Towards Intuitive Rigid-body Physics Through Parameter Search

Javier Felip[1] and David Gonzalez-Aguirre[1] and Omesh Tickoo[1]

***Abstract*— The ability to predict the future location of objects is key for robots operating in unstructured and uncertain scenarios. It is even more important for general purpose humanoid robots that are meant to operate and adapt to multiple scenarios. They need to determine possible outcomes of actions, reason about their effect and plan subsequent movements accordingly to act preemptively. The prediction ability of current robotic systems in is far from that of humans. Neuroscience studies point out that humans have a predictive ability, called intuitive physics, to anticipate the behavior of dynamic environments enabling them to predict and take preemptive actions when necessary, for example to catch a flying ball or grab an object that is about to fall off a table.**

In this paper, we present a system that learns to predict based on previous observations. First, object's physical parameters are learned through observation using parameter search techniques. Second, the learned dynamic model of objects is used to generate probabilistic predictions through physics simulation. The parameter search update rules proposed, are compared to other approaches from the state-of-the-art in physical parameter learning. Finally, the predictive capability is evaluated through simulated and real experiments.

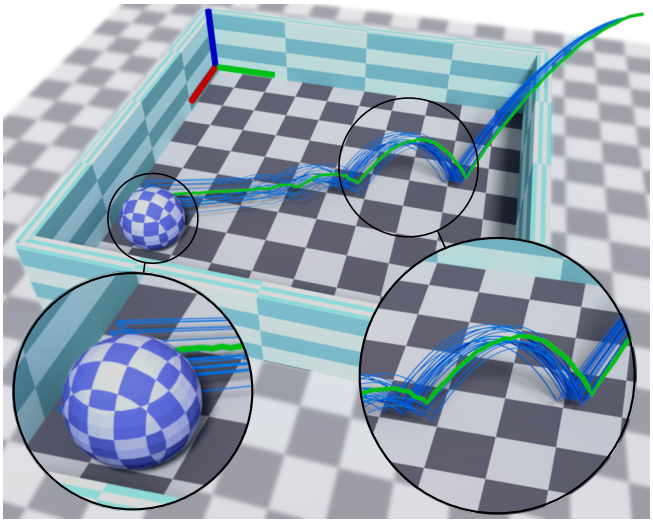


Fig. 1. Prediction results after looking at the first 3 points of the green trajectory. Thin blue lines: Predicted trajectory distribution. Thick green line: Observed trajectory

I. Introduction

Reasoning in cognitive systems requires a mechanism to foretell the evolution of an observed scene. Prediction is important to evaluate the outcome of a previously executed action and plan accordingly (e.g. to avoid a ball it is necessary to predict its trajectory). This prediction ability needs to be learned from experiences and adapt to new scenarios, which becomes critical in the specific case of multi-purpose humanoid robots.

There are many neuroscience studies that propose models of human intuitive physics [1], [2]. However, there is no clear consensus on internal representations of objects, environment, action and interactions. While on the one hand, it is accepted that the physics laws are not directly encoded in the brain and humans use simplifications and heuristics to make predictions [3]; on the other hand, some studies point out that a Newtonian representation of the environment can be used to mimic the prediction ability of humans [4], [5].

In this paper, an approach to intuitive physics is implemented and evaluated. It is able to observe a scene, learn the required physical properties of the objects and provide probabilistic predictions. For the learning phase, we approach the problem of determining the physical parameters of objects as a black-box optimization problem and evaluate several methods for our specific use-case. Once parameters are learned, predictions are generated using an implementation of the noisy-newton paradigm [5]. The approach is quantitatively and qualitatively validated through simulated and real experiments.

The contributions of this paper are: i) Application of novel search methods for learning object and environment material parameters. ii) Evaluation of search methods that outperform state-of-the-art for this particular problem. iii) Quantitative and qualitative evaluation of predictions, showing that it is possible to learn an intuitive physics model from observation and provide reliable predictions for a reduced object universe.

II. Related Work

Available prediction methods can be classified into three main types: data-driven, model-based and hybrid. In an effort to use a data-driven approach for prediction, Kopicki et. al. [6] used machine learning techniques to encode the effects of physical interaction on objects. However, the approach lacks generalization, produce implausible results and requires object-wise manual parameter tuning. This method was improved by Belter et. al. [7], combining a physics engine with the learning approach to force physically plausible predictions. As Belter et. al. claim, their hybrid method provides fairly good results useful for prediction. However, it inherits generalization problems from data-driven methods and high dimensionality from model-based approaches. More recently, data-driven approaches that use deep neural networks have

[1] Intel Labs. 2111 Northeast 25th Street, Hillsboro, OR 97124, USA {javier.felip.leon, david.i.gonzalez.aguirre, omesh.tickoo} @intel.com

been proposed [8]. Nevertheless, as analyzed by Zhang et. al. [9] model-based approaches naturally generalize (w.r.t materials and objects), while they also capture the way human judgment works and seem to encode causal relations where so far DNNs have not been successful.

The approach presented in this work is model-based and uses a physics engine in its core. Physics engines have already been used to predict the future state of the world for a variety of applications like trajectory prediction [10], manipulation [11] and tracking [12]. However, like most model-based systems they require a lot of parameters to be tuned. Some approaches make assumptions about those parameters such as friction coefficients, restitution coefficients and material density. In the presented approach, those parameters are obtained using parameter search techniques combined with scene observation. The use of model-based approaches has its advantages and disadvantages:

- ✓ Can be tuned through parameters, while data-driven approaches have to re-train the interactions with specific object instances.
- × Lack of robustness and adaptation to uncertainty of sensor data used to perceive the environment.

