Model Identification and Control of a Low-cost Mobile Robot with Omnidirectional Wheels using Differentiable Physics

Edgar Granados, Abdeslam Boularias, Kostas Bekris and Mridul Aanjaneya

***Abstract*— We present a new data-driven technique for predicting the motion of a low-cost omnidirectional mobile robot under the influence of motor torques and friction forces. Our method utilizes a novel differentiable physics engine for analytically computing the gradient of the deviation between predicted motion trajectories and real-world trajectories. This allows to automatically learn and fine-tune the unknown friction coefficients on-the-fly, by minimizing a carefully designed loss function using gradient descent. Experiments show that the predicted trajectories are in excellent agreement with their real-world counterparts. Our proposed approach is computationally superior to existing black-box optimization methods, requiring very few real-world samples for accurate trajectory prediction compared to physics-agnostic techniques, such as neural networks. Experiments also demonstrate that the proposed method allows the robot to quickly adapt to changes in the terrain. Our proposed approach combines the data-efficiency of classical analytical models that are derived from first principles, with the flexibility of data-driven methods, which makes it appropriate for low-cost mobile robots. Project website: `https://go.rutgers.edu/mqxn2x6h`**

I. Introduction

With the availability of affordable micro-controllers such as Arduino [1] and Beaglebone Black [2], light-weight high-performance computing platforms such as Intel's *Next Unit of Computing* (NUC), there is interest in low-cost robots. Motivated by this, the long-term goal of the present work is to develop affordable mobile robots that can be easily assembled using off-the-shelf components. Affordable robots can be used for exploration and scene understanding in unstructured environments. They can also be augmented with end effectors for object manipulation. The ultimate goal is to remove the economic barrier of entry that has limited research in robotics. To that end, we present mobile robot (see Figure 1), for exploration and scene understanding.

High-end robots can be easily controlled by software tools provided by manufacturers. Physical properties, such as inertial and frictional parameters, are precisely measured, eliminating need for further calibrations. Robots assembled and fabricated in-house are significantly more difficult to control due to uncertainties in manufacturing. Due to this, hand-crafting precise and shared models for these robots is challenging. For example, the wheels of the robot in Figure 1 cannot be precisely modeled manually because of the complex structure and unknown material properties. Moreover, friction between the wheels and terrain vary largely when the robot is deployed on unknown non-uniform terrain.

This work was supported in part by an NSF HDR TRIPODS award 1934924, NSF awards IIS 1734492, IIS 1846043 and 2132972. The authors are with the Department of Computer Science, Rutgers University, NJ 08901, USA. Email: `eg585, ab1544, kb572, ma635@cs.rutgers.edu`.

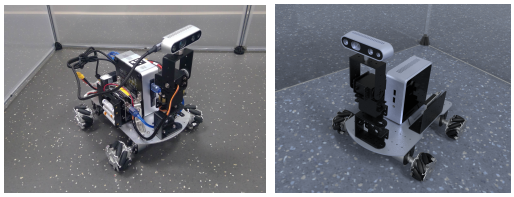

Fig. 1. (Left) A low-cost mobile robot with omnidirectional wheels (cost $1200). A differentiable physics engine automatically infers unknown friction coefficients on-the-fly, and a control algorithm allows for autonomously driving the robot along pre-specified curves. (Right) A virtual 3D model of the robot using Autodesk's Fusion 360 suite.

An alternative actively promoted by several research groups [3]–[8] is to first train robots in simulation, and subsequently transfer the learned control policies to the real world – the *sim2real* process. The virtual simulation environment provides a safe environment where (potentially) an unlimited number of experiments can be conducted while learning effective control policies. To this end, we built a virtual 3D model of the robot (Figure 1 (right)). However, to the best of our knowledge, none of the available rigid body physics engines are able to produce stable trajectories for this model that could reliably reproduce the acquired ground-truth. While a deeper investigation of this requires further research, we believe this arises due to the complex nature of the omnidirectional wheels[1]. Thus, one of the main contributions of our work is a differentiable physics engine that can be effectively used for learning unknown friction coefficients between the wheels and the ground that allow the simulation to match ground-truth training data.

We propose a hybrid data-driven approach combining the versatility of machine learning with the data-efficiency of physics-based models. The key contribution is a *self-tuning differentiable physics simulator* of the robot. Taking as input the pose, the generalized velocity and a sequence of controls, returning a predicted trajectory. Ground-truth trajectories, collected by executing controls on the real robot, are recorded and systematically compared to the predicted ones. The difference between predicted and ground-truth trajectories is used to identify the unknown coefficients of friction between each wheel and the present terrain (Figure 2). Since the identification process must happen on the fly, black-box optimization tools cannot be effectively used. Instead, we show how to analytically compute the derivatives of the reality gap with respect to each unknown coefficient of friction. The proposed method is shown to be more efficient

[1] https://pybullet.org/Bullet/phpBB3/viewtopic.php?t=12966&p=42863

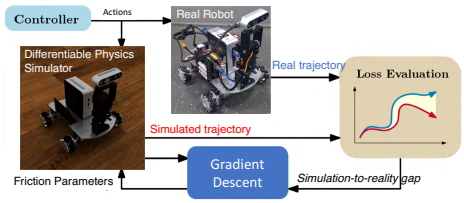

Fig. 2. Framework for identifying unknown friction coefficients of the robot wheels by minimizing the difference between simulated and real trajectories using differentiable physics simulation.

computationally than black-box methods, and more accurate than a neural network trained with the same small amount of data to predict velocities from control signals.

