An Online Gait Adaptation with SuperBot in Sloped Terrains

Teawon Han[1], Nadeesha Ranasinghe[2], Luenin Barrios[3], and Wei-Min Shen[4]

Abstract— **Among the different types of robots, modular and self-reconfigurable robots such as SuperBot have less limitations than their counterparts due to their versatility of gaits and increased dynamic adaptability. This results in a highly dexterous and adjustable robot suitable for many environments. This however, usually comes at the expense of a necessary human observer required to monitor and control the robot manually resulting in a waste of power and time. Thus, an intelligent system would be indispensable in optimzing the behavior and control of modular and self-reconfigurable robots. This paper presents an Intelligent Online Reconfiguration System (IORS) which through a combination of learning and reasoning, increases the efficiency in control and movement of the modular and self-reconfigurable robot called Superbot. Using this system, Superbot is able to learn and choose the best gait automatically by sensing its current environment (e.g., friction or slope). As a result, the IORS implementation in SuperBot achieves: 1) correct slope gradient sensing, 2) best gait learning to traverse different slopes, and 3) rational decision making for choosing the best gait.**

I. INTRODUCTION

In general, robots are designed with functional or environmental considerations in mind. The area of deployement and the robot's task requirements in that environment drive the physical design of the robot. Once the robot is manufactured and programmed, it would have only those capabilities sufficiently required to perform its task. This inexorably leads to limitations in hardware and software wherein activities or actions outside the scope of the robot's purpose or abilities results in either total failure or decreased unsatisfactory performance. Here, limitations in hardware and software means "Does the robot have any gaits to move in the encountered environment?" and "Does the robot have intelligence to choose appropriate gaits depending on the environment?" Modular and self-reconfigurable robots such as SuperBot can overcome the hardware limitation by changing their configurations [1]. For example, Superbot [2] has several gaits with each one being suitable to different types of tasks. The caterpillar gait of Superbot can go through pipes, but its locomotion speed is slow whereas the rollingtrack gait moves fast on flat ground, but is susceptible to the gradient of the slope[3]. However, there are still software limitations to consider. Even if the robot has a gait that operates well in a particular environment, if the appropriate operation (i.e., a condition-action pair) was not pre-programmed, then the robot will not be able to adequately select it. There are two possible solutions. The first method requires the introduction of a human operator who changes the gait using a remote control. This, of course, is not a permanent solution because of the inefficiency of redundant sensing, control, and communication which ineluctably results in power loss and decreased performance; the more a robot uses power, the less it can move. In addition, monitoring conditions is limited when the exploration is conducted underwater, underground, or on other planets. The second method is to uitlize the pre-programming of a lot of predictatble condition and action pairs to choose the proper gait. In 2004, a research team from the National Institute of Advanced Industrial Science and Technology (AIST) tried to make M-TRANII [7] adapt to various ground conditions (frictions, gradients of slope) by transforming from a 4-legged to an H-shaped robot. However, M-TRANII could only transform its shape if a given condition was satisfied. In other words, M-TRANII cannot recognize and overcome unexpected conditions. Therefore, pre-programmed pairs of conditions and actions are not enough to move in real-world environments because it is impossible to predict all possible conditions the robot might encounter. However, taking a modular and self-reconfigurable robot and providing it with its own decision making through the application of an intelligent and cognitive system would endow it with the capabilities required to overcome many of these obstacles. Intelligent cognitive systems such as Game Theory, SOAR, and Graphical Cognitive Architecture, have been researched and applied in various fields. Game theory itself has been applied in several U.S. territory guard systems to prevent terror attacks; a well known example is ARMOR[4] which schedules security check-points at the Los Angeles airport. SOAR supported an air traffic control task[5] and the Cognitive Graphical Architecture was used to solve several problems, e.g., SLAM(simultaneous localization and mapping) with path planning of robots [6] and bridging dichotomie for a virtual human [8]. Here, the challenge of intelligent cognitive systems is learning and reasoning from scratch with only noisy data. The challenge is further compounded in modular and self-reconfigurable robots because the sensor data includes a wide range of noise discrepancies depending on the different configurations of the modules. For example, there is no sensor to measure the gradient of slope directly,

