Noname manuscript No.
(will be inserted by the editor)

Evolving Controllers for High-Level Applications on a Service Robot: A Case Study with Exhibition Visitor Flow Control

**Alex Fukunaga · Hideru Hiruma ·
Kazuki Komiya · Hitoshi Iba**

the date of receipt and acceptance should be inserted later

Abstract We investigate the application of simulation-based genetic programming to evolve controllers that perform high-level tasks on a service robot. As a case study, we synthesize a controller for a guide robot that manages the visitor traffic flow in an exhibition space in order to maximize the enjoyment of the visitors. We used genetic programming in a low-fidelity simulation to evolve a controller for this task, which was then transferred to a service robot. An experimental evaluation of the evolved controller in both simulation and on the actual service robot shows that it performs well compared to hand-coded heuristics, and performs comparably to a human operator.

1 Introduction

Evolutionary robotics (ER) is a class of approaches that use genetic and evolutionary computation methods in order to automatically synthesize controllers for autonomous robots. Early work started with evolution and testing of controllers in simulation [1,17,9]. This was followed by the introduction of *embodied evolution*, where the evolution was performed on real robots [25,21], as well as controllers which were evolved in simulation and transferred to real robots [25]. Nelson, Barlow, and Doitsidis present a recent, comprehensive review of previous work in evolutionary robotics [24].

Previous work on ER involving mobile robots has focused on relatively low-level tasks, such as locomotion/gait generation, navigation (including object avoidance),

A. Fukunaga
The University of Tokyo, Tokyo, Japan (corresponding author: E-mail: fukunaga@idea.c.u-tokyo.ac.jp, TEL/FAX +81-3-5454-6792)

H. Hiruma
The University of Tokyo, Tokyo, Japan, E-mail: hiruma@iba.t.u-tokyo.ac.jp

K. Komiya
The University of Tokyo, Tokyo, Japan E-mail: komiya@iba.t.u-tokyo.ac.jp

H. Iba
The University of Tokyo, Tokyo, Japan E-mail: iba@iba.t.u-tokyo.ac.jp

box-pushing, wall-following, and searching/foraging tasks. By "low-level", we are not referring to the difficulty of solving the task with a robot, but rather to the fact that the tasks tend to be essential for the "survival" of an autonomous robot (e.g., obstacle avoidance, wall-following), or tasks that an animal-level intelligence can perform (e.g., object following, box-pushing). Furthermore, previously work in ER tended to generate "low-level" controllers that mapped sensor inputs to motor outputs (motor output levels or low-level commands such as "forward", "turn left").

Although there have been some experiments with large, service robots, such as the evolution of navigation and obstacle avoidance behaviors [7,4], and shepherding of a robot using another Nomad robot [31], previous ER research involving mobile robots has usually been performed using relatively small, research robots (robots which are primarily built for research purposes, as opposed for commercial deployment), as exemplified by the Khepera platform. A notable exception is the evolution of gaits and ball-chasing behavior for a commercial quadruped [10,11].

In this work, we investigate the use of genetic programming to solve high-level tasks on a *commercial*, human-sized service robot. This differs significantly from traditional ER research due to the nature of the robot platform. Compared to most of the robot platforms that have been used in traditional ER research such as the Khepera, service robot platforms are have more powerful onboard sensing and processing capabilities, as well as wireless networking, enabling them to access remote resources. Such platforms have become relatively mature in recent years, and basic capabilities such as navigation and collision avoidance are provided as reliable, robust, software libraries. For example, the Fujitsu `enon` service robot platform we used provides vision-based localization and navigation, integrated with collision avoidance. When used in their intended environments, traditional ER benchmark tasks such as navigation and object avoidance are of little interest for this kind of platform because high-quality implementations based on decades of mobile robot research are already provided with the platform. Furthermore, due to safety concerns and perceptions of risk, there are significant obstacles to deploying evolved, low-level behaviors for critical tasks such as collision avoidance or navigation on commercial service robots. On the other hand, as long as robust, low-level behaviors guarantee safety, there is substantially less risk involved with considering the deployment of evolved, high-level behaviors.

Therefore, this class of platform provides the opportunity to consider ER for higher level tasks, such as high-level task planning and interaction with humans. Instead of studying low-level tasks that map raw sensor inputs to motor outputs, we focus on higher-level applications which map symbolic information gathered by the system to high level commands executed by lower-level software, such as "move to location L while avoiding obstacles". These are higher level primitives than the commands output by evolved controllers in previous work with service robots [4,31].

For high-level tasks where a single fitness function evaluation requires a very long time (e.g., tens of minutes or longer), executing the evolutionary process on the real robot tends to be impractical, and the evolutionary learning must take place in simulation. However, an issue with evolving controllers in simulation is that the controllers might not perform well when transferred to the real robot. It might be possible to overcome this problem by investing significant effort in building an accurate simulator which closely reflects the real world, but in a practical setting, this brings up the question of time and cost-effectiveness: the effort of building a high-fidelity simulation for the ER task can end up being significantly more time-consuming than carefully programming the robot by hand to achieve the desired performance.

We investigate the use of a GP approach where a very low-fidelity simulation with many trials per controller (using an "envelope of noise" approach [14]) is used to evolve a robust controller for a high-level task which maps a symbolic description of the state of the system/environment to high-level actions (commands to a lower-level navigational system).

As an application case study, we consider the problem of visitor traffic flow management in an exhibition space such as a museum using a service robot. In order to maximize the enjoyment of visitors at an exhibition space with multimedia exhibits, a robot guide can rove the exhibition space, suggesting to visitors which exhibit they should go to next. In addition, because the robot guide is itself an interesting novelty, the robot should try to maximize interactions with visitors. We model this problem as a planning problem where the controller constantly decides where to go next in order to interact with visitors, and outputs a high-level goal (exhibit) which is given to the lower level, navigation software. We show that genetic programming using a very crude simulation is sufficient to generate a controller for this task which significantly outperforms hand-designed heuristics. This controller is transferred to a service robot and demonstrated to perform well in the real world compared to the hand-coded heuristics, as well as a human operator.

The rest of the paper is organized as follows. First, we review previous work on applying GP to evolutionary robotics (Section 2). In Section 3, we describe the exhibition space visitor traffic management problem, which is the motivation for this work. Section 4 describes the application of GP to generate a controller for a guide robot which addresses this task, and presents experimental results in simulation, as well as the results when the evolved controller is transferred to an actual service robot. In Section 5, we consider whether the ER approach is practical for our problem, compared to a standard software development process. Finally, Section 6 discusses our results and directions for future work.

2 Previous Work on GP for Robot Control

The first application of GP to evolutionary robotics was the evolution of wall-following and box-pushing behaviors in simulation by Koza [18]. An early survey of evolutionary robotics by Mataric and Cliff pointed out the lack of realistic noise model in Koza's simulation and questioned whether this approach would actually work on a real robot [20]. In simulation, Ito, Iba, and Kimura investigated the robustness of GP controllers for box pushing behavior and showed that evolved controllers were quite robust with respect to initial conditions and noise in sensing and actuation [13].

Lee, Hallam, and Lund showed the feasibility of using GP to evolve a controller for box-pushing in simulation, and then successfully transferring the evolved controller to a real Khepera [19]. Ebner and Zell successfully replicated Koza's wall-following experiment both in simulation and on a real service robot (RWI B21) [4].

Nordin and Banzhaf were the first to perform embodied evolution using GP, and evolved obstacle avoidance and object tracking (following) behaviors on a Khepera [26] by mapping sensor input to motor output values. They improved upon this by mapping sensor inputs to expected utility ("learning a world model") and demonstrated this approach on obstacle avoidance and wall-following behaviors on the Khepera [27]. Sharabi and Sipper used GP to evolved Sumo robots on small, custom-built, real robots [32].

While most of the work applying GP to robot control has been on wheeled robots (simulated and real), Wolff and Nordin used Linear Genetic Programming to generate locomotion on a simulated humanoid robot [38], and Tanev applied GP to evolve locomotion a simulated snake robot [36]. Kamio and Iba proposed a method for evolving box-moving behavior in simulation for a "generic" robot. This generic behavior was then transferred to real robots (an Aibo quadruped as well as a HOAP-1 humanoid robot) and adapted to the characteristics of the specific robots using reinforcement learning [15].

Evolving robot controllers in simulation has become a common benchmark in research which proposes new GP techniques. For example, Pilat and Oppacher performed a comparative study of hierarchical GP methods for wall-following, light avoidance, and obstacle avoidance on a Khepera simulator [30]. .

