Simulated Data Generation Through Algorithmic Force Coefficient Estimation for AI-Based Robotic Projectile Launch Modeling

Sajiv Shah⋆
Saratoga High School
Saratoga, California, USA
sajiv.shah@gmail.com

Ayaan Haque⋆
Saratoga High School
Saratoga, California, USA
ayaanzhaque@gmail.com

Fei Liu
University of California San Diego
San Diego, California, USA
f4liu@ucsd.edu

***Abstract*—Modeling of non-rigid object launching and manipulation is complex considering the wide range of dynamics affecting trajectory, many of which may be unknown. Using physics models can be inaccurate because they cannot account for unknown factors and the effects of the deformation of the object as it is launched; moreover, deriving force coefficients for these models is not possible without extensive experimental testing. Recently, advancements in data-powered artificial intelligence methods have allowed learnable models and systems to emerge. It is desirable to train a model for launch prediction on a robot, as deep neural networks can account for immeasurable dynamics. However, the inability to collect large amounts of experimental data decreases performance of deep neural networks. Through estimating force coefficients, the accepted physics models can be leveraged to produce adequate supplemental data to artificially increase the size of the training set, yielding improved neural networks. In this paper, we introduce a new framework for algorithmic estimation of force coefficients for non-rigid object launching, which can be generalized to other domains, in order to generate large datasets. We implement a novel training algorithm and objective for our deep neural network to accurately model launch trajectory of non-rigid objects and predict whether they will hit a series of targets. Our experimental results demonstrate the effectiveness of using simulated data from force coefficient estimation and shows the importance of simulated data for training an effective neural network.** [1]

***Index Terms*—Deep Neural Networks, Non-Rigid Objects, Robotic Launchers, Coefficient Estimation, Simulated Data, Launch Modeling**

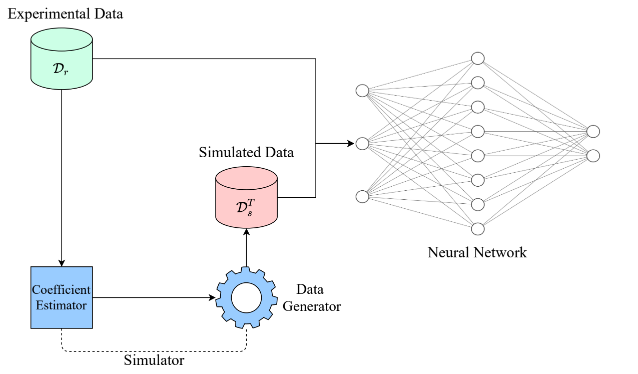


Fig. 1. Our general pipeline: Simulator is composed of algorithmic force coefficient estimation using experimental data and data generator which uses the estimated coefficients. Neural network is subsequently trained on experimental data as well as the simulated data.

I. Introduction

Real-world non-rigid object manipulation and launching is difficult to model, but required for advanced robots to achieve task flexibility and obtain a human-like skill-set [1]. This is because there are external factors and dynamics that govern the motion of an object which are unaccounted for in elementary physics models. Objects have an increasing amount of stress relaxation over time [2] as well as varying stiffness under a multitude of non-constant conditions such as temperature or humidity [3]. This creates a high amount of uncertainty as each object of a certain model will have varying conditions.

Furthermore, known dynamical factors that are non-constant have coefficient values that cannot be directly measured [4]. Specifically for aerodynamics, to accurately determine lift and drag force coefficients, experiments using expensive tools and sensors such as wind tunnels must be used [5]. Prototype or small-scale models are required to perform these tests, meaning the general shape and material of the object must be predetermined, making designing mechanisms that interact with objects of unknown or changing shape difficult. Without high-speed, accurate feedback from precise sensors and controllers, robots cannot imitate models well. In launching systems, friction from the mechanism and vibrations from actuation due to uneven mass distribution, in addition to unideal control loops results in a variance from targeted parameters to actual parameters. The combination of noninclusive modeling and an imperfect mechanical system produces a great variance in experimental and theoretical results.

To address control problems in robotics and train robots for unknown environments, Reinforcement Learning has been researched [6]–[8] (see [9], [10] for comprehensive reviews). Robots are able to learn novel behaviors through constrained trial and error behaviors, removing the need for a human

⋆Authors contributed equally, order arbitrarily assigned
[1]Code will be made available after review

operator to pre-program motion and test accuracy. Reinforcement learning on simulated models of physical mechanisms often garners poor results, meaning the physical system must be used for training, which poses a safety problem. Despite advances in safe learning systems [11] (see [12], [13] for comprehensive reviews), reinforcement learning in robotics is generally unsafe as training without human operator supervision can result in hardware failure. In a launching setting, a high-cost architecture of sensing, loading, and configuring is required. To determine whether the launched object has successfully reached its target, sensors must be used, and to reload the objects into the launcher, an automated system must be implemented. Lastly, to configure the launcher in various positions, additional mechanisms are necessary. It is of interest to model non-rigid object launching without the use of high-cost and dangerous experimental systems such as reinforcement learning methods.

Due to the drawbacks of reinforcement learning, other data-powered deep learning approaches have been investigated for various physics modeling tasks [14]–[16] (see [17] for comprehensive review). Deep learning is heavily reliant on large pools of data, and this restriction has grown in recent years as neural network architectures have become deeper and more complex. This is a significant challenge, as collecting data for neural-network based approaches is difficult. Similar to reinforcement learning, experimental setups must be built in order to acquire large amounts of samples and tests [18]. However, this may be infeasible because of cost or time limitations, begging the need for more approaches which can train learnable networks from limited experimental data. For robotic applications, collecting and producing large datasets would render prediction models obsolete as programmatically referencing the collected data would produce far more accurate results than generating predictions based off that data. Simply put, in the large dataset, any test case with its outcome could be found, meaning a simple searching algorithm would be sufficient.

