Solving Physical Reasoning Tasks in Simulated Environments

By
Zeyad Shureih

A THESIS

Submitted to
Oregon State University
Honors College

In partial fulfillment of
The requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented May 25th 2021,
Commencement June 4th 2021
AN ABSTRACT OF THE THESIS OF


We take for granted how quickly we, as humans, form mental models of the world around us. By the time we are toddlers, we have an observable intuition around the physical rules of the world. Stacking blocks such that they don’t fall over becomes such a trivial task, that it enters the sphere of common sense. We aim to develop an agent with such physical common sense capabilities inspired by the train of thought that humans build mental models and simulations of their environment in real time. Our agent forms a hypothesis for the dynamics of objects in a given scene through the reconstruction and subsequent “mental” simulation of observed objects. This hypothesis comes in the form of both a quantitative comparison of the scene to the hypothesis, and a qualitative assessment of the scene’s compliance with expectations. Our agent was evaluated on its ability to solve a gravity based task, and received top marks amongst competing agents.

Abstract approved: ________________________________________________________________

Alan Fern

Key Words: machine common sense, computer vision, physical reasoning, physics engines,

Corresponding e-mail address: shureihz@oregonstate.edu
Solving Physical Reasoning Tasks in Simulated Environments

By
Zeyad Shureih

A THESIS
Submitted to
Oregon State University
Honors College

In partial fulfillment of
The requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented May 25th 2021,
Commencement 2021
Honors Baccalaureate of Science in Computer Science project of Zeyad Shureih presented on May 25th, 2021.

APPROVED:

________________________
Alan Fern, Mentor, representing Department of Computer Science

________________________
Xiaoli Fern, Committee Member, representing Department of Computer Science

________________________
Stefan Lee, Committee Member, representing Department of Computer Science

________________________
Toni Doolen, Dean, Oregon State University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, Honors College. My signature below authorizes release of my project to any reader upon request.

________________________
Zeyad Shureih, Author
Solving Physical Reasoning Tasks in Simulated Environments

Zeyad Shureih
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR, USA
shureihz@oregonstate.edu

Abstract
We take for granted how quickly we, as humans, form mental models of the world around us. By the time we are toddlers, we have an observable intuition around the physical rules of the world. Stacking blocks such that they don’t fall over becomes such a trivial task, that it enters the sphere of common sense. We aim to develop an agent with such physical common sense capabilities inspired by the train of thought that humans build mental models and simulations of their environment in real time. Our agent forms a hypothesis for the dynamics of objects in a given scene through the reconstruction and subsequent “mental” simulation of observed objects. This hypothesis comes in the form of both a quantitative comparison of the scene to the hypothesis, and a qualitative assessment of the scene’s compliance with expectations. Our agent was evaluated on its ability to solve a gravity based task, and received top marks amongst competing agents.

Introduction

With the recent surge in the adoption and efficacy of deep learning in applications with human facing components, there has never been a time where the lack of common sense in these systems is most apparent. Up until very recently, most robotic agents were highly specialized, and kept away from humans unless the task directly required interaction. The field of machine common sense attempts to fill that void through two major avenues of research. The first, and the focus of this paper, follows the principle that in order for an agent to effectively coexist in an environment, it must be able to register and respond to stimuli in its environment. It establishes that to do this, the agent must have a sense of physical common sense, or physical reasoning. Much like a toddler, it has the ability to form hypotheses about the interactions between objects, and attempts to learn a reason behind a false assumption. The second, explores the sort-of social common sense humans exhibit during physical interactions and communication with other humans. While our agent is meant to be integrated with a physical, autonomous system, thus requiring some capabilities around identifying and interacting with humans in the environment; the task we are evaluated against does not include an interactive component.
Our lab developed what we call a Gravity Agent. Capable of determining whether the events in a given time sequence are indicative of a violation of physical expectations, the agent is able to recreate rudimentary environments it can see in with an on-board physics simulator. While typical approaches to similar problems involve training an agent with reinforcement learning to predict object dynamics given a previous series of observations, the only trained component of the agent identifies objects in the environment by their shape and task-specific role. PyBullet then takes these learned representations of the objects in the environment and computes the dynamics of the scene until all objects are at rest or sufficient time has passed.