In a different approach to high-level robot control, GP has been used in simulation to generate robot plans in a classical AI planning framework. Handley's work generated sequences of mid-level, discrete actions in a 2D grid world ("turn left 90°", "turn right 90°", "move forward one square", "toggle lightswitch") [8]. Spector [35], as well as Muslea [23] used GP in a classical AI planning framework to solve planning problems where possible actions can be arbitrary operators such as `pick-up-block` and `open-door`. While this line of work uses GP to generate sequences of actions that achieve a goal in a simulated, deterministic setting, we generate a policy/controller which selects actions to maximize fitness in a noisy setting for a real robot.

3 Directing Visitor Flow in an Exhibition Space Using a Guide Robot

Robot tour guides for exhibit spaces such as museums are a popular application of mobile robots. The earliest robot tour guide was Polly, which led visitors around the MIT robot lab [12]. Rhino, which gave interactive tours to visitors of the Deutsches Museum in Bonn started its tours at a predetermined start location, led visitors through the exhibits and played a recorded presentation at each exhibit, and returned to the start location, where it waited for new visitors [3]. Minerva, deployed at the Smithsonian Museum of American History, had a high-level tour planner designed to give tours which were approximately 6 minutes long (researchers determined that this was the amount of time that visitors would like to follow the robot). Since the rate of progress of a tour varies depending on the circumstances, Minerva dynamically re-planned the remainder of the tour at each exhibit in order to achieve the target 6-minute duration [37]. Ten Robox guides were deployed at the Swiss Expo.02 [34]. An average of 125 visitors were present in the exhibit at any given time, making management of visitor flow an important issue. To channel the visitor flow smoothly, the tours were constrained so that the tour led visitors closer to the exit, and centralized coordination was used in order to assign exhibits to robots in order to prevent multiple robots from simultaneously presenting the same exhibit.

Other robot guides include Sage [29], which gave tours at the Carnegie Museum of Natural History, the the Robovie guide robot, which was deployed at the Osaka Science Museum [33], and a robot guide deployed in a Japanese shopping mall [16].

Previous robot guide research has focused on giving tours to a single group of visitors at a time. However, this means that in a popular museum or exhibition, we must deploy many robots (as in [34]), or the number of visitors who can interact with the robot will be limited. As an alternative, we can have a semi-guided tour where the robot directs the visitors to an interactive exhibit. This frees the robot guide to go and give directions to another group of visitors. Decoupling exhibit presentation from the task of guiding visitors between exhibits allows a single robot to simultaneously attend to multiple groups of visitors. While there has been previous work on various aspects of robot-human interaction involving a single robot and multiple humans [2],we focus on high-level goal selection for this setting.

Previous, large-scale, long-term deployments have demonstrated that robot guides can successfully entertain and educate visitors with minimal human supervision over extended periods of time [3,37,29,34]. Basic capabilities such as localization, collision avoidance, and navigation have matured to the point that today, commercial "service robots" have become available (e.g., the Fujitsu Frontech `enon` [5]). It is now possible to work on high-level, algorithmic issues related to guide robots, such as the operational effectiveness of robot guides in achieving the overall goals of the museum or exhibit space where the robot is deployed.

One aspect of museum/exhibition operations which could be optimized by the use of a robot is the flow of visitor traffic through the exhibits. For example, a museum exhibition consists of a set of exhibits. If all of the visitors simultaneously flock to a particularly popular exhibit, it would cause congestion and overall dissatisfaction with the museum visit. Of course, the visitors will tend to autonomously adjust their course so that extreme cases (such as everybody simultaneously going to the same exhibit) are avoided. The question we wish to address is: Is it possible to use a robot guide to help direct traffic among the exhibits so that the visit experience is improved?

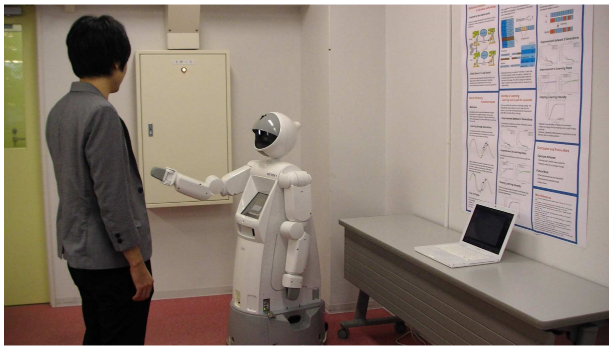

Fig. 1 Fujitsu `enon` robot guide directing visitors at a poster exhibit

3.1 A Model for the Robot Guide Problem

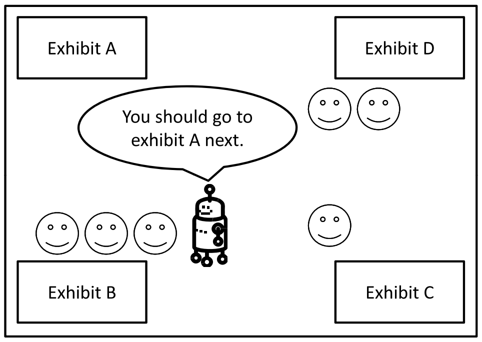

Fig. 2 A robot directs groups of visitors through a museum with 4 exhibits

We are developing a robot-guided tour environment which can be used in a museum, conference, or similar exhibition space. A single mobile robot guide is used to direct many groups of visitors simultaneously. Our model consists of a set of exhibits, visitors, and the guide robot (Fig. 2):

Exhibits: There is a set of k exhibits. At each exhibit, there is a pre-recorded presentation/demo with durations $d_1, ..., d_k$. Each exhibit can be in one of 2 states, `idle` or `presenting`, and initially starts in the `idle` state.

Presentations are started upon a visitor's request, and can not be interrupted. Additional requests to start a presentation while an exhibit is already `presenting` have no effect.

Visitors: A stream of visitors arrives at some initial location. When a visitor arrives at an exhibit i, there are 3 possibilities:

1. The exhibit is currently `idle` (not giving a presentation). In this case, the visitor requests a presentation. The exhibit state changes to `presenting`, and the visitor state changes to `listening`.
2. There is an ongoing presentation (exhibit state = `presenting`), and the time remaining until the end of the presentation is less than $d_i/2$ (the visitor has missed over half of the presentation). In this case, the visitor will listen to the remainder of the presentation, and then, at the end of the current presentation, will restart the presentation and stay until this second cycle of the presentation is completed.
3. There is an ongoing presentation (exhibit state = `presenting`), and the time remaining until the end of the presentation is greater than or equal to $d_i/2$ (i.e., the visitor can listen to the majority of the presentation). In this case, the visitor will remain at the exhibit until the end of the current presentation, and then leave for another exhibit.

When a visitor leaves a location (an exhibit or the entrance of the exhibit space), the next destination is decided as follows: If v has visited all exhibits, he exits the exhibition. If the robot guide suggests a destination, v will follow the suggestion and select the suggested destination with probability p_s (we refer to p_s as the *visitor compliance* parameter). Otherwise, v selects a destination according to a selection heuristic. If v has not received any suggestions from the robot, v will select a target according to a selection heuristic. The default selection heuristic is to select the least crowded exhibit which v has not yet visited (ties are broken randomly).

Guide Robot: There is a single, robot guide which moves freely in the exhibition area. Given a target location, the robot can plan a path to the goal and move towards the goal while avoiding any obstacles.

The robot is in one of 4 states:

– `idle`: This is the initial state for the robot.
– `moving`: Moving towards an exhibit (once a robot begins to move toward an exhibit, it will not change its destination midstream).
– `waiting`: The robot is at an exhibit where a presentation is being given; when the presentation ends, the robot state will change to `directing`, and it will begin to give directions to the visitors at the exhibit
– `directing`: The robot is giving directions to visitors at an exhibit.

If the robot arrives at an exhibit where there are visitors listening to a presentation, the robot waits until the presentation is complete, and then suggest to each of the visitors which exhibit to go to next. The process by which the robot makes the suggestion are described in Section 4. The visitors may ignore the robot's directions with probability $(1 - p_s)$ and decide their own destination, as described above.

3.2 An objective function for the robot guide

Our goal is to make the exhibition visit more enjoyable for the visitor, so we would like a *quantifiable* metric correlated with enjoyment. Although it is difficult to quantify "enjoyment", we can identify two factors related to enjoyment which can be influenced by our robot guide system.

