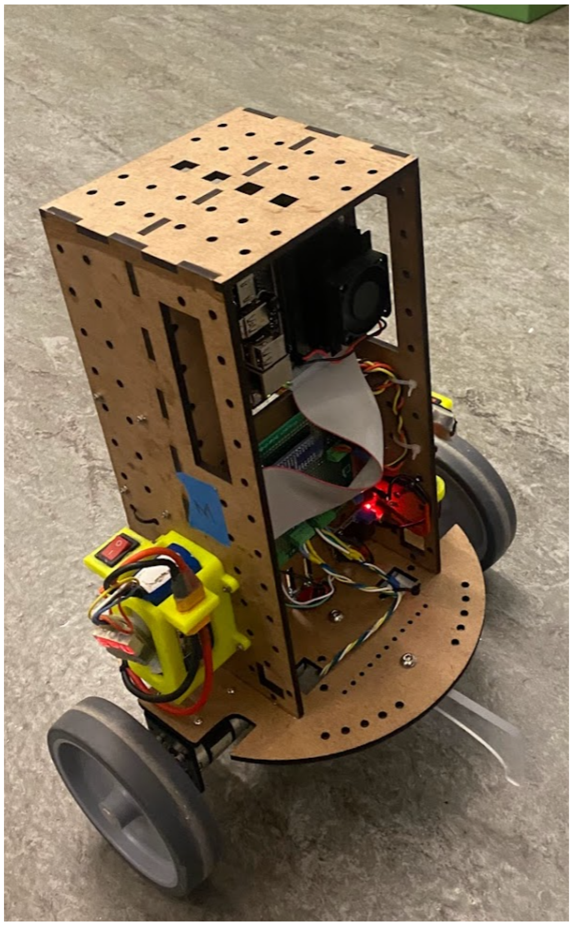
Gym2Real:
An Open-Source Platform for Sim2Real Transfer



Daniel Backhouse, Jonah Gourlay, Bereket Guta, Kevin Huang, Kobe Ng
Project Sponsor: The Engineering Physics Project Lab

Project Number: 2153
ENPH 479
Engineering Physics Project Laboratory
The University of British Columbia

April 2022

Contents

List of Figures

List of Tables

1 Executive Summary

Sponsored by the Engineering Physics Project Lab, we developed an open-source pipeline to facilitate the task of transferring a motion policy trained in simulation to a real-world robot, minimizing the barrier-to-entry for hobbyists and researchers. The project includes a two-wheeled robotic platform and software to deployed on a Nvidia Jetson Nano, with a bill of materials and instructions to build the robot and train and use policies. We used the platform to investigate reinforcement learning by training policies to balance the two-wheeled robot in Isaac Gym, a new simulation environment accessible to hobbyists. We conclude that our platform allows for successful Sim2Real transfer, but does not fully close the reality gap. We recommend further exploration of reinforcement learning through policies that execute high-level tasks using the platform we've developed.

2 Introduction

2.1 Background

Automatic control theory deals with the general task of developing a policy (aka. "controller") that provides a set of inputs into a dynamic system in order to achieve a desired state, using feedback from sensors to correct for errors. Automatic control methods are heavily integrated with modern information and communication systems. Classical techniques, such as Proportional-Integral-Derivative (PID) and Linear Quadratic Gaussian (LQG), are widely used today [3]. Yet PID, the most popular method, requires manual tuning while LQG requires an accurate state space model, making them difficult to tune when hardware is dangerous or expensive to test.

Recent advances in deep learning, enabled by powerful computer hardware and access to large amounts of simulated data, have reinforcement learning (RL) feasible as an approach to solving complex control problems. In contrast to classical algorithms, RL allows for parameters to be learned through experience. Given input observations, RL trains a policy to output actions to control an agent and maximize a designed reward.

Gym frameworks combine RL with physics simulation to train policies. Due to differences between simulation and reality (reality gap), domain transfer is not trivial. The problem of taking a policy trained in simulation and deploying it in reality is called "Sim2Real".

Policies are often trained to control agents in virtual environments with sparse success deploying these policies in reality. OpenAI has succeeded in training a robotic hand to manipulate blocks [6], but it must be noted that they use RL for high-level planning of target joint angles, while low-level PID control is still used for position control of servos. In this case, the dynamics of the robot are abstracted away. See Sec. 3.2.1 for discussion of this distinction.

2.2 Problem and Project Objectives

The reality gap is not the only disconnect in Sim2Real; there is a gap in knowledge within the field as RL researchers are not typically roboticists. A common problem is "Sim2Null", which refers to RL policies trained in simulation without consideration of deployment in the real world. [2]

Our goal is to create a platform to bridge these gaps and demonstrate a successful application of Sim2Real, by creating a unified software platform for training and deploying policies and an example robot with enough depth for further exploration. The platform should make it easy for RL researchers to deploy their policies into the real world, for roboticists to train an RL policy, and for hobbyists to learn about both ends of the problem.

2.3 Scope and Limitations

Our initial scope was overly broad: create a fully generalizable software platform that allows a user to train a policy in simulation and deploy this on hardware regardless of their robot.

The first major shift was to split our focus equally on the development of a robotic platform to demonstrate this capability, resulting in our design of a two-wheeled inverted pendulum (TWIP).

We reduced our scope on the software to focus on making the platform fit our hardware. While we anticipated potential features that other robots may need and designed our software with these in mind by including inheritable classes, we decided the user would still have to write their own code depending on their specific hardware.

2.4 Sponsor

This project is sponsored by the Engineering Physics Project Lab, with the intention of using the platform to enable future projects. Not only should our project demonstrate a novel application in a growing field of interest (Sim2Real), it should have educational value for hobbyists and Engineering Physics students.

3 Discussion

We begin with the requirements of our robotic platform, the type of control we implemented, the theory on the control and platform, and finish with a discussion on the design of the robot and software.

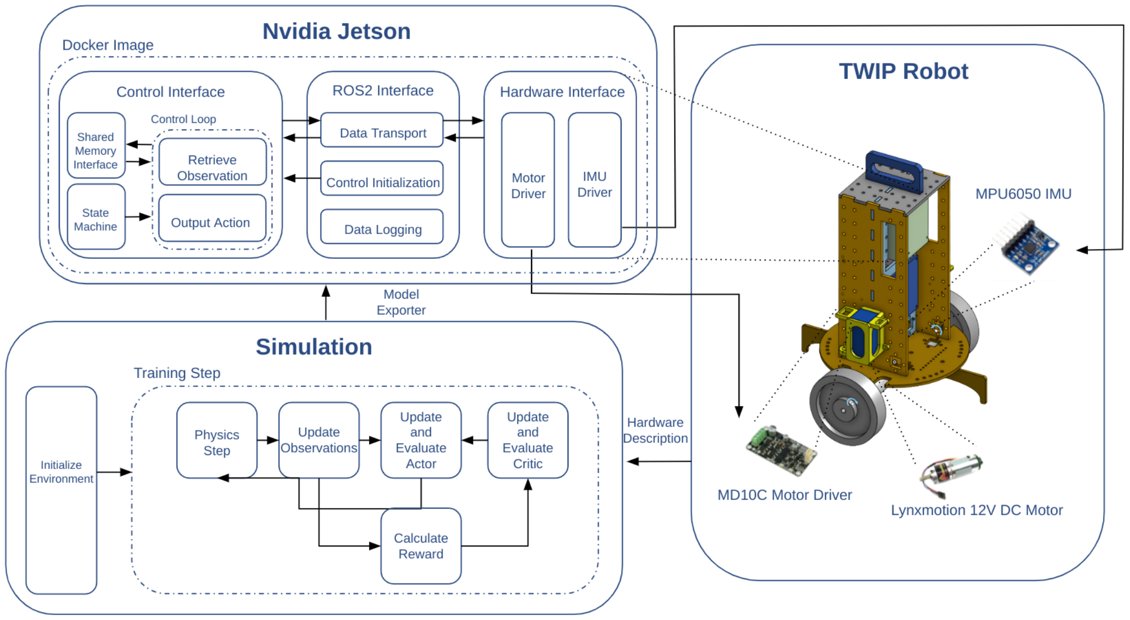


Figure 1: system level diagram showcasing the main components of our solution

3.1 Planning and Considerations

3.1.1 Robotics Platform

The choice for the robotics platform has three major considerations:

The platform must be simple to assemble given a bill of materials and CAD files for custom parts. A major goal is for hobbyists to be able to reproduce the design, so there must be no parts that are complex to manufacture, and no expensive machinery needed to assemble the robot.

