FraMoTEC: Modular Task-Environment Construction Framework for Evaluating Adaptive Control Systems

Thröstur Thorarensen[1], **Kristinn R. Thórisson**[1,2], **Jordi Bieger**[1] and **Jóna S. Sigurðardóttir**[2]

Abstract. While evaluation of specialized tools can be restricted to the task they were designed to perform, evaluation of more general abilities and adaptation requires testing across a large range of tasks. To be helpful in the development of general AI systems, tests should not just evaluate performance at a certain point in time, but also facilitate the measurement of knowledge acquisition, cognitive growth, lifelong learning, and transfer learning. No framework as of yet offers easy modular composition and scaling of task-environments for this purpose, where a wide range of tasks with variations can quickly be constructed, administered, and compared. In this paper we present a new framework in development that allows modular construction of physical task-environments for evaluating intelligent control systems. Our proto- task theory on which the framework is built aims for a deeper understanding of tasks in general, with a future goal of providing a theoretical foundation for all resource-bounded real-world tasks. The tasks discussed here that can currently be constructed in the framework are rooted in physics, allowing us to analyze the performance of control systems in terms of expended time and energy.

1 INTRODUCTION

To properly assess progress in scientific research, appropriate evaluation methods must be used. For artificial intelligence (AI) we have task-specific benchmarks (e.g. MNIST [21]) for specialized systems on one end of the spectrum and—proposed yet controversial—tests for more general human-level AI (e.g. the Turing test [40]) on the other end. Little is available on the middle ground: for systems that aspire towards generality, but are not quite close to human-level intelligence yet. A major goal of AI research is to create increasingly powerful systems, in terms of autonomy and ability to address a range of tasks in a variety of environments. To evaluate the general ability and intelligence of such systems, we need to test them on a wide range of realistic, unfamiliar task-environments [17]. To cover the entire spectrum of AI systems, we want to be able to analyze, compare and adapt the task-environments that we use [36].

A prerequisite to evaluating systems is the construction of task-environments. It has been suggested that successful regulation or control implies that a sufficiently similar model must have been built (implicitly or explicitly) [9, 32]. It is important for AI to understand how the construction of models of task-environment works: for evaluation, for the model-building (learning) and decision-making that goes on in the "mind" of a successful controller, and for the design process where match between agent and task must be considered. Tasks truly are at the core of AI, and in another paper (under review) we argued for the importance of a "task theory" for AI [37]. Other (engineering) fields often have a strong understanding of the tasks in their domains that allows them to methodically manipulate parameters of known importance in order to systematically and comprehensively evaluate system designs. A task theory for AI should provide appropriate formalization and classification of tasks, environments, and their parameters, enabling more rigorous ways of measuring, comparing, and evaluating intelligent behavior. Analysis and (de)construction capabilities could furthermore help any controller make more informed decisions about what (sub)tasks to pursue, and help teachers manipulate or select environments to bring about optimal learning conditions for a student [4].

Here we present an early implementation of a framework for the construction of modular task-environments for AI evaluation, based on the initial draft of a task theory. The modular nature of the construction makes it easy to combine elementary building blocks into composite task-environments with the desired complexity and other properties. It allows us not only to make simple variants on existing task-environments to test generalization ability and transfer learning, but also to measure (learning) progress by scaling environments up or down (e.g. by adding/removing building blocks). The framework is aimed at general systems that aspire to perform real tasks in the real world. Such systems have bounded resources that they must adequately manage, so we take a special interest in the amount of time and energy they use to perform assigned tasks. Since we are mainly interested in assessing cognitive abilities, the agent's body is defined as part of the task-environment so that we can evaluate the performance of the agent's controller (i.e. its "mind").

2 RELATED WORK

Since the inception of the field of AI, the question of how to evaluate intelligence has puzzled researchers [40]. Since then a lot of work in the area has been done [22, 17, 25]. Most AI systems are developed for a specific application, and their performance is easily quantifiable in domain-specific terms. Evaluation of more general-purpose algorithms and systems has always been more difficult. A common strategy is to define benchmark problems (e.g. MNIST written character recognition [21] or ImageNet object detection challenges [10]), AI-domain-restricted contests (e.g. the Hutter compression prize [19], planning competitions [8] or the DARPA Grand Challenge [39]) or iconic challenges (e.g. beating humans at chess [7] or Go [33]) in the hope that performance on these manually selected problems will be indicative of a more general kind of intelligence. These methods are good at evaluating progress in narrow domains of AI, where they

[1] Center for Analysis and Design of Intelligent Agents, School of Computer Science, Reykjavik University, Iceland. email: {throstur11,thorisson,jordi13}@ru.is
[2] Icelandic Institute for Intelligent Machines, Reykjavik, Iceland. email: jona@iiim.is

encourage innovation and competition, while also—unfortunately—encouraging specialized approaches for overfitting on the evaluation measure.

Many methods aimed at more general intelligence exist as well. Many researchers have turned to theories of human psychology and psychometrics to evaluate AI systems, resulting in human-centric tests (e.g. the Turing Test [40], the Lovelace Tests [5, 27], the Toy Box Problem [20], the Piaget-MacGuyver Room [6] and AGI Preschool [14, 15]; see also the latest special issue of AI Magazine [25]). These tests tend to either be very subjective, very human-centric or only provide a roughly binary judgment about whether the system under test is or is not (close to) human-level intelligent. They are again applicable to only a narrow slice of AI systems, which in many cases don't exist yet (i.e. we have no human-level AI that can genuinely pass most of these tests).

What we need is a way of evaluating AI systems that aspire towards some level of generality, but are not quite there yet [36]. Hernández-Orallo argued that in order to assess general intelligence, that the assessment should cover the testing of a range of abilities required for a range of tasks [17]. What is needed then is a battery of tasks that can be used to evaluate the cognitive abilities of a system. Ideally, this battery of tasks should be suitable to the system we want to test, but still comparable to tasks that are used on other systems.