First, visitor tours should be managed such that the amount of redundancy in the presentations heard is minimized. In a crowded space where exhibits have prerecorded presentations, it is desirable for visitors to synchronize their tour such that they avoid spending time listening to partial presentations. Ideally, the guest arrives at each exhibit exactly when a new presentation begins.

Second, we assume that interaction with the robot enhances enjoyment of the visit. Although lengthy interactions with robot guides is not necessarily enjoyable or sustainable, particularly compared to a human tour guide (c.f., [29]), mobile robots are still novel enough that we would like visitors to have *some* interaction with the robot during their visit.

Taking these considerations into account, the robot guide is defined as the problem of programming a robot guide which seeks to: (1) Schedule visitor arrivals at exhibits so that they arrive when the presentation has not yet started, or as close as possible to the start of the presentation; and (2) Maximize the number of visitors who interact with the robot.

More precisely, the objective is to minimize $f = \sum_v (w_P P(v) + w_I I(v))$, a weighted sum of 2 components: (1) the exhibit arrival component,

$$P(v) = \sum_e \left(\frac{-4(t - T_i/2)^2}{T_i^2} + 1\right)$$

which is minimized ($P(v) = 0$) when t, the arrival time of the visitor at exhibit i coincides with the start of a presentation, and is maximal (worst) when the visitor arrives exactly halfway through the duration T_i of the presentation; (2) the robot interaction component, $I(v)$, where $I(v) = 0$ if visitor v had at least 1 interaction with the robot guide, and 1 otherwise. In the experiments below, $w_P = 5$ and $w_I = 1$.

3.3 Controllers for the Robot Guide

The guide robot's behavior consists of moving to an exhibit where either a presentation is currently ongoing (or is expected to start when a visitor arrives), and when the presentation ends, suggesting to each visitor who has just finished listening to the presentation which exhibit to visit next. This can be decomposed into two decision problems: First, the robot must decide which exhibit to move to. Second, the robot must decide which exhibit to suggest to each idle visitor.

The second problem (selecting the exhibit to suggest to each visitor) is handled in a greedy manner using one-step lookahead as follows: The robot is aware of the current state of every exhibit. For each exhibit e, based on the current state of e and the estimates of visitor speed, we can estimate what state e will be in if the visitor moves to e from his current location. For each idle visitor, the robot suggests the exhibit which will minimize the presentation penalty $P(v)$.

In this paper we focus on the first problem, which is the problem of deciding which exhibit the robot moves to. This is difficult because the visitors behave autonomously and their actions are not entirely predictable. As described in Section 3, if the robot is not present when a visitor finishes listening to a presentation, the visitor chooses his destination autonomously; furthermore, even if the robot makes a suggestion, the visitor may ignore the suggestion and choose his own destination.

Several, obvious baseline strategies can be implemented for this problem:

- `Stationary` - Remain stationary at Exhibit #1. Since all visitors will eventually visit this exhibit, this will maximize the number of visitors who interact with the robot;
- `Random` - move toward a randomly selected exhibit;
- `Busiest-exhibit`: move toward the exhibit which currently has the largest number of visitors who are listening to the exhibit demo;
- `Most-fresh-visitors` - move towards the exhibit which currently has the largest number of visitors who have not yet interacted with the robot.

3.4 Prototype Robot Environment

We implemented a physical prototype of the robot guide system described above using a Fujitsu Frontech `enon` service robot [5] (Figure 3. The `enon` robot is a wheeled mobile robot with a humanoid torso and head, and has an onboard x86 PC with wireless communications. Standard capabilities of the `enon` (provided in the systems software) include obstacle detection and visually guided navigation.

All of the exhibits (consisting of a poster and a PC which is used to both give a multimedia presentation and track visitor locations) as well as the robot are networked, so all information needed by the robot controller would be readily available. Visitor location and history are tracked by having visitors register their arrival at each exhibit. In a real-world deployment, the system could track the location and visit history of each visitor by having visitors carry an RFID tag, and placing RFID readers/sensors at each exhibit.

The software architecture of our robot guide system is shown in Figure 3, and consists of the following, distributed components:

– Presenter (one instance at each exhibit): A process running at a notebook PC located at every exhibit. When a visitor arrives at an exhibit, he registers himself at the exhibit Presenter. Registration can be done by entering his unique visitor ID number, or an RFID tag-based tracking of visitors. Each Presenter sends this registration information, as well as the state of the exhibit, to the Commander, allowing the system to perceive the visitors' locations, as well as keep track of each visitor's history. The Presenter plays a prerecorded, audio presentation which explains the exhibit.
– `Enon`-Server: This process runs on the `enon` robot. It sends robot state information to the Commander, and receives destination commands from the Commander. The destination is given to the onboard navigation system, and the robot moves to its destination while avoiding obstacles.
– Commander: This is the central, decision-making component (running on the robot) which receives Presenter state and visitor registration data from all of the Presenters, as well as state information from the `enon`, and based on this information, decides upon the next action (which exhibit to move the robot to). The decision function can either be hand-coded, or evolved using a GP.

This prototype system has been implemented and tested in a physical environment. The `enon` robot guide guides student "visitors" in an environment with 4 poster exhibits. A notebook PC running a Presenter is located at each of the 4 exhibits. The current Commander uses a controller evolved in simulation. Our current research is focusing on evolving successful controllers for this physical platform, using techniques such as on-line adaptation of controllers which are initially evolved in simulation, then improved with on-board learning.

Unfortunately, experiments using the actual robot are time consuming – each experiment requires at least 30 minutes, and some time is required between each run to reset/initialize the robot and environment. Furthermore, the number of consecutive experiments that can be performed is limited because the robot must be recharged after several hours of operation. Under these adverse conditions, an experimental evaluation involving actual human "guests" moving around the exhibit hall and interacting with the robot was not feasible due to time and resource limitations. Instead, we used "virtual guests", which are simulated guests who behaved like the guests in the simulated environment (Section 4.1). Thus, in the robot experiments, the `enon` robot moves around in the actual physical exhibit environment using vision-guided navigation (which means that there is significant noise in the system); however, instead of interacting with real human guests who

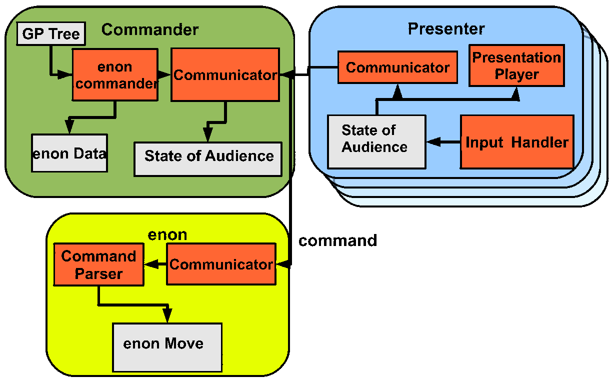

Fig. 3 Visitor Traffic Flow Management System Architecture

are tracked using RFID tags, the robot (and exhibits) "interact" with simulated, virtual humans.

This hybrid physical-virtual experimental setup approximates the intended real-world robot guide scenario and allows us to perform repeated experiments to evaluate the behavior of candidate guide controllers in the physical world.

4 Generating Robot Guide Controllers Using Genetic Programming

We use strongly-typed genetic programming [22], a variant of genetic programming (GP)[18], to automatically search for controller programs expressed in a domain-specific language for the robot guide domain. In strongly-typed GP, every slot has an associated *type*, and only functions or terminals which satisfy the type constraint can be used to fill the slot. The root node of every individual returns an object of type *exhibit* – in other words, all individuals generated by the GP system return a valid exhibit object. The functions in the domain-specific language defining the space of controllers searched by the GP system are described in Table 1.