The dynamics of the robotics platform must be well studied. Significant risk arises from reinforcement learning and Sim2Real transfer being relatively new methods. A platform that is poorly understood risks being fundamentally uncontrollable or difficult to reason about. Difficulty to reason about the dynamics would make it difficult to troubleshoot any issues in the Sim2Real process.

The dynamics of the system need to be interesting. This is difficult to quantify, but an uninteresting robot runs the risk of being dismissed by hobbyists as not being worth their time to explore. An overly simple platform would also be unable to highlight the capabilities of a reinforcement learning policy.

We chose a TWIP as the robotic platform to satisfy these criteria.

3.1.2 Isaac Gym

While most machine learning is done on GPU, most gyms perform physics simulations on the CPU, which introduces a bottleneck as many cores are required to simulate multiple robots in parallel. Nvidia's

Isaac Gym offers a solution by performing both physics simulation and network training on GPU. The reduced training times in [4] demonstrate a speedup by 2-3 orders of magnitude. This enables training on hobbyist-level hardware rather than large server clusters, making Isaac Gym far more accessible for our target audience.

3.2 Theory

3.2.1 Low-Level Control vs High-Level Planning

Often, robots require a hierarchy of controllers. A distinction should be made between two levels of control to clarify terminology and firmly categorize the TWIP-balancing problem:

Usually, low-level control directly drives actuators and operates a continuous action space. Examples include position control of servos to reach a target angle and velocity control of motors to reach a target RPM.

High-level planning typically has no visibility into the low-level controller; it interfaces with an abstraction. It can output either continuous or discrete actions. Examples include motion planners that generate trajectories for a robot to follow and policies that play discrete games like chess (requiring no low-level controller).

The delineation isn't always clear. Trajectory tracking is low-level control even though lower-levelled controllers drive actuators, because it has to be differentiated from the high-level motion planner which generates trajectories. For a TWIP, the distinction is also vague as the policy we developed is low-level enough to be dynamics-sensitive, but doesn't directly drive the motors. This difference leads to unique challenges, making the TWIP an effective case study for Sim2Real transfer. We will classify our TWIP-balancing policy as a low-level controller, while keeping these distinctions in mind.

3.2.2 Reinforcement Learning

The typical RL training process involves four components. As defined by OpenAI, the agent is the entity to control, such as a robot; the environment, real or simulated, is the world containing the agent; observations are partial states from the environment received as input into the agent; and actions are the agent's output, which allow it to affect the environment. [1] The policy controls the agent, so we will treat this as the main unit for discussion in place of the agent.

The goal of a RL policy is to maximize a reward function. In an actor-critic model such as Advantage Actor-Critic (A2C), the policy consists of two networks: an actor takes the observations and outputs actions; the critic learns a value function, which estimates the expected reward given the observations and actions. The critic is used only during training; inference requires only the actor. [5]

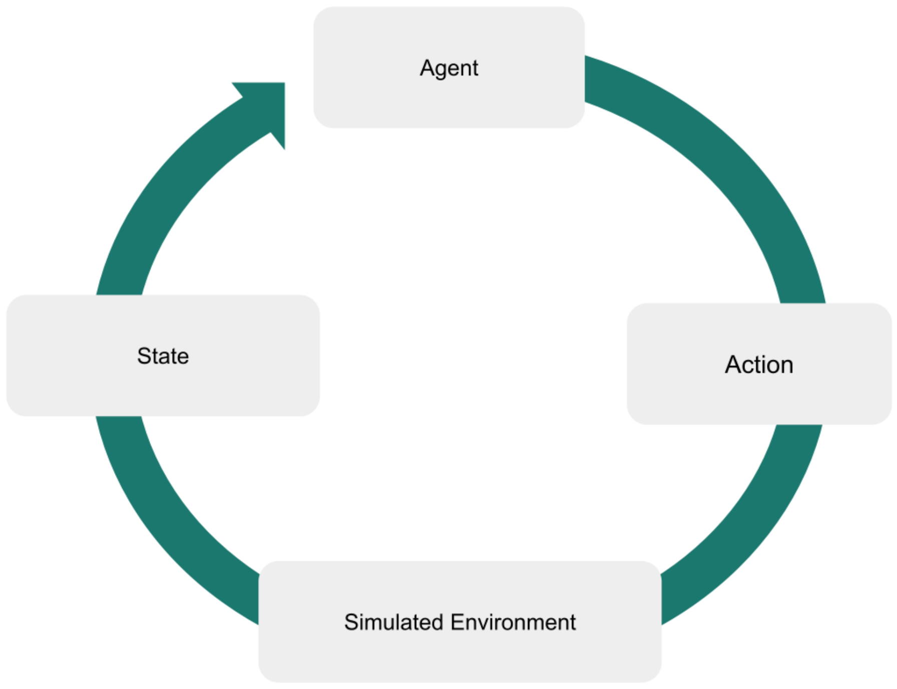


Figure 2: Main components of reinforcement learning

While the environment can be the real world, training a policy on hardware can be expensive, time-consuming, and unsafe. Gyms package a physics simulation engine (taking on the role of the environment) with a RL training framework to facilitate the training of policies for different tasks. In simulation, robots can be easily reset upon failure, and the only cost is computation time and power.

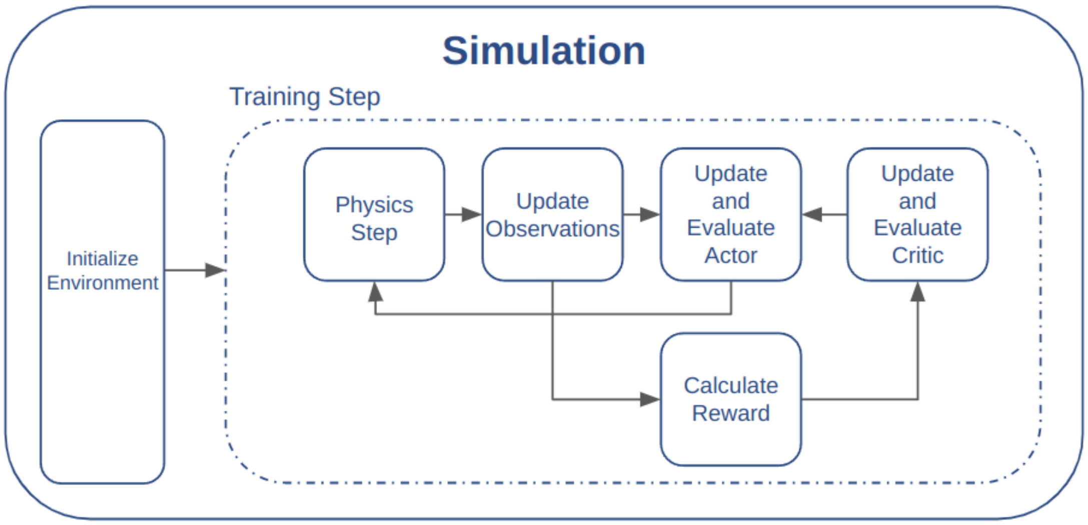


Figure 3: Computational flow of actor-critic model in a gym

3.2.3 Domain Randomization

The main technique for creating robust RL policies is domain randomization. If we can randomly vary physical parameters in simulation within a range, then the policy should generalize to a real robot with parameters within this range. Research has shown that domain randomization can also help a policy handle latency in controller feedback, but recommend against blindly applying domain randomization to all parameters [9].