Dynamic (kinetic) friction coefficients identified using analytical gradients *vary* as a function of different motor controls. Frictions coefficients are identified per wheel due to the use of a low-cost mecanum wheel . Instead of identifying individual friction values, we identify a *function* that maps the desired wheel velocity to a friction coefficient. First, we collect few trajectories with various velocities. Then, the friction values are automatically identified per trajectory using the differentiable physics engine. Finally, the collected data is generalized by training a small neural network that captures the dependence between the friction coefficients and desired wheel velocities. Once identified, the dynamic friction functions are used within the simulator to compute control signals (Figure 3) allowing the robot to drive along pre-specified curves. We refer the reader to our supplemental video for all our experiments reported in Section VII.

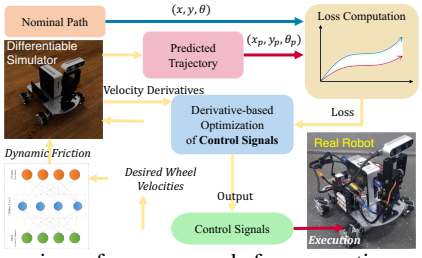

Fig. 3. Overview of our approach for computing control signals that drive the robot along pre-specified curves.

II. Related Work

The problem of *learning dynamic and kinematic models of skid-steered robots* has been explored in the past. Vehicle model identification by integrated prediction error minimization was proposed in [9]. A similar approach was used in [10] for calibrating a kinematic wheel-ground contact model for slip prediction. Our work builds on the model for feedback control of an omnidirectional wheeled mobile robot [11].

A learning-based *Model Predictive Control* (MPC) is used in [12] to control a robot in challenging outdoor environments. It uses a priori model and a learned disturbance model, which is modeled as Gaussian Process and learned from local data. A MPC technique is used for autonomous racing in simulation in [13], which consists on decomposing the dynamics as the sum of a known deterministic function and noisy residual. Our approach shares some similarities with [14], wherein the dynamics equations of motion are used to analytically compute the mass of a manipulator. Unlike in the present work that considers a full body simulation, [15] considered only a rigid wheel on deformable terrains. A dynamic model is also presented in [16] for omnidirectional wheeled mobile robots, including surface slip. However, the friction coefficients in [16] were experimentally measured.

Classical system identification builds a dynamics model by minimizing the difference between the model's output and real-world response data for the same input [17], [18]. A similar approach is the use of *natively differentiable physics engines*. In [3] the Theano framework is used to develop one for differentiate model and control parameters. First principles from numerical integration of rigid and flexible body dynamics were used to design a differentiable physics engine for tensegrity robots [19], shown to be more data-efficient than prior works [20]. A combination of a learned and a differentiable simulator was used to predict action effects on planar objects in [21]. Differentiable physics simulations were also used for manipulation planning and tool use in [5]. In [22], a differentiable contact model was used to allow for optimization of several tasks. The present work is another step toward the adoption of *self-tuning* differentiable physics engines as both a data-efficient and time-efficient tool for learning and control in robotics.

III. Robot Design

The design of our mobile robot consists of the hardware assembly, and the software drivers to provide the controls for the motors. These two components are described below.

A. Hardware Assembly

The robot is built from the hardware components shown in Figure 4(left): (a) a central chassis with 4 Mecanum omnidirectional wheels that are run by $12V$ DC motors, (b) 2 rail mount brackets for Arduino, (c) 2 Arduino, (d) Arduino shield for DC motors, (e) servo motor shield for Arduino, (f) 2 MG996R servos, (g) 2 2DOF servo mount brackets, (h) Intel Realsense D435, (i) Intel NUC5i7RYH, and (j) 2 1300mAh, $11.1V$ DC batteries. One battery powers the DC motors and the other the onboard electronics. The servo motors are powered with a separate $6V$ DC source.

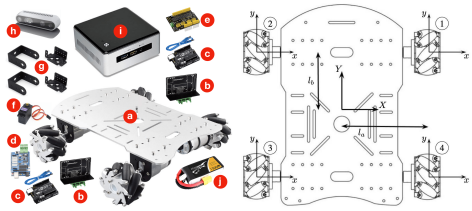

Fig. 4. (Left) Hardware components for the mobile robot. All hardware and electronic designs were developed in-house. (Right) Top view of the mobile robot.

B. Control Software

Arduinos (driving the motors) are connected to the CPU via USB. To send commands, the Serial protocol [23] is used. Each message is encoded as a byte, the limited buffer size is accounted for by using "acknowledge" messages. We have generalized the implementation in [23] to support two Arduinos, two servo motors, and the differential drive mechanism for the wheels.

