OPEB: Open Physical Environment Benchmark for Artificial Intelligence

Hamid Mirzaei
Dept. of Computer Science
University of California, Irvine
mirzaeib@uci.edu

Mona Fathollahi
Dept. of Computer Science and Engineering
University of South Florida
mona2@mail.usf.edu

Tony Givargis
Dept. of Computer Science
University of California, Irvine
givargis@uci.edu

***Abstract*—Artificial Intelligence methods to solve continuous-control tasks have made significant progress in recent years. However, these algorithms have important limitations and still need significant improvement to be used in industry and real-world applications. This means that this area is still in an active research phase. To involve a large number of research groups, standard benchmarks are needed to evaluate and compare proposed algorithms. In this paper, we propose a physical environment benchmark framework to facilitate collaborative research in this area by enabling different research groups to integrate their designed benchmarks in a unified cloud-based repository and also share their actual implemented benchmarks via the cloud. We demonstrate the proposed framework using an actual implementation of the classical mountain-car example and present the results obtained using a Reinforcement Learning algorithm.**

I. Introduction

Recent advancements in using Artificial Intelligence (AI) to solve continuous-control tasks have shown promise as a replacement for conventional control theory to tackle the challenges in emerging complex Cyber-Physical Systems, such as self-driving control, smart urban transportation and industrial robots. An example of AI approaches is Reinforcement Learning (RL). RL algorithms are mostly model-free, meaning that the explicit modeling of the physical system is not required. Also, RL-based agents can work under uncertainty and adapt to the changing environment or objectives. These unique characteristics of RL make it a good candidate to solve the control problem of complex physical systems. However, the RL solutions for continuous control are in their infancy, since there are limitations when applying them in real-world applications. Some examples are unpredictability of agent actions, lack of formal proofs of closed-loop system stability and not being able to transfer learning from one task to other tasks with slight modifications. This calls for extensive research to address these limitations and design RL and other AI algorithms that can be used in real-world applications.

While there are a number of widely-used benchmarks in different computing domains, for example MiBench [10] for embedded processing and ImageNet [7] for computer vision, the available AI benchmarks are very limited. This makes conducting research in AI difficult and expensive. Moreover, since there are not many available standard benchmarks, it is hard to evaluate and compare newly proposed AI algorithms. One of the reasons for the lack of AI benchmarks is the interactive nature of dynamical systems. In other words, while it is possible for many other domains to record and label datasets and make them publicly available, AI benchmark developers should provide an interactive "environment" which the AI agent must be able to interact with by applying actions and gathering the new system state (or observation) along with reward signals. This makes AI benchmark development a challenging task. Nevertheless, significant progress has been made recently towards building simulation/emulation based AI benchmarks such as OpenAI Gym and OpenAI Universe [5].

Although the recently developed AI benchmarks enable the researchers to apply their algorithms on a vast variety of different artificial environments, such as PC games or physical systems simulations, real-world physical environments such as industrial robots and self-driving cars are only available to a limited number of groups in big institutes due to the high costs of manufacturing and maintenance of those environments. The lack of physical benchmarks slows down the research progress in developing AI algorithms that can address challenges that usually exist in the real-world such as sensor noise and delay, processing limitations, communicational bandwidth, etc., and can be used in emerging Internet-of-things (IoT) and Cyber-Physical systems.

In this paper, we propose the Open Physical Environment Benchmark (OPEB) framework to integrate different physical environments. Similar to OpenAI Gym, in our approach a unified interface of the environments is proposed that enables research groups to integrate their physical environment designs to OPEB regardless of the details involved in the hardware/software design and implementation. To achieve the main goals of universality and affordability, we propose leveraging 3D printing technology to build the customized mechanical parts required in the environments and using low-cost generic hardware components such as bolts, ball bearings, etc. We also use popular and affordable embedded processing platforms, such as the Raspberry Pi [20], which is a promising processing solution for IoT and Industry 4.0.