The importance of using a range of tasks that are unknown to AI system designers is widely—although not nearly unanimously—recognized. For instance, the 2013 Reinforcement Learning Competition [42, 1] included a "polyathlon" task in which learning systems were presented with a series of abstract but related problems. We see the same in other competitions: the General Game Playing competition [12] presents contestants with the description of a finite synchronous game only moments before it needs to be played, and the General Video Game AI competition [24] does the same with video games. These competitions are very interesting, but their domains are still fairly restricted, the adversarial nature makes progress evaluation between years tricky, and the "tasks" each need to be carefully constructed by hand.

To really be able to evaluate a wide range of AI systems well, it is necessary that tasks—and variants of those tasks—can easily be constructed or preferably generated. Hernández-Orallo advocates this approach, but only covers discrete and deterministic environments [16, 18]. Legg & Veness developed an "Algorithmic IQ" test that attempts to approximate a measure of universal intelligence by generating random environments [23]. Unfortunately these methods cannot easily generate complex structured environments and are opaque to analysis and human understanding. Our own prior work on the MERLIN tool (for Multi-objective Environments for Reinforcement LearnINg) [11] followed in the footsteps of other Markov Decision Process generators like PROCON [2] and GARNET [3]. While Merlin does support using continuous state and action spaces and somewhat tunable environment generation, it fails to meet most of the requirements below.

In [36] we listed a number of requirements for the comprehensive evaluation of artificial learning systems. An ideal framework ought to cover the complete range of AI systems—from very simple to very advanced. Performance of various systems on different tasks should be comparable, so as to differentiate between different systems or measure progress of a single system. Such a framework would not only facilitate evaluation of the performance of current and future AI systems, but go beyond it by allowing evaluation of knowledge acquisition, cognitive growth, lifelong learning, and transfer learning. Most importantly, it should offer easy construction of task-environments and variants, the ability to procedurally generate task-environments with specific features, and facilitation of analysis in terms of parameters of interest, including task complexity, similarity and observability. Easy construction includes the ability to compose, decompose, scale and tune environments in terms of parameters like determinism, ergodicity, continuousness, (a)synchronicity, dynamism, observability, controllability, simultaneous/sequential causal chains, number of agents, periodicity and repeatability.

The framework we present here is a prototype aimed towards the requirements outlined above. Section 7 will elaborate on this more, as we evaluate our success so far.

3 TASK THEORY

The concept of *task* is at the core of artificial intelligence (AI): tasks are used in system evaluation, training/education and decision making. Tasks can vary from the classification of an image, to the clustering of data points, and to the control of a (physical or virtual) body to cause change in an environment over time. It is the latter kind of task that we are primarily concerned with here.

Most AI systems are designed to perform a *specific kind* of task, and most systems require a set of concrete tasks to train on in order to learn to later perform similar tasks in production. This requires the collection or construction of appropriate task examples.

Systems that aspire to a more general kind of intelligence aim to tackle a wide range of tasks that are largely unknown at design time. Upon deployment these systems are intended to not rely on their designer to decompose their future tasks into component parts and elementary actions – they will need to choose among different decompositions and (sub)tasks themselves, in terms of both priority (benefits) and feasibility (costs). Evaluation of more general cognitive abilities and intelligence can not simply be done by measuring performance on a single target task: we could just develop a specialized system for that[3]. Rather, we need a battery of tasks that can be modified to grow with the systems under test and facilitate the measurement of knowledge acquisition, cognitive growth, lifelong learning, and transfer learning.

While we don't have fully general systems yet, different systems will need to be evaluated on different task batteries, and we need the flexibility to tailor those to the systems under test and the ability to compare performance on various tasks in order to compare different AI systems. Yet in most cases tasks are selected ad hoc, on a case-by-case basis without a deep understanding of their fundamental properties or how different tasks relate to each other. We have argued elsewhere for the importance of a "task theory" for AI [37]. Such a theory should cover all aspects of tasks and the environments that they must be performed in, and cover:

1. *Comparison* of similar and dissimilar tasks.
2. *Abstraction* and *reification* of (composite) tasks and task elements.
3. Estimation of time, energy, cost of errors, and other resource requirements (and yields) for *task completion*.
4. Characterization of task complexity in terms of (emergent) quantitative measures like *observability*, *feedback latency*, form and nature of *information/instruction* provided to a performer, etc.
5. Decomposition of tasks into subtasks and their atomic elements.

[3] Unless the single target task is AI-complete (c.f. the Turing test and similar tests), but these are typically only applicable to a very narrow range of intelligence (i.e. humanlike intelligence), and no current systems can pass these tests.

6. Construction of new tasks based on combination, variation and specifications.

To accomplish all of this we need some way to formalize task-environments. Here we only give an overview of our initial attempt; for more detail see [37].

At the highest level, we use a tuple of task and environment—similar to Wooldridge's $\langle Env, \Psi \rangle$ [43], where Ψ represents the criteria by which success will be judged. As a first approximation a *task* may be formulated as an *assigned goal*, with appropriate constraints on time and energy. Wooldridge [43] defines two kinds of goals, what might be called *achievement goals* ("Ensure $X \approx G_X$ before time $t \geq 10$") and *maintenance goals* ("Ensure $X \approx G_X$ between time $t = 0$ and $t = 10$).[4] By introducing time and energy into the success criteria, this disparity is removed: Since any achieved goal state must be held for some non-zero duration (at the very minimum to be measured as having been achieved) an achievement goal is simply one where the goal state may be held for a short period of time (relative to the time it takes to perform the task which it is part of) while a maintenance goal is held for relatively longer periods of time. The highest attainable precision of a goal state is defined by the laws of physics and the resolution of sensors and actuators. Performing a task in the real world requires time, energy, and possibly other resources such as money, materials, or manpower. Omitting these variables from the task model is tantamount to making the untenable assumption that these resources are infinite [41]. Including them results in a unified representation of goals where temporal constraints on the goal state are provided.

Any action, perception and deliberation in the real world takes up at least some time and energy. Limitations on these resources are essentially the raison d'être of intelligence [35]—unbounded hypothetical systems will randomly stumble upon a solution to anything as time approaches infinity. Estimation of time, energy and other resource requirements (and yields) for task completion can be used to design effective and efficient agent bodies, judge an agent based on comparative performance, and make a cost-benefit analysis for deciding what (sub)tasks to pursue.

