Evaluating a Physics Engine as an Ingredient for Physical Reasoning

Erik Weitnauer, Robert Haschke, and Helge Ritter

Bielefeld University, Neuroinformatics Group and the Research Institute for Cognition and Robotics, Universitätsstr. 25, 33615 Bielefeld, Germany
{eweitnau,rhaschke,helge}@techfak.uni-bielefeld.de

Abstract. Physics engines have been used in robotics research for a long time. Beside their traditional application as a substitute for real world interactions due to their higher speed, safety and flexibility, they have recently also been used for motion planning and high level action planning. We propose to further explore the idea of using a physics engine as means to give a robot a basic physical understanding of its environment. In this paper, as a preliminary step, we study, how accurately the process of pushing flat objects across a table with a robot arm can be predicted in a physics engine. We also present an approach to adapt the engines parameters to enhance the simulation accuracy.

Keywords: physics engines, physical simulation, physical reasoning, robotics

1 Introduction

One prominent research objective in robotics is to build autonomous service robots that are fit to share our common living space. However, we are still very far from robots that can perform everyday human tasks like tidying a room as smoothly as we do. A major challenge is the integration of the different abstraction layers of information the robot has to deal with. While goals and actions are usually represented at a high level of abstraction, the sensory input and motor control information are represented and processed on a much lower level of abstraction.

To allow reasoning about goals and actions an understanding of the world's dynamics and the effects of actions is necessary. For example, when asked to arrange books on a table it is good to know how they behave – they might fall down if placed too close to the edge of the table. To interact with objects a physical understanding of the world would be very beneficial, as it would provide a very general and powerful way for both interpreting perceptions and planning actions. Such a model of the world's behavior should ideally be learned automatically from low-level sensor inputs, so that the robot can adapt to new situations and no hand-crafted models are needed.

In fact, humans also acquire 'physical' concepts and rules, which help them to interprete the world. The very basic concepts are already learned at a quite early age: Young infants of only three months already have a concept of solid objects. They are surprised when they see solid objects passing through each other or not falling down when they lack support [1]. This kind of 'naive physical understanding' of the world certainly plays an important role in structuring their perception and actions.

The artificial intelligence community has long been concerned with finding appropriate computational models for human-like physics knowledge and reasoning. The developed theories and algorithms such as qualitative process theory [2] and QSIM [3] are all based on qualitative representations. Their power lies in their ability to infer all possible behaviors of the modeled systems without the need for complete, quantitative knowledge of the system's initial state. In improved versions of the original QSIM algorithm, available numerical data can be additonally fed into the qualitative simulation to further restrain the number of possible behaviors [4].

Physics engines provide an alternative way of simulating physical interactions quantitatively and have become popular in the robotics community in the last ten years. Traditionally applied in the design and monitoring of actual physical systems, they have recently also been used in the context of evolutionary optimization and motion planning. Lipson et al. [5] use physical simulations to repeatedly evaluate the fitness of a large number of individuals in an evolutionary process to develop blueprints and motor programs of robots. The motion planning community employs physics engines to simulate the temporal progress of controlled motions of complex robots where kino-dynamic constraints must be taken into account [6].

In contrast to qualitative simulation techniques, physics engines operate at a lower abstraction layer and can be used for robot control more directly. However, they are so far not considered as means to endow a robotic system with the ability to reason about physical world interactions. We propose to take a new perspective on physics engines in this regard.

Reasoning processes guiding an autonomous robot naturally have to cope with questions concerning the stability of manipulated objects such as whether they could tip over or start to roll or how to shift several objects without changing their configuration. Other questions are related to the controllability of objects and the estimation of uncertainty in the prediction of the actions' effects. Physics engines provide answers to these questions in a general and detailed way. They can therefore complement classical rule-based reasoning systems by acting as a source of information or 'oracle' for them. Pasula's work [7], in which she uses a physics engine to build a simplistic qualitative world model, provides an example along this line.