Fig. 1. A scene from our development set at the corresponding points: a) The initial frame of the scene with the support object on the floor. b) The point at which both the pole and the target objects enter the scene. c) The ‘drop step’ in which the target is released from the pole. d) The final frame of the scene.

Our inspiration for utilizing a physics engine to solve this problem is motivated by two key works. First is a proposal presented by Kenneth Craik\textsuperscript{1}, which suggests that human brains build mental models of physical interactions that are used to create and run mental simulations of the physical world, much like engineers build simulations for prediction and analysis of complex physical problems. The second work, by Battaglia, Hamrick and Tenenbaum\textsuperscript{2} influenced by the former, notes that earlier attempts to implement these kinds of systems in computational agents focuses largely on symbolic reasoning and high-level problem solving. That is, approaches have been constrained in the abstract, and in order to understand physics in the context of environment, perception and action, quantitative and probabilistic approaches to formalizing mental models is a must. As such, they define an Intuitive Physics Engine (IPE)
that approximates simulation of physical situations using computer graphics and interactive game engines.

Agent Architecture

Following Battaglia, Hamrick and Tenenbaum, we aimed to utilize physics engines in the development of physical intuition. Thus, we designed the following agent architecture. At the beginning of each scene, the MCS Scene Controller steps the simulation of the task scene and outputs level specific metadata (see Task Domain). Our Gravity Agent receives the scene metadata and passes it through the vision module. The vision module (see Computer Vision and Convolutional Neural Networks) extracts physical properties about the objects in the scene at each time step. Once the vision module has extracted properties of each object in the scene and the agent has determined that the 'drop step' (see Fig. 1c) has occurred, the extracted object data is sent to the Pybullet Engine. The PyBullet Engine reconstructs the scene according to extracted object data (see Scene Reconstruction), and simulates the trajectory of each object in the scene until they are all at rest. After this simulation is run, the agent continues to observe the scene through the Vision and Violation Detection Modules, reporting observations back to the Scene Controller. Once the scene ends, quantitative (see Calculating Confidence) and qualitative (see Determining Violations) comparisons are made between the MCS and Pybullet scenes, which are then recorded and output by the scene controller.

Fig 2. Diagram of the agent's high level architecture. Components on the left of the dashed line are conditionally utilized at each time step in the task scene. Components on the right of the dashed line are only utilized when the task scene ends.
Background

Task Domain

The DARPA Machine Common Sense project has developed a series of tasks that evaluate an agent's physical intuition. Our agent was developed to detect violations in scenes, short simulations rendered with the Unity engine, from the Violation of Gravity task. In each scene, the agent is to determine whether there was a violation of gravity, when it occurred, and where in the scene. An example of a scene is shown in Figure 1.

Within the task, every scene consists of the following objects used to determine whether or not a violation occurred. The support, floor and wall objects will always appear at the beginning of the scene, the target and pole object will always descend from the top of the screen, and the pole object will always return to the top of the screen, detaching from the target object.

- Wall Object
- Floor Object
- Support Object
- Target Object
- Pole Object

Within each scene, the support and floor objects are always rectangular prisms, with the former stacked on top of the latter. The target object will descend from the top of the scene, suspended by the pole object. The pole object will always be a cylinder, while the target object could take one of the following forms:

- Rectangular Prism
- Cone
- Square frustum
- Cylindrical Frustum
- Cylinder
- L.Shape

![Fig 3. Examples of scenes with and without violations](image-url)
As well, the task specifies three levels of agent knowledge; Level 1, Level 2, and Oracle. At Level 1, at each time step in the scene the agent is given an RGB image and depth map of the scene from a camera positioned in view of the support object. At Level 2, the agent is additionally given a series of masks for each object currently in view. In Oracle, the agent is given all previous data, as well as metadata for the scene describing the physical properties of all objects in the scene. Our agent was iteratively developed for and evaluated on Oracle and Level 2 datasets.