For the environment part of our formalization we take inspiration from Saitta & Zucker [30]. The environment description contains variables describing objects in the world, as well as transition functions to describe the dynamics. Environments are considered perspectives on the world, and can be nested within each other if desirable, resulting in a certain amount of modularity. Since we are mainly interested in creating systems with advanced cognitive abilities, we define sensors and actuators simply by listing observable and controllable variables. This essentially places the agent's body inside the environment. From an evaluation perspective, this allows us to focus on the agent's *controller* (mind). From a task analysis point of view this allows to make statements about physical limits and feasibility without needing to consider the unknown cognitive abilities of a controller.

4 FRAMEWORK

Our Framework for Modular Task-Environment Construction "FraMoTEC" enables the construction and simulation of task-environments in a modular way. An early prototype has been implemented in Python in an object-oriented manner, using a layered composition of small building blocks to describe entire environments.[5] To aid in the explanation of our framework and its various components, we use the following running example of a task-environment:

Example 1 (Car Race)**.** A one-dimensional race is our simplest version. The agent must move a car across the finish line N meters away from the starting position. The controller determines the rate of energy expenditure, which results in higher or lower acceleration (pressing the gas pedal more increases the flow of fuel to the engine and through that the amount of energy that is converted in order to move the vehicle). Naturally the race must be finished using as little time as possible and using an amount of energy that doesn't exceed what is available from the gas tank.

Example 2 (Car Parking)**.** Example 1 can straightforwardly be converted to a parking task if we require the car to end up between two points (rather than across a finish line), or extended by e.g. adding a second dimension or adding/removing friction and air resistance.

4.1 Components

Constructing task-environments requires using the building blocks provided by the framework. These building blocks are designed to be as basic as possible to allow for a wide variety of behaviors to emerge from the different combinations of organization of the blocks. Most of the organizational complexity of the resulting tasks emerges from the various combinations of objects with custom transitions. The following components have been incorporated: objects, transitions, systems, goals, motors and sensors. When all building blocks come together they form what is finally called the "model"—i.e. the complete representation of the task-environment (and by extension, the natural system that the model represents).

Objects Objects are used to describe the "things" in a task-environment model, such as the `car` in the race example. The framework implements basic one-dimensional kinematics individually for each object. Objects have a main `value` x (in the example corresponding to the car's position) as well as physical properties like `velocity` v, `mass` m, `friction` μ_k and might even contain values for gravitational acceleration at some `angle` θ. This allows the object to naturally transition (as will be explained below): the new `velocity` is computed based on the current `velocity` and the input power P (and direction) from any affectors, which is then used to update the main `value`, as shown by these physics equations:

$$
\begin{aligned}
F_{\textit{input}} &= \frac{P}{v} \\
F_{\textit{gravity}} &= -mg \cdot \sin\theta \\
F_{\textit{friction}} &= -\operatorname{sgn} v \cdot \mu_k mg \cdot \cos\theta \\
F_{\textit{total}} &= F_{\textit{input}} + F_{\textit{gravity}} + F_{\textit{friction}} \\
v &\leftarrow v + \delta t \cdot \frac{F_{\textit{total}}}{m} \\
x &\leftarrow x + \delta t \cdot v
\end{aligned}
$$

where g is the gravitational constant and δt is the (theoretically infinitesimal) time over which the framework calculates each change.

Although the framework does not currently implement other object behavior, we envision extending the framework as experience with it accumulates.

[4] We use approximate rather than precise equivalence between X and its goal value G_X because we intend for our theory to describe real-world task-environments, which always must come with error bounds.

[5] The FraMoTEC code is available at https://github.com/ThrosturX/task-env-model.

Transitions Transitions or transition functions are used to change the (values of) objects in the task-environment. Transitions come in two forms: the *natural* form and the *designed* form. Natural transitions describe the natural change of the objects and systems in the task-environment. They are provided by the framework and executed automatically during simulation unless this is explicitly prevented. Transitions that are specified by the task-environment designer expand upon the natural behavior of an environment by adding custom logic to it without requiring the framework to be extended specifically. We could for example implement a transition that causes the car in our race example to lose mass as its energy is depleted: `t_mass: car.mass` $\leftarrow$ `1200 + car.energy / 46000`.

Motors Motors can be considered "effectors" or "actuators" of the controller, which it can directly interact with to affect the environment, to achieve goals and perform tasks. The controller sets the rate at which energy is transferred to each such motor (we refer to this energy transfer rate as "power"). When connected to an object (as they typically are), this energy is converted into a force that affects an object's current `velocity`. Motors can be placed in systems of objects with custom transitions to create new behavior. For instance, to add more realistic steering controls, instead of letting the controller use independent motors to affect the `x` and `y` position directly, we could add motorized `orientation` and `speed` objects, plus these transitions:

- `x.velocity` $\leftarrow$ `speed.velocity`$\cdot\cos($`orientation.value`$)$
- `y.velocity` $\leftarrow$ `speed.velocity`$\cdot\sin($`orientation.value`$)$

Sensors Standardized *access* to objects' `values` can be provided via sensors. A sensor reads an object's `value`, optionally applying some distortion due to noise or limited resolution. Sensors can also read other sensors, allowing designers to combine observations or apply multiple layers of distortions.

Systems Systems facilitate composition immensely by acting as a container for objects, transitions, sensors, motors and other elements. The natural transition of a system is to apply all transitions within. Systems can be used to create a hierarchy of larger building blocks that can easily be reused and rearranged. For instance, we could create a system to encapsulate the above object `car` and transition `t_mass` so that more cars whose mass depends on the contents of their fuel tank can easily be made. Or, when we define a car that can move in two dimensions, we need separate objects for the position in each dimension. We could make a system with two objects—for `x_position` and `y_position`—and motors to control each dimension separately (this would perhaps be more reminiscent of a helicopter) or motors for controlling speed and angle of the wheels. One kind of "car" could easily replace another kind in a larger system, without affecting everything else, making it easy to create slight variations. "Inaccessible" objects or other constructs can also be created, whose value is inaccessible directly via sensors. This facilitates the theoretical creation of systems with hidden states.

