching a flying ball with a hand-arm-system," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2010, pp. 2592–2599.

[6] S. Kim, A. Shukla, and A. Billard, "Catching objects in flight," *IEEE Trans. Robot.*, vol. 30, no. 5, pp. 1049–1065, Oct. 2014.

[7] T. Senoo, A. Namiki, and M. Ishikawa, " High-Speed batting using a multi-jointed manipulator," in *Proc. IEEE Int. Conf. Robot. Automat.*, vol. 2, 2004, pp. 1191–1196.

[8] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *Int. J. Robot. Res.*, vol. 32, no. 3, pp. 263–279, 2013.

[9] J. Peters, K. Mülling, J. Kober, D. Nguyen-Tuong, and O. Krömer, "Towards motor skill learning for robotics," *Robot. Res.*, pp. 469–482, 2011.

[10] Y. Huang, D. Büchler, O. Koç, B. Schölkopf, and J. Peters, "Jointly learning trajectory generation and hitting point prediction in robot table tennis," in *Proc. Int. Conf. Humanoid Robots*, 2016, pp. 650–655.

[11] Y.-B. Jia, M. Gardner, and X. Mu, "Batting an in-flight object to the target," *Int. J. Robot. Res.*, vol. 38, no. 4, pp. 451–485, 2019.

[12] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Schölkopf, and J. Peters, "Learning to play table tennis from scratch using muscular robots," 2020, *arXiv:2006.05935*.

[13] L. Guilamo, J. Kuffner, K. Nishiwaki, and S. Kagami, "Efficient prioritized inverse kinematic solutions for redundant manipulators," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2005, pp. 3921–3926.

[14] Y. Guan and K. Yokoi, "Reachable space generation of a humanoid robot using the monte carlo method," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2006, pp. 1984–1989.

[15] M. Vukobratović and B. Borovac, "Zero-moment point-thirty five years of its life," *Int. J. Humanoid Robot.*, vol. 1, no. 1, pp. 157–173, 2004.

[16] R. S. Sutton, D.Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1/2, pp. 181–211, 1999.

[17] S. Nishikawa, T. Kobayashi, T. Fukushima, and Y. Kuniyoshi, "Pole vaulting robot with dual articulated arms that can change reaching position using active bending motion," in *Proc. Int. Conf. Humanoid Robots*, 2015, pp. 395–400.

[18] S. Nishikawa, K. Shida, and Y. Kuniyoshi, "Musculoskeletal quadruped robot with torque-angle relationship control system," in *Proc. Int. Conf. Robot. Automat.*, 2016, pp. 4044–4050.

[19] D. F. Specht, "A general regression neural network," *IEEE Trans. Neural Netw.*, vol. 2, no. 6, pp. 568–576, Nov. 1991.

[20] Carl Edward Rasmussen, "Gaussian processes in machine learning," in *Proc. Summer Sch. Mach. Learn.*, 2003, pp. 63–71.

[21] C.-C. Chang and C.-J. Lin, " Training v-support vector regression: Theory and algorithms," *Neural Comput.*, vol. 14, no. 8, pp. 1959–1977, 2002.

[22] S. H. Martijn, C.Wisse, and A. Ruina, "A three-dimensional passive-dynamic walking robot with two legs and knees," *Int. J. Robot. Res.*, vol. 20, no. 7, pp. 607–615, 2001.

[23] K. Tanaka, S. Nishikawa, R. Niiyama, and Y. Kuniyoshi, "Humanoid robot performing jump-and-hit motions using structure-integrated pneumatic cable cylinders," in *Proc. IEEE-RAS Int. Conf. Humanoid Robot.*, 2017, pp. 696–702.

[24] M. Müller, S. Lupashin, and R. D'Andrea, "Quadrocopter ball juggling," in *Proc. Int. Conf. Intell. Robots Syst.*, 2011, pp. 5113–5120.

[25] R. G. Brown and P. Y. C. Hwang, *Introduction to Random Signals and Applied Kalman Filtering: With MATLAB Exercises*. John Wiley & Sons, New York, NY, USA, vol. 4, 2012.

[26] R. Smith, " Open Dynamics Engine, 2005.

[27] J. E. Bobrow and B. W. McDonell, "Modeling, identification, and control of a pneumatically actuated, force controllable robot," *IEEE Trans. Robot. Automat.*, vol. 14, no. 5, pp. 732–742, Oct. 1998.