IV. Analytical Model

Consider a simplified cylindrical model for the wheel. Let J be the scalar component of its 3×3 (diagonal) inertia tensor matrix about the rotational axis of the motor shaft (discarding the components in the plane orthogonal to this axis). Making the common assumption that the weight remains unchanged, balanced uniformly by all wheels, the equation of motion is:

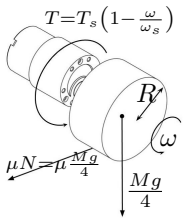

Fig. 5. Motor torque and friction on a simplified wheel.

$$J\frac{d\omega}{dt} = T_s\left(1-\frac{\omega}{\omega_s}\right) - \mu\frac{Mg}{4}R$$

where T_s is the motor stall torque, ω the wheel's angular velocity, ω_s the desired angular velocity, μ the coefficient of friction, g the gravity, R the wheel radius, and M the robot's mass. The first term on the right hand side was derived in [24]. The equation is integrated to obtain:

$$\omega = \omega_s\left(1-\frac{\mu MgR}{4T_s}\right)\left(1-\exp\left(-\frac{T_s t}{J\omega_s}\right)\right) \quad (1)$$

Since we are interested in large time spans for mobile robot navigation, only the steady state terms in equation (1) are important. Thus, we can discard the transient terms:

$$\omega = \omega_s\left(1-\frac{\mu MgR}{4T_s}\right) \quad (2)$$

which is an expression for the wheel's angular velocity as a function of ground friction forces. Assume the friction coefficient μ_j for each wheel j is different. The chassis of our mobile robot is similar to that of the *Uranus* robot [11] which linear and angular velocities using the wheel angular velocities as input is:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = \underbrace{\frac{R}{4l_{ab}}\begin{bmatrix} -l_{ab} & l_{ab} & -l_{ab} & l_{ab} \\ l_{ab} & l_{ab} & l_{ab} & l_{ab} \\ 1 & -1 & -1 & 1 \end{bmatrix}}_{B}\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \quad (3)$$

where $l_{ab} = l_a + l_b$ (see Figure 4(right)). v_x, v_y are the linear velocities along the X and Y axes, and ω_z is the angular velocity about the Z axis. Equations (2) and (3) define a model for the robot with the effects of friction.

V. Loss Function

Let (x^i, y^i, θ^i) be the generalized position of the mobile robot at time t^i, and $(v_x^i, v_y^i, \omega_z^i)$ be its generalized velocity. Then, its predicted state at time t^{i+1} can be computed as:

$$\begin{bmatrix} x^{i+1} \\ y^{i+1} \\ \theta^{i+1} \end{bmatrix} = \begin{bmatrix} x^i \\ y^i \\ \theta^i \end{bmatrix} + \Delta t\begin{bmatrix} \cos\theta^i & -\sin\theta^i & 0 \\ \sin\theta^i & \cos\theta^i & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} v_x^i \\ v_y^i \\ \omega_z^i \end{bmatrix} \quad (4)$$

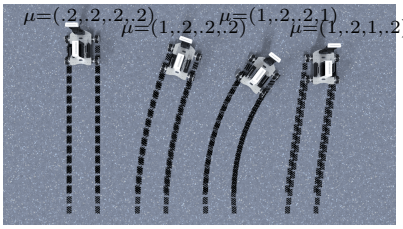

Fig. 6. Trajectories generated using the wheel friction coefficients (top of Eq (2)). Note that, ceteris paribus, with higher friction the angular velocity of the wheel should decrease. Thus, the motion model matches the intuition of how the robot moves under the influence of friction forces.

where $\Delta t = t^{i+1} - t^i$. The *simulation-reality gap* is computed as follows:

$$L_{\text{gt}} = \sum_{k=1}^{N}\left(|x^k - x_{\text{gt}}^k|^2 + |y^k - y_{\text{gt}}^k|^2\right)^{1/2}, \quad (5)$$

where $(x_{\text{gt}}^k, y_{\text{gt}}^k)$ are the ground-truth position values at time-step k. (x^k, y^k) are the predicted positions using Equation (4) and the same sequence of controls provided to the real robot. The video is recorded using the overhead camera at 60fps, whereas the PWM signal is sent to the motor at a 3-4$\times$ higher frequency (mostly determined by the network bandwidth). Thus, there are several simulation time steps between two consecutive ground-truth position values.

In practice, the predicted state often "lags behind" the ground truth values (Figure 7(Left)), even when the robot is *exactly* following the overall path, leading to a high loss value. Thus, we fit a spline curve to the ground truth data and compute another loss function that uses the point $(x_{\text{sp}}^k, y_{\text{sp}}^k)$ *closest* to this curve from the predicted state at time-step k:

$$L_{\text{sp}} = \sum_{k=1}^{N}\left(|x^k - x_{\text{sp}}^k|^2 + |y^k - y_{\text{sp}}^k|^2\right)^{1/2} \quad (6)$$

Note that the loss L_{sp} alone is not sufficient either, as it does not penalize the simulated robot for not moving at all from its starting position. Thus, a weighted linear combination of L_{gt} and L_{sp} is proposed as the actual loss function:

$$L = w_1 \cdot L_{\text{sp}} + w_2 \cdot L_{\text{gt}} \quad (7)$$

Where the weights are empirically determined as $w_1 = 0.8, w_2 = 0.2$ and using the parameters $M = 4, r = 0.03$, and $T_s = 0.6$ for equation (2) . The pseudocode for loss computation is shown in Algorithm 1, vectors shown in bold. B is defined in Equation 3, and μ is a vector of the friction coefficients of the different wheels. Line 4 uses component-wise vector multiplication. T the total time-steps, and t^i the time the control was applied to the