In the last decade, probabilistic approaches have become very popular to manage uncertainty in model based systems [13]. The probabilistic approach to physics simulation is known as approximate physics and is commonly implemented using the noisy-newton paradigm [5] which consists on performing several simulations adding noise to the initial state of the objects to obtain different possible results [4]. These kind of models have been used to mimic human intuitive physics [5], [14], [15]. Noisy-newton paradigm is model-based and parameters have huge impact on results. In this paper, we use the noisy-newton approach to provide probabilistic predictions. In addition, model parameters are estimated through observation.

Wu et. al. implemented a similar idea using object tracking to obtain observations of the object trajectories and fit the simulation parameters to match each observation [16]. The authors demonstrated human-like prediction. However, the learning is performed independently for each observed scene and does not integrate different observations, their parameter search method selection is not evaluated or compared to other relevant methods and their predictions do not provide an associated confidence value.

III. Methodology

The proposed intuitive physics system uses a physics engine to obtain expected trajectories of observed objects. Probabilistic predictions are provided using a noisy-newton approach with the observed dynamic parameters θ_d and the learned static parameters θ_s. Besides the parameters θ_d and θ_s, predictions also depend on the set of observed object primitives Ω_o and the set of environment primitives Ω_e. Only rigid object primitives (sphere, box and capsule) with uniform density are considered. For the application of our approach in a real scenario, we consider Ω_o, Ω_e and θ_d to be obtained by the perception system (see an example in Sec. IV).

In order to learn θ_s, a set of observations containing object identifier, shape primitive and trajectory are required. Next, a parameter search is performed to obtain the values of θ_s that generate predictions closer to observations (i.e. minimizes the cost function). Once the learning process converges, predictions can be generated by providing each object's identifier and initial dynamic parameters θ_d. The output is the expected probabilistic trajectory for each object.

A. Parameter space

Provided the set of observed objects Ω_o and the set of environment objects Ω_e, the behavior of simulations depends on two sets of parameters θ_d and θ_s;

1) Dynamic parameters: The first set, θ_d is associated to each object motion and can change over time. Each object requires its own dynamic parameter set:

- Position: $x \in \mathbb{R}^3$
- Orientation quaternion: $q \in \mathbb{H}$
- Linear velocity: $\dot{x} \in \mathbb{R}^3$
- Angular velocity: $\omega \in \mathbb{R}^3$

No external forces (except gravity) are assumed thus acceleration is not included as a state parameter. Other effects such as air flow, temperature, humidity etc. are not considered, their influence to the results is assumed to be negligible in our simulations.

2) Static parameters: The second set θ_s is bound to material properties. θ_s are shape and material dependent and we assume that they do not change over time. Friction and restitution coefficients are known to be pairwise. It means that the coefficient depends not only on one material but on the specific pair of materials that are in contact.

- Mass: $M \in \mathbb{R}$
- Body inertia matrix: $I \in \mathbb{R}^{3\times3}$
- Pairwise static friction coefficient: $\mu_s \in \mathbb{R}$
- Pairwise dynamic friction coefficient: $\mu_d \in \mathbb{R}$
- Pairwise rolling coefficient: $\mu_r \in \mathbb{R}$
- Pairwise restitution coefficient: $e \in \mathbb{R}$

Unlike other dynamic simulations, in this paper friction and restitution are modeled as pairwise coefficients. Thus, the number of parameters $\psi \in \mathbb{N}^+$ that each pairwise coefficient requires, depends on the number of materials $m \in \mathbb{N}^+$ present on each observation and it can be calculated by

$$\psi = \begin{cases} m + \frac{m!}{2!(m-2)!}, & \text{if } m \geq 2 \\ 1, & \text{otherwise.} \end{cases} \quad (1)$$

Mass M and inertia I depend only on object's material density $\rho \in \mathbb{R}$ and its volume $v \in \mathbb{R}$. Therefore, given that the object geometric primitive is an input to the system, both M and I can be replaced by ρ. After this simplification, the remaining parameter set is:

$$\theta_s = \{\rho, \mu_s, \mu_d, \mu_r, e\} \text{ and } \theta_d = \{x, q, \dot{x}, \omega\}. \quad (2)$$

θ_d is obtained through observation, thus only θ_s is learned leaving the dimensionality of the optimization problem to be:

$$|\theta_s| = m + 4\psi, \tag{3}$$

where ψ is obtained from Eq. 1 and $m \in \mathbb{N}^+$ is the number of different materials considered. On any observation there is at least one object and one material. However, in addition to the object's materials, an extra material for the environment static objects is added.

B. Trajectory representation and distance metric

Trajectories are represented by series of 4-tuples where the first three values represent the 3D position and the fourth the time-stamp. See Eq. 4 where $\Phi(\tau) : \mathbb{R}^4 \mapsto \mathbb{R}$ is a function to extract the time-stamp of a trajectory data point. The j-th data point in a trajectory is denoted as $\mathcal{T}_j$. In a valid trajectory, the time dimension is monotonically increasing.

$$\begin{aligned} \Phi(\tau) &= \tau \cdot [0, 0, 0, 1], \\ \mathcal{T}_j &:= \{\mathcal{T}_j \in \mathbb{R}^4 | \Phi(\mathcal{T}_{j+1}) > \Phi(\mathcal{T}_j)\}. \end{aligned} \tag{4}$$