*This work was not supported by any organization

[1]T. Han is with Information Sciences Institute, The University of Southern California, Marina Del Rey, CA 90292, USA teawonhan at gmail.com

[2]N. Ranasinghe is with Information Sciences Institute, The University of Southern California, Marina Del Rey, CA 90292, USA nadeesha at isi.edu

[3]L. Barrios is with Information Sciences Institute, The University of Southern California, Marina Del Rey, CA 90292, USA lueninba at usc.edu

[4]Wei-Min Shen is a research associate professor of Computer Science department and a project leader of Information Sciences Institute at The University of Southern California shen at isi.edu

so several modules must be used. Despite the propinquity of the modules, depending on the robot's configuration, the noise level for each module can be extremely disparate.

This paper shows how with the inclusion of an Intelligent Online Reconfiguration System (IORS), Superbot is able to deal with these challenges to learn and choose the best gait automatically given three gaits and different sloped environments. IORS allows the robot to be able to (1) recognize different sloped environments despite disparate noise levels, (2) learn the preference for each gait in different environments and (3) make an internal decision regarding the best gait for a particular environment.

II. RELATED WORK

A. Different Rollingtrack gaits of SuperBot

Three different kinds of rollingtrack gaits were used to test IORS. These were chosen because rollingtrack gaits provide fairly fast locomotion and transform shapes without lots of effort and obviate the need for additional docking actions. The rollingtrack configuration was inspired by Polynomial Robotics Laboratory at the Information Sicences Institute at the University of Southern California in 2007, and Jimmy Sastra et al [10], also analyzed about dynamic rolling for a modular loop robot. The rollingtrack consists of six SuperBot modules. The principle of the gait's movement is that changing the center of gravity by reconfiguring the robot's shape into a squeezed hexagon results in a forward motion[3]. While the rollingtrack is traveling, each module checks a value of its accelerometer to find its orientation and change its configuration following a given rule (sets of state-action). Previously, in [3] and [9], Superbot did not recognize its environment and thus was unable to adapt accordingly. It simply obeyed the specified gait rule. Depending on the given rule, the pattern of movement (gait) is different and has uniquely different characteristics. For example, the height of the Climbing rollingTrack gait is lower than the standard circular rollingtrack. Its movement speed is also slower, but it can climb steeper slopes. In this paper, a step is a process between transformations, and three gaits are used, namely: rollingtrack high, climbing rollingtrack, and the standard circular rollingtrack. These are abbreviated as rt_high, rt_low, and rt_circle respectively. Sequential positions of a step for rt_high are shown in Figure 1, (a)-(b)-(c), for rt_low sequential positions of a step are shown in (d)-(e)-(f), and for rt_circle it is simply (a). rt_high and rt_low consist of three configurations, and the transformations must occur six times for a forward roll[3]. In previous works, not only was Superbot unable to sense its environment, it was also incapable of choosing its gait automatically. Therefore, whenever the robot encountered unexpected environments such as steep slopes, it became stuck and unable to make any further forward progress. However, with IORS, Superbot can sense the environment to learn and choose the most suitable and appropriate gait without any outside human interference.

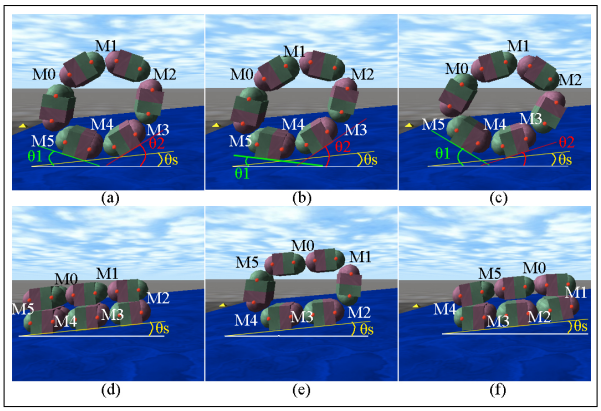

Fig. 1. Three different types of rollingtrack gaits: M0 ∼ M5 represent the six modules. θs is 5° gradient of the slope, and $\theta 1$ and $\theta 2$ represent the angles which the accelerometer sensor of M5 and M4 detect respectively. During a step, rt_high and rt_circle do not have a module guaranteed to be parallel to the slope, whereas rt_low does. The accelerometer sensor is located on the red part of a module.