Experimental data has often been paired with simulated/generated data to provide more data for deep learning models. In order to produce additional data, generative models have been explored. These approaches have been used in other fields, such as computer vision [19] and natural language processing [20]. For robotic launch modeling, few methods using generative models for simulated data have been proposed [21]–[23]. However, strong generative models require access to large datasets in order to accurately learn and reproduce the representations of the data. Therefore, a simpler approach would be to use the accepted physics equations and models to produce simulated data. While using these models alone would not be sufficient or accurate, using them to produce supplemental data could be a satisfactory solution. In literature, many methods have used regression or other statistical methods to produce data [24], [25], but none have been applied as supplements to neural networks. Ideally, having entirely experimental datasets would be more ideal, but in the case of neural network modeling, we hypothesize the benefits of larger datasets outweighs the inconsistencies of simulated data. However, as stated, these models rely on coefficients which are specific to each non-rigid object, meaning they need to be derived or estimated.

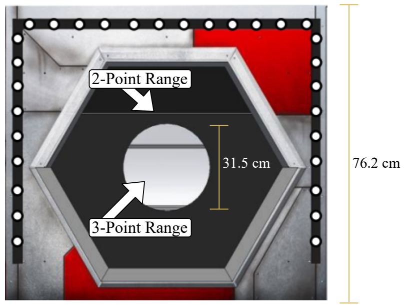


Fig. 2. A frontfacing view of the scoring target. The inner circle of the scoring target is the 3-point region, and the dark shaded hexagon is the 2-point region. The diameter of the inner 3-point region is $\approx$ 31.5 cm and the height of the entire target is $\approx$ 76.2 cm.

To address the aforementioned problems, we propose a novel method of modeling robotic launching of non-rigid objects using neural-networks which are trained with supplemental simulated data, generated from algorithmic force coefficient estimation. In section II, we define our problem, preliminaries, and notation. In section III, we describe in detail each component of our method. In section IV we present the results of our method in comparison to baselines. Finally, in section V we discuss and draw conclusions from our work.

II. Problem Definition and Notation

We assume access to data $p(X_r, Y_r)$ of measured launcher configurations X_r and launch outcomes Y_r, as well as data $p(X_s, Y_s)$ of simulated data with launcher configurations X_s and labels Y_s. X_r and Y_r together are used to form experimental dataset $\mathcal{D}_r$. X_s and Y_s are together used to form simulated dataset $\mathcal{D}_s$.

We aim to train our neural network to predict an outcome based on a certain set of launch configurations X_r, each of which has 3 features: distance to target (meters), motor speed ratio from range 0-1, and launcher angle from horizontal (degrees). More conventionally, X is the input for the neural network, which in this case are the launcher configurations, and Y is the label for the input, and in this case it is the outcome of the launch. The launch outcome (Y_r and Y_s) contains two binary results: whether the launch has hit a 2-point target and a 3-point target. Technically, there are only 3 labels of two dimensional vectors: both targets missed (0,0), the two-point target was hit while the 3-point target was missed (1,0), and both targets where hit (1,1); this is because the 3-point target is entirely contained within the 2-point target (see Figure 2).

Our objective, given by the FIRST Robotics Challenge and shown in Figure 2, is to score as many foam, 7-inch diameter balls as possible into the 3 point target from different launching zones, which range from 3 to 30 feet away from the base of the target. The center of the target is mounted $\approx$ 8 feet high.

Regarding object dynamics, the three known forces (Figure 3) that govern the acceleration of the ball once it is launched from the robot at a given angle and velocity are the force of

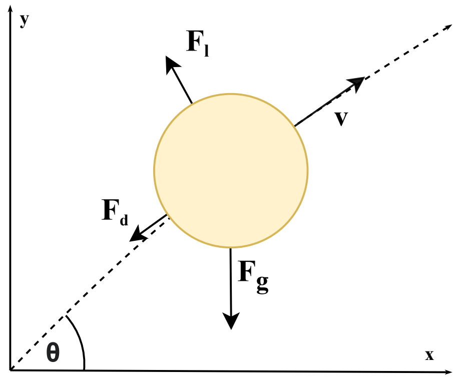


Fig. 3. The known forces acting on the ball during its trajectory are shown. The 7-inch diameter foam ball has an unknown and varying stiffness. Moreover, this stiffness decreases over time as the ball is used, and therefore varies between balls. As it is launched, the ball is deformed, but once again expands as it exits the launcher. The ball has a mass of $\approx$ 0.14 kg.

gravity (F_g), drag (F_d), and lift (F_l). Gravity always points downwards and is treated as constant as our launch height is relatively small. The drag force due to air friction will act parallel and in the opposite direction of the velocity of the ball at a given point in time, and is given by Eq. 2. Lift is a result of the magnus force, which is caused by the ball rotating and creating a variance in the fluid pressure around it. It acts perpendicular to the direction of the velocity of the ball and has an upward component, and is given by Eq. 3. Both lift and drag forces act at different angles on the ball during its travel.

$$F_g = mg \tag{1}$$

$$F_l = C_l\frac{4}{3}(4\pi^2 r^3 s\rho v) \tag{2}$$

$$F_d = \frac{C_d}{2}Av^2\rho \tag{3}$$

In Eq. 1-3 we define the variables r, s, v, and A, where r is the radius of the ball in meters, s is the spin in rotations per second, v is the velocity of the ball in m/s, and A is the cross-sectional area of the ball in m^2.