Goals Goals are used to define tasks that the controller must perform. A goal specifies a target object X along with a goal value G_X, tolerance ϵ and time restrictions. Tolerance values should be used because all measurements are constrained by real-world resolution. Time restrictions should allow the user to specify before and after which (absolute and relative) times the target value needs to be in the goal range. Goals can also depend on other goals to be satisfied (i.e. the goal can require other goals to be met before it can consider itself satisfied). This allows users to easily define composite tasks by sequencing goals. Once a goal has been satisfied, it is forever considered satisfied unless it is reset, in which case both the goal itself and any goals it was a prerequisite will be reset recursively. This allows the state of the *task* to be evaluated based on the number of achieved goals without regarding the *environment's* current state. In the car race example we might define a goal that says $G_X - \epsilon \leq$ `car.value` $\leq G_X + \epsilon \wedge t \leq 10$. Since the task is accomplished as soon as the goal becomes satisfied for the first time, `tolerance` could be set to zero. For the parking task, we might add a goal that says $0 - \epsilon \leq$ `car.velocity` $\leq 0 + \epsilon$ and add it as a prerequisite to the other goal to demand that the car is stopped at the end.

Task-Environment Model Finally, a model of an entire task-environment is created that can be run by the framework. The task part consists of a set of goals plus additional restrictions on time and energy. The environment is a system that contains all relevant objects, transitions, sensors and motors.

5 Task Construction

Tasks are constructed modularly using the smallest possible constructs, as building blocks with peewee granularity provide the most flexibility [38]. The simplest conceivable task-environment is essentially specified in our car race example: the environment is a single system containing one object with an associated motor and sensor, and the task simply specifies a single goal value for the only object along with some restrictions on time and energy. We can construct more complicated tasks by adding more objects, sensors and motors to the environment. We can enforce systems' behavior as desired by implementing appropriate transitions. We can for example create a system in which objects X and Y move independently except if $Y < 20$ by implementing a transition that checks if $Y < 20$, locking X in place (freezes it's state) if so, and unlocking it otherwise: $\textit{transition}_{\textit{lock-x}}$: `X.locked` $\leftarrow$ `Y.value` ≥ 20.

The framework has some built-in randomization options, allowing designers to set ranges of options for the values of objects and goals. This allows us to easily generate a set of different but highly similar concrete tasks that can be used to train or evaluate a system on the general idea of a task family, rather than just having it memorize the specifics of a single task.

5.1 Simulation

In order to evaluate or train intelligent controllers, it is important that the constructed task-environments can be executed or simulated. The designers of task-environments ultimately produce formalizable models—this is a natural implication of the framework building on simple, simulable causal processes (the building blocks and their interaction). A simulation of the model becomes a simulation of the natural system that the model represents, transmuting Church's thesis into an assertion (all systems that can be modeled by the framework are simulable) [28].

In order to simulate a task-environment, its model needs to be connected to a controller. Since different controllers have different requirements, this is largely left up to the user. FraMoTEC is implemented in the Python programming language and provides a simple API on the task-environment model. The typical method of usage would be to construct and instantiate a task-environment model as well as a control system—here we will use the example of a SARSA reinforcement learner. The user is free to access all aspects of the task-environment and should use this access to define the interaction with the controller in any way that they like. For instance, some implementations of table-based SARSA systems may need to be instantiated with knowledge of the range of possible actions and observations.

The task-environment model provides a `tick` method that takes `delta_time` as an argument. Whenever this method is called, the framework will run the simulation for `delta_time` seconds. When the method returns, the user can easily obtain the values from sensors to pass on to the controller. At this point it is also possible to e.g. compute a reward based on values from the sensors or other information that the user can access about the current state of task and environment. It should be noted that the framework itself does not yet support explicit rewards.[6] Not every control system requires explicit or incremental rewards; for more powerful learners rewards for ought to be intrinsic [34, 31, 13, 26]. Having said that, it is trivial to implement rewards via a special kind of sensor designated as a reward channel, and to possibly couple this with the state of goal achievement, which is of course tracked within the system. The user could then pass on the actions of the controller to the motors of the task-environment before calling `tick` again. This fully synchronous mode of interaction is required by most reinforcement learners, but FraMoTEC could also run simulations asynchronously if the controller supports it.

The simulation component of the framework would ideally be truly continuous, but the nature of the Von Neumann architecture encourages stepwise integration. `delta_time` can be viewed as the time resolution of the controller. The task-environment model has its own time resolution `dt`. As such, every simulation step regardless of length should optimally ensure that:

- For all systems: naturally transition for `dt` seconds—*recall that any system's natural transition fires all transitions within.*
- For all objects: naturally transition for `dt` seconds
- Goals should be asserted to evaluate whether success (or failure) conditions have been met
- The time passed during the frame must be recorded and added to an accumulator
- The energy used by any motor during that time frame should be recorded and added to an accumulator
- Current time and energy usage should be compared with time and energy limits

5.2 Analysis

In the current prototype implementation FraMoTEC offers limited functionality for analyzing task-environments and the performance of controllers. For a given task, the framework can produce a time-energy tradeoff plot (Pareto curve) that shows the minimal amount of energy that is required to finish the task in a given amount of seconds (or alternatively: the fastest the task can be completed given a certain amount of expended energy).

As previously established, time and energy usage are key metrics to consider when evaluating the performance of controllers in a given set of task-environments. It goes without saying that a controller that spends 2 minutes and 20 KJ of energy to solve a specific task-environment is worse at *completing the task* than a controller that spends 30 seconds and 700 J in that same task-environment. Maybe the first controller spent more time actually learning about the environment, in which case it might be much better suited for a set of similar task-environments than the second controller.

Naturally, we can continuously measure the time and energy expenditure of an controller to quantify the total amount of time and energy required to come up with a solution to some task. In this sense we are not evaluating a controller's ability, but its ability to improve some ability (i.e. the controller's ability to learn). We can further extend both these evaluation methods to a set of tasks in lieu of a single task, allowing for a more comprehensive evaluation and comparison of all kinds of controller.