As a first step towards the long-term objective of employing physics engines in physical reasoning, in this paper we closely analyze the Bullet engine, which is an arbitrary choice among current, freely available engines. We present our study on the accuracy of predicting outcomes of real world interactions in a well-defined scenario: a robot arm pushing flat objects across a table. In the following section, we describe the experimental setup and the data acquisition process. Section 3 gives an overview of the Bullet physics engine, its capabilities

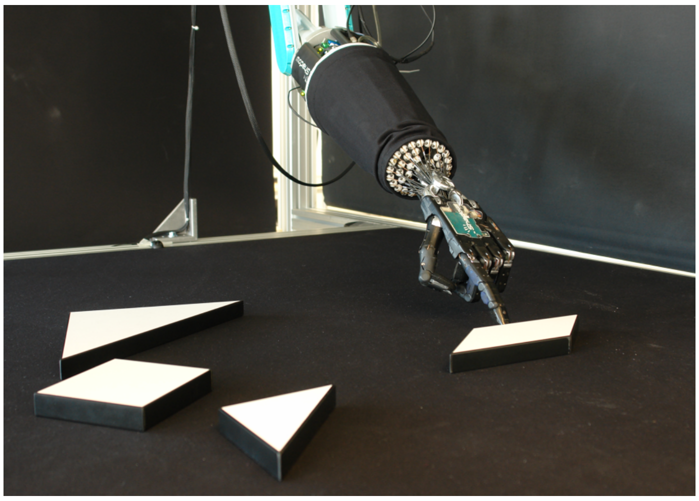


Fig. 1. The robot pushes one of the tangram pieces across the table. The results of this interaction are recorded and used to improve the accuracy of the according physical simulations.

and restrictions, many of those being typical for physics engines in general. In section 4 we present an optimization method to choose simulation parameters, which improve the resemblance of simulated and real world actions. Finally, we discuss the results in section 5 and give an overview of possible applications of the proposed techniques in section 6.

2 Experimental Scenario

2.1 Setup

We decided to test the simulation accuracy of a current physics engine in a scenario of flat objects being pushed across a table. This scenario was chosen, because it is simple and still several physical properties like friction, mass and shape of the objects have to be taken into account in simulations. Specifically, we used the four wooden tangram pieces shown in Fig. 1. In each pushing trial, one of the tangram pieces was placed on the table at a predefined position and then pushed against its side with a robot hand. The hand was attached to a redundant 7 DoF Mitsubishi PA10 robot arm.

The trials were conducted with the square, parallelogram, small triangle and large triangle placed at varying initial positions x_0 (see Fig. 2). For each trial, the kind of tangram piece, its initial position, its final position and the trajectory of the pushing fingertip were recorded. The path of the hand movement was always identical, although the duration it was actually pushing the pieces varied for their different initial positions (Fig. 3). In total we had 19 different start conditions across the objects, for each of which we performed five (identical)

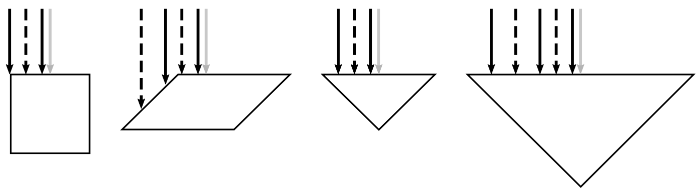


Fig. 2. The arrows in the figure mark the positions where the robot hand started to push the tangram pieces. The tangram pieces were positioned at $x_0 \in \{10\,\text{cm}, 11\,\text{cm}, 13\,\text{cm}, 15\,\text{cm}, 18\,\text{cm}, 21\,\text{cm}\}$, while the hand trajectory followed a straight line of 40 cm length at $x = 10\,\text{cm}$. The pushing trials with the dashed start positions were used as test data, the solid ones as training data in the optimization process outlined in Section 4.

pushing trials. Repeating the trials was done to find out how much the final poses of a pushed object vary under identical conditions, providing a baseline for the subsequent predictions.

2.2 Data Analysis

The camera system used to recognize the pushed object's shape as well as its initial and final pose provides an accuracy of ± 1 mm for position and $\pm 1°$ for rotation. The fingertip trajectory was obtained from applying the forward kinematics to the joint angle trajectory recorded with 100 Hz. Due to controller inaccuracies, the deviation of the fingertip trajectory from its ideal straight line amounts to ± 2 mm.

