

**Tumbling Robot Optimization Using a Learned Control
Policy**

**A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY**

Matthew Tlachac

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE**

Nikolaos Papanikolopoulos

December, 2021

© Matthew Tlachac 2021
ALL RIGHTS RESERVED

Acknowledgements

Foremost, I want to thank my advisor Nikolaos Papanikolopoulos for his constant guidance throughout my studies. He has supported me since my first year of college, and I wouldn't be where I am today without him. I deeply appreciate Vassilios Morellas and Junaed Sattar for serving on my committee, and for the many valuable lessons they've imparted to me.

I would also like to thank my labmates for their camaraderie over the years. Amalia Schwartzwald, Athanasios Bacharis, Luis Guzman, and Travis Henderson deserve special recognition for their invaluable contributions to the tumbling robot project.

Finally, I owe gratitude to my family and friends for their unending encouragement and love.

Abstract

Tumbling robots are simple, robust, and able to overcome large obstacles relative to their size. The specific body dimensions and leg lengths of tumbling platforms are often an uninformed choice. We introduce a framework for optimizing the parameters of a tumbling robot by referencing a control policy trained with deep reinforcement learning. We find a globally optimal tumbling robot in simulation, and provide proof for the real-world applicability of our framework by constructing and testing a physical robot based on our optimization results.

Contents

Acknowledgements	i
Abstract	ii
List of Tables	v
List of Figures	vi
1 Introduction	1
1.1 Related Works	2
2 Methodology	3
2.1 The Tumbling Robot Platform	3
2.2 Optimization Objective	4
2.3 Optimization Tasks	5
2.3.1 Setpoint Navigation	5
2.3.2 Flat Distance Traversal	5
2.3.3 Incline Climbing	6
2.4 Training Control Policies	6
2.5 Penalizing Power Usage	7
2.6 Optimization	11
3 Experiments and Results	12
3.1 The Universal Optimum	12
3.2 Sensitivity Analysis	14

3.3	The Real World	15
4	Conclusion	16
4.1	Discussion	16
4.2	Future Work	17
	References	18
	Appendix A. Power Usage Derivation	20
A.1	Full Derivation	20
A.2	Constants	24

List of Tables

3.1	Optimized robot parameters.	13
3.2	Sensitivity analysis	14
A.1	Robot constants	24

List of Figures

2.1	Tumbling robot with labeled axes	3
2.2	Terrain generated with Perlin noise	6
2.3	Network training progress	8
2.4	Tumbling robot geometry	8

Chapter 1

Introduction

In recent years, deep reinforcement learning has proved to be an effective means of generating a control policy for non-traditional robot platforms. For example, the tumbling locomotion explored in [1] was improved by applying reinforcement learning and simulation-to-real transfer in [2]. Tumbling platforms have many advantages, such as robustness and the ability to climb large obstacles relative to their size, and reinforcement learning helps address the difficulty of controlling them.

While a control policy learned in simulation can be applied to a physical robot, it is rarely used to inform the design of the robot itself. When a general robot strategy such as tumbling locomotion is chosen, the specific parameters of the fabricated robot - in our case, the body dimensions and leg length - are often an uninformed choice. We propose the use of a trained and fixed control policy to optimize robot performance while penalizing estimated power consumption by finding the best physical dimensions for several objectives, such as flat terrain traversal, hill climbing, and navigation accuracy.

Our contributions can be summarized as follows:

- We present a framework for optimizing physical robot parameters using a learned control policy.
- We discover an optimal tumbling robot by running experiments in simulation.
- We construct a physical tumbling robot based on our optimization results and prove that it outperforms an unoptimized platform.

1.1 Related Works

Automatic robot design is a relatively underexplored field. [3] was among the first to address the topic, and they demonstrate a procedure to design generic robots made of cylindrical bars and ball joints. [4] is even more generalized, allowing for a much wider variety of component types. The authors also evolve a control policy for the robot at the same time as the robot itself is evolved. These works are too automatic for our use case - rather than tackling robot design from scratch, we seek to optimize a robot that has been selected for characteristics like robustness and low cost.

[5] works on a more limited subset of robot designs, with arthropod-inspired robots that consist of multiple body segments and pairs of legs or wheels. Like our proposed approach, the robots can be designed with one or more specific environments in mind. However, this approach is limited to robots where a control policy can be developed using model predictive control, whereas we want to generalize to robots like the tumbling platform that require reinforcement learning to train a control policy.

This work relies heavily on [2], which introduced the sim-to-real pipeline for training tumbling bots to navigate with deep reinforcement learning.