B. Two purposes of 3D Accelerometor Sensor in SuperBot

In [3] and [9], the accelerometer sensor was used to recognize each module's orientation. This is useful in developing various movement patterns(gaits) by programming different pairs of conditions and actions. For example, if the orientation of a module in the three axes has some set of accelerometer values, namely: x=$A1$, y=$A2$, and z=$A3$, then we can order the module into a different orientation by commanding the three motors to new motor angles: $motor1$=$\theta 1$, $motor2$=$\theta 2$, and $motor3$=$\theta 3$. In this paper we propose that the accelerometer sensor can be used to recognize the gradient of slopes by using the same technique as analyzing tilt. Figure. 2 shows the configuration of the three axes in a SuperBot module and how to calculate the slope gradient. In a step of movement, the accelerometor data obtained from a module which is parallel to the slope is read. For example, when the gait is rt_low, the slope is detected by reading the accelerometor sensor of the module whose position is the same as M2 in (f) of Figure 1.

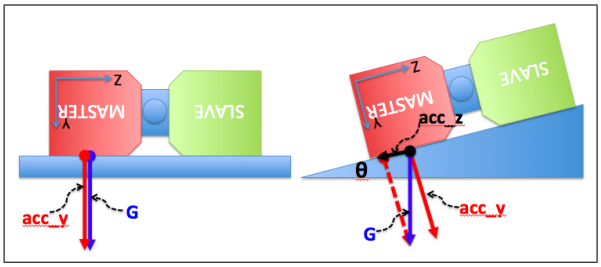

Fig. 2. Detection of slope gradient using accelerometor sensor in SuperBot: acc_y and acc_z represent sensed accelerometor data on the y and z axis respectively. The data on acc_y changes based on θ according to $'acc\_y = 1G \times -cos(\theta)'$ which is the same as equation (1). Therefore, we can use the acc_y to calculate gradients of slopes, i.e., when θ is 5°, acc_y is -0.9962.

III. THE APPROACH OF INTELLIGENT ONLINE RECONFIGURATION SYSTEM (IORS)

In the real world, environments possess many features such as slope and friction. To use IORS, a specified number of selected gaits is provided to the robot which is then placed in an environment composed of different features. For the nonce, we deal exclusively with different slope gradients. Under such conditions, IORS is executed iteratively using the following four steps: (1) recognize the feature of the environment, i.e., slope gradient (2) semi-learning the likelihood of the accelerometer sensor, (3) learning the preference of each gait under different features and (4) selection of the best gait having the highest preference for the recognized feature. In this paper, the robot is given three gaits, namely rt_high, rt_low, and rt_circle, and features of the environment consist of various slope gradients. Under these coditions, the focus is on how fast SuperBot can overcome different slope gradients using IORS.

A. Recognition of the gradient of slope with three gaits

To recognize the slope, two different types of data are used: sensed data from the accelerometer sensor on the y axis as discussed in the Section II-B and *time per step* of gait. For example, the *time per step* of rt_high is the time(sec) during which Superbot moves from (a) to (c) in Fig. 1. For rt_low, recognizing the slope's gradient can be calculated directly from the accelerometer data because at least one module is guaranteed to impinge the environment fully and be parallel to the slope at each step. This is shown in Fig. 1 (d), (e), and (f). For the remaining two gaits, such a condition is not guaranteed. Therefore, the *the time per step* is used for rt_high and rt_circle. Regardless of the method used, the sensed data includes uncertainties which arise from noise in the sensor. To overcome these uncertainties, the sensed data is converted to a Gaussian distribution as shown in Fig. 3.