3.2.4 Two-Wheeled Inverted Pendulum

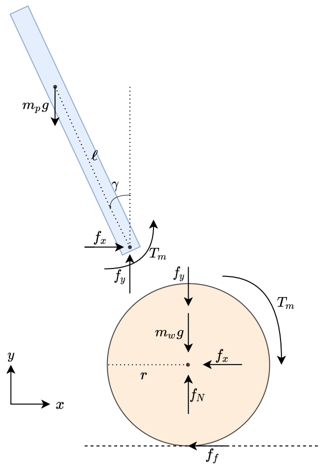


Figure 4: Free body diagram of the Two-wheeled inverted pendulum under analysis

The essential components of the TWIP are the wheels, body, and motors. Analysis can be carried out in two dimensions to reduce the degrees of freedom. The system is actuated by a motor mounted on the body. The wheels, which are mounted on the shaft of the motor, can rotate with respect to the body to maintain balance about the vertical axis. This inverted configuration is unstable by design and is not robust to disturbances without active control.
[7].

The appendix contains a detailed derivation of the dynamics of this system using Newtonian Mechanics App. A. The key outcome of the analysis presents the following set of coupled non-linear differential equations for the angle about the vertical (γ) and the linear distance of the wheel (x) as shown in Fig. 4.

$$\ddot{x} = \frac{\left[\frac{K_t}{R}(V - K_e\dot{\psi}) - b\dot{\psi}\right] r - \left[\dot{\gamma}^2 \sin(\gamma) - \ddot{\gamma}\cos(\gamma)\right] \frac{m_p \ell r^2}{2}}{\left[I_w + (m_w + \frac{m_p}{2}) r^2\right]} \tag{1}$$

$$\ddot{\gamma} = \frac{2\left[\frac{K_t}{R}(V - K_e\dot{\psi}) - b\dot{\psi}\right] + m_p g \ell \sin(\gamma) - \ddot{x} m_p \ell \cos(\gamma) + m_p \ell^2 \ddot{\gamma} \cos^2(\gamma) - m_p \ell^2 \dot{\gamma}^2 \sin(\gamma)\cos(\gamma)}{I_p} \tag{2}$$

Eq. 1 & Eq. 2 govern the interaction between wheel and body. The parameters for the design presented in this report are outlined in Tab. 1.

Parameter	Value	Unit	Description
m_p	2.271	kg	Mass of body
I_w	0.03088119	$\mathrm{kg\,m^2}$	Moment of inertia of body
ℓ	0.188	m	Wheel center to body COM distance
m_w	0.19	kg	Mass of wheel
I_w	0.0004358343	$\mathrm{kg\,m^2}$	Moment of inertia of wheel
r	0.0625	m	Radius of wheel
R	2.14	Ω	Resistance of motor
K_e	1.07	$\mathrm{V\,s\,rad^{-1}}$	Emf constant
K_t	0.611	$\mathrm{N\,m\,A^{-1}}$	Torque constant
b	0.03	$\mathrm{N\,m\,s\,rad^{-1}}$	motor viscous friction constant
g	9.81	$\mathrm{m\,s^{-2}}$	Gravitational acceleration

Table 1: Key parameters for the two-wheeled inverted pendulum designed.

3.2.5 Real-Time Programming

Robotics control requires a fast control loop frequency to quickly adjust to changing dynamics. This presents issues for an embedded system.

Memory operations can be time-consuming if an accessed address has not yet been loaded into random access memory (RAM). When this happens, an expensive page fault operation occurs and the processor attempts to load the next sequence into memory. This can be circumvented by pre-allocating a fixed amount of memory before the main program runs. Allocating and carefully managing shared memory buffers to be re-used within the program ensures that memory is used as efficiently as possible.

Embedded systems often communicate with many devices simultaneously. Processes with a high frequency tend to consume CPU resources, starving essential processes (i.e. a control loop). Most Unix operating systems schedule processes non-deterministically, and certain processes could wait forever until they are rescheduled. In a real-time system, deadlines are very important: every operation has to happen at a certain time to keep the whole system running smoothly. A Linux kernel can be made more deterministic by applying a patch called PREEMPT_RT. This patch allows processes to be scheduled with a higher priority than they normally could be, making deadlines “harder” than on the standard kernel.

3.3 Design

3.3.1 Two-Wheeled Inverted Pendulum

As outlined in Sec. 3.1.1, the main criteria for the design is ease of assembly given the bill of materials and readily obtainable parts. A CAD of the design is shown in Fig. 5. The design consists of a hardboard frame held by epoxy. This frame provides a durable structure that is resistant to compressional and torsional loads. The rest of the components are placed on this frame. The motors and wheels are mounted on the base plate using L-brackets placed near the edges of the TWIP (Fig. 6). The electronics lie vertically on the front for ease of access. Two battery holders are placed on the sides to supply power to the motor and Jetson independently. A detailed bill of materials is provided in App. C and a guide for building the robot is described on our website in App. D.

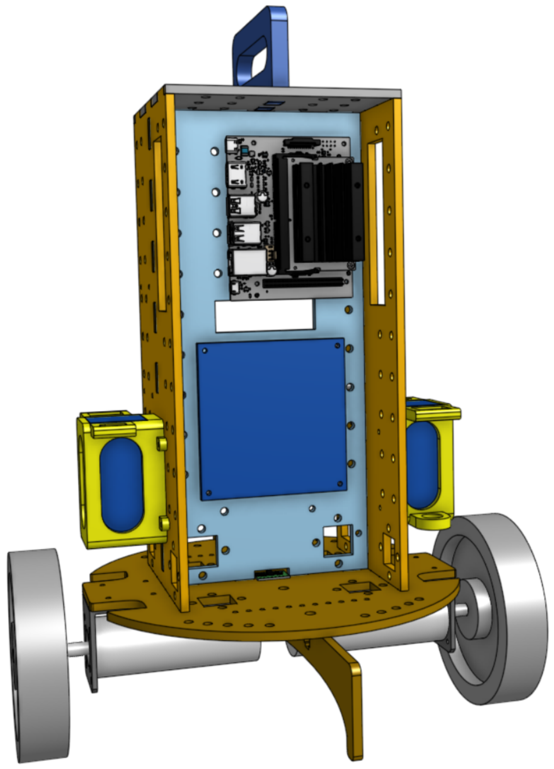


Figure 5: CAD of the presented robotics platform.

We modelled each part of the TWIP and generated a Unified Robot Description Format file (URDF). We saw a 0.3kg discrepancy, which is accounted for by domain randomization.

3.3.2 Sensor Integration

To measure wheel velocity and pitch angle, we used rotary encoders included with the Lynxmotion motors and a MPU-6050 inertial measurement unit (IMU). Measuring wheel velocity can be achieved by sampling encoder pulses over a period of time N_T, and dividing the number of pulses by the sample time t. This measurement was useful to compare the desired wheel speed against the target speed

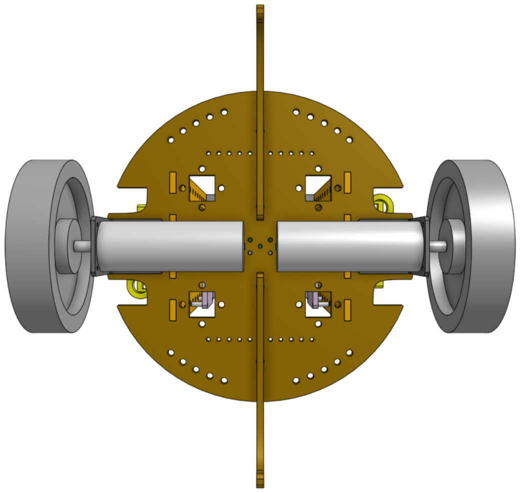


Figure 6: The mounting for the motors on the robotics platform.

outputted by the policy.

$$\omega_W = 2\pi * \frac{N_T}{t} \tag{3}$$

Measuring the pitch was more difficult. The IMU consists of a gyroscope and an accelerometer, each giving three separate measurements. The gyroscope measures the rate of change of the Euler angles (pitch, yaw, and roll), while the accelerometer measured acceleration in the Cartesian directions (X, Y, and Z). To convert these into a quaternion representation, we used a Madgwick filter (App. B). The final pitch θ used for control is calculated with Eq. 4 from a quaternion of form (w, x, y, z).

$$\theta = \arctan\left(\frac{2xw + 2zy}{1 - 2x^2 - 2y^2}\right) \tag{4}$$

We noticed that the parameter β had a large impact on the responsiveness of the Madgwick filter. For large angles ($> 5°$) the filter had slow convergence regardless of the magnitude of β, making angle measurements subject to a large error. However, substituting the Madgwick filter for a complementary filter made the resulting pitch data noisy in the steady-state ($\pm 3°$), causing severe instability when balancing. We decided to use the Madgwick filter to avoid this instability.