Chapter 2

Methodology

2.1 The Tumbling Robot Platform

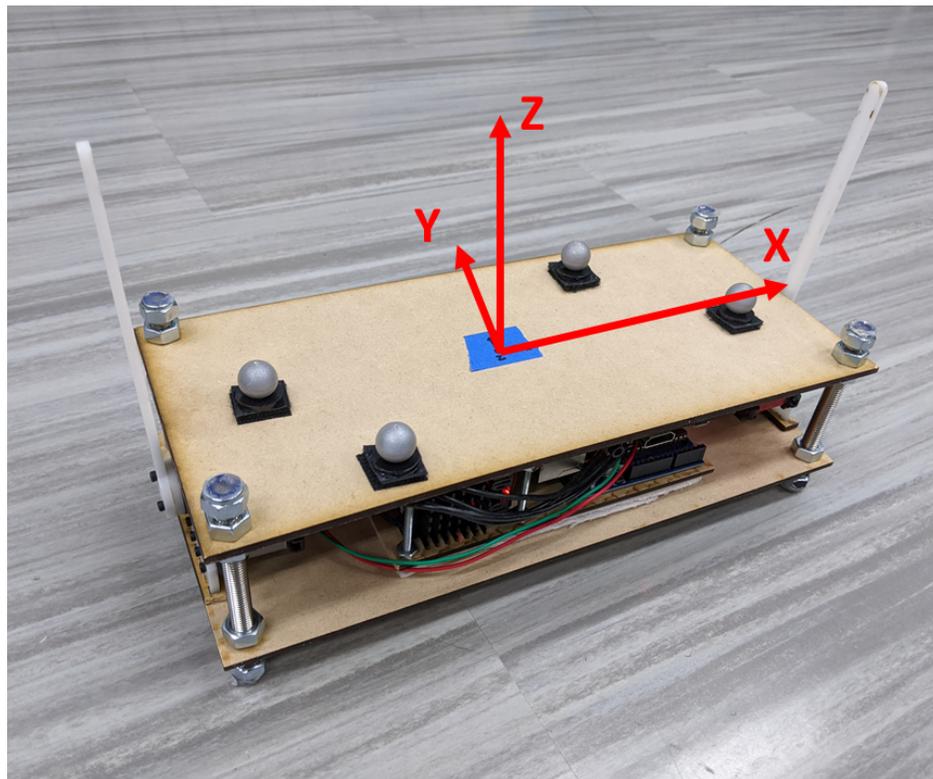


Figure 2.1: Tumbling robot with labeled axes.

Figure 2.1 shows a tumbling bot with labeled axes; these same axis labels will be used throughout the rest of this work. The bot has two legs that are fixed orthogonal to their shared axis of rotation, powered by continuous rotation servos. The outer robot shell is made of laser-cut MDF with threaded rods as offsets and laser-cut Delrin motor mounts and legs. The electronics are all contained within a small board in the center that can quickly be swapped between robot shells. This design allows for quick prototyping and easy construction and repair.

The robot is controlled in the same way as [2]. Spherical markers are affixed to both MDF plates so that the position and orientation of the robot can be tracked with a Vicon motion capture system. A Raspberry Pi on-board the robot communicates with a control computer remotely via the Robot Operating System. The remote computer runs inference for learned control policies using the Vicon tracking data and sends commands to the Raspberry Pi, which in turn commands an Arduino to move the motors.

2.2 Optimization Objective

In order to perform optimization, we must express our objective function. The optimization variable is defined as $\mathbf{w} = (x, y, z, \ell)$, where the elements denote the body dimensions as shown in Figure 2.1 and the leg length respectively. Suppose we have N different environment tasks with respect to which we want to optimize; these include objectives like navigating to a setpoint and climbing up an incline and are defined in section 2.3. Inference with our trained control policies (see section 2.4) produces an episode reward $R_i(\mathbf{w})$ for each $i \in [1..N]$.

The episode reward does not always fully encapsulate our idea of robot quality. It is simplest to train robots to perform as well as possible at a specific task, but for robot evaluation, we may want to consider factors like power expenditure; see section 2.5. Thus, we define wrapper functions $f_i(R_i(\mathbf{w}), \mathbf{w})$ to translate episode reward into an evaluation metric. We also define a weight λ_i for each term, allowing us to adjust the robot evaluation to a specific environment. When designing a robot that should have to traverse in a mostly flat environment, for example, we would set the weight corresponding to the incline-climbing task low. Finally, our environment and trained policies have stochastic elements, so for each \mathbf{w} , we run a batch of size B and average

the results. This leaves us with the following optimization objective:

$$\arg \max_{\mathbf{w}} \frac{1}{B} \sum_{b=1}^B \sum_{i=1}^N \lambda_i f_i(R_i(\mathbf{w}), \mathbf{w}).$$

2.3 Optimization Tasks

Here we define each of the specific environment tasks used for our experiments. Note that all of the reward expressions R presented below are functions of the optimization parameters \mathbf{w} . Note also that the environments are designed with a specific direction or location in mind (for example, the x axis or the origin) without loss of generality, since a simple frame transformation can be used to apply these environments to any related task a robot may encounter.

2.3.1 Setpoint Navigation

The first task is the navigation environment from [2]. It spawns a robot at a random distance from the origin in a random direction. It uses the reward function

$$R = \frac{1}{1 + \sqrt{v_x + v_y}} - \sqrt{p_x^2 + p_y^2}$$

where p_x and p_y are the components of position and v_x and v_y are the components of velocity in the plane. This is designed to reward the robot for getting close to its goal at the origin while penalizing repeatedly flipping over the goal once it is close.

This task can be performed on flat ground, and is also trained and evaluated on randomized terrain. The terrain, an example of which is shown in Figure 2.2, is comprised of tiles with varying heights generated using Perlin noise.