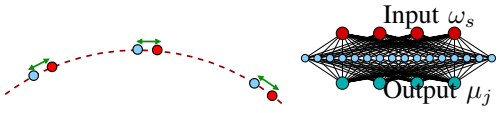

Fig. 7. (Left) Predicted positions (blue) can often "lag behind" the ground-truth (red), even when closely following the correct trajectory, leading to a high loss value (green). (Right) A small neural network learns the map between controls ω_s and learned friction coefficients μ_j.

motor at time-step i. Since there can be multiple simulation time steps between two consecutive ground-truth observations, the function `IsGroundTruthSample` checks if a ground truth position exists at time t^i, if so, the function `GroundTruthIndex` returns its index.

Algorithm 1 `LossComputation`$(B, \boldsymbol{\mu})$

Initialize $l \leftarrow 0$, $p \leftarrow (x^0_{\text{gt}}, y^0_{\text{gt}}, \theta^0_{\text{gt}})$
for $i = 1 \dots T$ **do**
  Compute $\Delta t \leftarrow t^{i+1} - t^i$
  Compute $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega}_s \left(\mathbf{1} - \boldsymbol{\mu} Mgr/4T_s\right)$
  Compute $(v_x, v_y, \omega_z) \leftarrow B\boldsymbol{\omega}$
  if `IsGroundTruthSample`(i) **then**
    $k \leftarrow$ `GroundTruthIndex`(i)
    $(\Delta p_x, \Delta p_y) \leftarrow (p_0 - x^k_{\text{gt}}, p_1 - y^k_{\text{gt}})$
    $(\Delta q_x, \Delta q_y) \leftarrow (p_0 - x^k_{\text{sp}}, p_1 - y^k_{\text{sp}})$
    $l \mathrel{+}= w_1 \left(\Delta q_x^2 + \Delta q_y^2\right)^{1/2} + w_2 \left(\Delta p_x^2 + \Delta p_y^2\right)^{1/2}$
  end if
  $p \mathrel{+}= \Delta t \left(c_{p_2} v_x - s_{p_2} v_y, s_{p_2} v_x + c_{p_2} v_y, \omega_z\right)$
end for
return l

VI. Differentiable Physics

To minimize the loss function in equation (7) with respect to unknown friction coefficients μ_j for all wheels, we derive analytical expressions for the *gradient* of the loss with respect to μ_j:

$$\frac{\partial L}{\partial \mu_j} = w_1 \cdot \frac{\partial L_{\text{sp}}}{\partial \mu_j} + w_2 \cdot \frac{\partial L_{\text{gt}}}{\partial \mu_j} \tag{8}$$

Let $(\Delta x^k_{\text{gt}}, \Delta y^k_{\text{gt}}) = (x^k - x^k_{\text{gt}}, y^k - y^k_{\text{gt}})$ and d^k denote the length $\left(|\Delta x^k|^2 + |\Delta y^k|^2\right)^{1/2}$. Then the second term in equation (8) can be expanded using the chain rule as follows:

$$\frac{\partial L_{\text{gt}}}{\partial \mu_j} = \sum_{k=1}^{N} \frac{1}{d^k} \left(\Delta x^k_{\text{gt}} \cdot \frac{\partial x^k}{\partial \mu_j} + \Delta y^k_{\text{gt}} \cdot \frac{\partial y^k}{\partial \mu_j}\right) \tag{9}$$

The derivatives in equation (9) can be computed using equations (3), (4) at the higher frequency of the PWM signal sent to the motors, used for predicting the next state, as:

$$\frac{\partial x^{i+1}}{\partial \mu_j} = \frac{\partial x^i}{\partial \mu_j} + \Delta t c_{\theta^i} b_{1j} \frac{\partial \omega^i}{\partial \mu_j} - \Delta t s_{\theta^i} b_{2j} \frac{\partial \omega^i}{\partial \mu_j} - \Delta t \left(v^i_x s_{\theta^i} + v^i_y c_{\theta^i}\right) \frac{\partial \theta^i}{\partial \mu_j} \tag{10}$$

$$\frac{\partial y^{i+1}}{\partial \mu_j} = \frac{\partial y^i}{\partial \mu_j} + \Delta t s_{\theta^i} b_{1j} \frac{\partial \omega^i}{\partial \mu_j} + \Delta t c_{\theta^i} b_{2j} \frac{\partial \omega^i}{\partial \mu_j} + \Delta t \left(v^i_x c_{\theta^i} - v^i_y s_{\theta^i}\right) \frac{\partial \theta^i}{\partial \mu_j} \tag{11}$$

$$\frac{\partial \theta^{i+1}}{\partial \mu_j} = \frac{\partial \theta^i}{\partial \mu_j} + \Delta t b_{3j} \frac{\partial \omega^i}{\partial \mu_j} \tag{12}$$

where b_{rs} is the (r, s) entry in the matrix B, (see equation 3), $c_\theta = \cos(\theta)$ and similarly for sin. The derivative $\partial \omega^i / \partial \mu_j$ is computed using equation (2). The expression for $\partial L_{\text{sp}} / \partial \mu_j$ in equation (8) can be derived similarly. Note that, the closest point $(x^k_{\text{sp}}, y^k_{\text{sp}})$ on the spline curve is a *function* of the point (x^k, y^k). However, we have empirically found that estimating its derivative can be ignored when computing the term $\partial L_{\text{sp}} / \partial \mu_j$, for $j \in \{1 \dots 4\}$.