The lift force coefficient C_l in Eq. 2 is dependent on the interaction between the object material and the fluid it travels in [26], [27], and the drag coefficient C_d in Eq. 3 is determined by a multitude of factors, including the shape of the object [28]. Therefore, each of these coefficients are specific to a given object. These quantities must be estimated or measured to accurately model the trajectory of an object.

III. Methods

A. Algorithmic Force Coefficient Estimation

To estimate the force coefficients based on experiment data, we designed a simulator based on the known forces to output a trajectory given a specific configuration. A configuration consists of three parameters: a distance (d), a motor speed ratio (ω), and angle (θ). The distance in meters is simply the horizontal distance between the exit point of the object from the robot to the base of the target. The motor speed is given as a percentage of the motor free speed, which is in rotations per minute. The angle of the launcher is the angle between the line connecting the center of the two launcher flywheels and the horizontal and is given in degrees. We develop a simulator that determines an output given these configurations.

Using the equations of the defined forces (Eq. 1,2,3), the individual accelerations from each force can be derived using the mass of the object.

As the object travels, the net force redirects the velocity of the ball in different angles. The aerodynamic forces, lift and drag, act in various directions according to the ball velocity. When considering the 2-D motion of the object as a combination of the horizontal velocity vector and the vertical velocity vector, we can determine the ball velocity angle, and use the previously defined equations to calculate the net acceleration in each direction due to the three known forces (Figure 3). We then compute the change in velocity using the following equations:

$$\frac{dv_x}{dt} = -(a_l\sin(\theta) + a_d\cos(\theta)) \tag{4}$$

$$\frac{dv_y}{dt} = a_l\cos(\theta) - a_d\sin(\theta) - a_g \tag{5}$$

We represent the angle of travel (θ) as a function of the initial launch angle (θ_0) and the change in angle, which is a trigonometric function of the horizontal and vertical velocities at a time t. The total displacement of the object in the x (horizontal) and y (vertical) directions are integrals of the velocities with respect to time added with the initial launch velocity.

$$\theta = \int_0^t \arctan(\frac{v_y}{v_x})dt + \theta_0 \tag{6}$$

$$v_x = \int_0^t (\frac{dv_x}{dt})dt + v_0\cos(\theta_0) \tag{7}$$

$$v_y = \int_0^t (\frac{dv_y}{dt})dt + v_0\sin(\theta_0) \tag{8}$$

$$x = \int_0^t (v_x)dt \tag{9}$$

$$y = \int_0^t (v_y)dt \tag{10}$$

The initial angle is a given parameter in X_r, but the initial velocity is not. Given the motor speed ratio, we can calculate the angular speed of each flywheel. Assuming that the frictional force is high enough to accelerate the object to the flywheel speed in a small time interval, the object will have a velocity given by:

$$v_0 = \frac{\omega_u r_u + \omega_l r_l}{2} \tag{11}$$

where ω_l is the angular speed of the lower flywheel, r_l is the radius of the lower wheel, ω_u is the angular speed of the upper flywheel, and r_u is the radius of the upper flywheel.

To calculate the lift force, a value for the spin of the object in rotations per second must be calculated. We assume this value to be constant throughout the trajectory, and it can be calculated by:

$$s = \frac{\omega_b}{2\pi} = \frac{\omega_l r_l - \omega_u r_u}{2\pi r_b} \tag{12}$$

where ω_b is the angular velocity of the object in radians per second.

By iterating through the set of differential equations, we can calculate the displacement in the x and y direction after a given amount of time. Knowing that the scoring target is a distance (d) away and that the center of the scoring area is at a height (h_{target}), we can determine whether the object has passed through the 2-point or 3-point target to create an outcome using the modeled trajectory for a set of configurations and pair of coefficients.

$$(d, h_{target} \pm 0.07) \rightarrow Y_r = (1,1) \tag{13}$$

$$(d, h_{target} \pm 0.35) \rightarrow Y_r = (1,0) \tag{14}$$

We aim to use this trajectory simulator to estimate the coefficients C_l and C_d by comparing the outcomes of the generated trajectories from configurations X_r to their outcomes Y_r. To estimate the coefficients that best fit the dataset, we simulate the trajectory for each configuration of X_r using multiple coefficient pairs. We evaluate 1000^2 combinations, with each coefficient being in a range of (0.000,5.000) inclusive with step-size 0.005.

For a given initial launch angle, launch velocity, and coefficient pair, the simulator is able to generate the object trajectory and determine whether it passed through the target. We can simply iterate through multiple coefficient pairs to determine which one models the trajectories of $\mathcal{D}_r$ best. Because the outcome dataset (Y_r) of $\mathcal{D}_r$ represents a range of points the object possibly passed through (shown in Eq. 13, 14), a large number of coefficient pairs will model the trajectory of a single configuration from X_r. We calculate the distance of the object from the center of the scoring target using the simulator and label this value the deviation. To select a single coefficient pair to use for data generation, we evaluate the accuracies in the 3-point and 2-point regions, and the mean and median deviations.

Our algorithmic force coefficient estimation method is generalizable to other objects interacting with aerodynamic forces. Figure 4 exhibits the effect of different aerodynamic force coefficients on the object trajectories, as for a given subset of configurations, the outcomes vary based on the assigned coefficient pairs.