2.3.2 Flat Distance Traversal

This environment spawns the robot at the origin and provides a reward of

$$R = p_x - |p_y|,$$

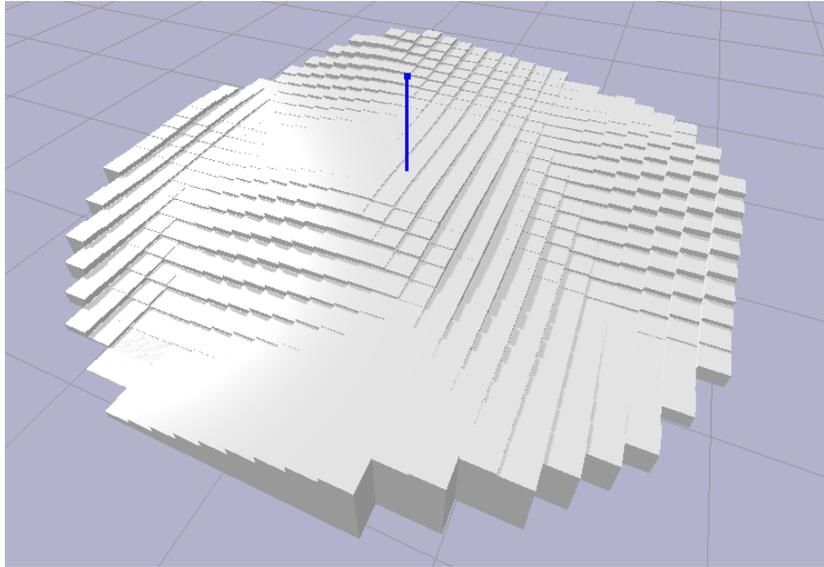


Figure 2.2: Terrain generated with Perlin noise. Rendered in the PyBullet simulator.

which simply rewards distance in the positive x direction while penalizing any deviation in the y direction. It is advantageous for a robot to move in a specified direction as directly as possible, though precise motion along a line is difficult for a tumbling platform.

2.3.3 Incline Climbing

The last environment includes a 15° incline starting at the origin and increasing in height along the positive x direction. The robot starts at the origin and is simply rewarded for distance traveled in the x direction; namely $R = p_x$. Hill climbing is a difficult task compared to flat ground traversal, so we reward distance achieved up the hill regardless of sideways motion that is also performed to make it happen.

2.4 Training Control Policies

We train a separate control policy for each environment task, and this section describes the aspects of the reinforcement learning process that are common across each environment. Each task is trained for 20 environment steps per episode, with each step

corresponding to one second of real time. At each step, the actions of the control policy determine the speed of each motor. The action space is discretized to simplify training, so for each motor, the control policy chooses between going full speed in either direction or staying still. The robot observes its 3D position, orientation quaternion, and the last action sent to the motors; note that this deliberately does not include angle position feedback from the motors, which reduces cost and complexity of the physical platform. Rather than goal-conditioning the policies, we train them with respect to a specific direction or location with the knowledge that we can perform frame transformations to apply the networks in any context (see section 2.3).

The control policies are trained in a custom OpenAI Gym [6] environment written using PyBullet [7] as the simulator. We use Proximal Policy Optimization [8] for training, the implementation of which comes from stable baselines [9]. Training was performed in 80 parallel environments and ran for about two days per network. Training progress is visualized in Figure 2.3, with plots generated using Tensorboard from Tensorflow [10].

Domain randomization [11] is used on the optimization variables in order to produce control policies that are effective for any robot geometry tested during optimization and evaluation. It is also applied to other parameters in the environment such as component masses, inertial matrices, and coefficients of friction, to learn robust policies that generalize to the real world as well as possible.

2.5 Penalizing Power Usage

As discussed in section 2.2, we want to define a wrapper function around the episode reward that penalizes power expenditure. This will allow us to optimize for a robot design that is more practically applicable in the real world, as it could run longer without needing to be charged. It also addresses an issue in some environments such as the flat distance traversal, where scaling up all the parameters of a robot design proportionally should always produce a better-performing robot. To design this wrapper function, we must first come up with an estimation for power usage based on the physical parameters of the robot. We choose to estimate the electrical power consumed by the system per unit of distance traveled (watts per meter).