3.3.3 Hardware Integration

To run an RL policy on our TWIP, we chose GPU capabilities as a criterion for computing hardware. In addition, we needed a device that was lightweight, real-time capable, and well-supported in the robotics community. We chose the Nvidia Jetson Nano development kit which satisfied all these criteria. We considered using a Raspberry Pi 4, but determined that the Nano was more flexible for future extension given its support for CUDA, an API required for many machine learning applications.

The Jetson Nano runs a customized version of ARM64 Ubuntu. This made development easier, because many modern tools are supported in this OS. Nvidia has also released a real-time kernel patch that incorporates the functionality of PREEMPT_RT into an easily-installed Debian package. We installed this for the advantages discussed in Sec. 3.2.5.

In addition to the real-time patch, we had to develop our own custom patch to solve a communication issue. The IMU communicates with the Jetson via an I2C channel, which was susceptible to noise from the motors. The motor noise disturbed the digital signal on the I2C channel and altered the bits, leading to invalid data. To remove errors, the Jetson has a kernel-level instruction to clear the I2C bus and wait 10 seconds before resuming transmission, resulting in a hangup that locked the robot into the same action as the pitch measurement couldn't change. By reducing the wait time from 10 seconds to 10 milliseconds in the kernel, we were able to prevent this hangup from occurring.

3.3.4 Software Design

We wrote our code with modularity in mind. We separated the functionality of our programs into two main components: the hardware interface, and the controller interface. These two components are bridged together with ROS2, a commonly used framework in robotics research. The main structure is shown in Fig. 7.

The hardware interface is responsible for low-level communications between the Jetson and other electronics. It reads/writes data from/to the IMU and motor drivers and publishes/receives messages via ROS2 topics. The motor and IMU drivers themselves are ROS2 nodes that can access GPIO, PWM, and I2C functions on the Jetson board.

The controller interface is responsible for control of the robot. It receives IMU and motor velocity data, performs inference with a policy, and outputs a target motor velocity. The controller interface can pre-allocate shared memory to avoid page faults at runtime, and has a state machine to enter a braking state if the IMU angle exceeds a certain threshold for a certain amount of time (a failure mode from which our policy cannot recover), allowing the TWIP to kick up to the desired angle.

A memory manager initializes memory buffers and controllers from a configuration file in YAML format. Rules can be added to the controller to perform arbitrary operations between memory buffers, allowing for heterogeneous data formats and non-contiguous memory segments to be copied to/from the buffers reserved for policy inference.

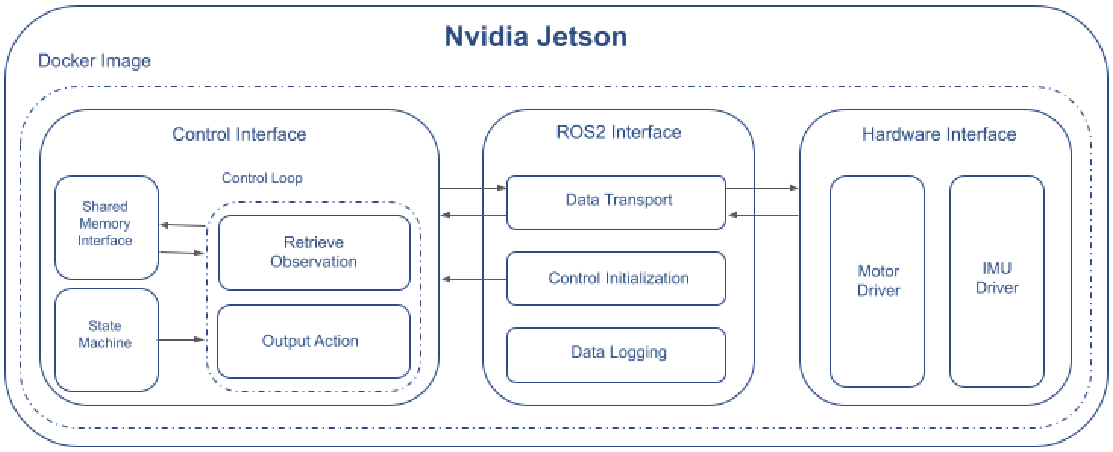


Figure 7: Code design system level diagram.

3.3.5 TWIP-Balancing Policy

We designed a policy to take in the pitch angle and velocity as observations and output motor velocity as an action. Velocity control is simple to implement and troubleshoot given a PWM motor driver and velocity feedback from rotary encoders, making it an accessible choice for hobbyists to perform their own experiments.

To design a reward function, we must consider the variables we want to maximize or minimize, as well as the shape of the reward function with respect to these variables. Balancing requires minimizing the TWIP's pitch angle from the vertical. Since we did not want the TWIP to constantly move at full speed, we also want to minimize the TWIP's action (motor speed). We experimented with a few different reward functions, including one designed for the cart-pole inverted pendulum included with Isaac Gym (Eq. 5).

$$r(\theta, v) = 1 - \theta^2 - 0.05|v| \tag{5}$$

This reward decays quadratically with increasing pitch angle, which is too slow for our TWIP since we have a smaller range of angles from which we can recover. With this reward, we saw that the TWIP would often favour staying at a tilt in simulation in order to minimize speed. In the end, we decided that the reward should decay at least linearly with pitch angle. We settled on a $\tanh$ function, which is linear for small angles, resulting in our final reward function (Eq. 6).

$$r(\theta, v) = 1 - \tanh 8\theta - 0.05 \tanh 2|v| \tag{6}$$

We noticed when deploying our policy in real that the TWIP was not robust except within a small range of angles. This was partially because the policy performed too well in simulation, reaching stability quickly and maintaining the angle within a small range. This made the brunt of its training experience limited to a small subset of the angles from which we want it to balance. We investigated a few different methods to solve this: adding random torque perturbations and random orientations whenever we reset the TWIP in simulation, and reducing the time of each “episode”, which means the TWIP has less time to reach stability.

3.3.6 Open Source Considerations

One of the major aspects of our project was to make it replicable by fellow researchers and hobbyists. This involved writing clean, extensible code and wrapping our dependencies into a Docker container. We applied a BSD 3-clause license to our work, as this has the most open permissions of the available open source licenses. We have made our work available through two GitHub repositories, a DockerHub repository, CAD files, and a website which can all be found in App. D.