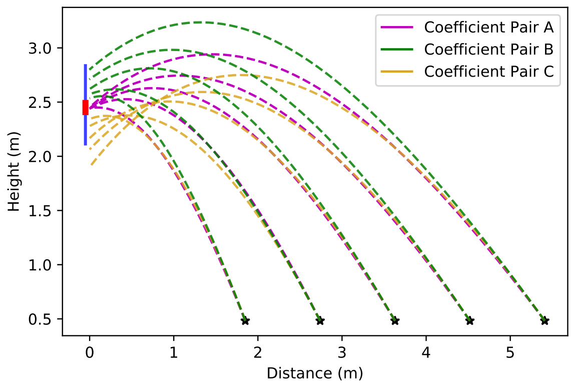


Fig. 4. Example of trajectory generation using algorithmic simulator. Vertical blue line represents the scoring range of the 2-point zone, and the vertical red line represents the scoring range of the 3-point zones. The black stars show the location/distance of the robot when the object is launched. We selected 5 configurations from X_r that each had a corresponding outcome of (1,1). As seen, coefficient pair A perfectly models the subset of data, while coefficient pairs B and C are inaccurate.

B. Simulated Data Generation

After coefficients are estimated, we replace the coefficient values in equations 2, 3. We then iterate through a range of values for each launcher configuration. For motor speed, we iterate through range (0.00,1.00) inclusive with step-size of 0.01; for launcher angle, we iterate through range (15.00,75.00) inclusive with step-size of 1.00; finally, for distance to target, we iterate through range (1.00,16.00) inclusive with step-size of 0.20. The range and step-size values for the motor speed and launcher angle were determined by the controllability of those systems and their operating range. The range and step-size for the distance was determined by the minimum and maximum distances we could accurately launch the object from as well as the variance in horizontal distance the robot could attain while still scoring launched objects. This results in a total of 375,000 simulated configurations, X_s. We then use the synthetic configurations and use our simulator of physics models and produce labels Y_s for each of the 375,000 configurations.

From this large pool of imbalanced samples, we randomly sample pairs of X_s and Y_s from each class in order to produce an approximately balanced dataset $\mathcal{D}_s$. We perform experimentation to determine the optimal amount of simulated data to use in conjunction with the experimental data. However, these configurations were not sampled in a balanced manner based on the configuration values, meaning certain configurations may have very similar values to one another. If visualized in a 3-dimensional feature space, there may be certain clusters that appear evident or a wide variety in their distribution. Sampling in such a method provides a more difficult challenge for the neural network to extract and learn the correct representations of the data, which could improve performance. Moreover, any set of configurations that already existed in the experimental dataset were removed and replaced. While this data has inadequate accuracy as the coefficients do not account for unknown forces on and deformations of the launched object, it allows us to further train a data-reliant neural network for improved predictions.

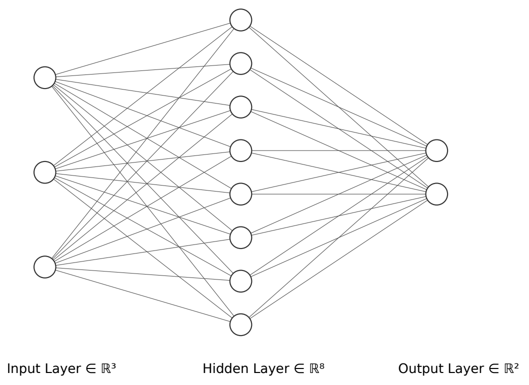


Fig. 5. Neural network schematic for robotic launch modeling.

C. Neural Network

We leverage deep neural networks in order to predict of a launch based on the given launcher configurations. In our method, we use a simple 3-layer forward-feed neural network (Figure 5) for classification ($\mathcal{F}$ with parameters θ). The input layer has 3 nodes, one for each feature in the launcher configuration (motor speed, launcher angle, distance to target). The feature-space is then up-sampled to 8 nodes (experimentally tuned), and finally down-sampled to a 2-dimensional vector, representing binary predictions for the 2-point and 3-point scores. We use the sigmoid activation function for each layer, as per [29]. This neural network can be substituted with a significantly improved and deeper network, however in order to investigate the effectiveness of just our contributions, primarily the simulated data from force coefficient estimation, we use a simple network.

The use of a neural-network allows for the consideration of inaccuracies in our mechanism and model. A major dynamical factor that is unaccounted for in our algorithmic simulator is the deformation of the non-rigid object as it is launched. In order to accelerate the object to its launch velocity, the flywheels must rely on the frictional force (F_f) between their surface and that of the object to pull the object. Because $F_f \propto F_N$, a higher normal force will ensure that in a small time interval, the object can match the flywheel speed. To balance F_N, the object applies a force F_s, which is proportional to the amount that the non-rigid object is compressed. As the object is compressed, its mass-distribution changes, as well as its cross-sectional area, which directly effects two major forces in our simulation algorithm, F_l and F_d.

We additionally cannot model energy transfer. To accelerate (through spin) and launch the object, kinetic energy must be transferred to the object. Evident from motor data, the motor power in the short time interval in which the object is accelerated is too small to solely provide enough energy, meaning energy must be transferred to the object by the flywheels. As the angular speeds of the flywheels decrease, the launch velocity of the object decreases as well, changing the trajectory. Since the moment of inertia of the object is non-constant due to object compression, and because our system cannot measure the changes in flywheel speed, it is not possible to model this energy transfer through traditional physics models.