In order to determine the error of a predicted trajectory and its corresponding observation, a method to calculate the difference between two trajectories is required. A common metric to compare curves is the Hausdorff distance [17], but this distance does not take into account the order of points. The Fréchet distance [18] takes into consideration the order of points but it does not reflect the simultaneous increment in the temporal dimension. Now by taking into account that time increases simultaneously along both trajectories, two specific distance calculations for discrete trajectories were implemented. Given two trajectories $\mathcal{T}^a \in \mathcal{T}$ and $\mathcal{T}^b \in \mathcal{T}$, their distance can be calculated as follows: First, a time-synchronized set $\mathcal{T}^c \in \mathcal{T}$ is obtained using the two time neighbors from $\mathcal{T}^b$ for each data point in $\mathcal{T}^a$:

$$\begin{aligned} \mathcal{T}_j^c = \Big\{ & (1-w) \cdot \mathcal{T}_i^b + w \cdot \mathcal{T}_{i+1}^b \mid \\ & \Phi(\mathcal{T}_{i+1}^b) \geq \Phi(\mathcal{T}_j^a) \geq \Phi(\mathcal{T}_i^b) \wedge \\ & w = \frac{\Phi(\mathcal{T}_j^a) - \Phi(\mathcal{T}_i^b)}{\Phi(\mathcal{T}_{i+1}^b) - \Phi(\mathcal{T}_i^b)} \Big\}, \end{aligned} \tag{5}$$

where w is the interpolation weight that depends on the time distance to each neighbor in $\mathcal{T}^b$. Second, we define the operator $\Gamma(\tau) : \mathbb{R}^4 \mapsto \mathbb{R}^4$ to extract the spatial position of a trajectory point as: $\Gamma(\tau) = \text{diag}(1, 1, 1, 0) \cdot \tau$.

Finally, the following functions to compute distance among trajectories can be used:

$$\bar{D}(\mathcal{T}^a, \mathcal{T}^c) = \frac{1}{|\mathcal{T}^c|} \sum_{i=1}^{|\mathcal{T}^c|} \left| \Gamma(\mathcal{T}_i^a) - \Gamma(\mathcal{T}_i^c) \right|_2 \tag{6}$$

$$\hat{D}(\mathcal{T}^a, \mathcal{T}^c) = \max_{|\mathcal{T}^c|} (\left| \Gamma(\mathcal{T}^a) - \Gamma(\mathcal{T}^c) \right|_2). \tag{7}$$

See trajectory comparison in Fig. 5. Due to the max operator, $\hat{D}$ is very sensitive to outliers and sensing errors. However, this metric is closer to well known trajectory comparison approaches [17], [18] that in the absence of outliers and small training sets yield good results. The selection of the distance metric to be used will depend on the quality and quantity of the training data. To improve the reliability of our experiments, due to noisy data we have selected $\bar{D}$.

When a contact occurs, object orientation has a critical role in resulting velocities. Due to physical interaction of objects, their trajectory in the orientation space has a direct effect on objects' translation trajectory. Therefore, a more accurate object orientation prediction will result in a closer trajectory using the metrics described above. For this reason we consider objects' orientation to be evaluated implicitly by the comparison of full trajectories only in translation.

C. Cost function

The cost function $J(\theta_s)$ is used to evaluate the fidelity of predictions given a set of ground-truth observations Y. Each observation $Y^i \in Y$ contains one trajectory Y_j^i for each object present in that observation. Therefore Y_j^i refers to the trajectory of the j-th object in the i-th observation. Conversely, X is the prediction set formed by predictions. For each observation $Y^i \in Y$ a prediction $X^i \in X$ is computed obtaining one trajectory X_j^i for each object present in the observation. Although the simulation depends on several other factors (θ_d, Ω_o and Ω_e), the system is able to observe them which leaves only θ_s to be learned, therefore the cost function only depends on the static parameters θ_s and is defined as follows:

$$J(\theta_s) = \lambda \sum_{i=1}^{|Y|} \sum_{j=1}^{|Y^i|} \bar{D}(X_j^i, Y_j^i). \tag{8}$$

The cost of the current parameter values θ_s is obtained as the mean trajectory difference between all the predicted and observed trajectories. The λ multiplier used to obtain the mean over all the trajectory differences is calculated as follows:

$$\lambda = \Big(\sum_{i=1}^{|Y|} |Y^i| \Big)^{-1} \tag{9}$$

D. Parameter search

Behavior of objects depends on 1) the simulator configuration and 2) laws of dynamics implementation, there is no guarantee of J to have only one local minimum and be smooth. In this scenario, gradient based methods do not guarantee convergence to a global minimum. For these reasons, we can approach the problem of determining θ_s as a black-box function optimization. A very similar problem is tackled by the Deep Learning community to determine the best value of neural network hyper-parameters. The state-of-the-art in hyper-parameter search is focused on rather simple update rules [19]. For the specific problem presented in this paper, Wu et. al. [16] have successfully

applied Monte-Carlo Markov Chains (MCMC) with one-dimensional Metropolis-Hastings (1D-MH). However, as our experimental results show, there are superior approaches and update rules outperforming this algorithm.

In this work, we use a parameter search approach to obtain θ_s. It is an iterative process (See Algorithm 1) that minimizes the cost function result by optimizing the parameter values θ_s given a set of observations Y and a convergence criteria ξ. First, the candidate parameters θ'_s are established depending on the search strategy used (namely update rule). Then, one simulation for each observation $Y_i \in Y$ is executed to obtain predictions $X_i \in X$. The cost of the candidate parameter set θ'_s is obtained using $J(\theta'_s)$ in Eq. 8. The parameter set θ_s is updated if the candidate parameters θ'_s used to obtain X have produced lower cost. The algorithm ends when the convergence criteria is met, this is usually implemented based on a limit number of iterations, number of iterations without reducing the error or the error being below a threshold.