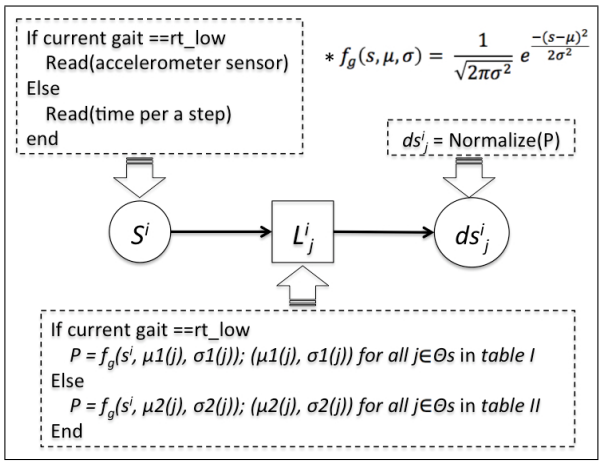

Fig. 3. The process of converting the sensed data to Gaussian distribution of slope: S is sensed data: *time per step* or *accelerometer data on y axis*, L is the likelihood for each slope's gradient, and ds is the distribution for the gradient of each slope.

For the Gaussian distribution, there are two likelihood tables, *table I* and *table II* each consisting of pairs of mean and standard deviation values for each slope gradient considered, e.g., downhill: -1, flat: 0, and uphill: 1∼20 degree. The results for the data in *table I* and *table II* is the data experimentally obtained on the accelerometer sensor for the y axis and *the shortest time per step* which was obtained from pre-experiments in simulation. Using this data, the values for both tables was initialized using equation (1) for the mean, and equation (2) for the standard deviation differences between neighbors' mean values.

TABLE I
LIKELIHOOD-TABLE FOR ROLLINGTRACK_LOW

A gradient of slope(θ)	mean $\mu1(\theta)$	standard deviation $\sigma1(\theta)$
-1	-1.0102	0.0072
0	-1.000	0.0002
1	-0.9998	0.0002
...	...	...
19	-0.9397	0.0029
20	-0.9336	0.0029

TABLE II
LIKELIHOOD-TABLE FOR ROLLINGTRACK_HIGH

A gradient of slope(θ)	mean $\mu2(\theta)$	standard deviation $\sigma2(\theta)$
-1	0.1	0.3514
0	0.597	0.0219
1	0.628	0.0358
...	...	...
8	1.312	0.1181
9	2.5	0.8400
...	...	...
19	2.5	0.8400
20	2.5	0.8400

B. Semi-Learning likelihood of sensor by Mutual Expectation and Correction

In table I and II, the initialized values are approximately correct, but should be further adjusted to the current status of the environment and sensors. This is because if the sensor's calibration is incorrect or unexpected factors arise, e.g., wind blows too hard, then the initialized likelihood of rt_low($\mu1$, $\sigma1$) or rt_high ($\mu2$, $\sigma2$) would not be helpful anymore in determining an accurate distribution of the slope.

$$mean(\theta) = -cos(\theta), -1 \leq \theta \leq 20 \quad (1)$$

$$std(\theta) = \sqrt{((A)^2 + (B)^2)/2} \quad (2)$$

$$A = mean(\theta - 1) - mean(\theta)$$
$$B = mean(\theta + 1) - mean(\theta)$$

Therefore, a simple technique called Mutual Expectation and Correction (MEC) is used. As shown in Fig. 3, pairs of mean and standard deviation are a likelihood L^i_j; i is an index of step, j is a gradient of slope -1$\leq j \leq$20, and are used to get Gaussian distributions of each slope gradient (ds^i_j). The MEC

runs based on two assumptions, $A1$: *likelihood is perfect* and $A2$: *sensed data is perfect*. The distribution of the gradient of a slope is obtained by assuming $A1$, and the likelihood is updated (or adjusted according to the likelihood of sensed data) under $A2$ in every step as shown in *table III*.