A neural-network is specifically meant to account for the inaccuracies of the physics models. The hidden layers allow for the network to model non-linear systems and account for the non-linearities [30]. The collected data is from the domain in which we deploy our AI system. If the data is collected from the target domain, it will account for the variances which the physics models cannot. Therefore, if the model is trained on this data which already accounts for the unpredictable variances from the specific test domain, it will be able to accurately learn the representations of these variances. So during evaluation and deployment, the model will have knowledge about these variances which humans and physics models cannot have, allowing it to still be accurate in the field.

D. Loss Objective

Our neural network is trained and optimized on the following loss function:

$$L(X_r, Y_r, X_s, Y_s) = L_R(\mathcal{F}_\theta(X_r), Y_r) + \lambda L_S(\mathcal{F}_\theta(X_s), X_s) \quad (15)$$

where X_r and Y_r are experimental feature sets and labels, X_s and Y_s are simulated feature and label sets, $\mathcal{F}_\theta$ is the model, L_R is the true loss term, L_S is the simulated loss term, and λ is the Lagrange multiplier for the simulated loss. Both L_R and L_S use conventional cross-entropy loss and are calculated identically. We specifically do not use physics-based loss functions such as [31], [32] because of the explained unknown factors which physics models cannot account for. Moreover, in order to optimize the network based on physics models, the acceleration values need to be predicted by the model and therefore the labels must be experimental acceleration values, which are intractable to acquire nor our main objective.

The λ term is significant in an approach which uses experimental and simulated data. Since $\mathcal{D}_s$ is significantly larger than $\mathcal{D}_r$, the loss function must be appropriately tuned in order to optimize the model to perform effectively in the target domain. Such auxiliary loss weight/Lagrange multiplier terms have seen success in various methods [33], primarily in semi-supervised or multi-task modes. The multiplier is intentionally low in order to force simulated data loss term to be an auxiliary penalty, as the model should primarily learn from the experimental data, as it is closest and most similar to the target domain.

E. Robotic Launcher Details

"Guppy", seen in Figure 6, is a 24-inch long, 24-inch wide robot that weighs $\approx$ 81 lbs. It has the ability to navigate itself, acquire balls from the ground, store up to 3 in its hopper, and then launch them rapidly and consecutively. It can launch balls up to 60 feet. The robot is powered by an on-board 12v, 18 amp-hour lead acid battery. To tele-operate the robot, human operators use joysticks connected to a laptop, which is connected to a WiFi router on the robot.

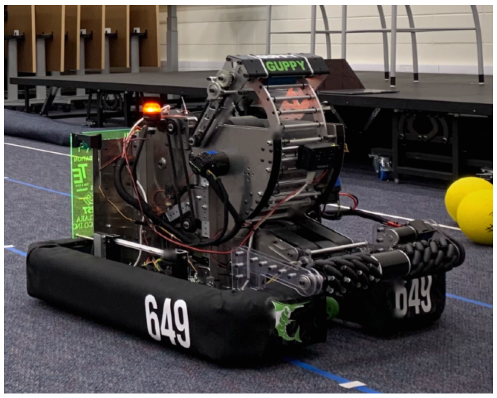


Fig. 6. CAD render (left) and image of assembled robot "Guppy" (right).

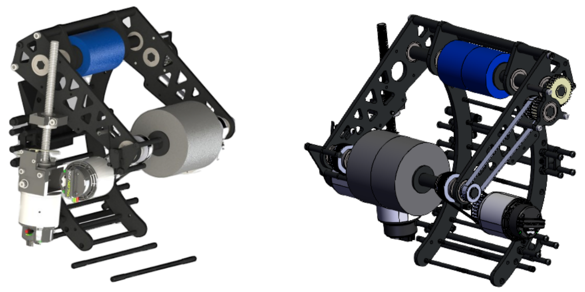


Fig. 7. CAD render of shooter mechanism.

The robotic launcher, seen in Figure 7, is a double-flywheel mechanism consisting of a 4-inch diameter lower (gray) flywheel and a 2.75-inch upper (blue) flywheel. Both flywheels are operated by the same motors, and the top flywheel is geared to spin at $\frac{9}{16}$th the speed of the bottom flywheel. We use two brushless DC motors which transfer power to the flywheel axles using a timing belt transmission. The top flywheel and bottom flywheel have a smaller distance between them than the diameter of the ball, as ball compression is required to consistently accelerate the ball to launch speed. The flywheels rely on the frictional force between their surfaces and the ball to move the ball. Adjusting the position of the top flywheel along an arc that is concentric to the bottom flywheel changes the launch angle of the ball. To do so, we implemented an acme screw mechanism. By turning a screw inside a nut fixed to the rotating hood, we convert rotational motion to linear power. The motor is mounted to this rotating screw, and is on a pivot in order to keep the screw and nut parallel as the linear distance between the motor and nut changes.

We use PID control loops to set the hood position and flywheel velocities, and use a vision system to determine the horizontal distance from the robot to the target.

IV. Experimental Results

A. Data

We conducted trials to produce an experimental dataset $\mathcal{D}_r$ which consists of a launcher configuration with 3 parameters and one outcome. The motor speed is set using a PID loop, and the data value added to X_r is calculated by a built-in quadrature encoder. The launcher angle is measured by an absolute encoder and is cross-checked by using the motor's built-in encoder for the angle change, and limit switches for the initial angle. The X-distance is measured using an on-board vision system, which uses lights and retro-reflective tape to determine the horizontal distance to the target with an accuracy of ± 0.05 meters.