Furthermore, the users are not only able to replicate physical environments using OPEB, but they can also share the implemented environment on the cloud enabling other users to evaluate their algorithms on the actual physical environment. This feature results in higher availability of physical benchmarks and facilitates collaborative research to design robust AI algorithms that can be applied on different realizations of an environment with slight variations in the physical properties of the hardware components. Since OPEB is based on low-cost fabrication solutions, it can be used for educational purposes for IoT, control, AI and other related courses.

The remainder of this paper is organized as follows. In Section II, we review the background and some related works in AI benchmarks. In Section III, the elements of a physical environment are introduced and it is explained how the required artifacts are provided in OPEB to replicate a physical environment. In Section IV, an example implementation of an OPEB, i.e., the classical mountain-car problem, is described, and the results of the experiments that are performed on the physical system using an RL-based method is presented in Section V. Finally, conclusions are presented in Section VI.

II. Background and related work

In this section, we first review existing literature about solving real-world tasks using AI algorithms. Next, we review recent simulation-based AI benchmarks that are widely used in academia. Finally, we review the related research projects to provide real-world benchmarks in robotic applications.

Using RL as a replacement for conventional control theory is an emerging trend in Cyber-Physical systems. In [15] an RL algorithm is proposed to autonomously navigate a humanoid Nao robot into a docking station used for recharging. An RL model is proposed in [12] to learn hand-eye coordination for grasping objects in an environment of robotic manipulators. In [13], RL methods have been applied on an actual cart-pole system to balance the pole. Researchers are exploring AI algorithms as a way to simplify and speed up the programming of industrial robots in factories. Fanuc [11], the world's largest maker of industrial robots, has used RL methods to train robots to precisely pick up a box and put it in a container. In the automotive industry, authors in [16] have proposed an RL-based approach to control robot morphology (flippers) to move over rough terrains that exist in Urban Search and Rescue missions.

Access to these physical environments (hardwares/robots) is not feasible for a lot of research groups. This hinders partnerships and cooperation between academia and industry. In this paper, for the first time, we propose the idea of providing low-cost and easy-to-construct physical environments that allow researchers and students to implement, evaluate and compare their AI algorithms on standardized benchmarks.

In a dynamic AI problem, the state of the environment depends on the actions that are chosen by the agent. This makes it almost impossible to store the environment as a fixed dataset similar to the supervised machine learning paradigm. Therefore, to facilitate reproducible research and accelerate the pace of education, researchers in this community are trying to design a standard programming interface for reinforcement-learning experiments.

One of the earliest efforts to design a standard tool is RL-Glue [19] which has been used for RL courses in several universities and to create experiments for scientific papers. A more recent effort, RLPy [8], is a software framework written in python that has focused on value-function-based methods with linear function approximation using discrete actions. ALE [4] is another software framework designed to make it easy to develop agents that play different genres of Atari 2600 games.

OpenAI Gym [5] is the most recent and comprehensive toolkit for developing AI algorithms. It provides a diverse suite of environments that range from classic control to 2D and 3D robots. It is designed to let the users evaluate the proposed AI algorithms with little background in AI. Researchers can compare the performance of their proposed algorithm with other approaches' scores reported on the scoreboard. These solutions are very effective in advancement of research and education within simulated environments because it is usually expensive and more challenging to implement AI algorithms in real-world scenarios.

Most similar to our work is [17] that has proposed an open hardware design for academic and research robots. They have leveraged 3D printing technology to allow users to create all required components except electronics parts. All basic code and libraries have been released under the GNU General Public License. Authors in [6] have made their research on aquatic swarm robots reproducible by providing the 3D printing models, CNC milling files and the developed software on Raspberry Pi. In this paper, we propose a framework that can be used to produce an arbitrary number of physical environments, not limited to robots. Contrary to the mentioned works where a specific physical environment is introduced, a unified benchmark framework is proposed in this paper to integrate a variety of physical environments. In other words, research groups can contribute by sharing their physical environment blueprints using the proposed framework. The other contribution is that users are able to share their actual implementation via a web-based software on the cloud to be used by others for research and education purposes.