An interesting aspect of the recorded pushing actions is the variance within the five identical trials for each of the starting conditions. Due to the limited precision with that the tangram pieces can be placed at the designated initial position and the controller inaccuracies, it is not possible to exactly reproduce a single trial. However, the variance between identical trials strongly depends on the starting condition. It is highest when the pieces are pushed close to their center (see Fig. 3). The standard deviations for all shapes are plotted in the results section, where we compare them to the simulation results.

3 Action Prediction with Physics Engines

3.1 Choosing a Physics Engine

The design of physics engines heavily depends on their intended application and focuses either on high precision (e.g. for weather forecasting, climate calculations or airplane design) or on real-time performance (e.g. for computer games). Since we want to use the physics engine for continuous simulation and planning of robot behavior, real-time capability of the engine is essential.

[8] compared ten major real-time physics engines with regard to five categories including material properties, constraint stability and stacking behavior.

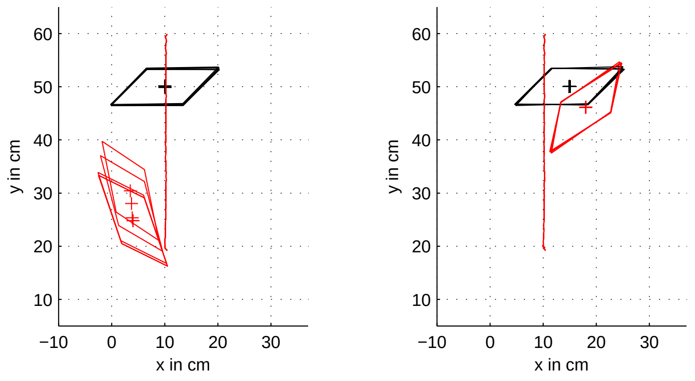


Fig. 3. The two figures show pushing trials recorded for the parallelogram-shaped object, which was placed at a different initial position in the left and in right figure. In each figure the final object positions of five repeated trials are plotted. The vertical red line marks the trajectory of the fingertip. When pushing close to the symmetry axis of the object, the variance of final position becomes bigger.

Although there was no physics engine that performed best in all categories they concluded that “of the open source engines the Bullet engine provided the best results overall, outperforming even some of the commercial engines.”

Based on this statement and considering the fact that it is freely available as open source[1], we decided to use the Bullet engine for our simulations.

This engine is capable of simulating rigid bodies as well as soft bodies which can be used to model cloth, ropes or deformable volumes. The engine implements Newton’s laws of motions and uses a simple friction model to simulate Coulomb friction. Bodies are prevented from penetrating each other by automatically introducing constraints, which are solved together with all user defined constraints, e.g. representing joints.

3.2 Stability and General Restrictions

One important issue about the physics simulations is how quickly two simulations diverge, when there are small differences in their initial conditions. There are two reasons for divergence. One is the limited numerical accuracy of floating-point operations, so that tiny numerical errors might accumulate in the simulation and cause different final results. The second reason are bifurcation points in the physical simulation state space. When placing a pyramid on its tip or pushing against one of the tangram pieces on one of their symmetry axes, the eventual direction of motion changes dramatically even for tiny changes in the initial conditions. Obviously, this also applies to real world experiments. However, the divergence of simulations can be reduced by changing parameters of the simulation algorithm, as this influences the way numerical errors accumulate.

[1] The code is open source under the MIT License, which permits anyone to use it for any purpose. The code is hosted at `http://code.google.com/p/bullet/`.

The implementations of most physics simulations focus on high performance at the cost of less accuracy. In fact, the physical laws like energy conservation or the constraint that objects should not intersect, are not *exactly* fulfilled. In most cases the resulting errors are quite small, but for certain parameter combinations, the simulated scene might become unstable and even 'explode'. This behavior will obviously also result in quick divergence of repeated simulations.