4 Results

4.1 Overall Balancing

The goal of the following test is to ensure that the RL model is capable of balancing the TWIP. To do this we flashed the model on the TWIP, let it rest on its side legs and turned on the motor. The TWIP would kick up, and we measured the time needed before the TWIP lost stability and reached $13°$. The results of the experiment with various policies can be seen in Fig. 8

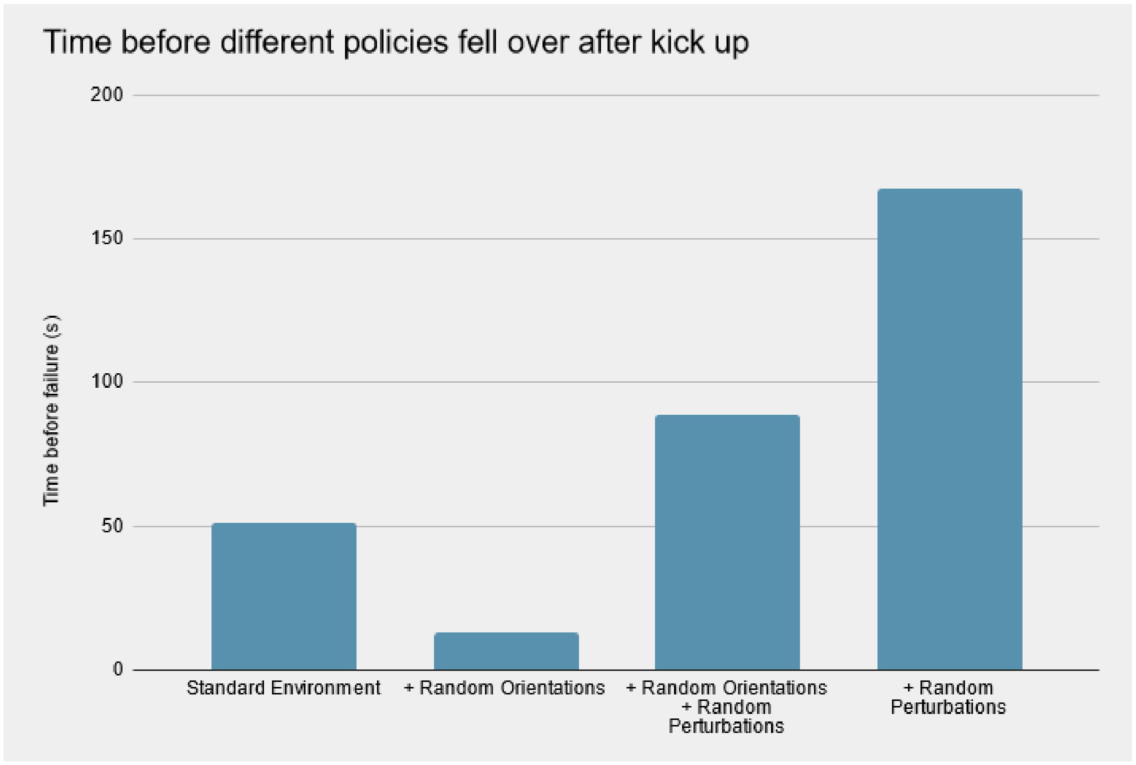


Figure 8: Comparison of policies trained with different initial conditions in simulation when deployed on the real robot. The standard environment is a naive implementation with only domain randomization. Random orientations means that the TWIP started with a random angle in simulation, and random perturbation means that the robot started with a random torque applied to the body.

The robot was capable of balancing for a reasonable period of time with all of the RL models. The best result comes from only initializing the robot in simulation with only a random torque.

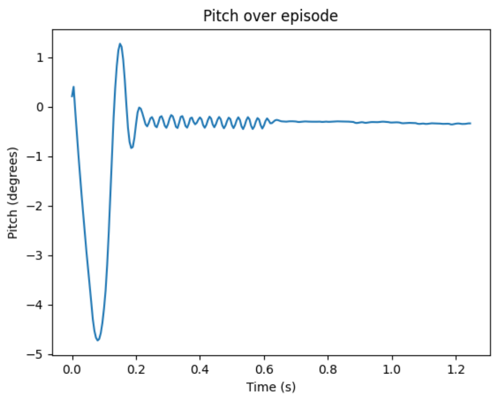
All orientations would eventually destabilize given enough time. This is due to IMU readings drifting over time creating inaccurate readings for pitch.

4.2 Physics in Simulation vs Real

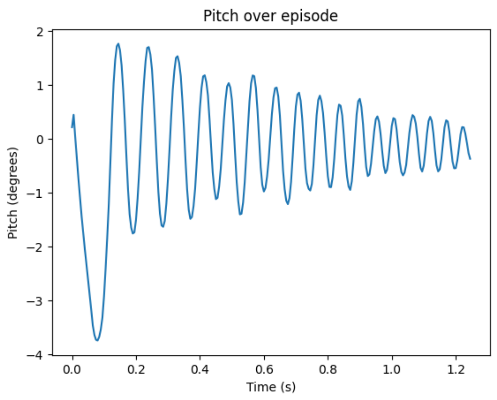
To explore the differences in physics between the simulation in Isaac Gym and reality, we designed tests that would be both interesting to hobbyists as well as industry professionals and researchers.

Some properties of the robotics platform, such as mass, are simple to measure; domain randomization can account for discrepancies between reality and simulation. Other physical properties such as damping and friction are much harder to measure, making it easy for an estimated range to not cover the real value. As this is a realistic situation to encounter, we designed tests to characterize the importance of training with realistic values of damping and friction. It's important to note that in the context of physics simulators, velocity control relies on an internal PD controller. Damping determines the torque output proportional to velocity error, which makes it different from the more common motor damping coefficient, which is used in effort control.

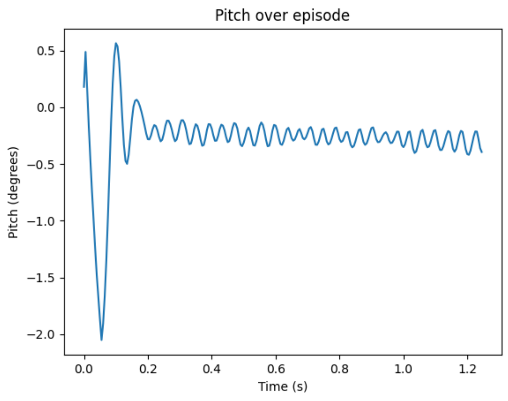
To test the importance of accurate friction and damping values we trained various models with values of damping and friction that were far enough apart that the values would not overlap even with domain randomization. We tested the TWIP's performance in two tasks, starting from $0°$ pitch ("Zero") and starting from $-13°$ pitch ("Kickup"), and then transferred the models to the real robot. If setting realistic values of these physical parameters in the model is important, we should expect to see that only one of the models performs similarly in both simulation and reality.



(a) Damping=600, Friction=0

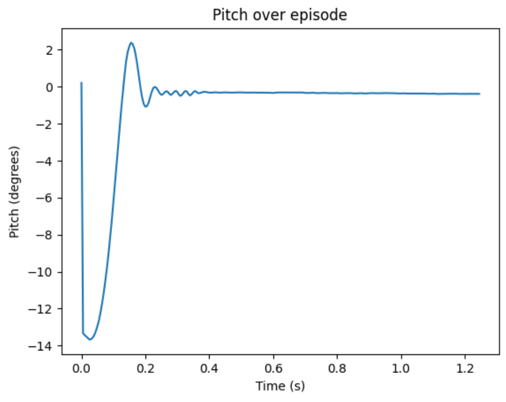


(b) Damping=600, Friction=1

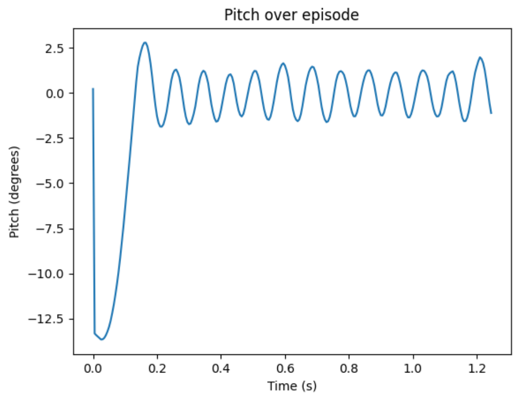


(c) Damping=0.1, Friction=0

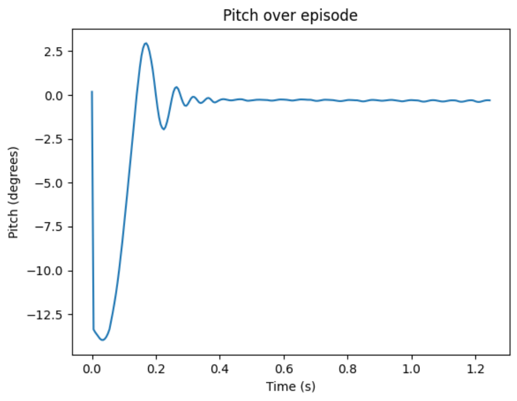
Figure 9: Comparison of policies trained with different physical parameters (these parameters were used during testing). Robot was initialized with zero pitch and no torque perturbation. Domain randomization (0.5 to 1.5 times the value) was enabled during training but disabled for this comparison.