With a given set of launcher configurations, the launcher has a repeatability R_2 and R_3, which represent the consistency of scoring in the 2 and 3 point target. Because the 2-point target encompasses the 3-point target, a score in the 3-point target is counted towards R_2. If $R_3 < 80\%$, for a configuration expected to outcome a score of 3, and $R_2 > 90\%$ it is assigned an outcome score of 2. For any case, if $R_2 < 75\%$ the configuration is considered illegitimate and is removed.

$\mathcal{D}_r$ has exactly 100 configurations with an even distribution of data representing our 3 possible outcomes. Only 2 data instances had to be manually rejected due to mechanism flaws. For train and test splits, we used 20% of the dataset as an external test set to evaluate the model performance. This test set contains no simulated data. As stated, the simulated dataset has 900 evenly balanced samples.

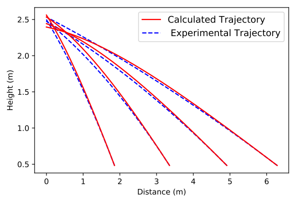
Collecting over 100 samples was intractable in this study due to limited lab time; moreover, we contend that having more samples would defeat the purpose of using a neural network to predict new test cases, as in our objective, with so many samples, a reference sheet could be produced and used in a program without the need for deep learnable models. As an additional evaluation metric, we sampled 50 of the generated X_s and use the configurations on our robot to determine the experimental outcome on our field. We used this to form an additional dataset $\mathcal{D}_s^T$. Acquiring the experimental labels for simulated inputs X_s allow us to not only evaluate the predictions from the simulator based on the estimated coefficients but also the predictions of the neural network.

B. Implementation Details

Our neural network was implemented in Keras and Python (code will be released after review) and trained with a NVIDIA K80 GPU. The network is optimized using the Adam Optimizer with a learning rate of 0.0001. The loss function used is cross-entropy. λ is experimentally tuned to 0.01 (see Section IV-D). We trained our network on varying amounts of $\mathcal{D}_s$: 500, 600, 750, 900, 1000, 1250. We trained on a batch size of 10. The epoch stepsize was dictated by $\mathcal{D}_s$, so $\mathcal{D}_r$ was repeatedly sampled until each batch in the simulated dataset was used.

C. Force Coefficient Estimation

While the training set contains 80 samples of trajectories, the most precise measurements are those with the outcome (1,1) as the ball must follow a strict path to score in the middle of the target. Therefore, the initial metric we used to select a coefficient pair (C_l, C_d) was the accuracy of the 3-point trajectories, as 3-point scores are desirable. After testing each trajectory using the 1000^2 combinations of coefficients, we



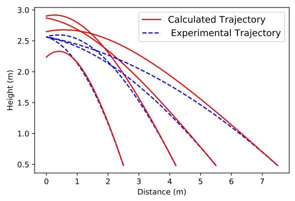


Fig. 8. Comparison of experimentally measured trajectories and trajectories generated by the simulator using estimated force coefficients (left). Inaccuracy of generated trajectories using estimated force coefficients in comparison to experimental trajectories on the test set (right).

TABLE I
ACCURACY OF FORCE COEFFICIENT ESTIMATION SIMULATOR ON THE TRAINING SET AND THE TEST SET.

$\mathcal{D}_r$ Train/Test	Metrics			
	2pt Acc (%)	3pt Acc (%)	Mean Dev	Median Dev
Training	100	92.31	0.051	0.058
Testing	92.86	**85.71**	0.283	0.124

selected 28 coefficient pairs to evaluate, as they all achieved identical accuracies for both outcomes: 3-point accuracy rate of $\approx 92.31\%$ and a 2-point accuracy rate of 100%.

The left plot in Figure 8 displays the close resemblance of a calculated trajectory with estimated force coefficients to a subset of X_r with $Y_r = (1, 1)$. To determine the coefficient pair that fit our model best, we examined the mean and median deviation from the center of the target for each trajectory, and discovered that both were at a minimum when $\boldsymbol{C_l = 0.06}$ and $\boldsymbol{C_d = 0.91}$.

To evaluate the performance of the simulator with our final estimated coefficients, we generated the trajectories for the 20 test samples in X_r and compared the outcomes from the simulator (Y_s) to the actual outcomes (Y_r). The right plot of Figure 8 demonstrates the inaccuracy when only using the simulator on the external test set due to the inability to generalize to a separate dataset. This prompts the need for a learnable method which can generalize to a test domain.

In Table I, we show the performance of the simulator on the training and test set and prove that is it able to very accurately predict the force coefficients to match the training set, which is optimal for the neural network, but performs poorly on the test set, which is the motivation to use the neural network. As stated, the simulator is meant to replicate the data distribution of the training set in order to generate new data that is indistinguishable from the training set, which would help the neural network performance the most. Therefore, because the simulator is very effective in accurately generating and modeling the training data, it can be a proper supplement for training a neural network.

D. Neural Network Performance

We evaluated the performance of our neural network trained on varying levels of simulated data.

TABLE II
ACCURACY AND CLASSWISE F1-SCORES FOR NEURAL NETWORK ON TESTSET WITH VARYING QUANITIES OF SIMULATED DATA.