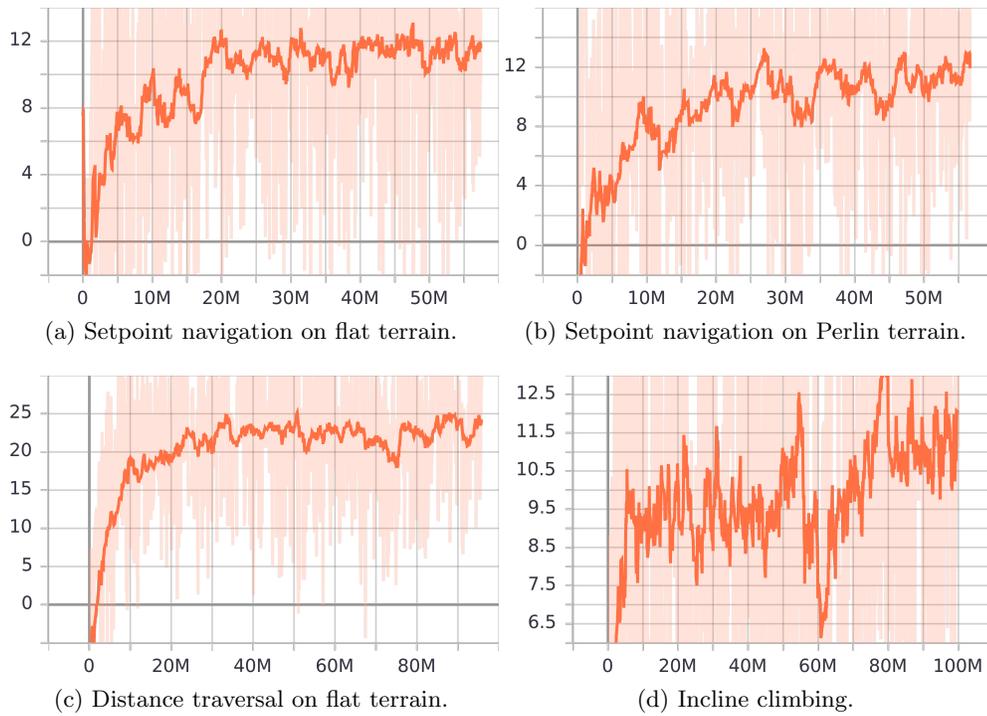


Figure 2.3: Episode reward vs environment steps during training. Light orange denotes the raw data, which is smoothed to produce the darker line.

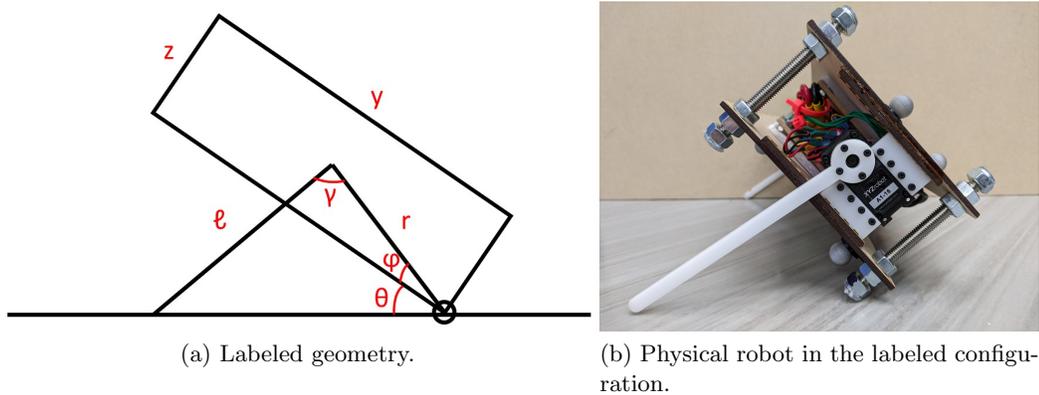


Figure 2.4: 2D geometry of the tumbling bot during a tumble.

We provide a brief description of the math here, with a full derivation available in the appendix. We first estimate the power consumed for the robot to lift itself off the ground in the simplest configuration, where both legs are pushing on the ground in tandem and we assume the bottom corner of the robot is a fixed axis about which the rotation happens; see Figure 2.4 for a side view of this situation.

The power consumption of each motor is estimated to be

$$P = \left(\frac{\tau}{k_e}\right)^2 R + \dot{\gamma}\tau,$$

where τ is the torque exerted on the motor shaft and $\dot{\gamma}$ is our fixed leg rotation speed. Constants such as back emf constant k_e and internal motor resistance R are reported in the appendix as well.

We simplify the torque estimation by modeling our system as a single motor with all the torque on it. See Figure 2.4 for angle and distance labels. At any given θ , the only torques acting on the system are the torque exerted by the leg and torque due to gravity, both of which act at the motor axis. The sum of torques gives us the equation

$$\begin{aligned} -\tau + \|\mathbf{r} \times (-m\mathbf{g})\| &= I\ddot{\theta} \\ \tau &= \|\mathbf{r}\| \|m\mathbf{g}\| \sin(\theta + \phi + \frac{\pi}{2}) - I\ddot{\theta}. \end{aligned}$$

Our moment of inertia is given by

$$I = m \left(r^2 + \frac{y^2 + z^2}{12} \right)$$

and we can easily estimate the robot mass m from our robot parameters by finding masses and densities of individual components.

Our only non-constants left in the torque equation are θ and $\ddot{\theta}$. We don't directly know these, but we do have a fixed leg rotation speed $\dot{\gamma}$ and therefore $\ddot{\gamma} = 0$. We can express θ in terms of γ (and constants) as

$$\theta = \arcsin \left(\frac{\ell \sin(\gamma)}{\sqrt{\ell^2 + r^2 - 2\ell r \cos(\gamma)}} \right) - \phi.$$

Note that we likewise know

$$\gamma = \pi - \arcsin\left(\frac{r \sin(\theta + \phi)}{\ell}\right) - \theta - \phi.$$

We then calculate the second derivative of this expression of θ to get $\ddot{\theta}$ in terms of our known $\dot{\gamma}$ and γ , finally giving us an expression for $\tau(\theta)$. These expressions are prohibitively long to report here and are therefore relegated to the appendix.

We can now calculate the power required to lift the robot from flat on the ground to the center of inertia being above the axis of rotation:

$$P = \int_0^{\pi/2 - \phi} \left(\frac{\tau(\theta)}{k_e}\right)^2 R + \dot{\gamma}\tau(\theta) d\theta.$$

We repeat this calculation starting in the tall configuration, where the only difference is the value of ϕ , and sum these two power figures together to get the total power consumed for a 180° rotation of the robot. Practically, we can't calculate a closed form for this integral, so we perform numerical integration when we evaluate it.