TABLE III
PSEUDO CODE FOR UPDATING LIKELIHOOD OF SENSOR

```
/* Initialization of Likelihood Start */
/* This part is called only one time when the robot is deployed */
For j from -1 to 20
   L_j^0 = {(μ1(j), σ1(j))^0 in table I, (μ2(j), σ2(j))^0) in table II}
    /* where L_j^i is the likelihood of a slope gradient(j) at a step(i) */
End for
i = i + 1;
/* Initialization of Likelihood End */

/* Converting Sensed data to Distribution of slope Start (figure 3) */
/* The below is called in every step i */
If current_gait == rt_low then
   s^i = accelerometer sensor data on y axis;
else
  If time per a step < threshold then
    s^i = time per a step;
  else
  /* when the robot is stuck or rolling back(fail) */
    s^i = threshold_high;
   /* threshold_high is the maximum time in a step which is bigger than
  the maximum time in a step of rt_low : 2.5 seconds */
  End if
End if
For j from -1 to 20
  If current_gait == rt_low then
    ds_j^i = f_g(s^i, μ1(j), σ1(j))^i;
  else
    ds_j^i = f_g(s^i, μ2(j), σ2(j))^i;
End for
ds_j^i = Normalize(ds_j^i);
/* Converting Sensed data to Distribution of slope End */

/* Updating Likelihood of Sensor by MEC Start */
θ_max = −1;
For j from -1 to 20
   If ds^i_θmax < ds_j^i then
    θ_max = j;
   End if
End for
Equation(3) with parameters: i, θ_max;
/* Updating Likelihood of Sensor by MEC End */
```

$$mean(\theta_{max})^{new} = W^i \times (s^i - mean(\theta_{max})) + mean(\theta_{max}) \quad (3)$$

$$\text{where } W^i = ds_j^{i-1} \times ds_j^i; j = \theta_{max}$$

The more a robot experiences, the more information it gets. Obviously, if a robot gets more information in several steps, it will be able to recognize more accurately. Therefore, Bayes' Rule (Posterior $\simeq$ Likelihood $\times$ Prior) is used to recognize the gradient of a slope more precisely by following equation (4) and figure (4). The *distribution of belief* at slope j in i step (db_j^i) means with how much confidence a robot believes what the gradient of a slope is.

$$db_j^i = \eta[(ds_j^i \times W) \times db_j^{i-1}], -1 \le j \le 20 \quad (4)$$

where $\eta[...]$ means normalization of $[...]$, and W is the same value as the one in equation (3).

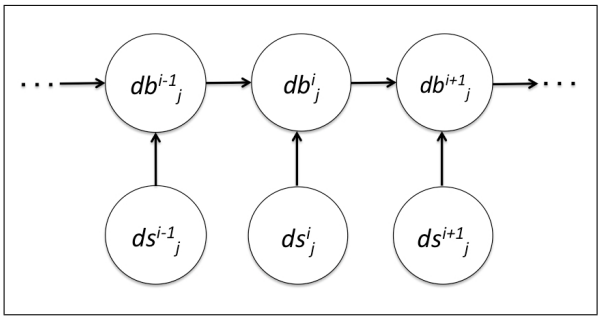

Fig. 4. The Bayesian network for the distribution of belief for continuous steps

C. Learning the preference of each gait for each feature

So far, we have considered how to recognize an environment's feature correctly. The next step, perforce, is to focus on learning the preference of each gait for each feature and to make a rational decision that chooses the best gait. *Table IV* called the decision-table maintains these preferences. The decision-table consists of the gait in columns and the feature in rows. The value in the table is the time(sec) per step of each gait, and it is updated by equation (4) and (5) in every step as shown in *Table V*; $t\_low_j^i$ and $t\_high_j^i$ are time per *step* i of each gait at *recognized slope* j where j is $\delta - 1 \le j \le \delta + 1$; δ is equal to j of db_j^i which is equal to $MAX(db_j^i)$ with $-1 \le j \le 20$.

TABLE IV
DECISION-TABLE

A gradient of slope	rollingtrack low	rollingtrack high
-1	$t\_low_{-1}^i$	$t\_high_{-1}^i$
0	$t\_low_0^i$	$t\_high_0^i$
1	$t\_low_1^i$	$t\_high_1^i$
...	$t\_low_j^i$	$t\_high_j^i$
19	$t\_low_{19}^i$	$t\_high_{19}^i$
20	$t\_low_{20}^i$	$t\_high_{20}^i$