Algorithm 1 Parameter search

```
function ParameterSearch(Y, ξ)
    ε_min = ∞
    while not(ξ) do
        θ'_s ← updateRule(θ_s)
        X ← RunSimulations(θ'_s, Y)
        if J(θ'_s) < ε_min then
            ε_min ← J(θ'_s)
            θ_s ← θ'_s
    return θ_s
```

In this paper we propose the following new update rules: **parameter gradient search, adaptive random search, hybrid search, hybrid-random search** and compare them with: 1) methods from the hyper-parameter search literature, namely grid search, random search, coordinate descent and 1D-MH; and 2) one of the most popular black-box function optimization methods: Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [20].

- **Grid search**: Tests all possible parameter combinations changing one value at a time with a fixed increment δp. The number of iterations for a complete grid search is $n^{\delta p^{-1}}$ where n is the number of parameters and δp^{-1} the possible values for each parameter. This makes the approach only suitable for low number of parameters that can take few values.
- **Random search**: All the parameter values are randomly generated from a uniform distribution over each parameter minimum and maximum values.
- **Coordinate descent**: Similar to grid search, but sweeps parameters one by one without testing all combinations. The number of iterations required is $n \cdot \delta p^{-1}$ for each sweep of all the parameters. The order used to modify the parameters is randomly generated before each sweep.
- **Parameter gradient search**: The first iteration uses random search. For the rest, a random parameter is targeted and the increment is set to the learning rate $\delta p = \alpha$. The parameter is updated with δp. Subsequent iterations continue adding the same δp if the cost was reduced. Otherwise $\delta p = -\delta p/2$. If δp is below a configurable threshold β the next parameter is targeted.
- **Adaptive random search**: The first iteration uses random search. Subsequent iterations use $\delta p = \mathcal{U}(-\epsilon_{i-1}, \epsilon_{i-1})$ where ϵ_l corresponds to the previous iteration cost. This strategy adapts the parameter accounting for the current cost of the parameter set.
- **Hybrid search**: A combination of parameter gradient search and random search that performs a configurable number of random search iterations i for initialization before switching to the parameter gradient update rule.
- **Hybrid-random search**: A variation of the hybrid search that switches to adaptive random search after the desired number s of hybrid search sweeps is performed.
- **1D-MH**: δp is sampled from $\mathcal{U}(-\alpha, \alpha)$ where α is the learning rate. On each iteration δp is added to a single parameter, this method is implemented in [16].
- **CMA-ES**: Is an evolutionary algorithm where in each iteration a set of λ candidate solutions are generated. On each iteration, for each candidate δp is obtained parameter-wise based on the covariance matrix of the parameters. The details of this derivative-free iterative optimization method can be found in [20].

E. Probabilistic physics simulation

As discussed by Gerstenberg et.al [5], the implementation of a human-inspired prediction engine based on Newtonian laws of physics, requires probabilistic results. Once the parameters θ_s of the observed objects are learned, a simulation can be executed to obtain a probabilistic prediction. However, a physics engine is deterministic and cannot provide likelihoods that represent confidence of predictions.

In the literature, probabilistic physics engines are implemented adding random noise to the initial dynamic state of the objects θ_d and counting the number of results that satisfy a specific condition [14]. These implementations are able to answer questions such as: *Will it fall?* or *Will it collide?* but are task-based and lack generality. Our implementation provides a more generic approach to intuitive physics engines that is not task-based and can answer queries about future states in a probabilistic fashion.

Similarly to other approaches, in our implementation a configurable number of simulations are run for each prediction. For each simulation, Gaussian noise is added to the perceived state (see Sec. III-A) of the objects θ_d. The difference is in the result which consists of a set of trajectories that are combined into a single prediction as shown in Fig. 1.

F. Physics engine selection

There are several available game physics engines. However, game oriented physics engines focus on obtaining visual appealing results instead of high fidelity simulation. An evaluation of the most important engines from the robotics perspective can be found in [11]. To ease the problem of

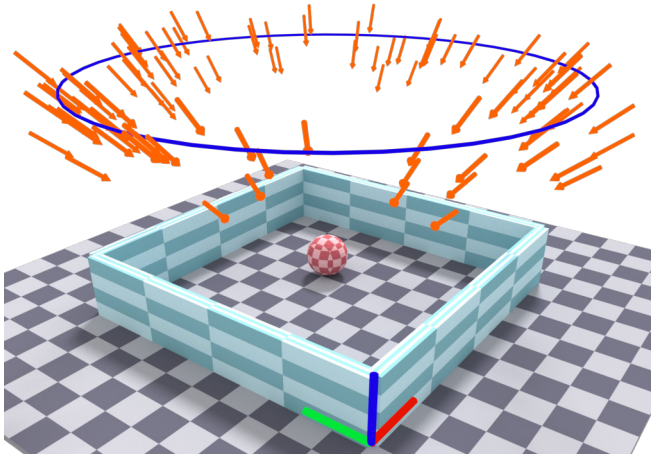


Fig. 2. Simulated experimental environment with a sample spherical object. Orange vectors show a sample set of 100 initial positions and velocities of the objects used for training. Blue circumference shows the seed initial positions to generate the objects' initial state. The area of the squares on the floor plane is $1m^2$ and the quadrilateral formed by the 4 walls is $10 \times 10m$.