III. Open Physical Environment Benchmark (OPEB)

In this section, we describe our OPEB framework. First, the elements of a physical environment (PE) are introduced and the requirements for each element are discussed. Next, we will explain how the required components to replicate the PE are encapsulated in OPEB and also how the actual implementation can be shared to other users on the cloud.

A. Physical Environment Elements

The PE consists of the following elements:

- Mechanical parts and structures
- Electromechanical components
- Electrical components
- Embedded processing unit
- Embedded software

To achieve the goal of affordability and universality of PE implementation, the physical parts should include either generic mechanical hardware such as bolts, ball-bearings, etc., or the parts that can be easily printed using a 3D printer. The electromechanical parts such as actuators, dc motors or transducers should be generic parts that can be easily found all over the world. For example, low cost hobby electromechanical parts can be used to build a PE. To drive and interface the electromechanical parts, some electrical parts such as motor drives should be included in the PE. Additionally, to measure the physical quantities, some sensors are required. Examples of such sensors are digital camera, thermometer and proximity sensor.

The embedded processing unit is needed to perform basic required tasks to run the environment such as timing, reading the sensors' outputs and the required signal processing, producing the environment observation, applying the action calculated by the AI algorithm, sending the monitoring data over the network to the monitoring node locally or over the cloud and running the AI algorithm. These tasks are implemented by the embedded software developed for the PE. All of the software components are provided by the OPEB except the AI algorithm which is developed by the PE user.

Emerging single-board embedded computing platforms can be used as the embedded processing unit in PE. Some examples of these solutions are Raspberry Pi [20], C.H.I.P. computer [1] and Arduino [3] platforms. Using a dedicated embedded processor instead of a general purpose computer reduces the cost of deployment of multiple instances of the PE on the cloud and simplifies interfacing the electrical and electromechanical elements because most of these platforms have on-board I/O capabilities.

B. OPEB Components

In Fig. 1, the different components of OPEB for each environment are shown. To realize an environment consisting of the elements listed in the previous subsection, the following components are provided in OPEB for that specific environment:

- Parts that should be 3D printed in STL [9] format.
- List of materials of the generic mechanical hardware.
- Diagrams and instructions required for mechanical structure assembly.
- List of electrical and electromechanical components.
- List of embedded processing units and peripherals.
- Wiring diagram of the electrical components.
- PE control and monitoring Embedded software.
- Web application for the cloud-based sharing of the PE.

The customized mechanical parts required by a PE are included in OPEB as 3D models in STL format that can be easily fabricated using a 3D printer. The specifications of other parts that are not printable or can be selected from off-the-shelf products are provided in OPEB. However, these parts are generic mechanical hardwares that are supplied by many manufacturers around the world.

Besides the information provided to obtain or fabricate the components, OPEB includes the complete instructions and diagrams to assemble the mechanical structures of the PE. The main goal of OPEB is that the environments can be reproduced with minimum discrepancy across different implementations. To achieve this goal, the user should be able to build the whole environment using the provided components in the OPEB without ambiguity. On the other hand, the instruction assembly should be of low complexity and easy to follow to be usable by users with different levels of expertise. For this purpose, a step-by-step assembly instruction approach proposed in [2] is employed for the mechanical and electromechanical parts.

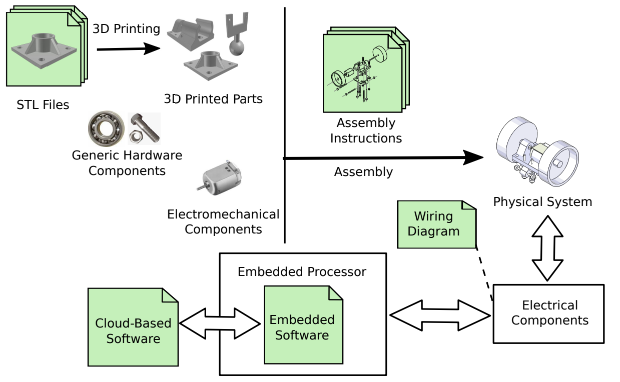

Fig. 1: OPEB framework components for each environment. Green blocks are provided in OPEB. All other components listed and specified in OPEB.