In addition to these functions, the terminals used by the GP system include standard numbers, comparator symbols ($<, >, \leq, \geq, =, \neq$), symbols representing each of the 4 exhibits (`+E1+`, `+E2+`, `+E3+`, `+E4+`). There are also 3 terminals of type "criterion" which are the criteria used by the functions `IF-EXHIBIT-COND`, `IF-EXHIBIT-COND-RATIO`, and `IF-EXHIBIT-COND-DIFF` to compare relevant properties of exhibits. These 3 terminals are `NUM-VISITORS`, `NUM-UNTOUCHED-VISITORS`, and `DISTANCE-TO-ROBOT`, which consider the number of visitors currently at an exhibit, the number of visitors at an exhibit who have not yet interacted with the robot, and the distance from an exhibit to the robot, respectively. For example, we can use this language to implement the `Busiest-exhibit` heuristic as:

```
(BUSIER-EXHIBIT
  (BUSIER-EXHIBIT +E1+ +E2+)
  (BUSIER-EXHIBIT +E3+ +E4+))
```

Similarly, the `Most-fresh-visitors` heuristic can be implemented in our language as:

Functions		
name	arguments (types)	description
`IF-RAND-LT`	p (float) e_1, e_2 (exhibit)	Let r be a randomly generated floating point number. If $r \leq p$, then return e_1, else return e_2
`IF-EXHIBIT-COND`	*comp* (comparator) *cri* (criterion) n (cond-int) e_1, e_2 (exhibit)	Compute $c_1 = criterion(e_1, cri)$ and $comp(c_1, n)$, where $comp(c_1, n)$ is the comparator *comp* applied to c_1 and n. Return e_1 if $comp(c_1, n)$ is true; otherwise return e_2. E.g., `(IF-EXHIBIT-COND = 0 +num-visitors+` e_1 e_2`)` returns e_1 if the number of visitors at exhibit e_1 is 0, and otherwise returns e_2.
`IF-EXHIBIT-COND-RATIO`	*comp* (comparator) *cri* (criterion) *ratio* (double) e_1, e_2 (exhibit)	Compute $c_2 = criterion(e_2, cri)$. If $c_2 = 0$ (special case to avoid division by 0), then return $criterion(e_1, cri)$. Next, compute $c_1 = criterion(e_1, cri)$. Let $r = c_1/c_2$. If $comp(r, ratio)$ is true, then return e_1, else return e_2, where $comp(r, ratio)$ is the comparator *comp* applied to r and *ratio*. E.g., `(IF-EXHIBIT-COND-RATIO >= 0.5 NUM-VISITORS` e_1 e_2`)` returns e_1 if the ratio of the number of visitors at e_1 and e_2 is greater than or equal to 0.5 (or if $e_2 = 0$), and otherwiese returns e_2.
`IF-EXHIBIT-COND-DIFF`	*comp* (comparator) *cri* (criterion) *ratio* (cond-int) e_1, e_2 (exhibit)	(Similar to `IF-EXHIBIT-COND-RATIO`): Compute $c_2 = criterion(e_2, cri)$ and $c_1 = criterion(e_1, cri)$. Let $d = c_1 - c_2$. If $comp(d, ratio)$ is true, then return e_1, else return e_2.
`MOST-UNTOUCHED-EXHIBIT`	e_1, e_2 (exhibit)	Compute u_1 and u_2, the number of visitors who are at e_1 and e_2, respectively, who have not yet interacted with the robot. Returns e_1 if $u_1 \geq u_2$, and returns u_2 otherwise.
`CLOSER-EXHIBIT`	e_1, e_2 (exhibit)	Returns e_1 if e_1 is closer to the robot than e_2; otherwise returns e_2.
`FURTHER-EXHIBIT`	e_1, e_2 (exhibit)	Returns e_1 if e_1 is further from the robot than e_2; otherwise returns e_2.
`BUSIER-EXHIBIT`	e_1, e_2 (exhibit)	Returns e_1 if there are more visitors at e_1 than e_2; otherwise returns e_2.
`EARLIER-END-EXHIBIT`	e_1, e_2 (exhibit)	Let end_i be the time remaining until the end of the current demonstration being played at exhibit i ($t_i = 0$ if exhibit i is currently not giving a demo). Return e_1 if $end_1 < end_2$; otherwise returns e_2.

Table 1 Functions in our domain specific language for Robot Controller Specification

```
(MOST-UNTOUCHED-EXHIBIT
  (MOST-UNTOUCHED-EXHIBIT +E1+ +E2+)
  (MOST-UNTOUCHED-EXHIBIT +E3+ +E4+))
```

The GP is a steady-state algorithm, where at every iteration, 2 individuals are selected using tournament selection and mated. The mating uses crossover and mutation to generates 2 children, which are then inserted into the population using a reverse tournament selection where the worst members in the tournament are replaced.

Since our fitness function requires a long experiment to evaluate, embodied evolution of the controller using the real robot (which would require hundreds or thousands of experiments) would be prohibitively expensive, so we rely on simulation-based evolution to synthesize a controller which can be transferred to the robot.

We did not have access to a simulation environment for our Fujitsu `enon` robot, so we had to implement our own simulator. Thus, we implemented an extremely simple, low-fidelity simulator which only models what we believed to be essential features of the problem. The robot, as well as the visitors, are treated as point masses, so collision avoidance is not modeled directly. [1]

Instead of attempting to implement a high-fidelity simulation which accurately models the inherent sources of noise in the system (obstacle avoidance, vision, navigation, etc), we apply the approach of Jakobi [14], and apply an "envelope of noise" (described below in Section 4.1) in the evaluation function in order to use this low-fidelity simulator to generate a controller that can perform well when transferred to the real robot. Rather than trying to evolve a controller that performs very well for one particular, simulated environment (where the simulation may differ significantly from reality), we evolve a controller to perform well on a wide range of simulated environments – it is hoped that an evolved controller which performs robustly across a wide range of environments will perform relatively well (i.e., better than hand-crafted controllers) when transferred to the real robot.

4.1 Simulation Results

We used the GP system described above to evolve a controller for a robot guide. As shown in Figure 2, the simulated exhibit space has 4 exhibits, located at the 4 corners of a room. The GP population size is 1000, and the GP system is executed until 20,000 candidates are generated and evaluated. The crossover probability was 0.6, and mutation was applied to both children if and only if crossover was not applied. Tournament selection was used (tournament size 20), and the max depth of the trees generated during the run is 6. Trees deeper than this limit are pruned to the depth limit by replacing subtrees which exceed the limit with a terminal with a matching type.

We evaluate the fitness of a candidate individual (controller) as follows: We generate 300 random environments, with three randomized parameters:

- The robot's speed is set to a value in the range $(0.1v, 0.3v)$, where v is the speed of the visitor (a constant value approximating actual human speed in a room with the same dimensions as the simulated room);
- The arrival rate r is selected randomly from $(0.01, 0.05)$ – this rate is used as the arrival rate in a Poisson process, so the Poisson arrival rate is set to r/step;
- The visitor compliance rate is set in (0.5,1.0).

The controller is evaluated once in each environment using a 1800 time step simulation (where each time step is equivalent to a second in the real world). Its fitness score is the sum of the scores on the 300 individual simulations. Thus, each controller is evaluated in a large variety of randomized test cases – since our

[1] The entire simulation system, including some unit test code, as well as a simple GUI for displaying simulation runs and setting some control parameters, is only 770 lines of Common Lisp code, and required approximately 25 hours to implement, so this is clearly not an elaborate simulator.

GP Run 1								
	Rank (out of 8 controllers)							
	1 (best)	2	3	4	5	6	7	8 (worst)
Evolved-best1	**83.7**	1.6	1.9	2.4	3.5	6.9	0.0	0.0
Initial-pop-best1	4.5	69.8	19.5	6.0	0.1	0.1	0.0	0.0
Initial-pop-median1	3.0	6.2	1.3	1.6	26.7	61.2	0.0	0.0
Initial-pop-worst1	0.0	0.0	0.0	0.0	0.0	0.0	20.5	79.5
Busiest-exhibit	0.5	9.1	41.4	39.0	5.8	4.2	0.0	0.0
Most-fresh-visitors	1.4	9.5	34.6	50.2	2.9	1.4	0.0	0.0
Stationary	6.9	3.7	1.3	0.9	60.9	26.3	0.0	0.0
Random	0.0	0.0	0.0	0.0	0.0	0.0	79.4	20.6

Table 2 Simulation-based evaluation of hand-coded and evolved controllers: Performance (rank) on 10,000 test cases (1,000 different randomized environments, 10 runs per environment). The fraction (%) of test cases where the controller was ranked n-th (out of the 8 controllers) is shown, e.g., the `Evolved-best1` controller was ranked 1st on 83.7% of the test cases.

simulation is extremely low fidelity, it is hoped that by searching for a controller which performs well across the entire spectrum of environments, we will obtain a controller that performs well when transferred to the real robot.

The GP was parallelized for multi-core architectures using a standard thread pool pattern so that one simulation run is executed on each thread, and linear scaling was observed up to 4 cores. The current implementation of the simulator-based GP requires 5-6 hours to execute on a 2.93GHz Core i7 workstation for a single run with 10,000 matings.