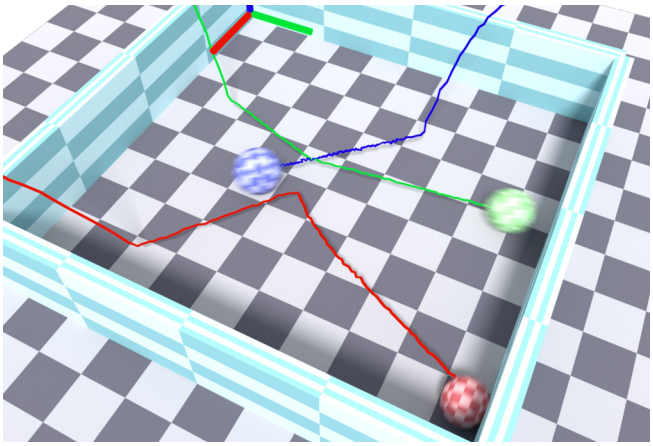


Fig. 3. Generated observation with three objects. The noise model used is $\mathcal{N}(0, 0.02)$ at 30 Hz sample rate.

selecting a physics engine there have been efforts on middleware implementation that can provide abstract interfaces to any supported physics engine such as PAL [21], OPAL [22] or FISICAS [23]. Unfortunately, those efforts have been abandoned and no support for new physics engines versions is provided. In our implementation, the physics engine used is ODE [24]. As pointed out by Erez et. al., ODE is the open source physics engine that provides better results for robotics related tasks [11]. Although ODE does not implement static, dynamic and rolling friction, in this paper we have extended it to include those coefficients.

IV. Experimental validation

For the validation of the proposed approach, we have designed a simulated experimental environment (See Fig. 2) that will be used to generate a synthetic dataset which in turn is used to test the learning and prediction of the implemented intuitive physics engine.

A. Experimental setup

The simulated environment Ω_e is depicted in Fig. 2 and consists of 5 boxes. One for the floor and 4 for the walls to keep the objects bounded and favor their interaction, the size of the quadrilateral formed by the walls is 10×10 meters. To reduce simulation complexity, the object universe is restricted to spheres, capped cylinders (i.e. capsules) and rectangular prisms (i.e. boxes).

B. Synthetic dataset generation

First, a pool of objects is created randomizing their shape type (box, sphere, capsule) and dimensions sampled from $\mathcal{U}(0.1, 0.7)$ meters. A different material is assumed for each object and static parameters θ_s are randomly generated. The dataset is generated by running multiple simulations and storing object trajectories until the desired number of samples per object is acquired.

For each simulation run, the number of objects to appear is calculated as $k = \lceil \mathcal{N}(1, 1) \rceil$. Then, the object set Ω_o, is created by randomly selecting k objects from the object pool. For each object in Ω_o, its initial dynamic parameters θ_d are set using Eq. 10, where r and c are the radius and center of the circumference that will be used as a hint to generate the initial states for objects as shown in Fig. 2.

$$\begin{aligned} x &= c + r\,[\,\sin(\alpha)\;\cos(\alpha)\;0\,]^T + \mathcal{N}(0,1), \\ \dot{x} &= c - x + \mathcal{N}(0,1), \\ \omega &= \mathcal{N}(0, 0.1). \end{aligned} \tag{10}$$

Finally the simulation is executed and the trajectory of each object is stored. Examples of initial states for the generated observations are shown by orange vectors in Fig. 2.

In order to provide a more realistic observation dataset, noise is added to each trajectory point. The noise model used is set to $\mathcal{N}(0, 0.02)$ according to state-of-the-art object recognition and pose estimation algorithms [25] and sample rate is set to 30 Hz. An example of a generated observation with three objects is shown in Fig. 3.

C. Execution

In our experiments a time step of 10ms was used and simulation time was set to 5 seconds in order to give enough time for objects to interact with each other. To determine the sensitivity of the approach to the number of samples, we have run experiments with 1, 5 and 10 objects and 10, 50 and 100 samples per object (See Fig. 4). To account for the possible bias of the random functions used by the parameter search methods, all the experiments in this work have been executed 10 times and the results averaged. The generated dataset of trajectories was divided into 80% for training and 20% for testing.

D. Results and discussion

An example of a prediction is shown in Fig. 1. Furthermore, Fig. 4 shows a comparison of the error obtained for training and test sets. Overall, the trajectory prediction

obtains an average error of 0.911 m $\pm$0.189 in the test datasets. The obtained error might seem high but a qualitative evaluation of prediction results shown in Fig. 5 gives us the intuition that despite the apparent high error, predictions are reliable and consistent. Especially if we consider that trajectories of objects are predicted 5 seconds into the future and the error added in the trajectory dataset impacts the initial state estimation that causes the prediction to drift along time.

Given that a prediction (with 20 noisy-newton samples) takes around 10 ms, predictions can be updated on-line according to the perceived state reducing error and drift. Experiments were executed on a Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz with 32Gb DDR4@2400Mhz.

V. Search algorithms evaluation

We have evaluated 9 different parameter search methods: grid search, random search, coordinate descent, parameter gradient search, adaptive random search, hybrid search, hybrid-random search, 1D-MH and CMA-ES. The evaluation criteria is the convergence speed (i.e. number of iterations) and the final error. Table I shows the parameters used for each method during the experimental evaluation. To account for the multiple candidates per iteration used by CMA-ES, we have counted each $J(\theta_s)$ evaluation as one iteration. Although not all the update rules have exactly the same per-iteration cost, the simulation roll-out required to evaluate each θ'_s is orders of magnitude more expensive than any of the update rules. For that reason we consider all the update rules to have a negligible per-iteration cost enabling us to compare convergence rate and speed with the number of iterations.