The reason for the omission of rollingtrack circle (rt_circle) from *table IV* is that rt_circle can be faster than the other gaits if and only if the slope is downhill. In addition, rt_circle always requires a few seconds to be faster than rt_high even on downhills. Therefore, rt_circle is selected if and only if *the time per step* of rt_high is less than *threshold_low* which is *the shortest time in a step* of rt_high on a slope of 0°(flat). We set the *threshold_low* to 0.588 sec.

$$t\_low_j^i = 0.5 \times (rt\_low(db_j^i) \times W + t\_low_j^{i-1}) \quad (4)$$

$$t\_high_j^i = 0.5 \times (rt\_high(db_j^i) \times W + t\_high_j^{i-1}) \quad (5)$$

where $\delta - 1 \le j \le \delta + 1$, and W is the same value as in an equation (3).

TABLE V

PSEUDO CODE FOR UPDATING DECISION-TABLE

```
/* Initialization of Decision-table Start */
/* This part is called only one time when the robot is deployed */
i=0;
For j from -1 to 20
   t_low_j^0 = 0.001;
   t_high_j^0 = 0.001;
End for
i = i+1;
/* Initialization of Decision-table End */

/* Updating Decision-table Start */
/* This part is called in every step i */
δ_max = −1;
For j from -1 to 20
  If current_gait == rt_low then
     If rt_low(db^i_δmax) < rt_low(db^i_j) then
       δ_max = j;
     End if
  else
     If rt_high(db^i_δmax) < rt_high(db^i_j) then
       δ_max = j;
     End if
   End if
End for
If current_gait == rt_low then
  Equation (4) with parameters (i, δ_max);
Else
  Equation (5) with parameters (i, δ_max);
End if
/* Updating Decision-table End */
```

D. Choosing the best gait

In this paper, the best gait is defined as the one which will result in the fastest forward movement for a particular environment. The decision table maintains the *seconds per step* of movement., Thus, for every step, the robot looks up values in the decision-table using equations (6) and (7) which together define the negative-preference of rt_low and rt_high respectively. The robot then compares the two negative-preferences, and chooses the gait whose value is smallest.

$$\rho_{rt\_low} = \sum_{j=-1}^{20} (db_j^i \times t\_low_j^i) \tag{6}$$

$$\rho_{rt\_high} = \sum_{j=-1}^{20} (db_j^i \times t\_high_j^i) \tag{7}$$

IV. EXPERIMENTAL RESULTS

For test purposes, six Superbot modular robots were used utilizing three given gaits (rt_high, rt_low, and rt_circle) in environments with different slope gradients$(-20^\circ \sim 20^\circ)$. The simulation physics were all performed using the Open Dynamic Engine(ODE). As previously mentioned, the challenge involves learning to select the best gait on a recognized environment. Figure 5 shows *the shortest time per step* at different slope gradients assuming Superbot is not allowed to change its gait; red (rt_high), blue(rt_low), and green (rt_circle) lines are representing *the shortest time per a step* at each gradient of slope when Superbot dosen't change its gait; dotted black-line is showing the shortest time per a step with choosing an optimized gait at each gradient of slope. For example, if Superbot uses rt_high, the maximum slope gradient it can climb is about 9°. In other words, a fixed-gait rollingtrack would not be able to travel through this environment, or it would be very slow it rt_low is used all the time.

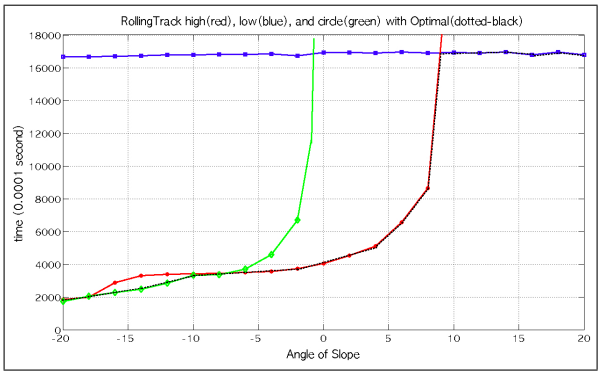

Fig. 5. *The shortest time per step* of each gait at different *slope gradients* in moving 11.5m. Without IORS, rt_circle, rt_high, and rt_low can only move in slope ranges of: $[-20 \sim -2]$, $[-20 \sim 8]$, and $[-20 \sim 20]$ respectively.