After simulation, successful attempts by the controller that resulted in completing the task can be added to the graph to compare time and energy usage compared to each other and the optimal Pareto curve. Different runs are color-coded according to the time at which they occurred (earlier attempts are lighter, while later ones are darker), which shows a controller's (hopefully improving) behavior over time.

6 USE CASES

In this section we will showcase some simple use cases of the system. Since this paper is not about advanced control systems themselves, we will use a simple SARSA reinforcement learner [29] and some domain-specific control systems to illustrate the capabilities of the framework in a simple way.

6.1 Learning Agent

An agent was implemented with an underlying SARSA reinforcement learning algorithm. The state exposed to the agent was an *n-tuple* of all sensor readings along with the velocity of one of the objects in the model. A scoring function was implemented to determine rewards[7].

Reinforcement learners generally learn slower as the $state{\cdot}action$ space increases, therefore the agent enumerates the available actions as the setting of a single motor at one of three power levels: (i) 0 (ii) P_{max} and (iii) $-P_{max}$. We experimented with an agent that included the settings (iv) $\frac{P_{max}}{2}$ and (v) $-\frac{P_{max}}{2}$, but we found that these settings unnecessarily crippled the agent and removed them. The agent implements a method `perform(self, dt)` that creates an *experience* for the agent by: (a) setting the current state (b) selecting and executing the *reward-maximizing action* (c) ticking the simulation by *dt* seconds (d) rewarding the learner based on the value of the scoring function and the new state. This method is called repeatedly in the evaluation, see Section 6.2.1.

6.2 Task-Environments

The agent was introduced to two similar environments. The first environment had the goal of moving the `position` object into `goal_position` with a tolerance of 5, with 5000J and 60 seconds as the maximum expendable time and energy (essentially, the 1D car race example):

[6] Rewards are appropriately seen as part of the information/training materials for a task, not as part of the task proper (although one may argue that a task will change drastically, perhaps fundamentally, depending on what kind of information is given about it up front and during its learning/performance).

[7] Implemented as $\left(-\sum_{i=0}^{N}|s_{object_i} - s_{goal_i}| - \epsilon_i\right)$ where s represents the position (value) of either the object or the goal associated with a `Goal` in the task-environment's `solution`.

- One object: `position`
- One fully reversible motor affecting `position` with 200W maximum input
- One sensor for `position`
- Goal: `position` within `goal_position` ± `goal_epsilon`
- Max time and energy: 60 s, 5000 J

The second environment expanded upon this environment, requiring a "plotter" to be activated when the position is correct—both goals needed to be satisfied to consider the task solved. An additional transition in the second environment locked the position while the plotter was activated.

- Two objects: `position` and `plot_it`
- One fully reversible motor affecting `position` with 200 W maximum input
- One non-reversible motor affecting `plot_it` with 5 W maximum output[8]
- One sensor for each object
- **New transition function**: If `plot_it` >0.5: `position` is locked, otherwise it is unlocked.
- Goal prerequisite: `position` between `goal_position` ± `goal_epsilon`
- Goal: `plot_it` is 1 ± 0.1
- Max time and energy: 60 s, 10000 J

The second task-environment increases the task difficulty when compared to 1D car racing by adding a new object (complete with sensor and motor), changing the behavior (with the transition function that locks the `position` object) and by expanding on the original goal.

6.2.1 Evaluation

First, the agent is tasked with solving some training environments which are copies of the target environment, except with a more favorable starting position. The training environments gradually get more difficult by increasing the distance between the starting position and the goal. Once this training is complete, the agent gets 200 chances to satisfy the goal(s) in each task-environment. The data is visualized using the methods described in Section 5.2. Figure 1 shows the results for the 1D car race task-environment. Figure 2 shows the results for the 1D locking plotter task-environment. Note that the agent continues to learn by creating *experiences* during the evaluation (i.e. learning is not "switched off"). The evaluation works as follows:

- While the task is not solved:
  1. If the task has been *failed*, stop
  2. Invoke the agent's `perform` method (with dt set to $0.25s$, see Section 6.1).
- Finally, report time and energy usage (and indicate if the task failed).

Note on reading the plots: The blue line represents the energy required to complete the task at each time. The red line represents the maximum amount of expendable energy due to motor power limitations. The dots represent data points for the evaluations, with lighter colored (greener) data points representing earlier runs and darker colored (redder) data points representing later runs.

[8] You can think of a solenoid with an off button.

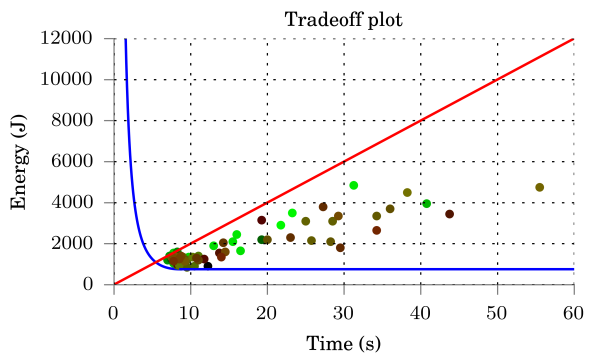

Figure 1. Resulting plot for 200 evaluations in the 1D car race environment.

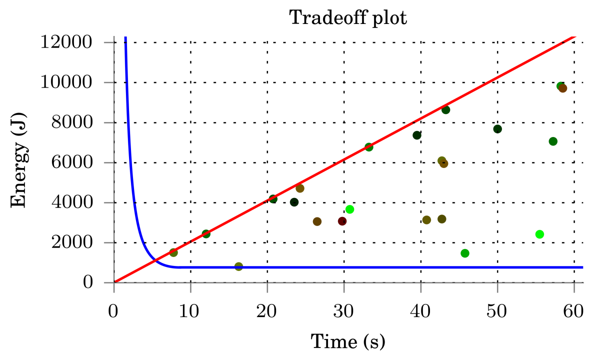

Figure 2. Resulting plot for 200 evaluations in the 1D locking plotter environment.