We evaluate the following controllers generated by a single, representative run of the GP system:

– `Evolved-best1` - the best individual (controller) found during the GP run (Figure 4);
– `Initial-pop-best1` - The best individual from the initial population in the GP run;
– `Initial-pop-median1` - The median individual in the initial population of the GP run;
– `Initial-pop-worst1` - The worst individual in the initial population of the GP run.

We compare these automatically generated controllers to the baseline `Busiest-exhibit`, `Most-fresh-visitors`, `Stationary`, and `Random` controllers described in Section 3.3.

We compared the controllers as follows. A set of 1000 environments was generated (using the same randomized process used to generate test cases for the GP test function in Section 4). For each of these 1000 environments, we generated 10 test cases, $c_{i,1}, ..., c_{i,10}$, where each test case consists of the environment E_i ($1 \leq i \leq 1000$), and a set of randomly generated visitor arrival times (based on the Poisson arrival rate, which is one of the environment parameters). On each of the 10,000 test cases, we executed each candidate controller once using a 5000 time step run, and evaluated its fitness function score on the test case, and ranked the controller's score compared to the score of the other controllers on the same test case.

Table 2 shows the frequency that each controller was ranked 1st (best), ..., 8th (worst) in each of these 10,000 comparisons. For example, the `Evolved-best1`

```
(EARLIER-END-EXHIBIT
 (MORE-TARGETED-EXHIBIT
  (MORE-UNTOUCHED-EXHIBIT
    *E3*
   (EARLIER-END-EXHIBIT
    (MORE-TARGETED-EXHIBIT
     (MORE-UNTOUCHED-EXHIBIT *E1* *E2*) *E1*)
    (CLOSER-EXHIBIT
     (MORE-TARGETED-EXHIBIT *E4* *E2*)
     (EARLIER-END-EXHIBIT *E1* *E2*))))
  (CLOSER-EXHIBIT (MORE-TARGETED-EXHIBIT *E4* *E1*)
                  (EARLIER-END-EXHIBIT
                   (MORE-TARGETED-EXHIBIT *E1* *E3*)
                   *E2*)))
 (CLOSER-EXHIBIT
  (MORE-TARGETED-EXHIBIT
   (MORE-UNTOUCHED-EXHIBIT
    (MORE-UNTOUCHED-EXHIBIT *E4* *E2*)
    (IF-EXHIBIT-COND-RATIO
     <= DISTANCE-TO-ROBOT 0.25d0
     (MORE-UNTOUCHED-EXHIBIT *E3* *E4*)
     (MORE-UNTOUCHED-EXHIBIT *E1* *E3*)))
   *E1*)
  (EARLIER-END-EXHIBIT
   (MORE-TARGETED-EXHIBIT *E2* *E4*)
   (EARLIER-END-EXHIBIT
    (MORE-TARGETED-EXHIBIT *E1* *E3*)
    (MORE-UNTOUCHED-EXHIBIT
     (MORE-UNTOUCHED-EXHIBIT *E3* *E2*) *E4*)))))
```

Fig. 4 The "`Evolved-best1`" robot controller generated by genetic programming

controller ranked 1st in 83.7% of the 10,000 trials, 2nd in 1.6% of the trials, and 6th in 6.9% of the trials, and never ranked 7th or 8th. These results show that:

- The best evolved controller significantly outperforms all other controllers, including the baseline controllers.
- The best randomly generated controller in the initial GP population (Initial-pop-best) performs significantly better than all of the baseline controllers.
- The simple, busiest-exhibit heuristic, which is arguably one of the most obvious heuristics for this problem, performs worse than the best of 1000 randomly generated controllers (i.e., the best member of the initial population).

The best evolved controller found by this particular GP run is shown in Figure 4. The behavior of a robot controller is quite complex, depending on both the code itself and the environment in which the code is executed. Our domain-specific language for robot controllers was carefully designed and constrained so that the GP system would generate controllers in our domain. Thus, the result that the best individual from the GP initial population performed significantly better than the busiest-exhibit and most-fresh-visitors heuristics indicates that this design was successful. This also indicates that in this domain, human intuitions on controller design can lead us astray. Thus, our data indicates that designing a successful controller for our robot guide problem is a difficult task which can be facilitated by automated design methods such as GP.

4.1.1 Robustness of the GP system

As mentioned above, the results in Table 2 were generated using the default settings for our GP library. In order test whether the success was dependent on that particular set of GP control parameters, we evaluated the performance of the GP system using a variety of control parameter settings.

We performed 5 additional runs of the GP (GP Runs 2-6). Each GP run used a different, randomly generated set of control parameter settings, where the population size was chosen from $[100, 2000]$, the crossover rate was chosen from $[0, 1.0]$, and the tournament size was chosen from $[5, 40]$. For each of these 5 runs, the set of parameters is shown in Table 3. The best controller generated by each GP run, as well as the best, median, and worst members of the initial population, and a the set of hand-coded heuristics (`Busiest-exhibit`, `Most-fresh-visitors`, `Random`, `Stationary`) using 10,000 test cases. Table 3 shows the frequency that each controller was ranked 1st (best), ..., 8th (worst) out of these 10,000 comparisons.

From the results in Table 3, it is clear that the results are qualitatively similar to the results of Table 2. The best evolved controller on each run is consistently the best performer, showing that the ability of the GP system to reliably evolve controllers which are better than hand-crafted controllers does not depend critically on a particular set of control parameter values.

These robust results suggest that perhaps GP is not necessary – given our domain-specific language for describing guide robot controllers (Table 1), perhaps random generation of expressions in this language is sufficient. We tested this hypothesis by evaluating Random-Generate-and-Test (RGT), a trivial algorithm which simply generates n expressions of up to depth 6 using the same random s-expression generator used by the GP code, evaluates them using the same fitness function used by the GP, and returns the best s-expression found from the n samples. We ran RGT 5 times, where each run of RGT, the number of individuals generated and evaluated was $n = 20,000$, and the best fitness scores found in the 5 runs were $Best_{RGT} = \{481.42, 476.27, 479.34, 483.77, 478.52\}$. In comparison, the fitness scores of the best individuals found by the 5 GP runs using randomized control parameters in Table 3 were $Best_{GP} = \{467.71, 463.57, 462.74, 470.84, 466.20\}$. We compared $Best_{RGT}$ and $Best_{GP}$ using a non-parametric, Mann-Whitney U-test and verified that there was a statistically significant difference ($p = 0.0079$). This indicates that a directed search using GP (even with randomized control parameters) performs significantly better than baseline, randomized generate-and-test.

4.1.2 Robustness of the evolved controller in various simulated environments

Although the previous data shows that evolved controller outperformed the other controllers overall across a range of test cases, it is possible that there are some specific classes of test cases in where the evolved controller does not perform well. To test whether this is the case, we evaluate the controllers on specific subclasses of test cases, perturbing a single parameter.

First, we consider the effect of perturbing visitor compliance. In the training cases used during GP-based evolution, compliance was set in the range $(0.5, 1.0)$, that is, visitors follow directions given by the guide robot at least $1/2$ of the time. We evaluated the evolved and hand-coded controllers by executing simulations