To address these issues, we did an iterative linear search over those simulation parameters, that significantly influenced the simulation stability to determine the values leading to the smallest variance of end results in repeated simulations. The obtained values are listed in the "global parameters" part of Table 1.

Another important question is how to adequately model real world objects in the physics simulation. As the Bullet engine is supposed to work best with objects not too big or small, a scaling factor applied to the dimensions of each of the real world object did improve the simulation for us. Last but not least, the friction is computed using a simple Coulomb friction model which is applied to each of the contact points between two bodies. In the case of the tangram pieces, the number of contact points is no higher than four and they are usually located at extremal points of the surface, in our case the corner points of the shapes. For an object rotating on the ground, this simplification leads to a strong overestimation of the friction-induced torque, because instead of correctly calculating the torque as

$$\tau_{\text{real}} = \mu F_N \frac{1}{A} \int r \, dA, \tag{1}$$

it is calculated as

$$\tau_{\text{sim}} = \mu F_N \sum r_i, \tag{2}$$

where μ is the dynamic friction coefficient, F_N is the normal component of the objects weight, A is its ground area and r is the distance of a point in A to the rotation center. Instead of integrating and thus averaging over all distances, only the extremal distances of the corner points r_i are taken into account by the engine. Therefore, in simulations the rotation of objects will slow down too fast. We solved this problem by slightly modifying the tangram objects in the simulations. Underneath each of the objects we attached a thin plate of identical shape, but reduced dimensions, thereby moving the contact points closer towards the rotation center, such that the friction-induced torque becomes smaller. Since the overall shape of the objects is merely affected, their behavior during collisions remains unchanged. We will refer to the shrinking factor as *shape factor* from here on. Every surface has a geometry-specific shape factor that can determined analytically.

4 Parameter Adaption

Our aim is to find a parameter setting for the Bullet engine so that its simulations yield the same results as observed in reality. We use the previously recorded data to define a cost function as the distance between simulation results and real world results. The recorded data consists of 19 sets of trials in which a single tangram piece was placed on the table and then pushed by the robot arm moving on a straight, well defined trajectory, as outlined in Section 2.

4.1 Simulation Parameters

Altogether we determined 17 parameters that affect the physical simulation. They can be divided into two groups: The first group comprises all parameters that reflect physical properties of objects. The parameters of the first group are friction and restitution of materials, the inertia tensor of rigid bodies, linear and angular damping of moving bodies, linear and angular factors that scale the forces acting on a rigid body per simulation step as well as the shape factor we introduced earlier. The second group contains global simulation parameters, most of them fine-tuning the Bullet engine's simulation algorithm: Gravity, the collision margin spanned around bodies, the simulation step size, the world scaling factor, the number of allowed solver iterations and the error reduction parameter (ERP).

By simulating pushing actions while varying these parameters one by one, we identified nine parameters that have significant influence on the simulation: angular factor and angular damping, friction, shape factor, gravity, world scaling factor and collision margin. All of these parameters are candidates for optimization. The latter three of them however, strongly influence the stability of the simulation, so they were fixed to their default values. The angular factor and angular damping parameters change the ratio between rotation and translation of the moved tangram pieces. They do not directly correspond to physical phenomena and the rotation-translation ratio can also be influenced by the shape factor, therefore we can exclude them from the training candidates, too. This leaves us with the three parameters to be adapted, listed in order of decreasing influence:

– shape factor of each tangram piece
– tangram/ground friction
– tangram/finger friction

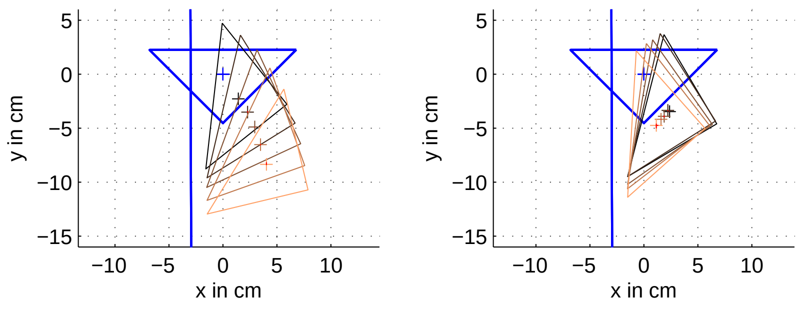
To show their effect on pushing simulations, in Fig. 4 typical simulation results for two of them are plotted. Before the actual optimization, all parameters were set to the values listed in Table 1. For the parameters to be optimized, these values were just a starting point.