6.3 Agent Comparison

In order to demonstrate how different agents can be compared just as different environments can be compared, a custom agent implementation was compared with the SARSA implementation in the 1D locking plotter environment. The custom agent roughly implements the following algorithm in the `perform` method:

- Compute `distance` between `position` and the corresponding goal
  - If the distance is small enough, deactivate the `position` motor and activate the `plot_it` motor.
  - If the distance is positive, maximum power to the `position` motor and deactivate the `plot_it` motor.
  - If the distance is negative, maximum negative power to the `position` motor and deactivate the `plot_it` motor.
- Tick the simulation by dt seconds

It should be obvious that the above algorithm is specifically tailored to outperform the SARSA agent, as it includes domain knowledge which the SARSA agent would need to come up with on its own. Figure 3 indeed shows that this controller performs much better and more constantly, but also that it's not improving over time.

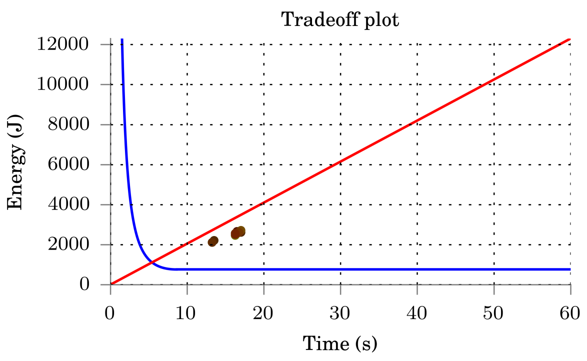

Figure 3. Resulting plot for 200 evaluations in the 1D locking plotter environment using an agent with a domain-specific implementation.

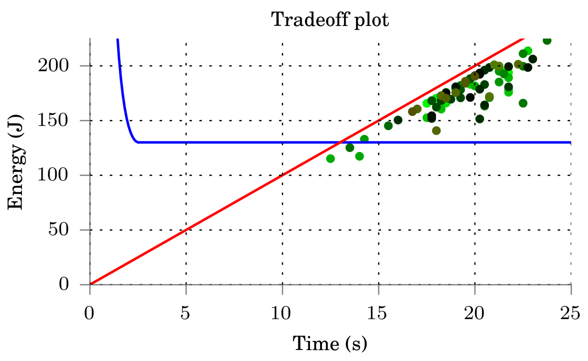

Figure 4. Resulting plot for 100 evaluations in a generic 10-dimensional task-environment.

6.4 N-Dimensional Task Comparison

6.4.1 Task-Environments

A generic N-dimensional task-environment generator is included in the samples as `sample_N_task`. The generator returns a task-environment with N objects and associated sensors with a default starting position of 3 ± 2 and a goal of reaching 10 ± 0.5. There are two systems: (i) a control system which contains two objects with associated motors and sensors and a transition function that sets the power level of some hidden motor to some value depending on the values of the objects in the control system (ii) a hidden motor system which ensures that activating the hidden motors for each of the N variables results in that power usage being counted

The control system includes the motors that the agent should have direct access to. The main power motor determines how much power is input into the hidden motors while the selection motor determines which hidden motor is activated.

6.4.2 Controller

A simple controller was created to solve the class of task-environments described in the previous section. The algorithm is quite simple, the below should demonstrate the agents `perform` method:

- Activate the main power motor
- Determine the object that is furthest from the goal, call it `min_o`
- Determine the direction of power required to enable `min_o`'s affector
- Activate the selection motor in the appropriate direction
- Tick the simulation by dt seconds

6.4.3 Results

Two variations of the N-dimensional task were attempted, one with 10 dimensions and one with 20 (Figures 4 and 5). It should not come as a surprise that the task-environment with fewer dimensions was solved in less time, with less energy. However, the difference was not double, as one might be inclined to suspect when doubling the size of the environment. This gives us an indication about the agent's ability to scale with environments (but could also give some indication of how well-formed the environment itself is). Since we know everything there is to know about the environment, we can assert that the agent seems to scale well from 10 to 20 dimensions.

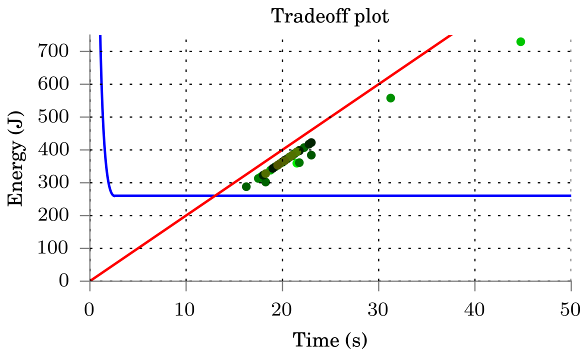

Figure 5. Resulting plot for 100 evaluations in a generic 20-dimensional task-environment.

7 CONCLUSION & FUTURE WORK

In this paper we have presented our ideas for and early prototype implementation of a framework for modular task-environment construction (FraMoTEC). FraMoTEC aims to facilitate the evaluation of intelligent control systems across the entire spectrum of AI sophistication on practical tasks. The framework is intimately intertwined with an equally early-stage "task theory" that is intended to deepen our understanding of fundamental properties of various types of task-environments and how they relate to each other. Such an understanding would help us compare control systems against each other and earlier versions in order to measure progress, learning and growth.

A major goal was to lay the groundwork for an evaluation tool that meets the requirements that we outlined in an earlier paper [36]: facilitation of easy construction, procedural generation and in-depth analysis. We believe that the framework does indeed make steps in the right direction. Current analysis capabilities of the framework are very limited, but already provide for instance some rudimentary understanding of tradeoffs between time and energy, and measuring a learning system's performance increase over time. One major piece of future work is to further develop task theory so that we can make predictions about the effects of combining tasks and environments: e.g. when we add a dimension or additional goal, how does that affect minimum time and energy requirements?

Procedural generation of task-environments is precursory at this point in time. The framework allows users to set ranges of acceptable options for initial values of objects and goals instead of concrete values. Slightly different concrete task-environments can then be instantiated automatically by the framework. However, the modular nature of tasks and environments should make it relatively easy to add functionality for e.g. adding dimensions or sequencing goals.