	Rank (out of 8 controllers)							
	1 (best)	2	3	4	5	6	7	8 (worst)
GP Run 2 (population=1274, crossover rate=0.1, tournament size=27)								
Evolved-best2	**86.3**	11.7	1.7	0.3	0.0	0.0	0.0	0.0
Initial-pop-best2	11.5	64.1	18.1	5.8	0.5	0.0	0.0	0.0
Initial-pop-median2	0.0	0.6	3.3	12.5	73.8	9.8	0.0	0.0
Initial-pop-worst2	0.0	0.0	0.0	0.0	0.0	0.0	34.0	66.0
Busiest-exhibit	0.7	9.1	33.6	46.7	9.7	0.2	0.0	0.0
Most-fresh-visitors	1.5	14.4	43.2	34.5	6.1	0.3	0.0	0.0
Stationary	0.0	0.0	0.1	0.3	9.8	89.8	0.0	0.0
Random	0.0	0.0	0.0	0.0	0.0	0.0	65.9	34.1
GP Run 3 (population=1604, crossover rate=0.9, tournament size=11)								
Evolved-best3	**71.2**	26.6	2.2	0.0	0.0	0.0	0.0	0.0
Initial-pop-best3	27.3	63.7	7.7	1.3	0.0	0.0	0.0	0.0
Initial-pop-median3	0.0	0.0	0.0	0.3	47.2	52.5	0.0	0.0
Initial-pop-worst3	0.0	0.0	0.0	0.0	0.0	0.0	30.3	69.7
Busiest-exhibit	0.4	3.1	36.3	59.5	0.7	0.0	0.0	0.0
Most-fresh-visitors	1.0	6.7	53.6	38.5	0.2	0.0	0.0	0.0
Stationary	0.0	0.0	0.0	0.6	51.9	47.5	0.0	0.0
Random	0.0	0.0	0.0	0.0	0.0	0.0	69.7	30.3
GP Run 4 (population=1804, crossover rate=0.83, tournament size=30)								
Evolved-best4	**87.2**	12.2	0.5	0.1	0.0	0.0	0.0	0.0
Initial-pop-best4	12.4	72.6	12.2	2.7	0.1	0.0	0.0	0.0
Initial-pop-median4	0.0	0.0	0.0	0.2	41.3	58.5	0.0	0.0
Initial-pop-worst4	0.0	0.0	0.0	0.0	0.0	0.0	33.2	66.8
Busiest-exhibit	0.2	6.6	36.9	55.8	0.5	0.0	0.0	0.0
Most-fresh-visitors	0.2	8.6	50.2	40.6	0.4	0.0	0.0	0.0
Stationary	0.0	0.0	0.2	0.6	57.7	41.5	0.0	0.0
Random	0.0	0.0	0.0	0.0	0.0	0.0	66.8	33.2
GP Run 5 (population=560, crossover rate=0.58, tournament size=12)								
Evolved-best5	**92.6**	7.2	0.2	0.0	0.0	0.0	0.0	0.0
Initial-pop-best5	6.9	67.7	20.4	4.9	0.1	0.0	0.0	0.0
Initial-pop-median5	0.0	0.0	0.7	2.0	65.7	31.6	0.0	0.0
Initial-pop-worst5	0.0	0.0	0.0	0.0	0.0	0.0	34.1	65.9
Busiest-exhibit	0.2	9.6	33.1	55.5	1.6	0.0	0.0	0.0
Most-fresh-visitors	0.3	15.5	45.6	37.3	1.2	0.1	0.0	0.0
Stationary	0.0	0.0	0.0	0.2	31.5	68.3	0.0	0.0
Random	0.0	0.0	0.0	0.0	0.0	0.0	65.9	34.1
GP Run 6 (population=900, crossover rate=0.84, tournament size=26)								
Evolved-best6	**72.0**	24.5	2.9	0.6	0.0	0.0	0.0	0.0
Initial-pop-best6	24.8	55.1	15.0	5.1	0.0	0.0	0.0	0.0
Initial-pop-median6	0.0	0.0	0.0	0.5	48.2	51.3	0.0	0.0
Initial-pop-worst6	0.0	0.0	0.0	0.0	0.0	0.0	31.8	68.2
Busiest-exhibit	1.0	8.5	36.4	53.5	0.6	0.0	0.0	0.0
Most-fresh-visitors	2.1	12.0	45.6	39.9	0.4	0.0	0.0	0.0
Stationary	0.0	0.0	0.1	0.4	50.8	48.7	0.0	0.0
Random	0.0	0.0	0.0	0.0	0.0	0.0	68.2	31.8

Table 3 Performance of GP using randomly selected control parameter values (population size, crossover rate, tournament size). Simulation-based evaluation of hand-coded and evolved controllers: Performance (rank) on 10,000 test cases (1,000 different randomized environments, 10 runs per environment). The fraction (%) of test cases where the controller was ranked n-th (out of the 8 controllers) is shown, e.g., the `Evolved-best2` controller was ranked 1st on 86.3% of the test cases.

where visitor compliance parameter varied between 0.0 (visitors never follow directions) and 1.0 (visitors always follow directions given by the robot guide). For each compliance parameter value, 1000 runs were executed. The results are shown in Figure 5. The evolved controller is quite robust with respect to visitor compliance and outperforms the other controllers regardless of the compliance value.

Next, we consider the effect of perturbing visitor arrival rate. During the GP-based evolution, the arrival rate was randomly selected from $(0.01, 0.05)$. We evaluated the evolved and hand-coded controllers by executing simulations where visitor arrival rate varied between 0.005 and 0.05. For each arrival rate, 1000 runs were executed. The results are shown in Figure 6. In this case, the evolved controller appears to be at least as robust as the other controllers. However, as the arrival rate increases significantly, the advantage compared to the hand-coded heuristics disappears. This is to be expected because when the arrival rate is too high, the exhibition space is simply too crowded – given the time constraints, it becomes physically impossible for the robot to interact with all of the visitors and guide them optimally, regardless of the controller strategy.

These results confirm that the evolved controller performs robustly across a wide range of environments, and there do not appear to be subclasses of environments which reveal obviously identifiable "holes" in the evolved controller's performance.

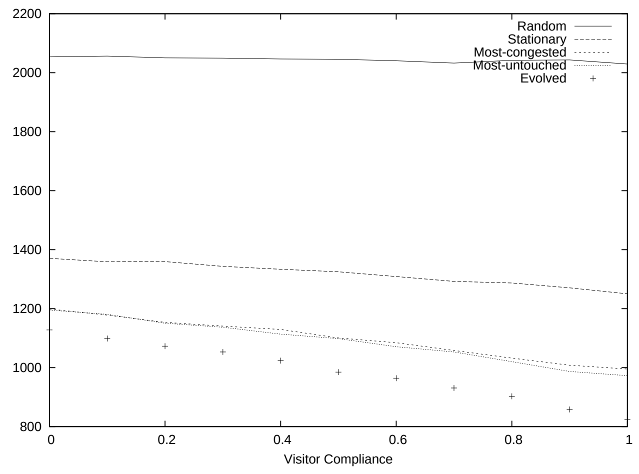

Fig. 5 Visitor Compliance vs Controller Score (scores are the sum of 1000 runs)

4.2 Results on the Physical Robot

We evaluated the performance of the evolved and hand-crafted controllers as follows. First, we ported the `Evolved-best1`, `Busiest-exhibit`, `Most-fresh-visitors`, `Random`, and `Stationary` controllers to the `enon` robot.

Seven experimental trials were performed to evaluate these controllers, where each trial is defined by a set of 60 visitor arrival times.

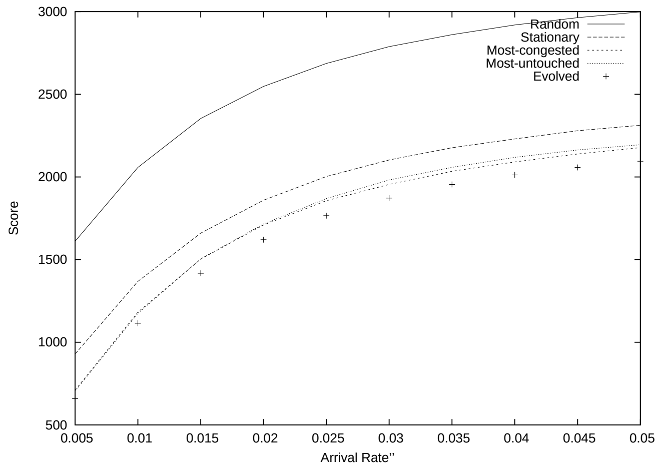

Fig. 6 Visitor Arrival Rate vs Controller Score (scores are the sum of 1000 runs)

	Fitness on each trial				
Trial	Evolved-best1	Busiest-exhibit	Most-fresh-visitors	Stationary	Random
1	**555.56**	591.03	582.83	583.47	567.31
2	**568.63**	621.64	611.10	589.48	592.03
3	**586.32**	629.85	643.41	638.39	633.25
4	**526.66**	618.64	579.84	621.93	576.08
5	576.80	**572.12**	593.67	653.64	628.53
6	**581.72**	629.34	643.44	683.88	676.56
7	**577.07**	581.06	585.15	584.09	583.19
p-value		0.02	0.03	0.02	0.02

Table 4 Evaluation of controllers on `enon` robot. The *p*-value for a Wilcoxson signed rank test comparison vs. the `Evolved-best1` controller are shown.