Displacement during that rotation is estimated as $d = y + z$ assuming the robot doesn't slide after a tumble, such as when it is climbing a very steep hill. On flat ground, given coefficient of friction μ between the ground and the robot, we have

$$d = y + z + \frac{1}{\mu} \left(2\sqrt{\left(\frac{y}{2}\right)^2 + \left(\frac{z}{2}\right)^2} - \frac{y}{2} - \frac{z}{2} \right).$$

Our final expression for power expenditure in watts per meter is given by $e = \frac{P}{d}$.

Our basic wrapper function to penalize power usage would look like $\lambda(R(\mathbf{w}), \mathbf{w}) = \frac{R(\mathbf{w})}{e(\mathbf{w})}$. However, depending on which term has greater variance, this may effectively optimize for the best performing robot or the most energy efficient one rather than a mix of the two. Note that $e(\mathbf{w}) > 0$. To add a hyperparameter $\xi > 0$ that lets us balance the relative importance of these terms, we instead formulate our wrapper as

$$f(R(\mathbf{w}), \mathbf{w}) = \frac{R(\mathbf{w})}{(e(\mathbf{w}) + 1)^\xi}.$$

2.6 Optimization

The first-order optimality condition is a common way to approach optimization problems. In this case the functions $R_i(\mathbf{w})$ are not known, so we can't calculate their derivatives. Even if we factor in the forward pass from the neural networks that are used to produce R_i , we can't feasibly take derivatives of each value calculated by PyBullet when simulating the motion of the robot over the course of an episode. Therefore, we consider our function to be a black box.

There are also several sources of randomness that affect when we evaluate the performance of a set of robot parameters. The PyBullet simulation itself has some stochastic elements. We also introduce some randomization to the motor commands to better simulate real motors. Bayesian optimization also starts with a prior based on random samples. We average a batch of robot trials to mitigate this. Because we are running 80 parallel environments, it is most efficient to make this batch size a multiple of 80. Thus, we set the batch size to 240 to balance between consistency of results and speed of optimization.

When it comes to optimizing a stochastic black box function that is expensive to evaluate, Bayesian optimization is a natural choice. We use the expected improvement acquisition function and run it for 1000 iterations with 1000 initial random samples.

Our optimization lower bounds are based on the smallest robot frame that can fit the modular electronics board (see section 2.1). The upper bounds are selected such that the largest possible bot can still perform a typical tumble with just one of the motors and is reasonably portable. They are also designed to minimize complications with the physical system; namely, the Delrin legs and motor mounts flex more the longer they get. The final bounds are given in meters: $x \in [0.189, 0.3]$, $y \in [0.102, 0.25]$, $z \in [0.084, 0.2]$, and $\ell \in [0.066, 0.25]$.

Chapter 3

Experiments and Results

3.1 The Universal Optimum

The original experiment design involved selecting two or more environments, optimizing one robot specifically to each environment, optimizing a third robot balanced between the two environments, and then evaluating the performances of these robots. We hypothesized that the robot designed for a specific environment would perform best on that environment, followed by the balanced robot, followed by the robot designed for the other environment.

Running Bayesian optimization on each task produced an interesting and unexpected result: the exact same robot was found to be optimal across all three environments, with or without terrain in the navigation task. This pattern persisted across values of ξ , the scaling factor of the power-consumption penalty. For each ξ , there is a single robot that is best at maneuvering to the extent that it is optimal for each different task.

To demonstrate the effect that ξ has on this optimal robot, Table 3.1 reports the results of optimizing the flat setpoint navigation task for several values of ξ . We include the optimal bot within the bounds considering no energy penalty ($\xi = 0$) and the default $\xi = 1$. We include $\xi = 3$ as the first integer scaling factor that produces a bot which is significantly different than the $\xi = 0$ bot, as well as being the first bot that can tumble unrestricted in the real world (see section 3.3). Finally, we examine what happens when we optimize exclusively with respect to estimated watts per meter, represented in the table as $\xi = \infty$. The column R denotes the average episode reward and e shows the

estimated power expenditure. We perform a t-test between the reward values of the first row and every subsequent row to determine if the differences in the reward are statistically significant; the p-values from these tests are reported in the p column.

The most important result to note is that, considering a typical t-test threshold of $p = 0.05$, none of these robots have a significantly different performance than the bot that was optimized with no energy penalty. This indicates that there is a large plateau in the objective function and that there are a wide variety of equally optimal tumbling robot platforms. Given that there is no significant loss in performance, we can conclude that the best overall tumbling robot platform happens to be the one that doesn't consider reward and only optimizes for power efficiency in terms of watts per meter. Thus, we have a globally optimal tumbling robot.

There are several additional insights to glean from this data. The y and z dimensions are interchangeable when power usage isn't factored in, but there is a pronounced difference between these parameters for $\xi = 0$, and this difference only closes as power usage is penalized more heavily. This implies that having a rectangular side profile rather than a square one is generally beneficial to robot performance. x is always near the maximum when robot performance is taken into account; it drops to the minimum for $\xi = \infty$ which is logical, as x only contributes to robot mass in the power estimation. While x and y are approximately equal to the optimization lower bounds for $\xi = \infty$, z and ℓ are not, which confirms that we have not simply optimized for the smallest and lightest possible robot. Interestingly, from $\xi = 3$ to $\xi = \infty$, x , y , and z all decrease, but ℓ actually increases.