This modularity also allows for easy construction, composition, decomposition and scaling of task-environments. Adding or removing objects or (sub)systems, as well as combining goals and tasks in various ways, allows us to make simple task-environments (slightly) more complex and vice versa; thereby allowing our battery of tasks to grow with the AI system under test. As the name suggests FraMoTEC is primarily a framework for task-environment *construction* and this is where it shines, even though much work remains to be done.

In [36] we listed a number of properties of task-environments that a framework should 1) support and 2) ideally let the user tune:

1. **Determinism** Both full determinism and stochasticity must be supported. The framework provides the option of partial stochasticity out-of-the-box, such as in the creation of objects (start values can be randomized), goals, sensor readings, and designed transitions.
2. **Ergodicity** Ergodicity controls the degree to which the agent can undo things and get second chances. The framework imposes no restrictions on this other than a fundamental rule: Expended time and energy cannot be un-expended. If the agent spends time or energy doing the wrong thing, that time and energy will still have been spent and the task-environment needs to be reset in order to give the agent a second chance with regard to the energy and time expenditure. Task-environment designers have full control over what states are reachable.
3. **Controllable Continuity** This point notes that it is crucial to allow continuous variables, and that the degree to which continuity is approximated should be changeable for any variable. All objects in the framework contain continuous variables, discretized only by floating-point inaccuracies by default. It is possible use sensors to further discretize (or distort) any accessible variables. It is also possible to tweak the time resolution of the simulation.
4. **Asynchronicity** Any action in the task-environment should be able to operate on arbitrary time scales and interact at any time. This must currently be done manually, and we aim to provide a more user-friendly solution in the future.
5. **Dynamism** The framework gives the user full control over how static or dynamic environments are. Natural transitions of objects can provide some limited dynamism, but controllers can be given a static experience by clever sampling. Most dynamics will come from designed transitions created by the framework user.
6. **Observability** The observability of task-environments is determined by the interface between the environment and the controller interacting with it. Sensors are the primary control for observability in the framework. Sensors can be tuned to tune the observability of a task-environment by distorting the value and/or discretizing it to a user-specified resolution.
7. **Controllability** Controllability is the control that the agent can exercise over the environment to achieve its goals. The controllability of the task-environment is controlled with the exposure of motors to the controller. By modifying motor properties and interactions between motors (specifically in custom transition functions), the controllability of a task-environment can be tuned.
8. **Multiple Parallel Causal Chains** Co-dependency in objectives can be programmed into designed task-environments without hassle. The framework does not place any restrictions on causal chains with a single exception that circular-references are currently not supported (two goals may not mutually depend on each other, one must depend on the other first).
9. **Number of Agents** The framework does not restrict the number of agents nor what interactions can take place. Even if multiple agents have access to the same motors, the framework regards the most recent setting to be the current setting. However, interactions are currently mostly defined by the user. We hope to provide more user-friendly support in the future, which will go hand-in-hand with implementing a better method for asynchronicity.
10. **Periodicity** The framework does not specifically handle periodicity, cycles or recurrent events. The user must implement this with designed transitions if they need this.
11. **Repeatability** By using the same random seed, repeatability can be guaranteed in most circumstances. However, agents and sensors must use their own random number generators (and seeds) to avoid tampering with task-environment repeatability.

While a lot of work remains to be done, we believe that this framework will be able to eventually fulfill the requirements we outlined and significantly contribute to the field of AI evaluation, task theory, and by proxy: AI itself.

ACKNOWLEDGEMENTS

This work was sponsored by the School of Computer Science at Reykjavik University, and a Centers of Excellence Grant (IIIM) from the Science & Technology Policy Council of Iceland.

REFERENCES

[1] ICML workshop on the reinforcement learning competition, 2013.

[2] T. W. Archibald, K. I. M. McKinnon, and L. C. Thomas, ‘On the generation of markov decision processes’, *Journal of the Operational Research Society*, 354–361, (1995).

[3] Shalabh Bhatnagar, Richard S. Sutton, Mohammad Ghavamzadeh, and Mark Lee, ‘Natural actor–critic algorithms’, *Automatica*, **45**(11), 2471–2482, (2009).

[4] Jordi Bieger, Kristinn R. Thórisson, and Deon Garrett, ‘Raising AI: Tutoring Matters’, in *Proceedings of AGI-14*, pp. 1–10, Quebec City, Canada, (2014). Springer.

[5] Selmer Bringsjord, Paul Bello, and David Ferrucci, ‘Creativity, the Turing test, and the (better) Lovelace test’, *Minds and Machines*, **11**, 3–27, (2001).

[6] Selmer Bringsjord and John Licato, ‘Psychometric Artificial General Intelligence: The Piaget-MacGuyver Room’, in *Theoretical Foundations of Artificial General Intelligence*, 25–48, Springer, (2012).

[7] Murray Campbell, A. Joseph Hoane Jr., and Feng-hsiung Hsu, ‘Deep Blue’, *Artificial Intelligence*, **134**(1–2), 57–83, (2002).

[8] Amanda Coles, Andrew Coles, Angel García Olaya, Sergio Jiménez, Carlos Linares López, Scott Sanner, and Sungwook Yoon, ‘A survey of the seventh international planning competition’, *AI Magazine*, **33**(1), 83–88, (2012).

[9] Roger C. Conant and W. Ross Ashby, ‘Every good regulator of a system must be a model of that system†’, *International Journal of Systems Science*, **1**(2), 89–97, (1970).

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, ‘Imagenet: A large-scale hierarchical image database’, in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, (2009).

[11] Deon Garrett, Jordi Bieger, and Kristinn R. Thórisson, ‘Tunable and Generic Problem Instance Generation for Multi-objective Reinforcement Learning’, in *Proceedings of the IEEE Symposium Series on Computational Intelligence 2014*, Orlando, Florida, (2014). IEEE.

[12] Michael Genesereth and Michael Thielscher, ‘General game playing’, *Synthesis Lectures on Artificial Intelligence and Machine Learning*, **8**(2), 1–229, (2014).