![RGB, Depth, and Mask image of the same scene](image)

**Fig 4.** RGB, Depth, and Mask image of the same scene

Lastly, while the reported results are only contingent on the decision made at the end of the scene, the task has additional submission requirements, including per scene, per frame, and per pixel continuous confidence score in physical intuition. Should a scene be completely plausible, and not contain any violations, these values should trend towards 1.0, and should the entire scene be implausible, these values should trend towards 0.

**PyBullet**

While the IPE treats each simulation of a given scene as a statistical sample, incorporating uncertainty about the scene into each prediction, our approach is fundamentally simpler. We work under the same assumption that each simulation contains a certain amount of uncertainty, provided by our perception methods, though we perform both qualitative and quantitative analysis on each simulation in order to compute similarity between observations and simulations in terms of low-level physical position of objects in the scene, as well as higher level abstractions that affirm or deny adherence to physical rules.

The Bullet Physics Software Development Kit\(^3\), is an open source repository written in C++ for real-time collision detection and physics simulation. PyBullet is a python module that allows users to import the Bullet Physics SDK into python programs. The decision to use PyBullet to solve our problem came down to two key factors. The first is that PyBullet is a light-weight and computationally efficient package that can be run with or without a GUI, allowing developers to debug from nearly any system. The second is that the documentation for the SDK is robust and in-depth, making it easy to begin development quickly.
The PyBullet module offers various methods for generating, monitoring, and controlling 3D mesh objects in real time or user controlled simulation. However, a key limitation is that it cannot construct custom mesh objects. Therefore, the team utilized solid-modeling design software to create approximate replicas of the solid-objects visualized in Unity. We accepted that this would lead to a small amount of error.

**Computer Vision and Convolutional Neural Networks**

Computer vision techniques are necessary when dealing with real world or simulated images. Most simple approaches have been made publicly available thanks to open-source packages like OpenCV. However, the problem poses an interesting problem in the form of replicating 3-dimensional scenes based on 2-dimensional observations, as well as classifying identified objects as one of six 3-dimensional shapes and one of five task specific roles.

Multiclass image classification is a complicated problem that has recently been simplified thanks to advances in deep learning. With a relatively small number of layers, and a decently sized dataset, most classification problems that would have given even the largest group of developers trouble have become somewhat arbitrary. We train a single network with two convolutional layers followed by a fully connected layer to predict the classification of masked objects, and then use a series of heuristics to determine the object’s role.

In order to determine the 3-dimensional position and dimensions of an identified object, we assumed a pin-hole camera model that takes the parameters of the Unity simulation camera and the depth map output of the scene controller, and converts objects in an image (pixel-space) into a 3-dimensional point-cloud (world-space). The centroid and 3-dimensional
bounding box of the object are then computed using existing work available with Open3D\textsuperscript{5}, an open-source software package for Python.

**Solution**

**Scene Reconstruction**

Our problem has been staged such that the accuracy and quality of our predictions is limited by two factors. The first is the ability of our vision model to accurately predict object position, orientation, shape, and role from given data. At Oracle, this becomes arbitrary, as all mentioned data is given to the agent at the beginning of each time step. At Level 2, this is quite a significant problem, and requires full use of all given image data (RGB, Depth-maps, Object Masks).

At the beginning of each time step at Level 2, our agent's vision component extracts the image data; segments the scene by objects determined in the given masks; computes heuristics to determine the object's role; calculates a 3-dimensional point cloud representation of each object; and then predicts the object's shape, position and bounding-box dimensions. If we determine the role of an object to be anything other than the target, or pole object, we know that it is a rectangular prism. We purposefully exclude the computation of orientation, and instead assume a yaw, pitch, and roll of all objects as [0 0 0]. This is done with the same understanding as before that we are accumulating an indeterminate amount of error, and are trying to predict the approximate object dynamics rather than calculate the exact dynamics.