Electrical and electromechanical parts, including actuators, sensors, processing units and drivers are usually selected from off-the-shelf products. The list of needed components and their specification are listed in OPEB for each environment. Also, unambiguous wiring diagrams are provided for electrical interconnections.

After building the hardware components, the embedded software should be deployed on the embedded processing unit. The embedded software is included in OPEB and can be deployed using installation manuals. To enable the OPEB users to evaluate their algorithms using different PEs, a standard API is defined similar to OpenAI Gym environments. More specifically, the AI agent can interact with the PE using functions that apply actions and returns the environment observations and reward signal. Furthermore, the environment can be reset to the initial state using the PE API.

Finally, the back-end and front-end software components are provided that enable the OPEB users to deploy their implemented PE over the cloud. Using this web-based application, other users can use the PE to upload and run their AI algorithms on the physical system and see the evaluation reports such as accumulated score over time and record the videos of the PE that runs their algorithm.

IV. Example implementation: Classical Mountain-Car Example

In this section, we discuss the process of developing an example OPEB environment, i.e., the Mountain-Car example, to demonstrate the methods mentioned in Section III.

In the Mountain Car example, which is first introduced in [14], the goal is to control the acceleration of a car inside a

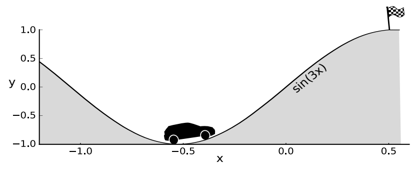

Fig. 2: Mountain Car example

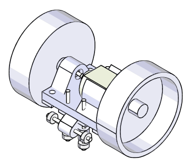
Fig. 3: Car Assembly in the MC-OPEB.

valley in order to move it to the top of the mountain (Fig. 2). However, the maximum acceleration of the car is limited and it can not be driven to the top of mountain in a single pass and the car has to go back and forth a number of times to get enough momentum to reach to the desired destination. An AI solution based on Q-learning and tile coding approximation is presented in [18] for this example with a fast convergence in a couple of hundred episodes. However, several simplifying assumptions are made in the original mountain car example including simplified dynamics equations, exact measurements without noise and nonlineariy, no sensor or processing delays and car motion with no friction and no slipping. The last assumption makes the learning process a fairly easy task since the kinetic energy delivered by the car's motor is preserved in the system. Consequently, the car can endlessly swing in the valley and the AI agent can make gradual progress towards the goal by increasing the swing range bit-by-bit using successive actions. In a real-world situation, none of these assumptions hold and the agent has to learn a successful policy in a limited time since the car is going to stop after a few swings. The mentioned limitations justify the importance of physical benchmarks that can evaluate the AI algorithms which are useful in real-world applications, for example industrial robotics or self-driving vehicles.

A. *Mechanical Structures*

The MC-OPEB consists of two mechanical structures: Car and Mountain rail. The car, which is shown Fig. 3, consists of only two large wheels because a car with two pairs of rear and front wheels might entangle around the positions of the path that have low radius of curvature. Also, using only two wheels results in less overall car weight which enables us to use a low power motor and simplifies the design or selection of electrical parts such as motor drive and power supply. Moreover, to prevent the motor from spinning and to constrain the car to move inside the mountain rail, 8 pieces of small ball-bearings are embedded in the car structure using short metal bars.

Each side of the mountain rail, which is shown in Fig. 4, is divided to two smaller parts to make them printable using 3D printers with small beds. Additionally, the whole rail surface is

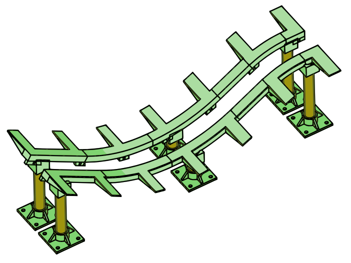
Fig. 4: Mountain Rail Assembly in the MC-OPEB.

not printed to preserve filament. A flexible cardboard should be placed on the support bars attached to the rail structure. The complete STL set of the 3D printed objects are shown in Fig. 5 and the set of required hardware is listed in Table I.