ξ	x (m)	y (m)	z (m)	ℓ (m)	R	e (W/m)	p
0	0.293	0.117	0.158	0.218	0.854	6.05	-
1	0.285	0.117	0.158	0.218	0.857	6.00	0.33
3	0.292	0.129	0.098	0.136	0.870	5.20	0.08
∞	0.189	0.104	0.098	0.152	0.853	3.76	0.53

Table 3.1: Robot parameters and performance in the flat setpoint navigation task, optimized with different ξ .

3.2 Sensitivity Analysis

In order to explore more about this ostensible plateau of robot performance, we perform a brief sensitivity analysis. Our default robot is the one optimized with $\xi = 0$, and each evaluation is done on the flat setpoint navigation task. For each parameter, a perturbation of one inch in either direction has no statistically significant effect on the resulting reward. Thus, Table 3.2 shows the results for perturbations of 0.1 meters. As before, R denotes the reward and p denotes the p-value of a t-test comparing the modified robot to the base robot of $\xi = 0$. An asterisk denotes a row that results in a value outside of the bounds of integration, and p-values that fall under the threshold of 0.05 are bolded to highlight rows that are statistically significant.

This analysis reveals a few interesting patterns. Increasing any of the body parameters does not have a statistically significant effect on the performance of the robot, while decreasing each of them does result in a significant drop in performance. We also see that ℓ is a special case, in that making it larger or smaller results in worse performance. Thus, we can conclude that ℓ is the most finely-tuned parameter, and the dimension for which the plateau is the smallest. Reducing x results in the largest drop in performance that we observe, so the drop-off in reward may be steepest in that direction.

Perturbation	R	p
None	0.854	-
$+x^*$	0.869	0.46
$-x$	0.785	1.9e-06
$+y$	0.868	0.22
$-y^*$	0.837	6.9e-4
$+z^*$	0.843	0.278
$-z^*$	0.827	8.9e-4
$+\ell^*$	0.823	0.019
$-\ell$	0.813	3.7e-4

Table 3.2: Sensitivity analysis of robot performance, with perturbations of 0.1m in each parameter. Asterisks denote a parameter that falls outside the bounds of optimization.

3.3 The Real World

As discussed in section 3.1, the original experiment would have necessitated building three robots and comparing performance between them. Once it was discovered that we had globally optimal robots across the different environments, we had to adjust the experiment design. The new plan was to construct the optimally-performing robot ($\xi = 0$), run it for the flat setpoint navigation task, and compare the results to the corresponding data collected in [2] for an unoptimized tumbling platform. After constructing this new optimal robot, we ran into another setback. There is an emergent behavior arising from the deep reinforcement learning of the control policy wherein the robot forces both legs down into the ground simultaneously in opposite directions; this allows the robot to quickly achieve a significant angular change without a corresponding position displacement, albeit imprecisely. We did not account for this maneuver when we set the upper optimization bounds by estimating the torque required to lift the robot. Thus, the motors on the newly constructed bot had insufficient torque to perform the emergent behavior. To solve this, we increased ξ until we found a robot at $\xi = 3$ that could perform this motion with our existing motors.

In [2], 400 physical trials were performed starting from 5 locations around a circle 1.5m away from the origin. 100 of these trials were on flat ground and used a control policy trained for flat ground, and are therefore directly comparable to our approach here. We ran 50 similar trials, 10 from each location, with our new optimized platform. We achieved an average setpoint error of 0.330 m with the optimized robot, compared to the 0.433 m of the unoptimized robot. This result is significant, with a p-value of 0.038. It's worth noting the estimated watts per meter of each robot: the unoptimized robot has 4.97 W/m, which slightly out-performs the 5.20 W/m of the optimized bot. This is an expected result, as it is still significantly higher than the energy-optimal bot ($\xi = \infty$) with 3.76 W/m.

Chapter 4

Conclusion

4.1 Discussion

We set out to create a pipeline for optimizing tumbling robots based on a description of the environment in which they would run. In doing so, we discovered a globally optimal robot: one that consumes the minimum amount of power while performing statistically no worse than the best possible robots within our bounds of optimization. This performance is consistent across a variety of environments: navigation to a setpoint across flat ground and terrain generated with Perlin noise, traversal across flat ground along a straight line, and incline climbing.

We analyzed the plateau around the optimal robots and achieved some insights as to what makes tumbling effective. We found that wide robots and robots with a rectangular cross-section tend to be best at tumbling, unless power consumption is being heavily penalized. We also found that perturbing leg length is more likely to result in significantly worse performance, but that the sharpest declines in performance compared to the optimal robot comes from lowering the width of the bot.

We constructed one such robot with optimal performance to prove that these results extend well to the real world. We found that the optimized robot performed significantly better on the flat setpoint navigation task than a robot that was constructed before optimization had been attempted.