Our agent stores and references the trajectories of objects that it detects in order to compute the location of the “drop step.” In each scene, until the drop-step occurs, the dynamics of the target object are determined by the pole object. After this point, in a plausible scene, the target object will then be acted upon by forces of gravity, falling from its position at that point onto the support object or floor. The time step that this occurs in differs across scenes. Once it is determined, we execute a simulation in PyBullet that initializes the target, floor, and support objects given their calculated positions and dimensions at the drop step. We run this simulation until the target object is determined to be at rest; collecting positional, rotational, dimensional, and contact point data at each time step.

**Determining Violations**

The evaluation submission expects an output from the agent at each time step detailing where it predicts a violation of gravity to be in the scene, as well as a final prediction once the scene has concluded. Due to the fact that proper gravity compliant dynamics do not begin until the drop-step, as is defined in the task, we determined three cases for these in-step predictions.
Our agent reports that no violation has occurred with full confidence at each step prior to the drop step. Should the drop step have occurred, and the PyBullet simulation has been run, we then determine whether objects in the Unity scene have come to rest. If the target object has not come to rest, the agent reports no violation with full confidence. If the target object has come to rest, we compute our confidence in the observed dynamics of the scene based upon our observations of the PyBullet simulation. If the computed confidence is below a reasonable threshold, we report a violation at the centroid of the target object in pixel coordinates, along with a heatmap visualizing our confidence in the position of the target object relative to the rest of the scene.

Once the scene has ended, we recompute our confidence in the scene in case the point at which the target object is at rest is not the last step of the scene. This provides more time-series information for our confidence calculation, suggesting more accurate calculations of similarity. As well, we used the collected contact, position, rotation, and bounding box data from our PyBullet simulation to qualitatively compare the two scenes. We look for agreement between both simulations with respect to the following two heuristics:

1. Are the target and support objects in contact with each other?
2. Are the target and floor objects in contact with each other?

Once we calculate these heuristics for each scene independently, we perform a binary comparison of each heuristic. If there is a disagreement with regards to criteria between the two simulations, we raise an error. As well, if in the Unity scene we find that neither heuristic is true, we raise a violation. For example, if in the Unity scene the target object is at rest on the support object, while the PyBullet simulation predicted the target object to be at rest while in contact with the support, we raise a violation and output the aforementioned heatmap.

Confidence Calculation

We decided to approach the confidence function similarly to how we’d approach an error function in machine learning. Because the only difference in the PyBullet and MCS Unity simulations should be the position and orientation of the target object, we found it best to
represent the confidence we have in the Unity scene based upon how similar the trajectories of the target objects are in both scenes, ie:

Given |P| !|= |U|:
Let P = [ p_t | p ∈ R^3, t ∈ Z:1 ≤ t ≤ |P| ]
Let U = [ p_t | p ∈ R^3, t ∈ Z:1 ≤ t ≤ |U| ]

\[ C(P, U) = \tanh\left(\frac{a}{\text{distance}(P, U)}\right): a \in R \] (1)

We chose to represent the similarity of the two trajectories P and U as the euclidean distance between corresponding points in each time series/trajectory. Due to simulation differences between PyBullet and Unity, each trajectory will have different lengths, thus requiring an algorithm to measure the similarity between two temporal sequences. We determined that the python module, fastDTW, which provides a Dynamic Time Warping algorithm with user specified distance functions, would be the best tool to solve the problem. Dynamic Time Warping is an algorithm which computes the optimal discrete matching of existing elements in one set to another, that is the one with the minimum cost. In other words, given ||.||_2 is the euclidean vector norm, fastDTW computes a list of |P| points lofastDTWng where each entry is a point from the PyBullet and Unity target object trajectories such that each, and computes the total distance between each matching point.