An example of assembly instruction documents is provided in Fig. 6 which shows the exploded-view diagram of car assembly. The assembly instruction includes the step-by-step action diagrams as explained in [2].

B. *Electromechanical Parts*

The only electromechanical part needed for MC-OPEB is the widely-used and low-cost 1.5-3 (V) hobby motor. To reduce the friction and simplify the mechanical design this

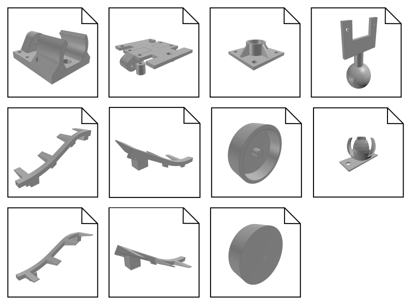
Fig. 5: STL files included for MC-OPEB for all required 3D printed parts.

TABLE I: List of Materials of required generic hardware parts

Item	Quantity
3mmx10mm bolt and nut	29
32mmx2mm steel bar	7
2x6x2.5mm ball bearing	8
10mmx100mm wooden bar	1

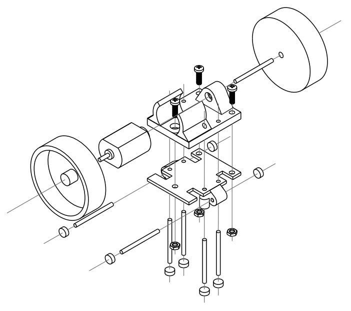
Fig. 6: Exploded-view of car assembly as an example of assembly instruction diagrams in MC-OPEB.

Algorithm 1: Hand-engineered policy for the mountain-car environment

```
Data: $x, v$                    ▷ Instantaneous car position(x) and speed(v)
Result: $a$                     ▷ action(a): acceleration direction
1 if $|v| < 50$ then
2     Choose $a \leftarrow$ left or $a \leftarrow$ right randomly with same probability.;
3 else
4     if $v > 0$ then
5         $a \leftarrow$ left;
6     else
7         $a \leftarrow$ right;
```

motor is directly coupled to one of the large wheels on the car. Also, no transducers, such as potentiometer or a shaft encoder is coupled to the motor to reduce the weight of the car and overall cost of MC-OPEB.

C. Electrical Parts

The required electrical parts are: motor driver, two 5V power supplies for the Raspberry Pi board and driving the motor, and Raspberry pi camera. We have used low-cost HG7881 motor drive with PWM inputs. Since the Raspberry Pi has two on-board pwm outputs we can directly connect it to the motor drive without any additional interfacing circuit.

The Raspberry Pi camera is used to measure the motion quantities of the car, i.e., position and speed. The captured image of the car also can be used to evaluate emerging deep reinforcement learning algorithms that can control a physical system only by raw visual data.

D. Embedded Processing Unit

We have used “Raspberry Pi Zero W” platform which is a powerful and affordable processing unit for different embedded applications.

E. Embedded Software

The Embedded software used in MC-OPEN is a C++ program that is executed on the Raspbian Jessie OS. The embedded software is responsible for implementing the 0.01(s) control timing, capturing and processing the camera image, running the AI routine supplied by the environment user, applying the motor voltage command using PWM outputs, sending monitoring data consisting of instantaneous speed, position and other status variables, running the learned policy and recording the performance video upon user’s request.

The camera image is post processed to calculate the position and speed of the car which are the observations of the MC-OPEB. First, the HSV pixel values are filtered by some fixed thresholds to extract the pixels of the yellow marker attached to the car. Next, the spatial moments of filtered pixels are calculated and used to obtain the single (x, y) coordinate of the car. To reduce the noise and estimate the car’s speed, a linear Kalman filter is implemented in the embedded software.