$\mathcal{D}_s$ Size	Metrics (Combined evaluation of $\mathcal{D}_r$ and $\mathcal{D}_s^T$)		
	Overall Acc (%)	F1-3pt (%)	F1-2pt (%)
0	44.29	4.17	66.66
500	78.57	50.00	87.50
600	87.14	70.83	91.67
750	91.43	79.16	95.83
900	**98.56**	**95.83**	**100.00**
1000	95.71	91.67	95.83
1250	92.85	87.50	91.67

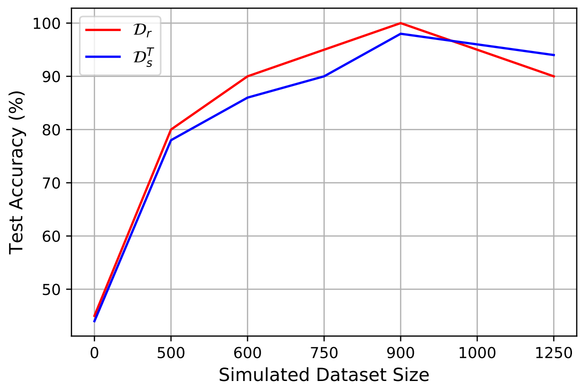


Fig. 9. Comparison of network accuracy on $\mathcal{D}_r$ and $\mathcal{D}_s^T$.

Table II displays the performance of the neural network at varying amounts of simulated data. The network is evaluated on 70 total samples, the 20 samples from the experimental test set as well as 50 samples from the simulated dataset $\mathcal{D}_s^T$ which are evaluated with the actual robot in the environment to receive a real prediction. The results show that with any amount of simulated data, the network has stronger performance than when the network is trained on no simulated data. However, after 900 samples, the performance begins to decrease, indicating overfitting to the simulated data which decreases the performance on the experimental test set. The results confirm the importance of using simulated data and shows statistically significant increases (p-value $\leq$ 0.05) in performance. Moreover, the accuracy of the neural network is much higher than just the force coefficient estimation. The classwise scores are consistent with the overall accuracy, as the model is able to accurately predict the outcome of the launch given the configurations, with a tendency to predict 2-pt scores more accurately than the other classes.

Figure 9 shows the accuracy of the neural network on $\mathcal{D}_r$ and $\mathcal{D}_s^T$. The model is more accurate on the data in the same domain as the test set, which is $\mathcal{D}_r$. However, it performs quite effectively on configurations which are out of domain but verified in the same manner. This evaluation further demonstrates the effectiveness of our network and force

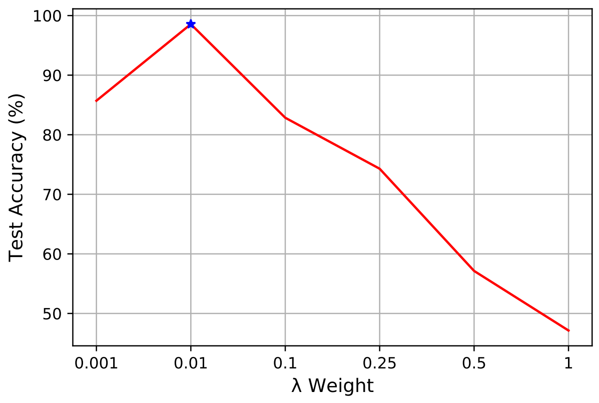


Fig. 10. Tuning of Lagrange multiplier λ in order to determine optimal value.

coefficient estimation process in accurately modeling launched object trajectories.

Figure 10 shows the importance of accurately tuning λ in order to train the neural network effectively. Since the $\mathcal{D}_s$ highly outnumbers $\mathcal{D}_r$ in number of samples, weighing the simulated data too high results in poor performance in the target domain of experimental data. As expected, once the Lagrange multiplier approaches the equilibrium weight of 1.0, the performance sharply decreases due to overfitting to the simulated data. After tuning, the optimal value for the weight was 0.01 (starred).

V. Conclusion

In this paper, we present a new method of modeling the trajectory of non-rigid objects launched by robotic systems using force coefficient estimation and deep neural networks. We introduce an algorithmic process for estimating drag and lift coefficients using experimental launcher data and a physics simulator. We importantly use this algorithmic simulator to generate additional data to train a data-starved neural network. We leverage deep neural networks trained simultaneously on simulated and experimental data to predict and model the outcomes of the launcher based on the input configurations, allowing for real-world modeling of non-rigid object launching which can account for external factors on object flight. Our experimental evaluations prove the effectiveness of our simulator as well as the importance of using additional generated data to train the neural network. We show that the neural network with additional data is able to accurately model and predict the outcome of both experimental and simulated configurations.

We envision such a system to be deployed on a tele-operated mobile robot with automated variable-launchers such that live-measured launcher configurations can be fed to the model to predict the outcome of launching. The configurations could be sent to a operator dashboard where the web-deployed network could efficiently return predictions on whether to fire the launcher. Our future work will investigate new neural network architectures as well as reverse-engineering neural networks to derive force coefficients.

Acknowledgements

We would like to thank FRC team #649 and their sponsors, mentors, and donors for providing resources which enabled us to conduct this research. Without their support experimenting our work would not have been possible. We would especially like to thank the senior mentors for their support and encouragement in exploring science, technology, engineering, and math.

References

[1] F. Nadon, A. J. Valencia, and P. Payeur, “Multi-modal sensing and robotic manipulation of non-rigid objects: A survey,” *Robotics*, vol. 7, no. 4, 2018. [Online]. Available: https://www.mdpi.com/2218-6581/7/4/74

[2] S. A. Ashter, *Thermoforming of Single and Multilayer Laminates: Plastic Films Technologies, Testing, and Applications*. William Andrew, 2013.

[3] E. Linul, L. Marsavina, T. Voiconi, and T. Sadowski, “Study of factors influencing the mechanical properties of polyurethane foams under dynamic compression,” in *Journal of Physics: Conference Series*, vol. 451, no. 1. IOP Publishing, 2013, p. 012002.