(a) Damping=600, Friction=0



(b) Damping=600, Friction=1

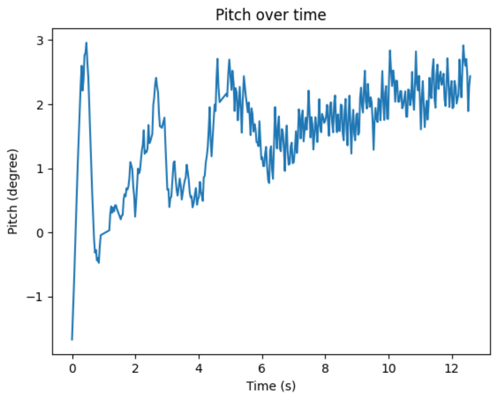


(c) Damping=0.1, Friction=0

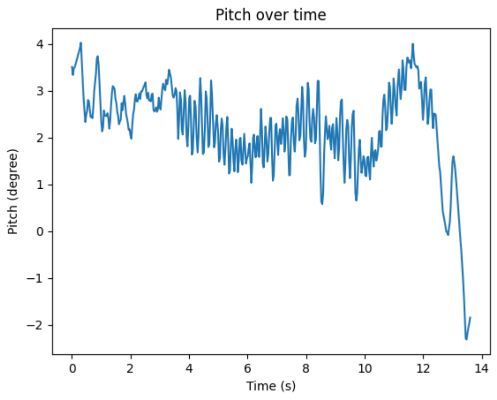
Figure 10: Comparison of policies trained with different physical parameters (these parameters were used during testing). Robot was initialized with -13° pitch and no torque perturbation. Domain randomization (0.5 to 1.5 times the value) was enabled during training but disabled for this comparison.

With high damping and no friction, the policy was able to reach stability in the steady-state in both tasks. With high damping and friction, the policy had large oscillations in both tasks. With low damping and no friction, we observed oscillations in the steady-state in both tasks. The policy was capable of reaching stability with all tested parameters.

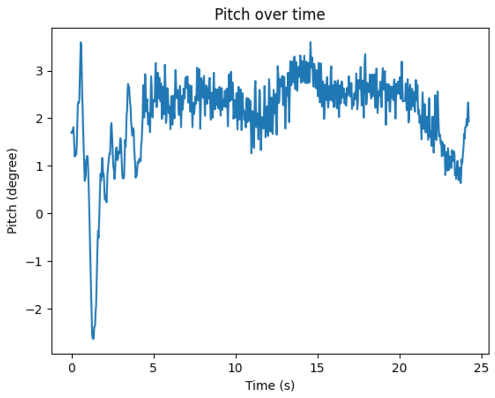
We then performed the same tests in reality, by testing each policy on the real robot.



(a) Damping=600, Friction=0

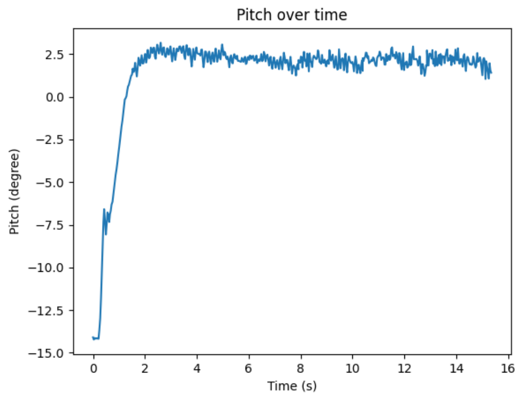


(b) Damping=600, Friction=1

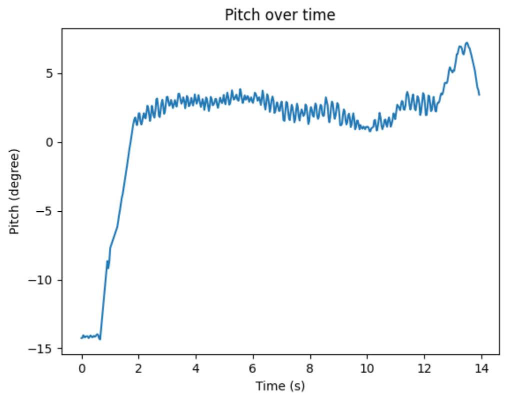


(c) Damping=0.1, Friction=0

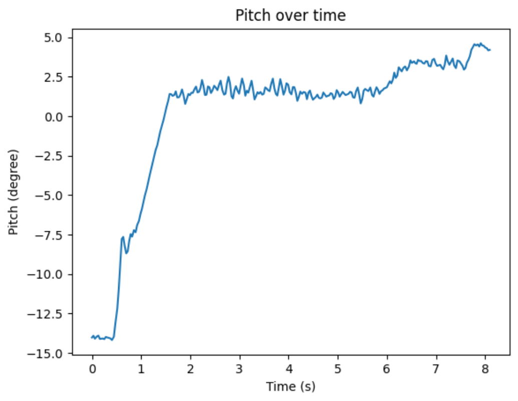
Figure 11: Comparison of policies trained with different physical parameters, tested on the real robot. Robot was initialized with zero pitch and no torque perturbation. Pitch measurement contains a bias of around $2.5°$ which we correct for in the input to the policy.



(a) Damping=600, Friction=0



(b) Damping=600, Friction=1



(c) Damping=0.1, Friction=0

Figure 12: Comparison of policies trained with different physical parameters, tested on the real robot. Robot was initialized with -13° pitch and no torque perturbation. Pitch measurement contains a bias of around 2.5° which we correct for in the input to the policy.

In the Zero task, all policies performed poorly when compared to simulation. The policy trained with high damping and friction performed the worst, losing stability after about 10 seconds. The other policies were more stable, but performed worse than in simulation.

In the Kickup task, the policy with high damping and friction and the policy with low damping and no friction both failed to remain stable after less than 10 seconds. Only the policy with high damping and no fiction remained stable for a significant period of time.

In all cases, the TWIP stabilized quicker and had less noise in simulation compared to reality. This is because the model is able to react much more quickly in simulation without additional latency from IMU readings. Other factors like motor backlash and inherent non-linearities in the motor further increase the Sim2Real gap.

Certain values of damping and friction will create models that balance in simulation but not in reality. Overall, the models are still able to balance for short durations of time regardless of the values that the model are trained on. If high performance is a requirement, some amount of tuning and measurement of physical parameters is required, but the values do not have to be precise.

5 Conclusions

We designed and assembled a TWIP and made a pipeline for training RL models in IsaacGym, which we used to transfer models onto the robot. The model was successful at balancing both in simulation and reality. Using this robotic platform, we ran several tests to explore how different parameters affect the Sim2Real transfer.

Training in simulation without consideration of reality will likely lead to poor performance in reality as simulations provide overly idealized environments. This is especially true for low-level controllers that are sensitive to dynamics. Even groups with more resources, such as OpenAI, carefully choose the problems they solve with RL, focusing on high-level planning instead of low-level control.

Sim2Real is difficult. There are still many considerations we have not accounted for between simulation and reality, such as motor behaviour, IMU behaviour, latency. We haven't solved the Sim2Real problem, but provide methods and tools to bridge the gap. For the Project Lab, our TWIP platform and our software pipeline will make it easier to develop future projects researching Sim2Real.

6 Recommendations

6.1 High-Level TWIP Planning

We have explored the training of a low-level balancing controller. Future work can investigate high-level planning using our low-level controller, as defined in Sec. 3.2.1, to execute complex tasks (eg. braking near obstacles, navigating a room). In simulation, this can be achieved by incorporating the low-level controller into the physics step, treating it as a distinct module that the high-level planner can rely on.