F. Web Application

The web application is an optional component that can be run on a secondary general purpose computer. Using the web application, the MC-OPEB user can see the monitoring data online and share the implemented physical environment on the cloud. The cloud user can upload a c++ routine that implements any custom AI application and evaluate the algorithm performance using the web application. The cloud user can also pause the learning and run the learned policy and see the recorded view of the actual AI algorithm performance.

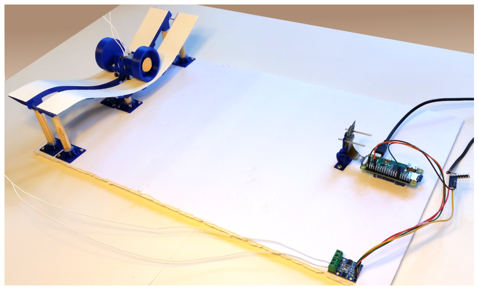
Fig. 7: Actual Implementation of MC-OPEB.

Fig. 7 shows a picture of the actual MC-OPEB. In the next section, we show the results of running a reference algorithm and an RL-based algorithm on the built environment.

V. Results

In this section, we present the results of the experiments performed on the MC-OPEB to show the effectiveness of a low-cost PE to perform real-world experiments using AI methods. The objective is to move the car to a certain height on the left side of the rail which corresponds to 80 pixel displacement of the car to the left in the captured image. The reward is defined is as -1 for all the sampling times that the car has not reached the destination. Each episode starts from the car being at the bottom of the valley and ends when it reaches the desired height on the left side. Therefore, the total reward which is the RL “return” is proportional to the negated total episode time. The action is the car’s acceleration direction assuming that the car moves with the maximum acceleration and only changes the direction of the acceleration.

A. Reference Solution

To ensure the possibility of moving the car from the lowest point in the valley to some certain height by any algorithm, a hand-engineered solution is proposed in Algorithm 1. The performance of the AI-based solution can be compared with the reference solution to evaluate the AI algorithm. Fig. 8 shows the result of the reference solution.

B. AI-based solution

The Q-learning algorithm with tile-coding function approximation is used to show that the proposed MC-OPEB can be used to evaluate AI algorithms on a physical environment in real-time.

Fig. 9 shows the learning curve of the AI agent where the accumulated return vs the episode number is shown. Fig. 10 shows the learned policy at episode 37 which is the best performance obtained using the AI algorithm. The results show that the RL algorithm is able to achieve the performance of hand-engineered reference solution. The less number of swings made by the RL agent might be due to slight variations in the physical system and does not necessarily mean the superiority of the RL algorithm.

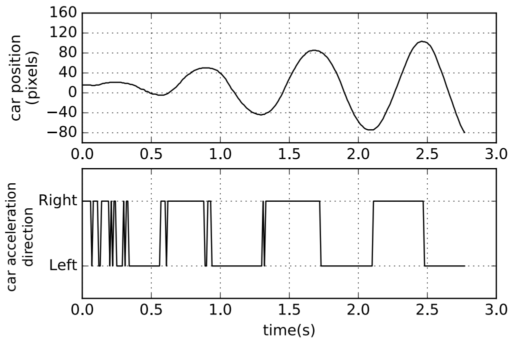

Fig. 8: The upper plot represents the car position vs. time. The lower plot represents the car acceleration command computed by the hand-engineered algorithm. After a few swings, the designed algorithm is able to move the car to desired height on the left side which corresponds to -80 pixel coordinate of the car in the captured image.

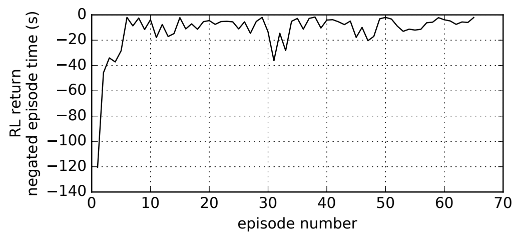

Fig. 9: Learning curve of the RL agent. x-axis is the episode number and y-axis shows the RL return translated to the total time of each episode. Less absolute value of return means less episode time and better performance.