4.2 Cost Function

For optimization we need a cost function, that maps each parameter setting onto a quality of the simulation, describing how well the physical interaction in the training data is reproduced. To judge the quality of a parameter setting π, a simulation with the Bullet engine is conducted for each of the trials using the corresponding tangram piece, the given initial position and the recorded trajectory of the fingertip. Afterwards, the final position P_{sim} of the tangram piece in

Table 1. Parameter Settings for Optimization

Parameter Name	Value	Parameter Name	Value
object parameters (adapt)		restitution ground	0.2
shape factor square	0.541	restitution finger	0
shape factor parall.	0.552	correct inertia tensor	on
shape factor tri. (s, l)	0.461	*global parameters (fixed)*	
friction tangram/ground	0.3	collision margin	0.04
friction tangram/finger	0.3	simulation stepsize	1/60
object parameters (fixed)		erp	0.2
angular, linear factor	1	world scaling factor	3
angular, linear damping	0	gravity	-9.81
restitution tangram	0.2	solver iterations	10



(a) shape factor (b) friction of tangram

Fig. 4. In both plots the final positions of the pushed triangle as a result of varying one simulation parameter are shown. In the left, the shape factor was varied between 0.3 and 1. In the right, the tangram/ground friction was varied between 0.05 and 1. With increasing values of the parameters, the final positions are plotted in a brighter color.

the simulation is compared to the real final position P_{real} in the training data. The polygon position P is described by a tuple of corner coordinates $(c_0, \ldots, c_n)$. The distance between the two final positions is measured as the mean distance of these corner points:

$$E\left(P, P'\right) = \frac{1}{n} \sum_{j=1}^{n} \|c_j - c'_j\|.$$

The error on the whole training set with N trials is defined as the mean distance between simulated and real final position:

$$E_D(\pi) = \frac{1}{N} \sum_{i=1}^{N} E(P_{\text{sim}}(\pi), P_{\text{real}})$$

4.3 Training and Testing Data Set

We divided the recorded data set into a training set D_{train} and a test set D_{test} as follows:

$x_0 =$	10 cm	11 cm	13 cm	15 cm	18 cm	21 cm
D_{train}		$\times$		$\times$		$\times$
D_{test}			$\times$		$\times$	

We do not use the trials where the objects were pushed at their axis of symmetry ($x_0 = 10$ cm), since for them the final positions in the real world vary too strongly. Although the number of trials in the training set is quite small ($5{\times}2$ for square and small triangle), we still expect that the found parameter values will generalize well. This is due to two reasons:

1. We strongly limited the number of parameters that are trained and they relate directly to physical phenomena.
2. The mapping between parameters and final positions of the tangram pieces is complex and computed in a huge number $N \gg 1$ of consecutive steps. So the simulation of one trial can also be interpreted as N consecutive one-step mappings from one world state to the next. To come up with the correct simulation result, all of these mappings have to be rather accurate as well. This reduces the danger of over-fitting.

4.4 Optimization Algorithms

We combined the simplex algorithm [9] and differential evolution [10] to search for optimal parameter values. While the simplex algorithm only performs a single simulation per step, the differential evolution algorithm does one simulation for individual in the current population. With a population of 30 individuals, the differential evolution took about 50 times longer to converge than the simplex algorithm. However, it is also less prone to get stuck in local optima than the simplex algorithm. To combine the strength of both algorithms, we ran 100 iterations with the differential evolution first and then took the resulting parameter vectors as initial conditions for further search using the simplex algorithm.

5 Results

We have chosen two different settings for the training. In the first setting, the shape factor and friction parameters were optimized separately for each of the tangram shapes. In the second setting, only the shape factor was optimized separately for each shape, while all shapes had to share the same friction values which is more realistic. The obtained parameter values and corresponding error values are summarized in Table 2.