4.2 Future Work

Several improvements would allow the tumbling robot platform to run in the real world, rather than a lab setting. Achieving robust state estimation for a tumbling platform with an inertial measurement unit rather than the Vicon system remains an open problem. If the robot had more powerful on-board computing resources, it could run inference locally, rather than needing to connect to a nearby computer with more processing power. If the robot could reason about the environment around it, it could theoretically navigate through a complex environment by continually referencing different pre-trained control policies, based on whatever task best mirrors the immediate surroundings.

Due to time constraints, we had to settle for estimating the power consumption of the physical robots with the same equation used for the simulated robots; a much more accurate solution would have been to add electronics that directly measure power consumption.

As far as our optimization framework is concerned, we would ideally create more diverse environments to see if one robot remains optimal, or if we can specialize robots for different environments. Some additional ideas for tasks, though much harder to simulate, include navigation through small pieces of rubble or deformable terrain such as sand or snow. Finally, our optimization pipeline could be applied to any robot platform where certain design choices are fixed and the specific geometric parameters have a significant impact on performance. For example, someone seeking to create a bipedal robot with joints mimicking those of a human may be able to apply this optimization framework to figure out what link lengths lend themselves best to successful walking.

References

- [1] Brett Hemes, Dario Canelon, J. Danes, and Nikolaos Papanikolopoulos. Robotic tumbling locomotion. pages 5063 – 5069, 06 2011.
- [2] Amalia Schwartzwald, Matthew Tlachac, Luis Guzman, Athanasios Bacharis, and Nikolaos Papanikolopoulos. Tumbling robot control using reinforcement learning. *IEEE Robotics and Automation Magazine*, in review.
- [3] Hod Lipson and Jordan Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–8, 09 2000.
- [4] Tingwu Wang, Yuhao Zhou, Sanja Fidler, and Jimmy Ba. Neural graph evolution: Towards efficient automatic robot design, 2019, 1906.05370.
- [5] Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Robogrammar: Graph grammar for terrain-optimized robot design. *ACM Trans. Graph.*, 39(6), November 2020.
- [6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016, 1606.01540.
- [7] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. <http://pybullet.org>, 2016–2019.
- [8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017, 1707.06347.

- [9] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. <https://github.com/hill-a/stable-baselines>, 2018.
- [10] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [11] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017, 1703.06907.

Appendix A

Power Usage Derivation

A.1 Full Derivation

Here, we provide the full derivation for power usage per unit distance traveled. The equations from the main body of the paper are repeated here along with additional context, justification, and results.

We first estimate the power consumed for the robot to lift itself off the ground in the simplest configuration, where both legs are pushing on the ground in tandem and we assume the bottom corner of the robot is a fixed axis about which the rotation happens; see Figure 2.4 for a side view of this situation.

We assume that we have a fixed leg rotation speed $\dot{\gamma}$. $\dot{\gamma}$ is related to V_{emf} , the voltage drop across the motor, by

$$V_{emf} = k_e \dot{\gamma},$$

where k_e is the motor back EMF constant. Because $\ddot{\gamma} = 0$, load torque and motor torque are equal, so we represent them both with τ . τ is likewise related to the current in the motor circuit i by

$$\tau = k_e i.$$

For an internal motor resistance R and input voltage V_{in} , Kirchoff's loop rule and Ohm's

law give us

$$\begin{aligned}
 V_{in} - iR - V_{emf} &= 0 \\
 V_{in} &= iR + k_e \dot{\gamma} \\
 V_{in} i &= i^2 R + k_e \dot{\gamma} i \\
 P &= \left(\frac{\tau}{k_e} \right)^2 R + \left(\frac{\tau}{k_e} \right) k_e \dot{\gamma} \\
 &= \left(\frac{\tau}{k_e} \right)^2 R + \dot{\gamma} \tau,
 \end{aligned}$$

which is the instantaneous power consumption of each motor.

We simplify the torque estimation by modeling our system as a single motor with all the torque on it. See Figure 2.4 for angle and distance labels. At any given θ , the only torques acting on the system are the torque exerted by the leg and torque due to gravity, both of which act at the motor axis. The sum of torques gives us the equation

$$\begin{aligned}
 -\tau + \|\mathbf{r} \times (-m\mathbf{g})\| &= I\ddot{\theta} \\
 \tau &= \|\mathbf{r}\| \|m\mathbf{g}\| \sin\left(\theta + \phi + \frac{\pi}{2}\right) - I\ddot{\theta}.
 \end{aligned}$$

Assuming our robot has uniformly distributed mass, the moment of inertia of the robot about the leg rotational axis (which passes through the center of mass) is given by

$$I = m \left(\frac{y^2 + z^2}{12} \right).$$

The parallel axis theorem then tells us that the moment of inertia r away at the rotational axis of the robot is given by

$$I = m \left(r^2 + \frac{y^2 + z^2}{12} \right).$$

We can easily estimate the robot mass m from our robot parameters by finding masses and densities of individual components.