All methods were tested with datasets composed of 1, 10 and 20 random objects with 1, 10 and 50 samples per object. The datasets were generated as explained in Section IV-B. The different datasets test how dimensionality and number of samples impact performance. As shown in Fig. 6a, for low dimensions (14 as obtained by Eq. 3 with $m = 2$) all the methods converge fast (around 50 iterations) except 1D-MH that apparently gets stuck in a local minimum.

As expected, in higher dimensional spaces; grid search, coordinate descent and parameter gradient do not provide good results. Surprisingly, as shown in Fig. 6b the adaptive-random strategy obtains fast convergence and low error for 275 parameters (m=11). As the parameter dimensionality keeps increasing, the CMA-ES method provides better results but adaptive-random is still close to the best performance. Results for a 20 object dataset, with 945 parameters are depicted in Fig. 6c. Considering the simplicity of the adaptive-random strategy versus the more sophisticated CMA-ES, it is still a good choice for the tested dataset configurations.

VI. Real experiments

Experiments were performed to showcase the application of the presented ideas on a real scenario. To capture the scene, an RGBD sensor was used. Point cloud sequences of bouncing balls were captured and used as input to our system, each one containing one observation of a single object.

Grid and Coordinate descent	$\delta p = 0.05$
Parameter-gradient and adaptive random	$\alpha = 0.1, \beta = 0.001$
Hybrid	$i = 300$
Hybrid-random	$s = 5$
1D-MH [16]	$\alpha = 0.05$
CMA-ES [20]	$\sigma = 0.2, \lambda = 20$

TABLE I

Parameter settings used for the parameter search method evaluation.

A. Experimental setup

The experimental objects used were 3 spheres with 3 cm, 5 cm and 7 cm radius. The object universe to be observed was restricted to spherical shapes to ease object detection. The environment can be composed by any number of planes with the floor plane perpendicular to the gravity vector being the dominant plane. The experimental setup can also be seen in the accompanying video, during the real experiments data acquisition scene.

B. Environment modeling

Our experimental environment is composed of a supporting floor plane and two walls. However, there are no assumptions made on the number of planes that can be present in the scene. The environment is reconstructed for each observation. The plane that contains more points, is assumed to be the supporting plane and the gravity vector is assumed to be parallel to the supporting plane normal. An example of an observation with the detected planes and gravity vector is shown in Fig. 7. Planes are detected fusing the first frames of each observation file applying the RANSAC [26] algorithm with a plane model.

C. Real dataset generation

We collected 90 observations of the three objects described in Sec. VI-A which were thrown into the scenario by hand with different velocities and initial positions. The accompanying video shows the data acquisition process, the parameter search and several predictions with the measured prediction error.

For each frame of each observation file, all the points that belong to a detected plane are removed. The remaining points are used to run a RANSAC algorithm using a spherical model to detect the bouncing spheres. The detection is performed frame-to-frame and no temporal data is used. After processing all the files, the dataset for learning is generated using the trajectory provided by the frame-to-frame object detection and averaging the detected object's dimensions. See an example of a trajectory in Fig. 7.

D. Results and discussion

We executed the parameter search process for each group of 30 samples and obtained the mean error shown in Fig. 8. Although errors are lower than results obtained with the

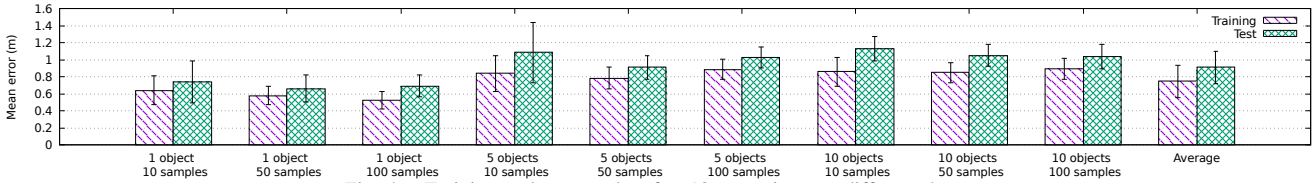


Fig. 4. Training and test results after 10 executions on different datasets.

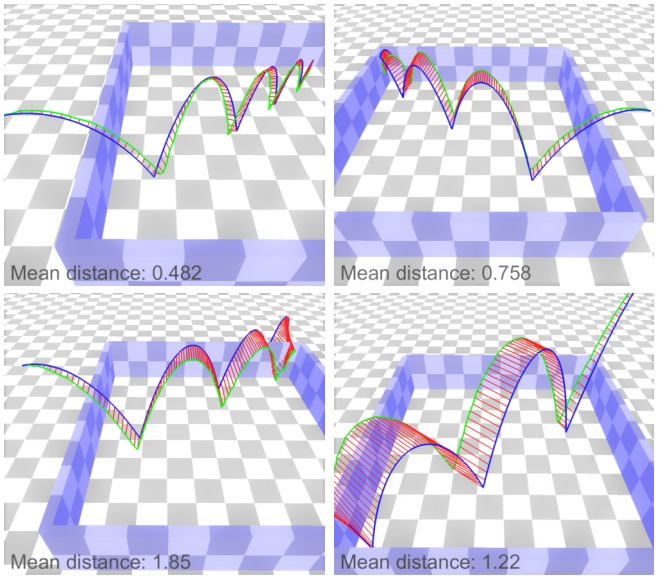


Fig. 5. Examples of predicted trajectories (blue) compared with the test set trajectories (green) and the error between each data point (red). Although mean errors are high, trajectories are qualitatively approximate. Mean distance corresponds to $J(\theta_s)$ for each shown observation-prediction pair.

synthetic dataset, it is important to note that the scale of objects and trajectories is also smaller.