A. Experimental Setup

As shown in Figure 6, the test environment is varied and includes diverse slope gradients of 0° flat, 5° and 13° uphill, and −14° downhill. The reason two different uphill gradients were chosen is that both rt_high and rt_low can climb a 5°slope, but only rt_low can overcome a 13° slope. Initially the robot has neither information anent its environment nor preferences of gaits for different slope gradients. The initial gait is set to rt_high and the robot begins to sense its environment, learning and choosing the best gait in every step. Through experimentation, the aim is to show if IORS can help SuperBot independently and simultaneously learn and choose the best gait.

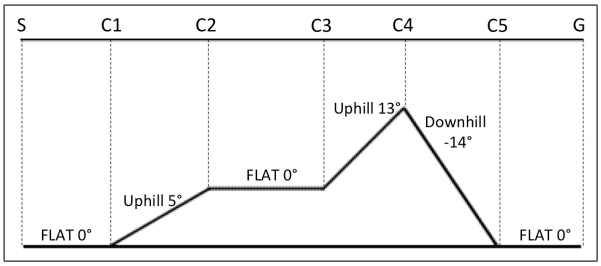

Fig. 6. The setting of experimental environment .

B. Experimental Results

In two continuous trials, comparisons are made of *the best gait* chosen by Superbot and *the optimal gait* chosen by a human for each slope gradient. Trial #1 and trial #2 refer to first and second executions. If Superbot learns the best gait correctly, the result of trial #2 should be better than that of trial #1. *Table VI* shows the number of times Superbot chose each gait while traversing each slope. In the table, *OPT* is the optimal gait chosen by a human for the slope, and *Incorrect*

is the number of chosen gaits which are different from the optimal gait.

TABLE VI
THE RESULTS OF THE CHOSEN GAIT AT EACH GRADIENT OF SLOPE

	Flat (0°)	Uphill (5°)	Flat (0°)	Uphill (13°)	Downhill (−14°)
OPT	rt_high	rt_high	rt_high	rt_low	rt_circle
	=====	=====	Trial #1	=====	====
	-	-	-	1stuck,2fails	-
rt_high	7	12	20	6	7
rt_low	0	2	0	24	3
rt_circle	0	0	0	0	11
Incorrect	0	2	0	6	10
Steps	7	14	20	30	21
	=====	=====	Trial #2	=====	====
	-	-	-	1 stuck	-
rt_high	7	13	18	2	3
rt_low	0	1	2	15	1
rt_circle	0	0	0	0	11
Incorrect	0	1	2	2	4
Steps	7	14	20	17	16

The total number of incorrect choices is reduced from 18 in trial #1 to 9 in trial #2. In addition, the total number of steps has also decreased from 92 in trial #1 to 74 in trial #2. Cleary, trial #2 is faster than trial #1 as show in Figure 7. The results confirm that using IORS, the more Superbot experiences, the better it will be able to learn and select the best gait. A video of IORS test is available at http://youtu.be/nxSt0egp2XI.

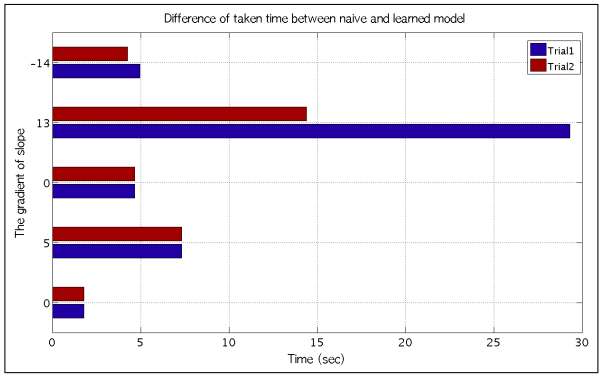

Fig. 7. The time taken for both first and second trials of the intelligent SuperBot to traverse different slope gradients. Blue bar and red bar represent first trial and second trial respectively. At the slope 13°, taken time is reduced significantly after more learning.