Algorithm 2 `GradientComputation`$(B, \boldsymbol{\mu})$

Initialize $\boldsymbol{g} \leftarrow [0]_{4\times 1}$, $J \leftarrow [0]_{3\times 4}$, $p \leftarrow (x^0_{\text{gt}}, y^0_{\text{gt}}, \theta^0_{\text{gt}})$
for $i = 1 \dots T$ **do**
  Compute $\Delta t \leftarrow t^{i+1} - t^i$
  Compute $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega}_s \left(1 - \boldsymbol{\mu} Mgr/4T_s\right)$
  Compute $(v_x, v_y, \omega_z) \leftarrow B\boldsymbol{\omega}$
  Compute $\boldsymbol{d\omega} \leftarrow -\boldsymbol{\omega}_s Mgr/4T_s$
  for $j = 0 \dots 3$ **do**
    $J_{0j} \mathrel{+}= \Delta t \{\boldsymbol{d\omega}_j (cB_{0j} - sB_{1j}) - (v_x s + v_y c) J_{2j}\}$
    $J_{1j} \mathrel{+}= \Delta t \{\boldsymbol{d\omega}_j (sB_{0j} + cB_{1j}) + (v_x c + v_y s) J_{2j}\}$
    $J_{2j} \mathrel{+}= \Delta t B_{2j} \boldsymbol{d\omega}_j$
  end for
  if `IsGroundTruthSample`(i) **then**
    $k \leftarrow$ `GroundTruthIndex`(i)
    $(\Delta p_x, \Delta p_y) \leftarrow (p_0 - x^k_{\text{gt}}, p_1 - y^k_{\text{gt}})$
    $(\Delta q_x, \Delta q_y) \leftarrow (p_0 - x^k_{\text{sp}}, p_1 - y^k_{\text{sp}})$
    $d_{\text{gt}} \leftarrow \left(\Delta p_x^2 + \Delta p_y^2\right)^{1/2}$
    $d_{\text{sp}} \leftarrow \left(\Delta q_x^2 + \Delta q_y^2\right)^{1/2}$
    $\boldsymbol{g} \mathrel{+}= J[\{0,1\}, :]^T \left(\frac{w_1}{d_{\text{sp}}} \begin{bmatrix} \Delta q_x \\ \Delta q_y \end{bmatrix} + \frac{w_2}{d_{\text{gt}}} \begin{bmatrix} \Delta p_x \\ \Delta p_y \end{bmatrix}\right)$
  end if
  $p \mathrel{+}= \Delta t \left(c_{p_2} v_x - s_{p_2} v_y, s_{p_2} v_x + c_{p_2} v_y, \omega_z\right)$
end for
return $\boldsymbol{g}$

As shown in Section VII, using the gradients derived in Section VI to minimize the loss function in equation (7) yields friction parameters that give good agreements with the observed trajectory. However, the computed friction parameters can differ in values for two different trajectories (with different control signals). This implies that the friction μ_j for each wheel is not a constant, but a function $\mu_j(\omega_s)$ of the applied control signal. Thus, we first generate a sequence of trajectories with fixed control signals, and estimate friction parameters for each of them by separately minimizing the loss function using gradient descent. We then train a small neural network with 4 input nodes, 16 hidden nodes, and 4 output nodes. The input to the neural network are the applied control signals to the wheels, and the output are the friction parameters estimated via gradient descent using our differentiable physics engine (see Figure 7(Right)). As shown in Section VII, a sequence of only 8 input trajectories is enough to obtain reasonable predictions from the neural network, and allowed us to autonomously drive the mobile robot along an "eight curve" with high precision.

The pseudocode for computing the gradient of the loss function is in Algorithm 2. Matrix J stores the accumulated gradients for the generalized position. The notation $J[\{0,1\}, :]$ refers to the first two rows of J.

VII. Experimental Results

The robot was driven in a 3×5 m^2 area with an Apriltag [25] attached on top of it. An overhead camera is used to record the motion. We remotely controlled the robot

for 6-8 seconds and collected 8 different trajectories, with 2 samples for each trajectory, as shown in Figure 9 in red and blue. For simplicity, the applied controls were constant for the entirety of each trajectory, but different per trajectory.