Each trial (row in Table 4) shows the objective function score for each of the 5 controllers using the same sequence of visitor arrival times (this design provides the same initial conditions for all 5 controllers and allows us to use a repeated measurements / paired difference test in our statistical analysis). The data shows that the `Evolved-best1` controller consistently performs ranked 1st or 2nd compared to the other controllers on each of the seven trials

We compared the results of the `Evolved-best1` controller against each of the other controllers using the Wilcoxson signed rank test, a standard, non-parametric, paired difference test used to test whether the median difference between pairs of observations is zero. As shown in Table 4, for all four comparisons (`Evolved-best1` vs `Busiest-exhibit`, `Evolved-best1` vs `Most-fresh-visitors`, `Evolved-best1` vs `Random`, `Evolved-best1` vs `Stationary`), $p < 0.05$, indicating that the median differences between `Evolved-best1` and each of the other controllers is significant.

4.3 Comparisons vs. Human Operators

In the experiments above, we have compared the evolved guide controller to some baseline heuristic controllers and show that the evolved controller outperforms the baseline controllers. Next, we consider human performance on our guide task. We implemented a remote control interface to assist in decision making, where relevant state information (e.g., for each guest, the list of exhibits which the guest has not visited yet, the amount of time remaining in the current presentation at every exhibit, etc.) is displayed on a GUI. In order to further assist the human the output of the `Evolved-best1`, `Busiest-exhibit`, `Most-fresh-visitors`, `Random`, and `Stationary` controllers are also displayed on the GUI.

Furthermore, the controller interface allows the human operator to suspend the state of the entire system, including the robot, the exhibit presentations, and all of the guests. This allows the human to carefully consider the state of the system as well as all of the outputs from the automated controllers before making each decision.

Thus, the human operator must perform the same decision-making task as the automated controllers given the same raw information as the fully automated controllers, as well as additional "advice" (i.e., the outputs of all of the automated controllers at each particular instant), and the ability to pause the entire system in order to make deliberate decisions.

We generated single arrival sequence of 30 visitors (using a Poisson arrival rate of 0.03, as in the simulation). Using this arrival sequence, we evaluated the `Evolved-best1` and `Random` controllers (6 trials each). The `Evolved-best1` controller scores were 258.87, 279.80, 290.41, 297.03, 300.10, and 307.16 (mean=288.90, SD=15.87). The random controller scored 295.06, 308.57, 308.99, 313.71, 320.87, and 346.67 (mean=315.65, SD=15.87). Although the initial conditions are identical, each trial results in different results because there is considerable noise in the system.

We then evaluated human performance on 19 trials by the same, experienced human operator (the first author) on the same arrival sequence. The human operator's score (shown in the order the trials were actually executed) were: 293.15, 282.09, 241.04, 272.49, 300.7, 300.6, 279.38, 259.15, 309.32, 266.11, 309.52, 292.0, 286.85, 234.56, 276.33, 262.7, 285.94, 315.92, and 282.02 (mean=281.54, SD=21.51).

A Mann-Whitney U-test (a nonparametric test) was used to compare the `Evolved-best1`, `Random`, and human results, with the following results:

- Human vs `Random`: $p = 0.002$
- `Evolved-best1` vs `Random`: $p = 0.015$
- Human vs `Evolved-best1`: $p = 0.598$

The comparisons against the `Random` controller verify that both the human operator and the `Evolved-best1` controller significantly outperform the `Random` controller in this experiment. Furthermore, there does not appear to be a significant difference between the experienced human operator and the `Evolved-best1` controller. Since all 19 trials use the exact same visitor arrival sequence, the human operator was capable of on-line learning and improving performance on the repeated trials, and these conditions should favor the human controller. However, the data does not indicate a pattern of improvement over the trials (which are listed in chronological order). Note that this human performance data was originally obtained for the purpose of providing training data for a supervised learning approach to fine-tune the behavior of the evolved controller (not described here), so the human operator was highly motivated to obtain the best results possible.

These results demonstrate that evolution in simulation is sufficient to generate a controller which, when transferred to a physical robot, performs comparably to an experienced human.

5 Is the Evolutionary Robotic Approach Practical?

We now consider the cost-effectiveness of applying evolutionary robotics for our application, compared to the baseline alternative, which is to design and implement a controller by hand.

The total human effort to generate the evolved controller (Figure 4) was less than 30 hours. This includes approximately 25 hours to develop and debug the simulator (including the simulation code, GUI, unit tests, and scripts), as well as approximately 4 hours to interface and test the simulator with a pre-existing genetic programming library (a GP library which was implemented by one of the authors for a previous, unrelated project [6]). We do not include the development time for the software for tracking visitor locations and interfacing the robot to the exhibits, which would be required regardless of the method used to develop the controller (evolved or hand-coded). The time for building this infrastructure was on the same order as the time to develop the code specific to evolutionary robotics system.

Furthermore, a single run of the multi-threaded GP requires 5-6 hours to execute on a 4-core workstation, but this is a completely unattended operation (no human involvement), and as we showed in Section 4.1.1, the GP system can reliably generate good solutions, so very little human effort is required. No time was spent on GP parameter tuning – we used the default configuration for the GP library.

Although we did not succeed in developing a hand-coded controller which performed comparably to the evolved controller, we can attempt to estimate lower bounds on the time required, assuming a successful controller could be developed by hand. There are two basic approaches to developing a controller by hand. The first approach is to use a simulator (by itself or in conjunction with the real robot) to develop and test a controller. In this case a lower bound on development time is the time required to implement the simulator, assuming that once the simulator was available, a good controller could be implemented in no time – this is an unrealistic assumption but provides a lower bound. Any simulator implemented for this purpose is likely to require at least as much effort to implement as the extremely crude simulator used in our GP. Therefore, any process that involved a simulator would have taken a minimum of 80% (total time minus the time to interface to the GP library) of the human effort that our ER implementation required.

The other possibility for hand-coding a controller is to develop a controller on the robot without any use of simulation. The evolved controller significantly outperforms all of the baseline heuristics, which were the best heuristics we were able to come up with. It is possible that there are significantly better heuristics that could be discovered and implemented manually by an iterative development process using only the robot. While it is not possible to estimate how long this process might take, we note that testing a single controller with our fitness function (i.e., testing the robot behavior against one particular visitor arrival pattern) requires 30 minutes, not including the time to actually implement and debug the controller, so a controller development method which does not require simulation must complete in less than 30 iterations in order to be competitive with our simulator-based evolutionary approach. This seems non-trivial, considering our results in Section 4.3, which showed that it was quite difficult for a human operator to outperform the evolved controller, even when presented repeatedly with the same input (visitor arrival pattern) – developing a controller which performs well in general seems quite difficult.

Based on these observations, our evolutionary approach seems to be a practical choice for our application, compared to an alternative, manual approach to designing a controller.

6 Discussion and Future Work

We investigated an evolutionary robotics (ER) approach for high-level applications on a service robot platform. In particular, we present a case study in applying evolutionary robotics to synthesize a controller for a guide robot which manages visitor traffic flow in an exhibition space such as a museum. We defined a objective function which models visitor enjoyment, and focused on the problem of designing a controller which optimizes this objective function. We used GP to evolve a controller for this task in simulation, and transferred the controller to an actual service robot. We experimentally evaluated several baseline heuristics, as well as our evolved controller, and showed that the evolved heuristic significantly outperformed the baseline heuristics in both simulation as well as on the service robot. We also showed that the evolved controller performs comparably to an experienced human operator manually controlling the real robot. Thus, we showed that with relatively little effort and no GP control parameter tuning, it is possible to obtain a controller that is human-competitive in two respects, i.e., the evolved controller is competitive with straightforward, hand-crafted controllers, as well as manual control by a human operator.

The main contribution of this work is a case study of applying evolutionary robotics to a new, high-level application (visitor traffic flow management), which is a significantly higher-level application task than standard evolutionary benchmarks such as navigation, locomotion and foraging. Since the commercial service robot platform we use provides a high-level API for navigation with collision avoidance, the controller we evolve outputs targets for the navigation API.

One advantage of high-level evolutionary robotics on this kind of service robot platform is that when we seek controllers that output high-level commands, safety is not an issue, because the lower level software (e.g., collision avoidance) ensures safety constraints, and the worst that could possibly happen is that the evolved controller is ineffective. This suggests that as commercial service robot platforms continue to mature, the opportunities to apply ER in service robot applications will likely be similar to ours case study, where the evolved controller outputs high-level commands for APIs provided by the platform.