Our only non-constants left in the torque equation are θ and $\ddot{\theta}$. We don't directly know these, but we do have a fixed leg rotation speed $\dot{\gamma}$ and therefore $\ddot{\gamma} = 0$. We can

calculate the base of our triangle using the law of cosines:

$$b = \sqrt{\ell^2 + r^2 - 2\ell r \cos(\gamma)}.$$

We can then express θ in terms of γ (and constants) using the law of sines:

$$\begin{aligned}\theta + \phi &= \arcsin\left(\frac{\ell \sin(\gamma)}{b}\right) \\ \theta &= \arcsin\left(\frac{\ell \sin(\gamma)}{\sqrt{\ell^2 + r^2 - 2\ell r \cos(\gamma)}}\right) - \phi.\end{aligned}$$

We can use the law of sines on the unlabeled angle in our triangle to easily show

$$\gamma = \pi - \arcsin\left(\frac{r \sin(\theta + \phi)}{\ell}\right) - \theta - \phi.$$

We can now calculate $\dot{\theta}$ and $\ddot{\theta}$ in terms of our know $\dot{\gamma}$ and γ , which we can substitute in to give us $\dot{\theta}$ and $\ddot{\theta}$ in terms of θ . For notational simplicity, let us define

$$\begin{aligned}v &= \ell^2 + r^2 - 2\ell r \cos(\gamma) \\ A &= \left(1 - \frac{\ell^2 \sin^2(\gamma)}{v}\right)^{-\frac{1}{2}} \\ B &= \ell \cos(\gamma) v^{-\frac{1}{2}} \\ C &= \ell^2 r \sin^2(\gamma) v^{-\frac{3}{2}}\end{aligned}$$

and the derivatives

$$\begin{aligned}\dot{A} &= \dot{\gamma} \left(1 - \frac{\ell^2 \sin^2(\gamma)}{v}\right)^{-\frac{3}{2}} \frac{v (\ell^2 \sin(\gamma) \cos(\gamma)) - (\ell^3 r \sin^3(\gamma))}{(v)^2} \\ \dot{B} &= -\ell \dot{\gamma} \sin(\gamma) v^{-\frac{1}{2}} - \ell^2 r \dot{\gamma} \sin(\gamma) \cos(\gamma) v^{-\frac{3}{2}} \\ \dot{C} &= 2\ell r \dot{\gamma} \sin(\gamma) \cos(\gamma) v^{-\frac{3}{2}} - 3\ell^3 r^2 \dot{\gamma} \sin^3(\gamma) v^{-\frac{5}{2}}.\end{aligned}$$

We can then express

$$\begin{aligned}\dot{\theta} &= A(B - C)\dot{\gamma} \\ \ddot{\theta} &= A(\dot{B} - \dot{C})\dot{\gamma} + \dot{A}(B - C)\dot{\gamma}\end{aligned}$$

which finally gives us an expression for $\tau(\theta)$.

We can now calculate the power required to lift the robot from flat on the ground to the center of inertia being above the axis of rotation:

$$P = \int_0^{\pi/2 - \phi} \left(\frac{\tau(\theta)}{k_e} \right)^2 R + \dot{\gamma}\tau(\theta) d\theta.$$

We repeat this calculation starting in the tall configuration, where the only difference is the value of ϕ , and sum these two power figures together to get the total power consumed for a 180° rotation of the robot. Practically, we can't calculate a closed form for this integral, so we perform numerical integration when we evaluate it.

Displacement during that rotation is estimated as $d = y + z$ assuming the robot doesn't slide after a tumble, such as when it is climbing a very steep hill. On flat ground, given coefficient of friction μ between the ground and the robot, we assume that all of the potential energy from height lost during a tumble is converted to kinetic energy that is dissipated during sliding. Starting when the robot center of mass is directly above the corner of the robot about which we are rotating and we are tumbling from y being vertical to z being vertical, we have

$$\begin{aligned}\Delta u &= mg \left(\sqrt{\left(\frac{y}{2}\right) + \left(\frac{z}{2}\right)} - \frac{z}{2} \right) \\ &= \frac{1}{2}mv_0^2 \\ v_0^2 &= 2g \left(\sqrt{\left(\frac{y}{2}\right) + \left(\frac{z}{2}\right)} - \frac{z}{2} \right).\end{aligned}$$

The frictional force of $F_f = -\mu mg$ gives us a horizontal acceleration during sliding of $a = -\mu g$. We can use kinematics to get a horizontal displacement as the robot comes

to a stop:

$$\begin{aligned}
 v^2 &= v_0^2 + 2a\Delta x \\
 0 &= 2g \left(\sqrt{\left(\frac{y}{2}\right) + \left(\frac{z}{2}\right)} - \frac{z}{2} \right) - 2\mu g\Delta x \\
 \Delta x &= \frac{1}{\mu} \left(\sqrt{\left(\frac{y}{2}\right) + \left(\frac{z}{2}\right)} - \frac{z}{2} \right).
 \end{aligned}$$

We have a similar term for the other half of the 180° rotation:

$$\Delta x = \frac{1}{\mu} \left(\sqrt{\left(\frac{y}{2}\right) + \left(\frac{z}{2}\right)} - \frac{y}{2} \right).$$

Summing these two terms with the fixed movement of $y + z$ gives us a total of

$$d = y + z + \frac{1}{\mu} \left(2\sqrt{\left(\frac{y}{2}\right) + \left(\frac{z}{2}\right)} - \frac{y}{2} - \frac{z}{2} \right).$$

Our final expression for power expenditure in watts per meter is given by $e = \frac{P}{d}$.

A.2 Constants

Name	Value	Unit
$\dot{\gamma}$	3.67	rad/s
k_e	1.64	
μ	0.645	
R	16.96	Ω

Table A.1: Constants used in the power consumption estimation derivation.