A. Model Identification

We use the L-BFGS-B method to estimate the unknown friction coefficients, where the gradient is computed using equation (8), as shown in green in Figure 9. Additionally, we imposed the constraints that estimated friction values should lie in $[0, 2]$. Our chosen values for M, g, r, T_s ensure that $\omega_j \to 0$ as $\mu_j \to 2$ for $j \in \{1 \dots 4\}$. For comparison, we show the result of the Uranus model [11] (does not account for friction), and a neural network trained using the same small amount of data to directly predict velocities from applied control signals. For each control signal, we use the remaining 14 trajectories in our data set from Figure 9 that correspond to different control signals for training the neural network, and 2 trajectories corresponding to the current control signal for testing. We only show the best result for each control signal in Figure 9. To ensure that our minimization problem is well-defined in an unconstrained setting, we slightly modified equation (2) as follows:

$$\omega = \omega_s \left(1 - \frac{\sigma(\mu) M g R}{2 T_s}\right) \tag{13}$$

where $\sigma(\mu) \in [0, 1] \forall \mu \in (-\infty, \infty)$ is the *sigmoid* function. As shown in Figure 9, our model has the best agreement with the ground-truth values.

The total run-time and iteration counts for our method, Nelder-Mead, and CMA-ES [26], two derivative-free optimization methods, are shown in Figure 10(right). Our method requires very few iterations to converge, and is generally faster than both Nelder-Mead and CMA-ES. We did not show the trajectories predicted using CMA-ES and Nelder-Mead in Figure 9, as they both converge to the same answer as L-BFGS-B, just take longer. The top portion of Figure 10(right) shows the loss value with increasing iteration counts of L-BFGS-B. The use of accurate analytic gradients allows for rapid progress in the initial few iterations. The bottom portion of Figure 10(right) shows the effect on loss when the training data is reduced according to the percentage on the X-axis. As can be seen, our method converges to almost the final loss value with only 40% of the total data, making it *data-efficient*, and potentially applicable in real-time settings for dynamically detecting changes in the friction of the terrain.

B. Path Following

We also used our learned model to compute control signals, such that the robot could autonomously follow pre-specified curves within a given time budget T. Taking as input the total way-points n the robot should pass in a second, and discretize the given curve with nT way-points. We assume the controls are constant between consecutive way-points, which is reasonable provided the sampling density of the way points is high enough. To compute the control signals, we again use L-BFGS-B, but optimize for the control signals $\boldsymbol{\omega}_s$, instead of the friction coefficients $\boldsymbol{\mu}$, when minimizing equation (7). Apart from changing the primary variable from $\boldsymbol{\mu}$ to $\boldsymbol{\omega}_s$, the only other change required is to use the derivative of equation (2) with respect to $\boldsymbol{\omega}_s$.

Note that the function $\boldsymbol{\mu}(\boldsymbol{\omega}_s)$ is crucial for path following. Once the control signal $\boldsymbol{\omega}_s$ is fixed, equations (2) and (3) can be used to predict the generalized velocity, which can be integrated in time to predict the position and orientation of the mobile robot in the next time step.

Figure 11 illustrates our results when the specified path is a circle and a "figure 8". Shown are the reference path and the real robot trajectory after applying control signals that were computed using our method. Our differentiable framework is accurate enough that the robot follows the specified path closely. To test the robustness of our method, we parametrized the 8-curve such that the robot drives the right lobe *backwards*, and the left lobe forwards. The robot overshoots the specified path because our method only optimized for position constraints, not for velocity constraints, when computing the control signals.

C. Terrain Adaptation

To demonstrate the ability of our method to quickly adapt to changes in the terrain, we conducted experiments driving the robot in one author's home (due to the pandemic) using the trained model. As shown in Figure 12(left), the executed path deviates from the reference path. To correct for this, we repeat model identification on this recorded trajectory by minimizing equation (7) again (see Figure 12(middle)), add the computed friction values to our training set from Figure 9, and retrain the neural network for approximating the function $\boldsymbol{\mu}(\boldsymbol{\omega}_s)$. (Note that the recorded trajectory from Figure 12(left) does *not* have constant applied controls. However, using the timestamps of each control, and the robot's position, the recorded trajectory is *segmented* into curves with constant controls and minimized the loss equation on each one. This approach has the additional advantage of *single* executed path generating many new data points in the training set.) Figure 12(right) shows the executed path after applying control signals computed from the retrained model, which is much closer to the reference path. Although not tried in our experiments, it is conceivable that the slight deviation in Figure 12(right) can be further reduced by repeating the steps above for a few iterations.

D. Motion Planning

We also performed motion planning using the learned model. First, a set of 8 primitives were defined (front, back, right, left and diagonals). Then, ground-truth trajectories per primitive were collected in a high friction environment (figure 13 left and center), using the proposed method to learn a friction coefficients per primitive. The learned model is used by the AORRT [27] algorithm from the ML4KP library [28] to compute plans (ordered sequence of piecewise-constant controls) to drive the robot to the desired location. The computed plans were tested on an open-loop setting and the trajectories recorded for visualization. As a comparison, AORRT was also used to plan using primitives with zero