While the Sim2Real problem necessitates analysis of the dynamics, we must consider whether using an RL policy for low-level control of a robot like a TWIP is necessary if the same or better result can be achieved with a PID controller. The benefit of RL is that it can solve problems that are infeasible by manual design. While verification of the dynamics is important, it is not the end goal; it may be more interesting and fruitful when working with simple hardware to leave low-level control to familiar technology and instead explore problems on a higher level.

6.2 Effort Control

Effort control (torque or force control) is commonly used in RL, especially for robots that interact with humans, since it is safer than position or velocity control (because the actuator will not try to apply more force than the target). The state-of-the-art in robot motion uses Riemannian Motion Policies or geometric fabrics which are acceleration-based policies, making effort control of continued interest to researchers. [8] However, torque control is typically found only in high-end servo motors. An affordable option may be the ODrive which can be used with cheap brushless DC motors to achieve torque control by controlling current.

6.3 Non-TWIP Robots on Gym2Real Platform

The TWIP's low-level controller is capable of balancing, which is a major milestone for the TWIP as a robotic platform. While more work can be done to explore high-level planning on the TWIP, other robots can also be investigated. Our open-source software can be modified to train and transfer models for other tasks such as air hockey tables given a URDF model and code to interface with hardware.

7 Deliverables

1. CAD files detailing design of TWIP
2. Full bill of materials needed to assemble the robot
3. Assembled robot with RL model flashed to the Jetson Nano
4. Github repository containing code to control the hardware and high level software of the Jetson Nano
5. Github repository containing code for simulation and training reinforcement learning models
6. Website for hobbyists detailing how to use recreate the robot and model

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[2] Sebastian Höfer, Kostas Bekris, Ankur Handa, Juan Camilo Gamboa, Florian Golemo, Melissa Mozifian, Chris Atkeson, Dieter Fox, Ken Goldberg, John Leonard, C. Karen Liu, Jan Peters, Shuran Song, Peter Welinder, and Martha White. Perspectives on sim2real transfer for robotics: A summary of the r:ss 2020 workshop, 2020.

[3] Štefan Kozák. State-of-the-art in control engineering. *Journal of Electrical Systems and Information Technology*, 1(1):1–9, 2014.

[4] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning. *CoRR*, abs/2108.10470, 2021.

[5] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[6] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018.

[7] Lal Bahadur Prasad, Barjeev Tyagi, and Hari Om Gupta. Optimal control of nonlinear inverted pendulum dynamical system with disturbance input using pid controller & lqr. In *2011 IEEE International Conference on Control System, Computing and Engineering*, pages 540–545. IEEE, 2011.

[8] Mandy Xie, Karl Van Wyk, Anqi Li, Muhammad Asif Rana, Dieter Fox, Byron Boots, and Nathan D. Ratliff. Geometric fabrics for the acceleration-based design of robotic motion. *CoRR*, abs/2010.14750, 2020.

[9] Zhaoming Xie, Xingye Da, Michiel van de Panne, Buck Babich, and Animesh Garg. Dynamics randomization revisited: A case study for quadrupedal locomotion. *CoRR*, abs/2011.02404, 2020.

Appendix A Two-Wheeled Inverted Pendulum

The TWIP system in Fig. 4 can be analyzed in 2D to simplify the problem. We analyze each component (body & wheel) separately, we begin with the equation for each Wheel

$$\begin{aligned}\ddot{x}m_w &= f_f - f_x \\ \ddot{y}m_w &= f_N - f_y - m_w g = 0 \\ I_w\ddot{\psi} &= -f_f r + T_m\end{aligned}$$

Next, the equation for pole/body

$$\begin{aligned}\ddot{x}_p m_p &= 2f_x \\ \ddot{y}_p m_p &= 2f_y - m_p g = 0 \\ I_p\ddot{\gamma} &= 2f_y\ell\sin(\gamma) - 2f_x\ell\cos(\gamma) + 2T_m\end{aligned}$$

The relationship between wheel rotation (ψ) and wheel linear displacement (x) is

$$\begin{aligned}x &= \psi r \\ \dot{x} &= \dot{\psi} r \\ \ddot{x} &= \ddot{\psi} r\end{aligned}$$

The center of gravity is displaced as governed by

$$\begin{aligned}x_p &= x - \ell\sin(\gamma) \\ y_p &= \ell\cos(\gamma)\end{aligned}$$

Taking time derivatives,

$$\begin{aligned}\dot{x}_p &= \dot{x} - \dot{\gamma}\ell\cos(\gamma) \\ \ddot{x}_p &= \ddot{x} - \ell\ddot{\gamma}\cos(\gamma) + \ell\dot{\gamma}^2\sin(\gamma) \\ \dot{y}_p &= -\dot{\gamma}\ell\sin(\gamma) \\ \ddot{y}_p &= -\ell\ddot{\gamma}\sin(\gamma) - \ell\dot{\gamma}^2\cos(\gamma\end{aligned}$$

Simplifying a little . . .

$$\begin{aligned}I_p\ddot{\gamma} &= m_p g\ell\sin(\gamma) - \ddot{x}_p m_p\ell\cos(\gamma) + 2T_m \\ &= m_p g\ell\sin(\gamma) - (\ddot{x} - \ell\ddot{\gamma}\cos(\gamma) + \ell\dot{\gamma}^2\sin(\gamma))m_p\ell\cos(\gamma) + 2T_m \\ I_p\ddot{\gamma} &= 2T_m + m_p g\ell\sin(\gamma) - \ddot{x}m_p\ell\cos(\gamma) \\ &\quad + m_p\ell^2\ddot{\gamma}\cos^2(\gamma) - m_p\ell^2\dot{\gamma}^2\sin(\gamma)\cos(\gamma)\end{aligned}$$

Simplifying further . . .

$$\begin{aligned}
I_w\ddot{\psi} &= -(\ddot{x}m_w + f_x)r + T_m \\
I_w\ddot{\psi} &= -(\ddot{x}m_w + \frac{\ddot{x}_p m_p}{2})r + T_m \\
I_w\frac{\ddot{x}}{r} &= -\ddot{x}m_w r - \ddot{x}_p\frac{m_p r}{2} + T_m \\
I_w\frac{\ddot{x}}{r} &= -\ddot{x}m_w r - (\ddot{x} - \ell\ddot{\gamma}\cos(\gamma) + \ell\dot{\gamma}^2\sin(\gamma))\frac{m_p r}{2} + T_m \\
I_w\ddot{x} &= T_m r - \ddot{x}m_w r^2 - \ddot{x}\frac{m_p r^2}{2} - \ell\dot{\gamma}^2\sin(\gamma)\frac{m_p r^2}{2} + \ell\ddot{\gamma}\cos(\gamma)\frac{m_p r^2}{2} \\
I_w\ddot{x} &= T_m r - (m_w + \frac{m_p}{2})\ddot{x}r^2 - \left[\dot{\gamma}^2\sin(\gamma) - \ddot{\gamma}\cos(\gamma)\right]\frac{m_p\ell r^2}{2}
\end{aligned}$$

To understand how the torque T_m is generated, consider the circuit in Fig. 13.

