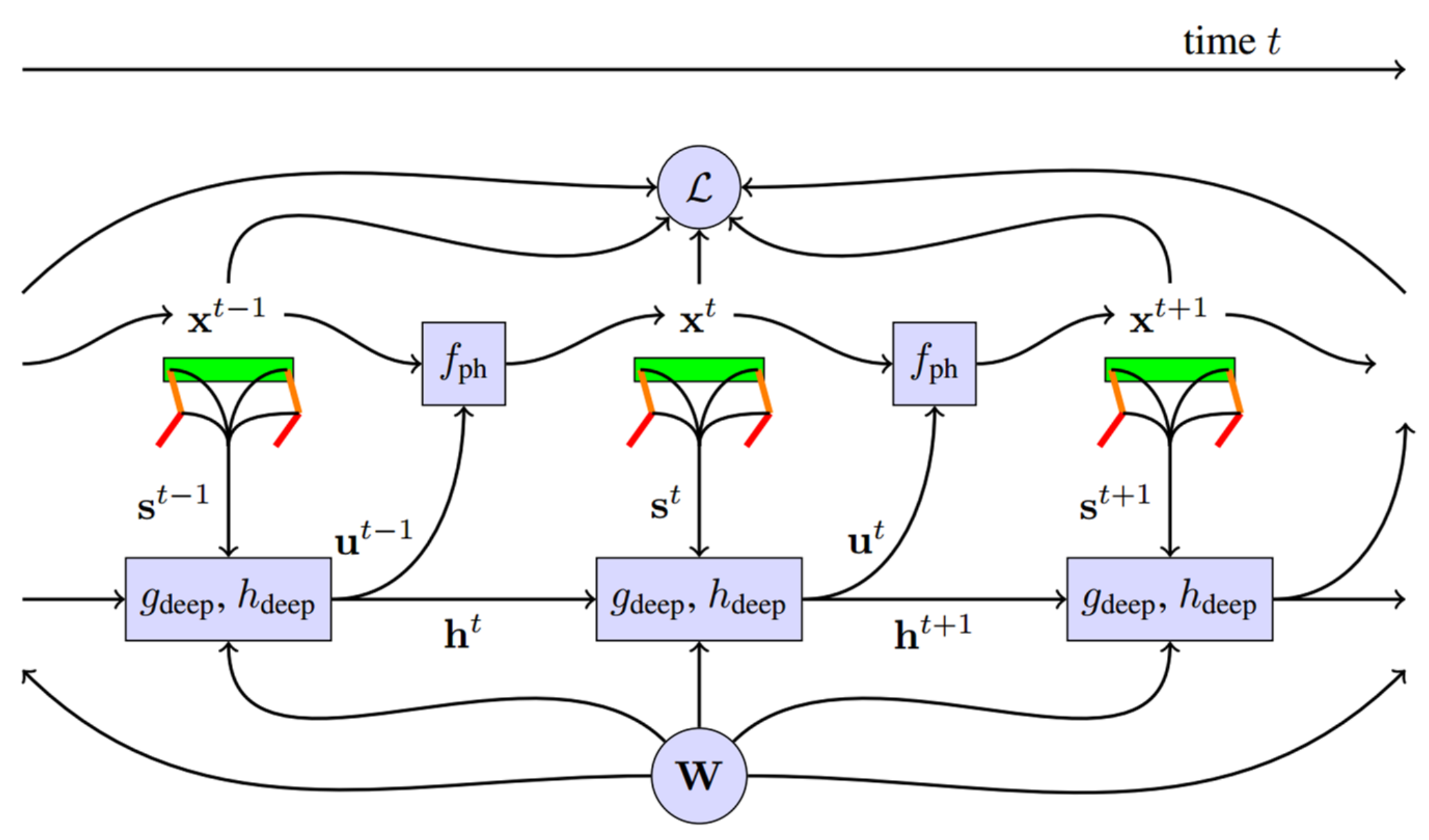
A Differentiable Physics Engine for Deep Learning in Robotics