[13] Olivier L. Georgeon, James B. Marshall, and Simon Gay, ‘Interactional motivation in artificial systems: between extrinsic and intrinsic motivation’, in *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*, pp. 1–2. IEEE, (2012).

[14] Ben Goertzel and Stephan Vladimir Bugaj, ‘AGI Preschool: a framework for evaluating early-stage human-like AGIs’, in *Proceedings of AGI-09*, pp. 31–36, (2009).

[15] Ben Goertzel, Cassio Pennachin, and Nil Geisweiller, ‘AGI Preschool’, in *Engineering General Intelligence, Part 1*, number 5 in Atlantis Thinking Machines, 337–354, Atlantis Press, (2014).

[16] José Hernández-Orallo, ‘A (hopefully) non-biased universal environment class for measuring intelligence of biological and artificial systems’, in *Proceedings of AGI-10*, pp. 182–183, (2010).

[17] José Hernández-Orallo, ‘AI Evaluation: past, present and future’, *CoRR*, **abs/1408.6908**, (2014).

[18] José Hernández-Orallo and David L. Dowe, ‘Measuring universal intelligence: Towards an anytime intelligence test’, *Artificial Intelligence*, **174**(18), 1508–1539, (2010).

[19] Marcus Hutter. 50’000€ Prize for Compressing Human Knowledge, 2006.

[20] Benjamin Johnston, ‘The toy box problem (and a preliminary solution)’, in *Proceedings of AGI-10*, (2010).

[21] Yann LeCun and Corinna Cortes, *The MNIST database of handwritten digits*, 1998.

[22] Shane Legg and Marcus Hutter, ‘Tests of Machine Intelligence’, *CoRR*, **abs/0712.3825**, (2007). arXiv: 0712.3825.

[23] Shane Legg and Joel Veness, ‘An approximation of the universal intelligence measure’, in *Proceedings of the Ray Solomonoff 85th Memorial Conference*, volume 7070, pp. 236–249. Springer, (2011).

[24] John Levine, Clare Bates Congdon, Marc Ebner, Graham Kendall, Simon M. Lucas, Risto Miikkulainen, Tom Schaul, Tommy Thompson, Simon M. Lucas, and Michael Mateas, ‘General Video Game Playing’, *Artificial and Computational Intelligence in Games*, **6**, 77–83, (2013).

[25] *Beyond the Turing Test*, eds., Gary Marcus, Francesca Rossi, and Manuela Veloso, volume 37 of *AI Magazine*, AAAI, 1 edn., 2016.

[26] Pierre-Yves Oudeyer, Adrien Baranes, and Frédéric Kaplan, ‘Intrinsically motivated learning of real-world sensorimotor skills with developmental constraints’, in *Intrinsically motivated learning in natural and artificial systems*, 303–365, Springer, (2013).

[27] Mark O. Riedl, ‘The Lovelace 2.0 Test of Artificial Creativity and Intelligence’, *CoRR*, **abs/1410.6142**, (2014).

[28] Robert Rosen, ‘On models and modeling’, *Applied Mathematics and Computation*, **56**(2), 359–372, (1993).

[29] Gavin A. Rummery and Mahesan Niranjan, ‘On-line Q-learning using connectionist systems’, Technical Report CUED/F-INFENG/TR 166, Cambridge University, (1994).

[30] Lorenza Saitta and Jean-Daniel Zucker, *Abstraction in Artificial Intelligence and Complex Systems*, Springer New York, New York, NY, 2013.

[31] Jürgen Schmidhuber, ‘Formal theory of creativity, fun, and intrinsic motivation (1990–2010)’, *Autonomous Mental Development, IEEE Transactions on*, **2**(3), 230–247, (2010).

[32] Daniel L. Scholten, ‘Every good key must be a model of the lock it opens’. 2010.

[33] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and others, ‘Mastering the game of Go with deep neural networks and tree search’, *Nature*, **529**(7587), 484–489, (2016).

[34] Satinder Singh, Andrew G. Barto, and Nuttapong Chentanez, ‘Intrinsically Motivated Reinforcement Learning’, in *Advances in Neural Information Processing Systems 17*, Vancouver, Canada, (2004).

[35] Kristinn R. Thórisson, ‘Reductio ad Absurdum: On Oversimplification in Computer Science and its Pernicious Effect on Artificial Intelligence Research’, in *Proceedings of AGI-13*, volume 7999 of *Lecture Notes in Computer Science*, Beijing, China, (2013). Springer.

[36] Kristinn R. Thórisson, Jordi Bieger, Stephan Schiffel, and Deon Garrett, ‘Towards Flexible Task Environments for Comprehensive Evaluation of Artificial Intelligent Systems & Automatic Learners’, in *Proceedings of AGI-15*, pp. 187–196, Berlin, (2015). Springer-Verlag.

[37] Kristinn R. Thórisson, Jordi Bieger, Thröstur Thorarensen, Jóna S. Sigurðardóttir, and Bas R. Steunebrink, ‘Why Artificial Intelligence Needs a Task Theory — And What it Might Look Like’. arXiv: submit/1536181, 2016.

[38] Kristinn R. Thórisson and Eric Nivel, ‘Achieving artificial general intelligence through peewee granularity’, in *Proceedings of AGI-09*, pp. 220–221. Springer, (2009).

[39] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, and others, ‘Stanley: The robot that won the DARPA Grand Challenge’, *Journal of field Robotics*, **23**(9), 661–692, (2006).

[40] Alan M. Turing, ‘Computing machinery and intelligence’, *Mind*, **59**(236), 433–460, (1950).

[41] Pei Wang, ‘The assumptions on knowledge and resources in models of rationality’, *International Journal of Machine Consciousness*, **03**(01), 193–218, (2011).

[42] Shimon Whiteson, Brian Tanner, Adam White, and others, ‘The reinforcement learning competitions’, *AI Magazine*, **31**(2), 81–94, (2010).

[43] Michael Wooldridge, *An Introduction to MultiAgent Systems*, John Wiley & Sons, 2009.