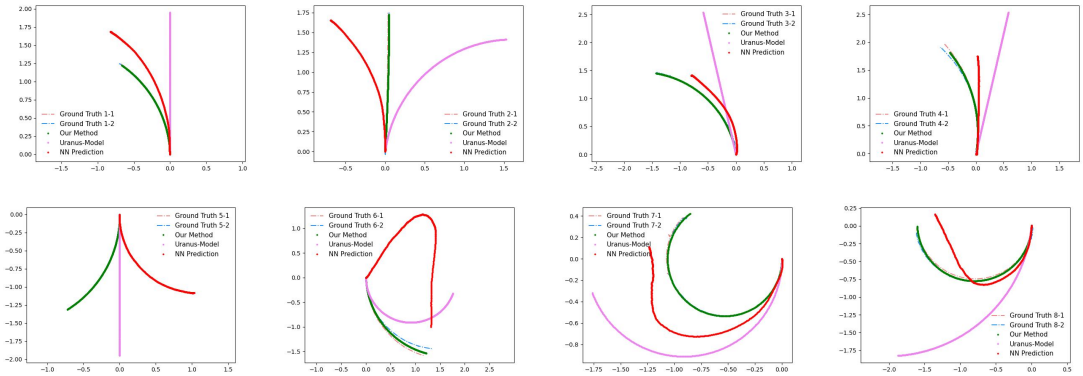

Fig. 9. Robot trajectories for 8 different motor control signals. Ground truth values estimated with Apriltags are shown in red and blue. Our method uses L-BFGS-B to estimate friction parameters, the result is shown in green. For comparison, also shown the result of the Uranus model [11] and a neural network. Note that ground truth overlaps with our method in most cases.

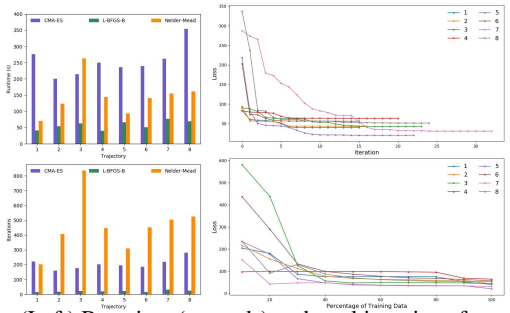

Fig. 10. (Left) Run-time (seconds) and total iterations for estimating friction coefficients for 8 trajectories using CMA-ES, L-BFGS-B with the gradient function, and Nelder-Mead method. All methods use the analytical model, but only L-BFGS-B makes use of the analytical gradients. (Right) Loss value vs iteration (top), making rapid progress in the initial iterations. Efficiency of our method with less training data (bottom), according to the percentage.

values for the friction coefficients. The resulting trajectories are shown in figure 13 (right). Note that the planner generates different plans for the zero friction values and the learned vectors due to: a) the predicted trajectories being different, and b) the planner being asymptotically-optimal. Note that the diagonal trajectories are shorter for the same execution time, therefore the planner prefers the other primitives for the learned system. This result in the robot being able to execute a plan that drives it towards the goal faster.

VIII. Conclusion and Future Work

We presented an analytical model that closely describes real-world recorded trajectories of a low-cost mobile robot designed in our lab. To estimate unknown friction coefficients for the robot wheels, we designed a differentiable physics engine, and showed that it is computationally efficient and more accurate than existing methods. To autonomously drive

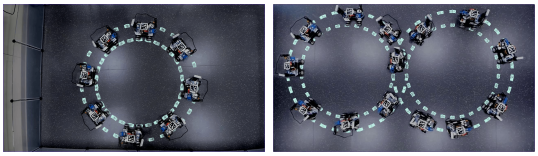

Fig. 11. Reference path highlighted as tire tracks (cyan) and snapshots of real-world recorded trajectory using control signals computed by our method are shown for a circular path (left) and a "figure 8" (right).

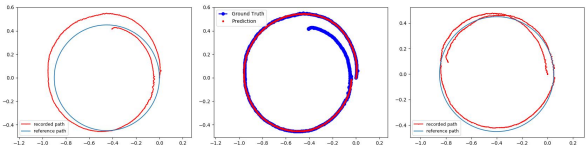

Fig. 12. (Left) Executed (red) and reference (blue) paths when driving the mobile robot in one author's home, using the model trained in the lab. The new terrain causes deviations from the reference path. (Middle) Recorded (blue) and predicted (red) paths after model identification on the previous path to get friction values that minimize equation (7). (Right) Executed path after retraining the model on the newly computed friction values and controls.

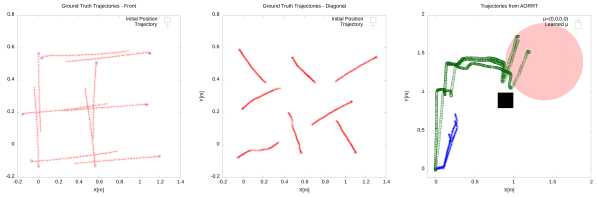

Fig. 13. Ground truth trajectories for front (left) and diagonal (center) primitives, executed for the same duration. Note the difference in length. (Right) Trajectories planned with AORRT: when using the learned friction coefficients (green) the robot is able to reach the goal vs not using the learned values (blue)

the robot along pre-specified paths, we also designed a hybrid approach that uses a neural network to approximate the function $\boldsymbol{\mu}(\boldsymbol{\omega}_s)$, where the training set is generated by our model identification pipeline. Our method combines the data-efficiency of differentiable physics engines with the flexibility of data-driven neural networks. Finally, we showed that our method can quickly adapt to changes in the terrain.