Where \( w_k = (P_i, U_j), w_{k+1} = (P_i', U_j'), 1 <= i <= i+1, j <= j <= j+1: \)

\[ \text{distance}(P, U) = \sum_{k=1}^{K} ||w_{kj} - w_{k'i}||_2 \] (2)

It is important to remember that in this task, the confidence of the agent represents how plausible it believes the scene to be, continuously scaled between 0 and 1. We know that \( \text{distance}(P, U) \) is a positive real number for all P and U, and want small differences in trajectory to not influence the overall confidence. This is the key intuition behind using the inverse cost of the matching sequence as the input to a hyperbolic tangent function. When the difference in the object trajectories is near zero, the confidence approaches 1.0, as the difference in trajectories increases, the confidence approaches 0.0.

**Evaluation**

Figures 7, 8 and 9 visualize the results of the DARPA MCS 3.5 Gravity evaluation between three teams, and an included baseline. Each team was evaluated against the same set of scenes at both Oracle and Level 2. Each “Plausibility” on the y-axis breaks down the number of correct and incorrect predictions by whether or not the given scene contained a violation or not. The x-axis for Figures 7 and 8 are the number of tests in each criteria, while the x-axis in Figure 9 is the percentage of total tests matching the plausibility criteria.
Fig 7. Gravity Task Test Results at Oracle Level by Plausibility

Fig 8. Gravity Task Test Results at Level 2 by Plausibility
In the Violation of Gravity task, our agent performed best overall at both Oracle and Level 2, achieving a 99.7% detection accuracy in the former and an 89% detection accuracy in the latter, with an overall performance of 95% accuracy. In the Level 2 setting, our agent performed better when detecting plausible scenes than when detecting implausible scenes. We outperformed all other agents in all metrics with the exception of detecting implausible scenes correctly.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Plausible Correct</th>
<th>Plausible Incorrect</th>
<th>Implausible Correct</th>
<th>Implausible Incorrect</th>
<th>Overall Correct</th>
<th>Overall Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPICS (us)</td>
<td>0.9</td>
<td>0.1</td>
<td>0.88</td>
<td>0.12</td>
<td>0.89</td>
<td>0.11</td>
</tr>
<tr>
<td>IBM-MIT-Harvard-Stanford</td>
<td>0.69</td>
<td>0.31</td>
<td>0.99</td>
<td>0.01</td>
<td>0.84</td>
<td>0.16</td>
</tr>
<tr>
<td>MESS*</td>
<td>0.01</td>
<td>0.99</td>
<td>0.99</td>
<td>0.01</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Baseline</td>
<td>0.7</td>
<td>0.3</td>
<td>0.7</td>
<td>0.3</td>
<td>0.7</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Table 1. Level 2 Agent Performance by Plausibility. *MESS includes labs from The University of Washington, The University of Michigan, and The University of California, Berkeley
Table 2. Oracle Agent Performance by Plausibility. *MESS includes labs from The University of Washington, The University of Michigan, and The University of California, Berkeley

<table>
<thead>
<tr>
<th>Agent</th>
<th>Plausible Correct</th>
<th>Plausible Incorrect</th>
<th>Implausible Correct</th>
<th>Implausible Incorrect</th>
<th>Overall Correct</th>
<th>Overall Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPICS (us)</td>
<td><strong>0.99</strong></td>
<td><strong>0.01</strong></td>
<td><strong>1.0</strong></td>
<td><strong>0.0</strong></td>
<td><strong>0.99</strong></td>
<td><strong>0.01</strong></td>
</tr>
<tr>
<td>IBM-MIT-Harvard-Stanford</td>
<td>0.69</td>
<td>0.31</td>
<td>0.99</td>
<td>0.01</td>
<td>0.84</td>
<td>0.16</td>
</tr>
<tr>
<td>MESS*</td>
<td>0.01</td>
<td>0.99</td>
<td>0.99</td>
<td>0.01</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Baseline</td>
<td>0.7</td>
<td>0.3</td>
<td>0.7</td>
<td>0.3</td>
<td>0.7</td>
<td>0.3</td>
</tr>
</tbody>
</table>

It is worth noting that each team’s agent performed identically between both evaluation levels with the exception of our own. It is likely that other teams used the same system between evaluation levels, while our Oracle level agent does not utilize the vision system at all, instead using the object metadata given to it by the scene controller.