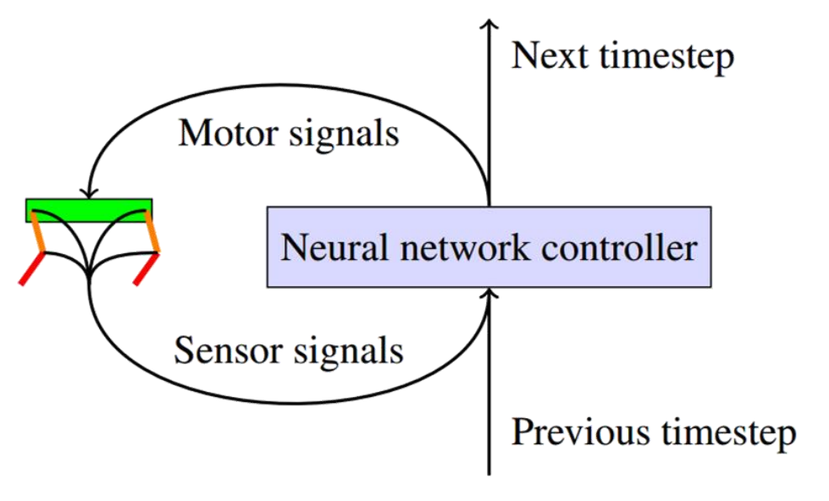
We developed a differentiable physics engine. We show how backpropagation through physics is remarkably fast, demonstrate some simple use cases and indicate how this can be used to improve a series of current approaches for optimization processes in robotics, especially optimizations regarding deep learning.

Implementation



We implemented a rigid body engine similar to physics engines commonly used to simulate robotics, in particular PhysX, Bullet, Havok and ODE. These are 3D rigid body engines, where all bodies have 6 degrees of freedom, and the relation between them are defined as constraints, e.g. to prevent the bodies from penetrating each other. The velocities, positions and constraints of the rigid bodies define a linear complementary problem (LCP-problem) which we solve using a Gauss-Seidel projection method. The solution of this problem are the new velocities of the bodies, which is then integrated using a semi-implicit Euler integration to get the new positions. Because this system is not always numerically stable, the constraints are softened.

In order to differentiate through this engine, we implemented the entire engine in Theano. Theano is an automatic differentiation library, developed with a focus on deep learning. We find that Theano is able to symbolically differentiate through the physics engine and that it produces quite efficient code to evaluate the Jacobians.



Therefore, we are able to look in a new way at the deep learning control optimization problem in robotics. By implementing a neural network controller for a robot, which receives sensor signals and outputs motor signals, together with this engine, we are able to treat this system as a recurrent neural network. And this we can optimize using backpropagation through time. Or backpropagation through physics, if you will.

How fast is it?

Because differentiation is done symbolically with an automatic differentiation library, we did not know how long it would take to find a gradient and perform an update step. We reckoned it could be an intractable computational problem. However, we found that on GPU, it is only about 7 times slower as evaluating the physics model.