[4] M. T. Mason and K. Lynch, “Dynamic manipulation,” in *Proceedings of (IROS) IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, July 1993, pp. 152 – 159.

[5] W. Wu, Q. Sun, S. Luo, M. Sun, Z. Chen, and H. Sun, “Accurate calculation of aerodynamic coefficients of parafoil airdrop system based on computational fluid dynamic,” *International Journal of Advanced Robotic Systems*, vol. 15, no. 2, p. 1729881418766190, 2018.

[6] J. Gruenstein, T. Chen, N. Doshi, and P. Agrawal, “Residual model learning for microrobot control,” *arXiv preprint arXiv:2104.00631*, 2021.

[7] M. Al-Gabalawy, “A hybrid mpc for constrained deep reinforcement learning applied for planar robotic arm,” *ISA Transactions*, 2021.

[8] M. Wang, B. Zeng, and Q. Wang, “Research on motion planning based on flocking control and reinforcement learning for multi-robot systems,” *Machines*, vol. 9, no. 4, p. 77, 2021.

[9] J. Kober, J. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *The International Journal of Robotics Research*, vol. 32, pp. 1238–1274, 09 2013.

[10] P. Kormushev, S. Calinon, and D. G. Caldwell, “Reinforcement learning in robotics: Applications and real-world challenges,” *Robotics*, vol. 2, no. 3, pp. 122–148, 2013. [Online]. Available: https://www.mdpi.com/2218-6581/2/3/122

[11] S. Shalev-Shwartz, S. Shammah, and A. Shashua, “Safe, multi-agent, reinforcement learning for autonomous driving,” *arXiv preprint arXiv:1610.03295*, 2016.

[12] J. Garcıa and F. Fernández, “A comprehensive survey on safe reinforcement learning,” *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[13] F. Berkenkamp, “Safe exploration in reinforcement learning: Theory and applications in robotics,” Ph.D. dissertation, ETH Zurich, 2019.

[14] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, “Tossingbot: Learning to throw arbitrary objects with residual physics,” *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307–1319, 2020.

[15] C. Kong and S. Lucey, “Deep non-rigid structure from motion,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[16] L.-h. Hou and H.-j. Liu, “An end-to-end lstm-mdn network for projectile trajectory prediction,” in *International Conference on Intelligent Science and Big Data Engineering*. Springer, 2019, pp. 114–125.

[17] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, “Integrating physics-based modeling with machine learning: A survey,” *arXiv preprint arXiv:2003.04919*, 2020.

[18] M. Lutter, C. Ritter, and J. Peters, “Deep lagrangian networks: Using physics as model prior for deep learning,” in *International Conference on Learning Representations (ICLR 2019)*. OpenReview. net, 2019.

[19] A. Haque, “Ec-gan: Low-sample classification using semi-supervised algorithms and gans,” 2020.

[20] R. Puri, R. Spring, M. Shoeybi, M. Patwary, and B. Catanzaro, “Training question answering models from synthetic data,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 5811–5826.

[21] L. de Oliveira, M. Paganini, and B. Nachman, "Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis," *Computing and Software for Big Science*, vol. 1, no. 1, Sep 2017. [Online]. Available: http://dx.doi.org/10.1007/s41781-017-0004-6

[22] A. B. Farimani, J. Gomes, and V. S. Pande, "Deep learning the physics of transport phenomena," *arXiv preprint arXiv:1709.02432*, 2017.

[23] Q. Zheng, L. Zeng, and G. E. Karniadakis, "Physics-informed semantic inpainting: Application to geostatistical modeling," *Journal of Computational Physics*, vol. 419, p. 109676, 2020.

[24] R. Swischuk, L. Mainini, B. Peherstorfer, and K. Willcox, "Projection-based model reduction: Formulations for physics-based machine learning," *Computers & Fluids*, vol. 179, pp. 704–717, 2019.

[25] O. Sigaud, C. Salaün, and V. Padois, "On-line regression algorithms for learning mechanical models of robots: a survey," *Robotics and Autonomous Systems*, vol. 59, no. 12, pp. 1115–1129, 2011.

[26] J. D. Anderson Jr, *Fundamentals of aerodynamics*. Tata McGraw-Hill Education, 2010.

[27] J. Katz and A. Plotkin, *Low-speed aerodynamics*. Cambridge university press, 2001, vol. 13.

[28] S. Lee, Y. Park, and J. Kim, "An evaluation of factors influencing drag coefficient in double-deck tunnels by cfd simulations using factorial design method," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 180, pp. 156–167, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167610518302307

[29] R. Jennings, "Development of physics based machine learning algorithms," *Bucknell University*, Mar 2019. [Online]. Available: https://digitalcommons.bucknell.edu/masters-thesis/221

[30] R. Grzeszczuk, D. Terzopoulos, and G. Hinton, "Neuroanimator: Fast neural network emulation and control of physics-based models," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 1998, pp. 9–20.

[31] R. Cang, H. Li, H. Yao, Y. Jiao, and Y. Ren, "Improving direct physical properties prediction of heterogeneous materials from imaging data via convolutional neural network and a morphology-aware generative model," *Computational Materials Science*, vol. 150, pp. 212–221, 2018.

[32] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler, "Deep fluids: A generative network for parameterized fluid simulations," in *Computer Graphics Forum*, vol. 38, no. 2. Wiley Online Library, 2019, pp. 59–70.

[33] Y. Grandvalet, Y. Bengio *et al.*, "Semi-supervised learning by entropy minimization." in *CAP*, 2005, pp. 281–296.