V. DISCUSSION & FUTURE WORK

This paper presented simultaneous learning and reasoning in choosing the best gait based on environment without human interruptions. IORS filters noises from the sensed data and learns the perference of each gait for different types of environment, then chooses the best gait autonomously. Due to using probability with Bayes' rule, the modular robot doesn't have to maintain previous sensed data or histories of decisions. Strictly speaking, IORS is giving modular robot a partial intelligence to make an own decision because a requisited pre-condition should be satisfied: gaits and the sensible method corresponded with environment should be given. Here, three gaits and time or acceleration which is representiable to gradients of slope were given. However, IORS is efficient for a modular robot to overcome various environments. Rather than pre-programming condition-action pairs, the robot autonomously learns and makes a rational decision to choose the best gait from experiences as shown in *Table VI* and Figure 7.

To make this technique more robust, we will consider a definition of the best gait. Here, the definition is the gait taken the shortest time in a step. However, depending on types of mission or robot, different hueristics (definition of the best gait) should be chosen. One of possible idea is changing the hueristic based on robot's battery status, e.g. if the current of battery is lower than threshold value, then change the hueristic from speed to power. We plan to achieve this by maintaining another decision table to choose a hueristic.

ACKNOWLEDGMENT

We would like to thank the reviewers & USC PRL group.

REFERENCES

[1] Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S. Chirikjian., Modular Self-Reconfigurable Robot Systems – Challenges and Opportunities for the Future, IEEE Robotics and Autonomation Magazine, March():4353, 2007.

[2] Behnam Salemi, Mark Moll, and Wei-Min Shen., SUPERBOT: A Deployable, Multi-Functional, and Modular Self-Reconfigurable Robotic System. In Proc. 2006 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, Beijing, China, October 2006.

[3] Harris Chiu, Michael Rubenstein, and Wei-Min Shen., Multifunctional SuperBot with Rolling Track Configuration. In Proc. 2007 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, San Diego, CA, November 2007. IROS 2007 Workshop on Self-Reconfigurable Robots, Systems & Applications.

[4] James Pita, Manish Jain, Craig Western, Christopher Portway, Milind Tambe, Fernando Ordonez, Sarit Kraus, and Praveen Paruchuri., Deployed ARMOR protection: The application of a game-theoretic model for security at the Los Angeles International Airport. In AAMAS 2008.

[5] Chong, R.S., Wray, R.E. Inheriting constraint in hybrid cognitive architectures: Appyling the EASE architecture to performance and learning in a simplified air traffic control task. In K. Gluck and R. Pew, eds, Modeling Human Behavior with Integrated Cognitive Architectures: Comparison, Evaluation, and Validation, 237–304. Lawrence Erlbaum Associates, 2005.

[6] J. Chen, J., Demski, A., Han, T., Morency, L-P., Pynadath, P., Rafidi, N. & Rosenbloom, P. S. Fusing symbolic and decision-theoretic problem solving + perception in a graphical cognitive architecture. Proceedings of the Second International Conference on Biologically Inspired Cognitive Architectures, 2011.

[7] Akiya Kamimura, Haruhisa Kurokawa, Eiichi Yoshida, Kohji Tomita and Shigeru Kokaji, & Satoshi Murata, Distributed Adaptive Locomotion by a Modular Robotic System, M-TRAN II (From Local Adaptation to Global Coodinated Motion using CPG Controllers), Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2004), pp. 2370-2377, 2004.

[8] Rosenbloom, P. S. (2011), Bridging dichotomies in cognitive architectures for virtual humans, Proceedings of the AAAI Fall Symposium on Advances in Cognitive Systems.

[9] Wei-Min Shen, Harris Chiu, Michael Rubenstein, and Behnam Salemi. Rolling and Climbing by the Multifunctional SuperBot Reconfigurable Robotic System. In Proc. Space Technology and Applications Intl. Forum (STAIF-08), AIP Conference Proceedings No. 969, American Institute of Physics, pp. 839848, Melville, NY, February 2008.

[10] Jimmy Sastra, Sachin Chitta & Mark Yim. Dynamic Rolling for a Modular Loop Robot. The International Journal of Robotics Research 2009 28: 758-773.