Seconds of computing time required to simulate a batch of robots for 10 seconds

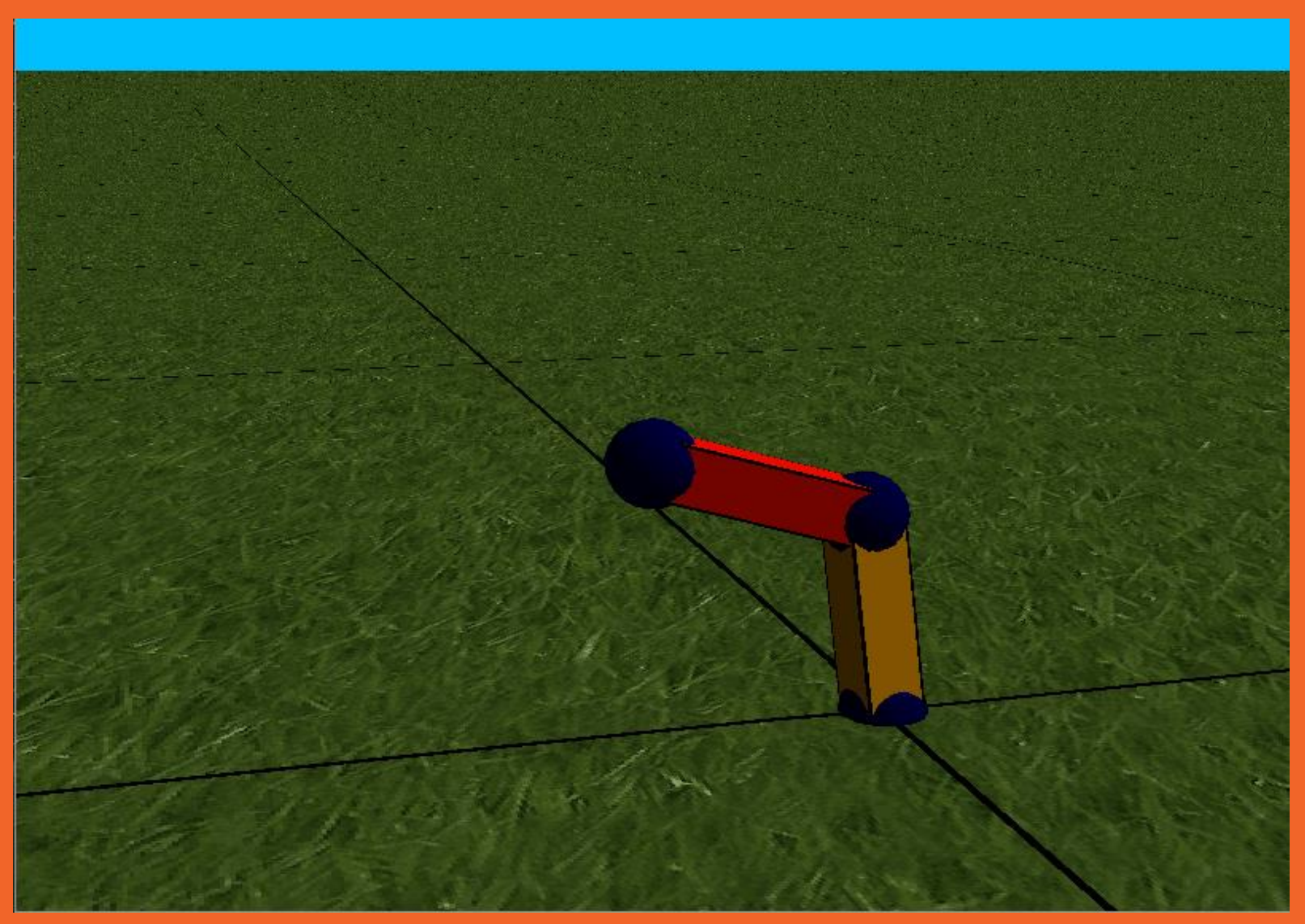
		with gradient		without gradient	
		CPU	GPU	CPU	GPU
1 robot	1 296 parameters	8.17	69.6	1.06	9.69
	1 147 904 parameters	13.2	75.0	2.04	9.69
128 robots	1 296 parameters	263	128	47.7	17.8
	1 147 904 parameters	311	129	50.4	18.3

Some test cases:

Ball Trajectory

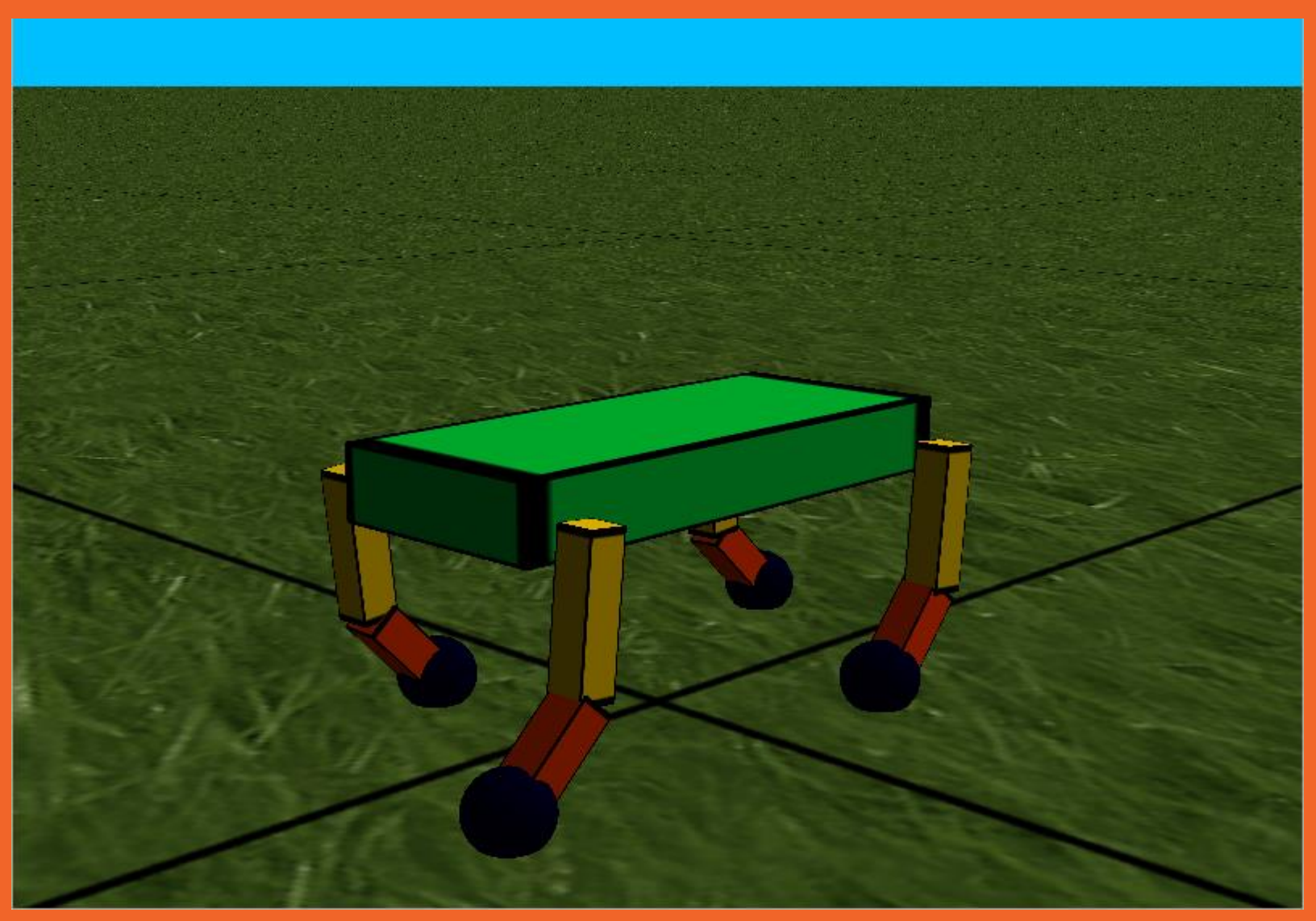
We implemented a ball model into our physics engine, and optimized the initial velocity of our ball in order to be at rest after 10 seconds on a point 10 meter away. This 6 dimensional problem is optimized with backpropagation through physics in 88 iterations or 16,3 seconds, which is faster than the 2.422 iterations or 59,9 seconds required by the gradient-free CMA-ES optimization. This shows that backpropagation through physics can already be faster from as little as 6 parameters.

Robot Arm



We implemented a 1 meter, 4 degree of freedom robot arm model and used a 17.000 parameter neural network controller . We were able to learn a global policy neural network controller with an average 5cm accuracy in its entire reachable domain. This took 2.563 update steps, or 656.128 evaluations of the model. CMA-ES did not show any sign of convergence, even when optimizing to reach only a single point, as it could not handle the sheer number of parameters.

Quadruped



We tried optimizing a controller for a quadrupedal robot. The robot starts from standstill, without any prior knowledge of a walking motion. The goal is for the neural network to transform a periodic sine signal into motor signals for the motors, in order for the robot to maximize its velocity during 8 seconds.
In only 500 model evaluations, a gait with a summer sault every three steps is found, which is 50% faster than the hand tuned gait we found before.

... and many more use cases

We see many ways in which this approach can be extended further. The neural network controller can be made more advanced, to profit from recent improvements in deep learning. You could also use the physical model as a strong prior to train a forward or backward model in a neural network. We also see no reason why hardware parameters could not be optimized either, or why a differentiable camera could not be implemented. We imagine that adversarial robotics training (ART) could bring a boost to the domain of evolutionary robotics.

Jonas Degrave, Michiel Hermans, Joni Dambre and Francis wyffels

elis UNIVERSITEIT GENT IWT Institute for the Promotion of Innovation by Science and Technology in Flanders NVIDIA