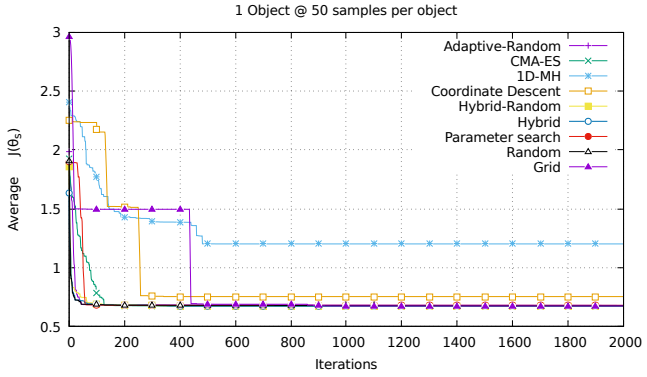
Predictions are very sensitive to initial state estimation errors that produce early trajectory drift. This idea is supported by the inverse correlation of object size and prediction error as shown in Fig. 8. However, results show that feasible trajectories can be predicted even with an unfiltered dataset obtained with straightforward methods.

VII. Conclusions

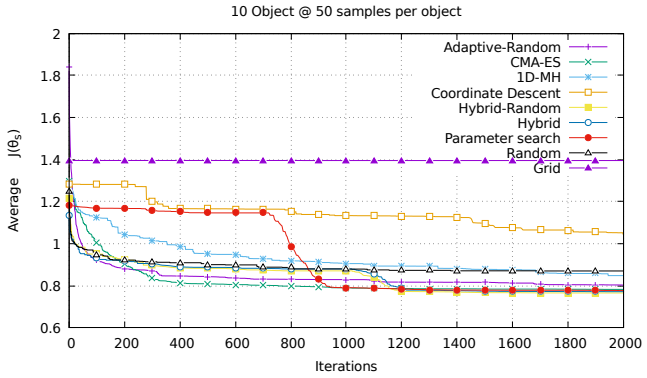
Our intuitive physics engine is able to learn a reduced universe of object models from observations and make probabilistic predictions of their motion. Predictions were validated by simulated and real experiments. Our results show that it is possible to use probabilistic simulation to provide fairly accurate predictions. However, the learning process is time consuming and has to be substantially improved to allow on-line learning and prediction.

Additionally, our experimental evaluation of 9 different parameter search methods, has shown that both CMA-ES and our Adaptive Random are a good choice for learning.

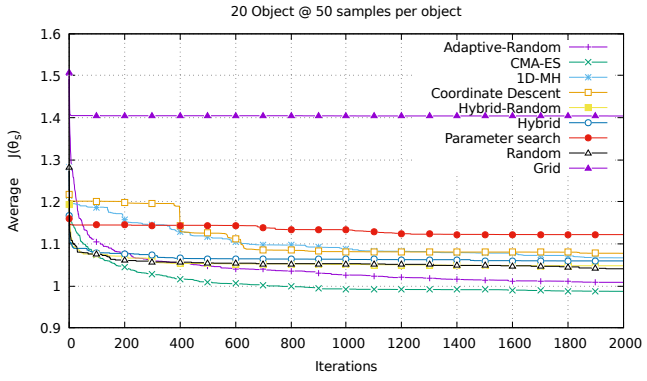
Such an intuitive physics engine will enable further research in probabilistic predictive models. It is our firm belief that on-line geometric and physical modeling will play a major role for the future of humanoid robots.



(a)



(b)



(c)

Fig. 6. Parameter search performance evaluation: $J(\theta_s)$ vs. iterations for datasets of 1, 10, 20 objects and 50 samples per object. For each iteration and update rule, vertical axis shows the $J(\theta_s)$ value averaged over 10 runs.

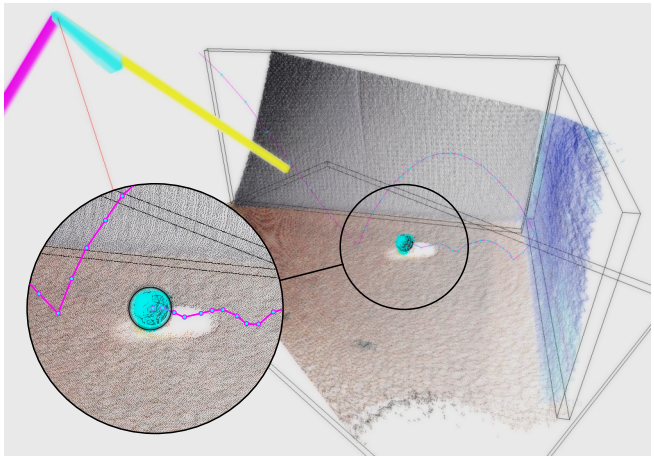


Fig. 7. Observation process of a bouncing ball on the real experimental setup. Detected planes wire-frame in black. Detected sphere shown in cyan. Observed trajectory is shown in magenta and each sample point is shown in cyan. Gravity vector is shown in red.

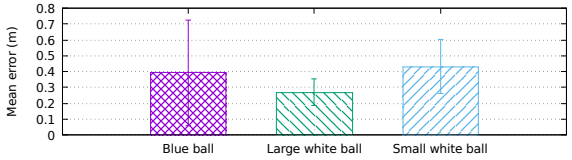


Fig. 8. Prediction error for each of the objects in the real dataset.