In the future, we would like to improve our analytical model to also account for control signals that are not powerful enough to induce rotation of the wheels. In such cases, the wheel is not completely static and can still turn during robot motion, due to inertia. This causes the robot to change orientations in a manner that cannot be predicted by our current model. Accounting for such control signals would allow for more versatile autonomous control of our robot. Additionally, improving the method's run-time is needed interleave execution and motion planning which would help keeping a small error rate.

REFERENCES

[1] M. Banzi, *Getting Started with Arduino*. Sebastopol, CA: Make Books - Imprint of: O'Reilly Media, ill ed., 2008.

[2] C. A. Hamilton, *BeagleBone Black Cookbook*. Packt Publishing, 2016.

[3] J. Degrave, M. Hermans, J. Dambre, and F. Wyffels, "A differentiable physics engine for deep learning in robotics," *CoRR*, vol. abs/1611.01652, 2016.

[4] F. de Avila Belbute-Peres and Z. Kolter, "A modular differentiable rigid body physics engine," in *Deep Reinforcement Learning Symposium, NIPS*, 2017.

[5] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Proc. of Robotics: Science and Systems (R:SS 2018)*, 2018.

[6] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, "Difftaichi: Differentiable programming for physical simulation," *ICLR*, 2020.

[7] W. Yu, V. Kumar, G. Turk, and C. Liu, "Sim-to-real transfer for biped locomotion," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3503–3510, 2019.

[8] M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros, "Add: Analytically differentiable dynamics for multi-body systems with frictional contact," *ACM Trans. Graph.*, vol. 39, no. 6, 2020.

[9] N. Seegmiller, F. Rogers-Marcovitz, G. A. Miller, and A. Kelly, "Vehicle model identification by integrated prediction error minimization," *International Journal of Robotics Research*, vol. 32, pp. 912–931, July 2013.

[10] N. Seegmiller and A. Kelly, "Enhanced 3d kinematic modeling of wheeled mobile robots," in *Proceedings of Robotics: Science and Systems*, July 2014.

[11] P. F. Muir and C. P. Neuman, "Kinematic modeling for feedback control of an omnidirectional wheeled mobile robot," in *Autonomous Robot Vehicles*, 1987.

[12] C. Ostafew, J. Collier, A. Schoellig, and T. Barfoot, "Learning-based nonlinear model predictive control to improve vision-based mobile robot path tracking," *Journal of Field Robotics*, vol. 33, 06 2015.

[13] U. Rosolia, A. Carvalho, and F. Borrelli, "Autonomous racing using learning model predictive control," *CoRR*, vol. abs/1610.06534, 2016.

[14] C. Xie, S. Patil, T. Moldovan, S. Levine, and P. Abbeel, "Model-based reinforcement learning with parametrized physical models and optimism-driven exploration," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[15] K. Iagnemma, Shinwoo Kang, H. Shibly, and S. Dubowsky, "Online terrain parameter estimation for wheeled mobile robots with application to planetary rovers," *IEEE Transactions on Robotics*, vol. 20, no. 5, pp. 921–927, 2004.

[16] R. L. Williams, B. E. Carter, P. Gallina, and G. Rosati, "Dynamic model with slip for wheeled omnidirectional robots," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 3, pp. 285–293, 2002.

[17] J. Swevers, C. Ganseman, D. B. Tukel, J. De Schutter, and H. Van Brussel, "Optimal robot excitation and identification," *IEEE TRO-A*, vol. 13, no. 5, pp. 730–740, 1997.

[18] L. Ljung, ed., *System Identification (2nd Ed.): Theory for the User*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.

[19] K. Wang, M. Aanjaneya, and K. Bekris, "A first principles approach for data-efficient system identification of spring-rod systems via differentiable physics engines," *Proceedings of Machine Learning Research*, vol. 120, pp. 1–15, 2020.

[20] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, *et al.*, "Interaction networks for learning about objects, relations and physics," in *Advances in neural information processing systems*, pp. 4502–4510, 2016.

[21] A. Kloss, S. Schaal, and J. Bohg, "Combining learned and analytical models for predicting action effects," *CoRR*, vol. abs/1710.04102, 2017.

[22] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Trans. Graph.*, vol. 31, pp. 43:1–43:8, July 2012.

[23] A. Raffin, "Autonomous racing robot with an arduino and a raspberry pi," Nov. 2017.

[24] R. Rojas, "Models for DC motors." Unpublished Document, 2004.

[25] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2016.

[26] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evol. Comput.*, vol. 11, p. 1–18, Mar. 2003.

[27] M. Kleinbort, E. Granados, K. Solovey, R. Bonalli, K. E. Bekris, and D. Halperin, "Refined analysis of asymptotically-optimal kinodynamic planning in the state-cost space," in *ICRA*, 2020.

[28] E. Granados, A. Sivaramakrishnan, T. McMahon, Z. Littlefield, and K. E. Bekris, "Machine learning for kinodynamic planning (ml4kp)." https://github.com/PRX-Kinodynamic/ML4KP, 2021–2021.