VI. Conclusion

In this paper, a novel physical environment benchmark is presented for AI algorithms. The environments can be implemented using low-cost parts and fabrication methods such as 3D printing. The proposed benchmarks enable researchers to easily replicate physical benchmarks to evaluate their AI algorithms and also share their implemented physical environments on the cloud with other users. Such collaborative benchmarking accelerates development of AI algorithms which can address challenges from real-world physical systems by engaging many researchers that can replicate the physical environments or access them on the cloud. We also presented an example implementation of the proposed physical environment framework. The results show the effectiveness of the proposed methods to develop a simple and low-cost physical benchmark.

Some possible future directions are adding more physical benchmarks, addressing the resource limitations of Raspberry PI for more computationally expensive algorithms and easy deployment of the whole framework on cloud solutions such as Amazon AWS.

VII. Acknowledgement

This work was supported in part by the National Science Foundation under NSF grant number 1563652.

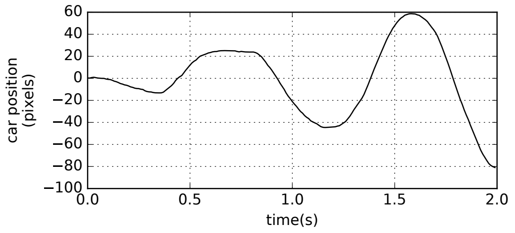

Fig. 10: Car position vs. time plot obtained by RL algorithm after 37 episodes which its best performance in the experiment.

References

[1] C.h.i.p. website. http://getchip.com/.

[2] M. Agrawala, D. Phan, J. Heiser, J. Haymaker, J. Klingner, P. Hanrahan, and B. Tversky. Designing effective step-by-step assembly instructions. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 828–837. ACM, 2003.

[3] M. Banzi and M. Shiloh. *Getting Started with Arduino: The Open Source Electronics Prototyping Platform*. Maker Media, Inc., 2014.

[4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[6] V. Costa, M. Duarte, T. Rodrigues, S. M. Oliveira, and A. L. Christensen. Design and development of an inexpensive aquatic swarm robotics system. In *OCEANS 2016-Shanghai*, pages 1–7. IEEE, 2016.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[8] A. Geramifard, C. Dann, R. H. Klein, W. Dabney, and J. P. How. Rlpy: a value-function-based reinforcement learning framework for education and research. *Journal of Machine Learning Research*, 16:1573–1578, 2015.

[9] T. Grimm. *User's guide to rapid prototyping*. Society of Manufacturing Engineers, 2004.

[10] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14. IEEE, 2001.

[11] K. Katsuki. Machine learning system and motor control system having function of automatically adjusting parameter, Mar. 30 2017. US Patent 20,170,090,434.

[12] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *arXiv preprint arXiv:1603.02199*, 2016.

[13] J. Mattner, S. Lange, and M. Riedmiller. Learn to swing up and balance a real pole based on raw visual input data. In *International Conference on Neural Information Processing*, pages 126–133. Springer, 2012.

[14] A. W. Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, Computer Laboratory, 1990.

[15] N. Navarro-Guerrero, C. Weber, P. Schroeter, and S. Wermter. Real-world reinforcement learning for autonomous humanoid robot docking. *Robotics and Autonomous Systems*, 60(11):1400–1407, 2012.

[16] M. Pecka, K. Zimmermann, M. Reinstein, and T. Svoboda. Controlling robot morphology from incomplete measurements. *IEEE Transactions on Industrial Electronics*, 64(2):1773–1782, 2017.

[17] R. Sheh, H. Komsuoglu, and A. Jacoff. The open academic robot kit: Lowering the barrier of entry for research into response robotics. In *Safety, Security, and Rescue Robotics (SSRR), 2014 IEEE International Symposium on*, pages 1–6. IEEE, 2014.

[18] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[19] B. Tanner and A. White. Rl-glue: Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10(Sep):2133–2136, 2009.

[20] E. Upton and G. Halfacree. *Raspberry Pi user guide*. John Wiley & Sons, 2014.