Table 2. Optimization Results

Shape	Friction T./Ground	Friction T./Finger	Shape Factor	$E_{D_{\text{train}}}$	$E_{D_{\text{test}}}$
Square	0.1246	0.7372	0.5274	0.62 cm	0.97 cm
Parall.	0.1055	0.4403	0.7178	0.59 cm	1.07 cm
Tri. (S)	0.1320	0.7346	0.4615	0.37 cm	0.77 cm
Tri. (L)	0.5951	0.7143	0.5658	0.49 cm	0.69 cm
All	0.3065	0.7495	see bel.	1.25 cm	1.42 cm

The optimized shape factors for the combined training of all shapes are (0.6279, 0.7131, 0.46, 0.5597) for the square, the parallelogram, the small triangle and the large triangle, respectively. When optimizing the three parameters individually for all shapes, the predicted final corner positions are only about 0.5 cm off the real positions in the training data. For testing data, the errors are about double that size, which shows that, although experiencing some over-fitting, the optimized parameters still generalize well over new pushing actions. The over-fitting is most likely due to the very small training set. In the chosen distribution of the data into training and testing set, all test trials are situated 'between' the training trials, and the optimized parameter vector successfully interpolates between them. In order to explore the ability to extrapolate we additionally trained the parameters for the large triangle with a different data distribution. We used the $x_0 = 18$ cm trial for training and the $x_0 = 21$ cm trial for testing, so extrapolation is necessary to achieve low test errors. In fact, the resulting errors are as low as the ones in the interpolation setting.

In the second optimization setting where the recorded training data of all four tangram pieces is used, the difference between training and testing error becomes much smaller. Except for the tangram/ground friction in second setting, the optimal parameter values did not resemble the empirically determined ones, that are shown in Table 1. The reason for this is most likely the imperfect modeling of the real interaction in the physics world, which mainly stems from the simplifications of the physical laws used in the simulations.

We further compared the prediction errors of the trained parameters $\boldsymbol{\pi}_{\text{opt}}$ with those of the default parameter set $\boldsymbol{\pi}_0$ on the complete data set D. The errors are listed in Table 3, together with an upper limit E_{base} which is the average distance the tangram corners moved in the real world and would be obtained when simply using their initial positions as predictions. As a measure of how reproducible the trials were in the real world, which is a lower limit for the prediction error, we list the standard deviation σ_{real} of tangram final positions in trials with identical start conditions, averaged over all start conditions.

Table 3. Comparison of Results

Tangram Shape	E_{base}	$E_D(\boldsymbol{\pi}_0)$	$E_D(\boldsymbol{\pi}_{\text{opt}})$	σ_{real}
Square	12.55 cm	3.28 cm	0.74 cm	0.46 cm
Parallelogram	8.39 cm	1.69 cm	0.82 cm	0.23 cm
Triangle (Small)	8.5 cm	0.99 cm	0.51 cm	0.26 cm
Triangle (Large)	17.36 cm	2.73 cm	0.57 cm	0.31 cm
All shapes	12.35 cm	2.23 cm	1.32 cm	0.33 cm

Fig. 5 shows the exact final positions predicted by the Bullet engine together with the actual final positions reached using the robot arm for the large triangular object. For the other shapes, the simulations provide equally good predictions of the places to which the tangram pieces were pushed.

6 Conclusions and Future Work

Our general objective is to explore the use of physics engines in robot's physical reasoning systems. We showed that accurate prediction of real world interactions is possible with a current physics engine for the case of flat objects that are pushed across a table. By adapting the simulation parameters, the predictions errors were reduced to less than half of the errors obtained with the default parameters. Apparently, the engine does a better job in predicting the objects' motions than humans are capable of, which would have to be evaluated in an additional study.

One interesting next step is to apply the physics engine to more complex scenarios and to adopt its parameters continuously. Enabling physics engines to handle incomplete data about a scene or making more of their inner workings available to learning processes also seems promising. We believe that a successful approach to physical reasoning in robotics will eventually have to combine the strengths of both qualitative simulations systems and quantitative simulation systems.