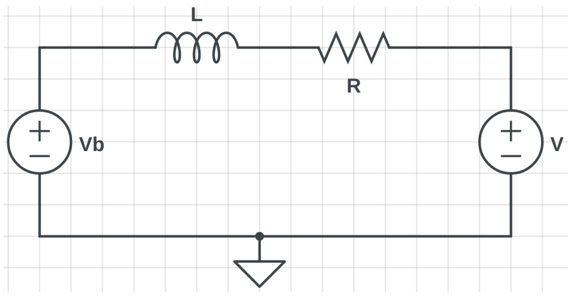


Figure 13: A model for a brushed DC motor

As the motor rotates, a potential (V_b) is induced which is equal to $V_b = K_e\dot{\psi}$, where K_e is the emf constant. The motor torque is equal to $T = K_t i$, where K_t is the motor torque constant. Solving for the steady state torque,

$$\begin{aligned}
i &= \frac{V - V_b}{R} = \frac{V - K_e\dot{\psi}}{R} \\
T &= K_t\frac{V - K_e\dot{\psi}}{R}
\end{aligned}$$

To obtain the torque T_m, damping be considered. The final torque generated is

$$T_m = \frac{K_t}{R}(V - K_e\dot{\psi}) - b\dot{\psi}$$

Note, it is assumed the moment of inertia of the motor shaft is negligible when compared to the wheel. Finally, the analysis above results in the following two equations

$$\ddot{x} = \frac{\left[\frac{K_t}{R}(V - K_e\dot{\psi}) - b\dot{\psi}\right]r - \left[\dot{\gamma}^2\sin(\gamma) - \ddot{\gamma}\cos(\gamma)\right]\frac{m_p\ell r^2}{2}}{\left[I_w + (m_w + \frac{m_p}{2})r^2\right]}$$

$$\ddot{\gamma} = \frac{2\left[\frac{K_t}{R}(V - K_e\dot{\psi}) - b\dot{\psi}\right] + m_p g\ell\sin(\gamma) - \ddot{x}m_p\ell\cos(\gamma) + m_p\ell^2\ddot{\gamma}\cos^2(\gamma) - m_p\ell^2\dot{\gamma}^2\sin(\gamma)\cos(\gamma)}{I_p}$$

Appendix B Madgwick Filter

Madgwick's filter is used to calculate the quaternion representation of an IMU's orientation $\boldsymbol{q} = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}$ from gyroscope and accelerometer measurements. The filter is a complementary combination of the orientation derived from angular rate, $\boldsymbol{q}_{\omega,t}$ and the orientation derived from optimizing an inertial objective function, $\boldsymbol{q}_{\Delta,t}$.

$$\boldsymbol{q}_t = \gamma_t \boldsymbol{q}_{\Delta,t} + (1-\gamma_t)\boldsymbol{q}_{\boldsymbol{\omega},\boldsymbol{t}} \tag{7}$$

Where γ_t are the weights at time-step t ranging between 0 and 1. To calculate $\boldsymbol{q}_{\omega,t}$, we can numerically integrate its derivative

$$\dot{\boldsymbol{q}}_{\omega,t} = \frac{1}{2}\boldsymbol{q}_{\omega,t}\boldsymbol{\omega}_t$$

Where $\boldsymbol{\omega}_t = \begin{bmatrix} 0 & \omega_x & \omega_y & \omega_z \end{bmatrix}$ is the tri-axial angular rate measured in the IMU's frame. The numerical integration step is simply

$$\boldsymbol{q}_{\omega,t} = \boldsymbol{q}_{t-1} + \dot{\boldsymbol{q}}_{\omega,t}\Delta t \tag{8}$$

The numerical integration is then

$$\boldsymbol{q}_{\omega,t} = \boldsymbol{q}_{\omega,t-1} + \frac{1}{2}\boldsymbol{q}_{\omega,t-1}\boldsymbol{\omega}_t\Delta t \tag{9}$$

Finding $\boldsymbol{q}_{\Delta,t}$ involves an optimization of an objective function $f(\boldsymbol{q},\boldsymbol{d},\boldsymbol{s})$ which aligns a reference $\boldsymbol{d} = \begin{bmatrix} 0 & d_x & d_y & d_z \end{bmatrix}$ with a measurement $\boldsymbol{s} = \begin{bmatrix} 0 & s_x & s_y & s_z \end{bmatrix}$. For our purposes, the reference $\boldsymbol{d}$ is chosen to be a normalized accelerometer reading $\boldsymbol{a} = \begin{bmatrix} 0 & a_x & a_y & a_z \end{bmatrix}$, and the measurement $\boldsymbol{s}$ is defined to be Earth's gravitational field $\boldsymbol{g} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$ in normalized quaternion form. The objective function is

$$f(\boldsymbol{q},\boldsymbol{d},\boldsymbol{s}) = \boldsymbol{q}^*\boldsymbol{d}\boldsymbol{q} - \boldsymbol{s} \tag{10}$$

The solution to the optimization problem $\boldsymbol{q}$ can be found by minimizing the objective function using a gradient descent algorithm.

$$\boldsymbol{q}_{\Delta,t} = \boldsymbol{q}_{\Delta,t-1} - \mu_t\frac{\nabla f}{||\nabla f||} \tag{11}$$

Where μ_t is the update rate defined as

$$\mu_t = \alpha||\dot{\boldsymbol{q}}_{\omega,t}||\Delta t$$

With $\alpha > 1$ being a parameter to offset the effects of noise present in the IMU data. The best choice of γ_t in the complementary filter Eq. 7 ensures that the divergence of $\boldsymbol{q}_{\omega,t}$ and the convergence of $\boldsymbol{q}_{\Delta,t}$ are equal so as to stabilize the algorithm. This optimal fusion is represented by

$$(1-\gamma_t)\beta = \gamma_t\frac{\mu_t}{\Delta t}$$

Where $\frac{\mu_t}{\Delta_t}$ can be seen as the convergence rate of $\boldsymbol{q}_{\Delta,t}$ and β can be seen as the divergence rate of $\boldsymbol{q}_{\omega,t}$. For large α, μ becomes large, making $\boldsymbol{q}_{\Delta,t-1}$ negligible in Eq. 11, such that

$$\boldsymbol{q}_{\Delta,t} \approx -\mu_t \frac{\nabla f}{||\nabla f||}$$

In this approximation, $\beta\gamma_t$ will become negligibly small compared to β, such that

$$\gamma_t \approx \frac{\beta \Delta t}{\mu_t}$$

Now using Eq. 7 and Eq. 8 with these approximations and the numerical integration for $\boldsymbol{q}_t$, we get

$$\boldsymbol{q}_t = \boldsymbol{q}_{t-1} + \left(\dot{\boldsymbol{q}}_{\omega,t} - \beta \frac{\nabla f}{||\nabla f||} \right) \Delta t \tag{12}$$

Assuming that $(1 - \gamma_t) \approx 1$.

Appendix C Bill of Materials

The bill of materials used for the design are found in Table 2. The materials were sourced from a combination of DigiKey, RoboShop, McMaster Carr and Amazon.

Appendix D Website and Repos

Website:
`https://jonah-gourlay44.github.io/gym2real/`

Gym2Real Repo (Hardware and software platform):
`https://github.com/jonah-gourlay44/gym2real`

Gym2Real-IsaacGym Repo (Simulation and reinforcement learning):
`https://github.com/kevinh42/gym2real_isaacgym`

TWIP CAD:
`https://cad.onshape.com/documents/430d4af740243dc0e842d2a6/w/42cfe57eb2477e6d1bc0bfd3/e/bfa82f8828e33e337c029dbe?renderMode=0&uiState=62508753c9046e2c4c613c4c`

Item	Category	Quantity
MPU-6050 Gyro and Acelerometer	SENSOR	1
Lynxmotion 12V 1:26.9 Brushed Motor + Encoder	ELEC	2
MD10C R3 - Motor Driver	ELEC	2
Ribbon Cable H3CCS-4006G	ELEC	1
Fuse Holder BK1-HTJ-606I	ELEC	3
Turnigy 1300mAh 3S 30C Lipo Pack	ELEC	2
Nylon XT60 Connectors	ELEC	2
HE1WPR/12 Terminal Block	ELEC	1
94669A104 Standoffs	MECH	24
Devantech 125mm Wheel	MECH	2
NEMA17 bracket	MECH	2
HCL8-12 Jumpers	ELEC	1
RA11131121 Switch	ELEC	2
Nvidia Jetson Nano Developer Kit	CONTROLLER	1
Micro USB to USB Cable	CONTROLLER	1
Micro SD Card (30 GB minimum)	CONTROLLER	1
2-56 $\times$ 5/8 bolt + lock washer + nut	MISC	2
2-56 $\times$ 1/2 bolt + lock washer + nut	MISC	8
M6 $\times$ 20 bolt + washer + nut	MISC	4
M3 $\times$ 8 bolt + lock washer + nut	MISC	8
M2.5 $\times$ 8 bolt + lock washer + nut	MISC	2

Table 2: Bill of materials for the the designed TWIP