REFERENCES

[1] J. Pearl, *Causality: Models, Reasoning, and Inference*. New York, NY, USA: Cambridge University Press, 2000.

[2] P. Wolff, "Dynamics and the perception of causal events," 2006, (in press). In T. Shipley & J. Z In T. Shipley & J. Zacks (Eds.), Understanding events: How humans see, represent, and act on events. Oxford University Press. [Online]. Available: http://philsci-archive.pitt.edu/3127/

[3] I. E. K. Andersson and S. Runeson, "Realism of Confidence, Modes of Apprehension, and Variable-Use in Visual Discrimination of Relative Mass," *Ecological Psychology*, vol. 20, no. 1, pp. 1–31, 2008.

[4] A. N. Sanborn, V. K. Mansinghka, and T. L. Griffiths, "Reconciling intuitive physics and Newtonian mechanics for colliding objects." *Psychological review*, vol. 120, no. 2, pp. 411–37, apr 2013.

[5] T. Gerstenberg, N. D. Goodman, D. A. Lagnado, and J. B. Tenenbaum, "Noisy Newtons: Unifying process and dependency accounts of causal attribution," 2012.

[6] M. Kopicki, S. Zurek, R. Stolkin, T. Morwald, and J. Wyatt, "Learning to predict how rigid objects behave under simple manipulation," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, may 2011, pp. 5722–5729.

[7] D. Belter, M. Kopicki, S. Zurek, and J. Wyatt, "Kinematically optimised predictions of object motion," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, sep 2014, pp. 4422–4427.

[8] T. Xue, J. Wu, K. L. Bouman, and W. T. Freeman, "Visual Dynamics: Probabilistic Future Frame Synthesis via Cross Convolutional Networks," jul 2016. [Online]. Available: http://arxiv.org/abs/1607.02586

[9] R. Zhang, J. Wu, C. Zhang, W. T. Freeman, and J. B. Tenenbaum, "A Comparative Evaluation of Approximate Probabilistic Simulation and Deep Neural Networks as Accounts of Human Physical Scene Understanding," may 2016. [Online]. Available: http://arxiv.org/abs/1605.01138

[10] N. Kyriazis, I. Oikonomidis, and A. Argyros, "Binding Computer Vision to Physics Based Simulation: The Case Study of a Bouncing Ball," in *Procedings of the British Machine Vision Conference 2011*. British Machine Vision Association, 2011, pp. 43.1–43.11.

[11] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2015, pp. 4397–4404.

[12] K. Pauwels and D. Kragic, "SimTrack: A simulation-based framework for scalable real-time object pose detection and tracking," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, sep 2015, pp. 1300–1307.

[13] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[14] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum, "Simulation as an engine of physical scene understanding," *Proceedings of the National Academy of Sciences*, vol. 110, no. 45, pp. 18 327–18 332, nov 2013.

[15] T. Ullman, N. Goodman, and J. Tenenbaum, "Learning physics from dynamical scenes," in *In Proceedings of the Thirty-Sixth Annual Conference of the Cognitive Science society*, 2014.

[16] J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. Tenenbaum, "Galileo: Perceiving Physical Object Properties by Integrating a Physics Engine with Deep Learning," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 127–135.

[17] F. Hausdorff, *Felix Hausdorff - Gesammelte Werke Band III: Mengenlehre (1927,1935) Deskripte Mengenlehre und Topologie*. Springer Berlin Heidelberg, 2008.

[18] M. Fréchet, *Sur quelques points du calcul fonctionnel*, 1906.

[19] G. Luo, "A review of automatic selection methods for machine learning algorithms and hyper-parameter values," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 5, no. 1, p. 18, 2016.

[20] N. Hansen, "The CMA evolution strategy: a comparing review," in *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, J. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, Eds. Springer, 2006, pp. 75–102.

[21] A. Boeing and T. Bräunl, "Evaluation of real-time physics simulation systems," in *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia - GRAPHITE '07*. New York, New York, USA: ACM Press, dec 2007, p. 281.

[22] (2004) Opal: Physics abstraction layer. [Online]. Available: http://opal.sourceforge.net/index.html

[23] (2011) Fisicas: Physics abstraction layer. [Online]. Available: http://opengrasp.sourceforge.net/FISICAS.html

[24] R. Smith, "Open Dynamics Engine - ODE," 2007. [Online]. Available: www.ode.org

[25] A. Aldoma, F. Tombari, R. B. Rusu, and M. Vincze, "OUR-CVFH–Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram for Object Recognition and 6DOF Pose Estimation," in *Pattern Recognition*. Springer Berlin Heidelberg, 2012, pp. 113–122.

[26] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, June 1981.

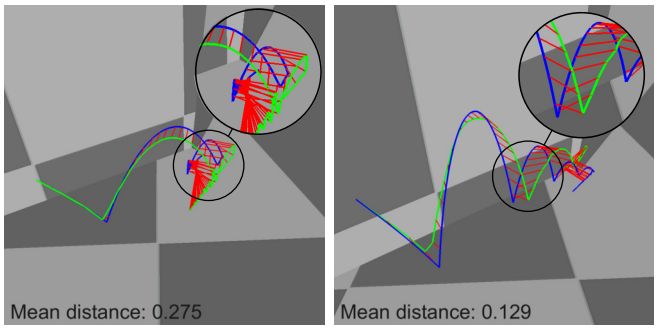


Fig. 9. Prediction result of a real observation. Predicted trajectory (blue) compared with the observed trajectory (green) and the error between each data point (red). Please see more examples in the attached video. Mean distance corresponds to $J(\theta_s)$ for each shown observation-prediction pair.