References

1. Baillargeon, R.: How do infants learn about the physical world? Current Directions in Psychological Science, Cambridge University Press **3** (1994) 133–140
2. Forbus, K.D.: Qualitative process theory. Artificial intelligence **24** (1984) 85168
3. Kuipers, B.: Qualitative simulation. Artificial intelligence **29** (1986) 289338
4. Berleant, D., Kuipers, B.J.: Qualitative and quantitative simulation: bridging the gap. Artificial Intelligence **95** (1997) 215255

5. Lipson, H., Pollack, J.: Evolving physical creatures. In: Artificial Life VII: Proceedings of the seventh international conference on artificial life. MIT Press. (2000) 282–287
6. Sucan, I., Kavraki, L.: Kinodynamic Motion Planning by Interior-Exterior Cell Exploration. In: Intl. Workshop on the Algorithmic Foundations of Robotics, Guanajuato, Mexico. (2008)
7. Pasula, H., Zettlemoyer, L., Kaelbling, L.: Learning symbolic models of stochastic domains. Journal of Artificial Intelligence Research **29** (2007) 309–352
8. Boeing, A., Braeunl, T.: Evaluation of real-time physics simulation systems. In: GRAPHITE, Perth, Western Australia. (2007)
9. Nelder, J., Mead, R.: A simplex method for function minimization. The computer journal **7** (1965) 308
10. Storn, R., Price, K.: Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. International Computer Science Institute – publications (1995)

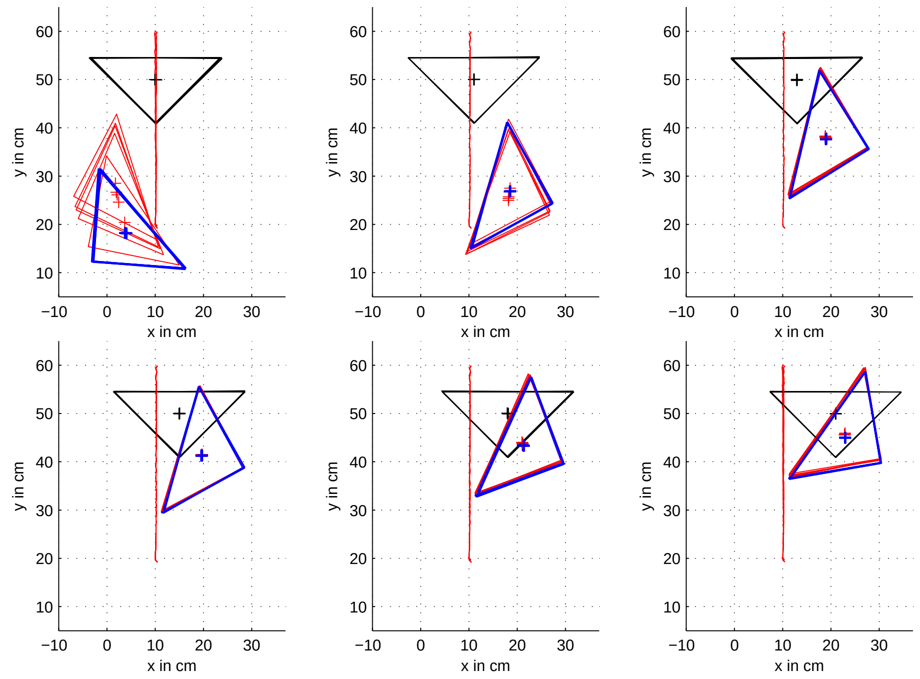


Fig. 5. The six figures show the recorded pushing trials of the large triangle-shaped tangram piece placed at different initial positions $x_0 \in \{10\,\text{cm}, 11\,\text{cm}, 13\,\text{cm}, 15\,\text{cm}, 18\,\text{cm}, 21\,\text{cm}\}$. The initial positions are printed in black, while the final positions and the trajectories of the finger are in red. The bold blue triangles are the final positions predicted by the Bullet engine. In each figure, five trials with identical start conditions are plotted.