Another contribution is the demonstration that in a high-level application, a very crude simulation can be successfully used to evolve a controller which works well when transferred to a real robot. By applying large amounts of noise to the essential parameters of simulation, and evaluating each candidate controllers on a large number of noisy trials in a wide range of operational environments, we force the GP system to evolve controllers which functioned robustly in a range of environments and operating conditions. As our simulation experiments show, the evolved controllers perform relatively well regardless of the visitor arrival rate or visitor compliance. The results on the real robot show that the competence of the evolved controller extends to at least one, physical setting. Thus, our results confirm the utility of the "envelope of noise" approach of Jakobi [14] in a different context – while Jakobi was addressing the problem of noise in the environment, i.e., the "reality gap" between simulation and the real world, our use of randomized environment parameters addresses both the reality gap as well as uncertainty about the operating conditions (e.g., visitor arrival rates).

While we have focused on GP in this paper, the main lesson from this case study is general: when robust, high-level navigation commands are available, human-competitive results for high-level tasks can be obtained relatively easily using a low-fidelity, simulation-based learning method to synthesize a controller that maps some symbolic representation of the system state to high-level navigation commands. Other learning methods such as reinforcement learning could be used to generate a controller based on the same state representation. A comparison of the quality of controllers obtainable using different machine learning methods such as reinforcement learning, as well as the time and effort required to develop the controller using other learning methods (including the effort required to transfer the controller from simulation to the robot), is an avenue for future work.

With respect to the visitor traffic flow management problem, this paper describes the first step in formulating robot guide behavior as an optimization and approaching it as a machine learning problem. The next step is to scale up the system to larger number of exhibits and consider deploying the robot guide system to manage visitor flow in a real setting (e.g., at a poster session at an academic conference). This will require addressing additional challenges. For example, so far, we used a model where visitors behave in a fairly simplistic, rational manner. However, visitor behavior can be more complex in reality. In fact, previous researchers have noted that visitors can actually behave in an adversarial manner by not following directions given by the robot, blocking the robot's progress, or trying to confuse the navigation system [3,28]. Therefore, future work needs to address how our robot guide system can achieve its objectives under worst-case, adversarial conditions. While we used a weighted objective function that considers the "smoothness" of the visitor experience and the amount of unique interactions the guide robot has, another direction for future work is to explicitly explore trade-offs between these components by using a multi-objective optimization approach.

References

1. R. Beer and J. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122, 1992.

2. M. Bennewitz, F. Faber, D. Joho, M. Schreiber, and S. Behnke. Towards a humanoid museum guide robot that interacts with multiple persons. In *Proc. IEEE-RAS Int. Conf. on Humanoid Robots*, pages 418–423, 2005.
3. W. Burgard, A. Cremers, D. Fox, D. Hhnel, G. Lake-meyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2):3–55, 1999.
4. M. Ebner and A. Zell. Evolving a behavior-based control architecture- from simulations to the real world. In *Proceedings of GECCO*, pages 1009–1014, 1999.
5. Fujitsu Ltd. Fujitsu Service Robot (Enon), 2010. http://www.frontech.fujitsu.com/en/forjp/robot/servicerobot/.
6. A. S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):31–61, 2008.
7. J. Grefenstette and A. Schultz. An evolutionary approach to learning in robots. In *Proceedings of the Machine Learning Workshop on Robot Learning*, 1994.
8. S. Handley. The genetic planner: The automatic generation of plans for a mobile robot via genetic programming. In *Proceedings of IEEE International Symposium on Intelligent Control*, pages 190–195, 1993.
9. I. Harvey, P. Husbands, and D. Cliff. Seeing the light: artificial evolution, real vision. In *Proc. Int. Conf. on Simulation of Adaptive Behavior (SAB)*, pages 392–340, 1994.
10. G. Hornby, S. Takamura, J. Yokono, O. Hanagata, M. Fujita, and J. Pollack. Evolution of controlelrs from a high-level simulator to a high DOF robot. In J. Miller, editor, *Evolvable Systems: from biology to hardware; proc. 3rd Int. Conference (ICES2000)*, Springer Lecture Notes in Computer Science Vol 1801, pages 80–89, 2000.
11. G. Hornby, S. Takamura, J. Yokono, O. Hanagata, M. Fujita, and J. Pollack. Evolving robust gaits with AIBO. In *IEEE Int. Conf. on Robotics and Automation*, pages 3040–3045, 2000.
12. I. Horswill. Polly: A vision-based artificial agent. In *Proc. AAAI*, pages 824–829, 1993.
13. T. Ito, H. Iba, and M. Kimura. Robustness of robot programs generated by genetic programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA, 28–31 July 1996. MIT Press. 321–326.
14. N. Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behavior*, 6(2):325–368, 1997.
15. S. Kamio and H. Iba. Adaptation technique for integrating genetic programming and reinforcement learning for real robots. *IEEE Transactions on Evolutionary Computation*, 9(3):318–333, June 2005.
16. T. Kanda, M. Shiomi, Z. Miyashita, H. Ishiguro, and N. Hagita. An affective guide robot in a shopping mall. In *Proc. 4th ACM/IEEE Int. Conf. on Human Robot Interaction (HRI)*, pages 173–180, 2009.
17. J. Koza. Evolution of subsumption using genetic programming. In *Proceedings of First European Conference on Artificial Life (ECAL)*, pages 110–119, 1992.
18. J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection.* MIT Press, 1992.
19. W.-P. Lee, J. Hallam, and H. H. Lund. Applying genetic programming to evolve behavior primitives and arbitrators for mobile robots. In *Proceedings of IEEE 4th International Conference on Evolutionary Computation*, volume 1. IEEE Press, 1997. to appear.
20. M. J. Mataric and D. Cliff. Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems*, 19(1):67–83, 1996.
21. F. Mondada and D. Floreano. Evolution of neural control structures: Some experiments on mobile robots. *Robotics and Autonomous Systems*, 16:183–195, 1995.
22. D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.
23. I. Muslea. SINERGY: A linear planner based on genetic programming. In S. Steel and R. Alami, editors, *Fourth European Conference on Planning*, volume 1348 of *Lecture notes in artificial intelligence*, Toulouse, France, 24–26 Sept. 1997. Springer-Verlag.
24. A. Nelson, G. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370, 2009.
25. S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. In R. Brooks and P. Maes, editors, *Proc. 4th Int. Workshop on Artificial Life*, 1994.

26. P. Nordin and W. Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior*, 5(2):107–140, Fall 1996.
27. P. Nordin, W. Banzhaf, and M. Brameier. Evolution of a world model for a miniature robot using genetic programming. *Robotics and Autonomous Systems*, 25(1-2):105–116, 31 Oct. 1998.
28. I. Nourbakhsh, C. Kunz, and T. Willeke. The mobot museum robot installations: A five year experiment. In *Proc. of IEEE/RSJ IROS*, pages 3636–3641, 2003.
29. I. R. Nourbakhsh. An affective mobile robot educator with a full-time job. *Artif. Intell.*, 114(1-2):95–124, 1999.
30. M. L. Pilat and F. Oppacher. Robotic control using hierarchical genetic programming. In *GECCO (2)*, pages 642–653, 2004.
31. A. Schultz, J. J. Grefenstette, and W. Adams. Robo-shepherd: Learning complex robotic behaviors. In *In Robotics and Manufacturing: Recent Trends in Research and Applications, Volume 6*, pages 763–768, 1996.
32. S. Sharabi and M. Sipper. Gp-sumo: Using genetic programming to evolve sumobots. *Genetic Programming and Evolvable Machines*, 7(3):211–230, 2006.
33. M. Shiomi, T. Kanda, H. Ishiguro, and N. Hagita. Interactive humanoid robots for a science museum. *IEEE Intelligent Systems*, 22(2):25–32, 2007.
34. R. Siegwart, K. O. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, M. Meisser, R. Philippsen, R. Piguet, G. Ramel, G. Terrien, and N. Tomatis. Robox at expo.02: A large-scale installation of personal robots. *Robotics and Autonomous Systems*, 42(3-4):203–222, 2003.
35. L. Spector. Genetic programming and ai planning systems. In *Proc. AAAI*, pages 1329–1334, 1994.
36. I. Tanev. Genetic programming incorporating biased mutation for evolution and adaptation of snakebot. *Genetic Programming and Evolvable Machines*, 8(1):39–59, 2007.
37. S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. R. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *I. J. Robotic Res.*, 19(11):972–999, 2000.
38. K. Wolff and P. Nordin. Learning biped locomotion from first principles on a simulated humanoid robot using linear genetic programming. In *GECCO*, pages 495–506, 2003.