Analysis

Upon dissection of scenes in which our agent failed to correctly determine the presence of a violation, we uncovered three key issues in our system.

An issue we didn’t encounter during development or development, that some might have noticed in the Confidence Calculation section above, is that if the distance between trajectories is exactly zero, our result is undefined. Unfortunately this oversight propagated into a bug that resulted in the agent automatically ending the scene and reporting a violation, whether or not there was one. This specific failure accounted for all of our incorrect predictions in the Oracle evaluation, and three in the Level 2 evaluation. Specifically, scenes in which the target object is a cube and is already fully above the support object; such that it does not move from its initial position at the drop step, trigger this bug. As well, this only occurs with target objects in the shape of a cube, likely because the PyBullet rendering of the object is most similar to the Unity rendering.

The biggest issue we encountered, as expected, was with our vision system’s perception abilities. While calculating position in the world relative to other objects seemed to occur without issue, our system had difficulty assessing the bounding box of the object with respect to depth. Objects closer to the center of the 2-dimensional image would have increasingly reduced depth scales applied to them. In other words, objects in the center of the screen were calculated to be especially thin, causing unpredictable behavior, and occasionally clipping through other objects. This obviously creates a large amount of potential error, if an object is not deep enough to hold
itself up, the resulting trajectory will almost certainly deviate greatly, and no longer represent an approximate trajectory to use as a viable prediction. We believe that this fault represents roughly half of our prediction failures.

Lastly, during development we had noticed issues with confirming whether or not the floor object is in contact with the target object using traditional axis-aligned bounding-box methods. As such, we opted to assume that the target object was in contact with the floor when it’s centroid was sufficiently close to a height of zero. While this worked for smaller objects whose centroid was inherently closer to the ground, larger objects observed in Unity would not necessarily register as in contact with the ground. This created enough false positives in the Level 2 evaluation for us to further consider a new approach to the vision system in the coming evaluations.

Future Research

Generalization of Model

Due to the performance of our model, we have decided to attempt to generalize our Gravity Agent to solve other kinds of physical reasoning tasks. DARPA’s MCS program has outlined other tasks that revolve around detecting other physical phenomena and violations, centered around object permanence. We plan to use PyBullet to form hypotheses on the dynamics of objects regardless of “role.” Every object in the scene will be subject to the same procedure for violation detection that we created for the gravity task, and additional robustness will appear in the form of detection of violations of expectations relating to the presence, absence, position, and appearance of objects throughout the scene.

We would also like to join our generalized model with an agent capable of interacting with its environment. An agent capable of forming accurate predictions of the motion of objects in its environment will create opportunities for novel research in the fields of robotics and human computer interaction. Such an agent could provide a solid foundation for the development and learning of more complicated tasks, such as those revolving around the presence of other observable agents and the traversal of complicated and/or dangerous pathways.

Automatic Mesh Generation

The bulk of future work will focus on automatic mesh generation, an extension of our existing vision work. As we move towards task evaluations featuring objects of irregular shape moving across the scene, approximating dynamics becomes much more difficult than whether or not the object will fall and land on another object.

We hope to apply an amodal 3-D reconstruction of observed objects in the scene, combining deep learning and existing open3D features to extrapolate point-clouds created from a single perspective into fully 3-D mesh objects that can be directly inserted into PyBullet without any algorithmic manipulation of a reference mesh. Assuming this reconstruction process
provides fruitful results, we will be able to model the dynamics of nearly any object with a much higher degree of accuracy. This will also replace the current object classification model in the vision module, as we will no longer need to assign a shape in order to find a corresponding mesh template.

Appendix