ROWDY RUNNER II: AN INDEPENDENTLY ACTUATED RIMLESS WHEEL ROBOT

by

ERIC SEBASTIAN SANCHEZ, B.Sc.

THESIS
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

MASTER'S OF SCIENCE IN MECHANICAL ENGINEERING

COMMITTEE MEMBERS:
Pranav Bhounsule, Ph.D., Chair
R. Lyle Hood, Ph.D.
HungDa Wan, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Engineering
Department of Mechanical Engineering
December 2018

DEDICATION

I would like to dedicate this thesis to my parents Eric and Lya. Thank you.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Pranav Bhounsule for the help and guidance all of these years. It is thanks to you that I got into robotics. Thank you all of the RAM Lab members, especially Ezra and Robert, without whom I would have gone crazy during graduate school. Lastly, thank you mom and dad, for your continued support and understanding.

December 2018

ROWDY RUNNER II: AN INDEPENDENTLY ACTUATED RIMLESS WHEEL ROBOT

Eric Sebastian Sanchez, M.Sc.
The University of Texas at San Antonio, 2018

Supervising Professor: Pranav Bhounsule, Ph.D.

Wheeled robots are energy efficient but cannot travel across uneven terrain. Legged robots, however, can traverse rough terrain, but are much more energy expensive. Rimless wheels are a wheel-leg hybrid that blends the benefits of both together. This thesis presents the design and control of a rimless wheel robot that achieves straight line walking and turning. The robot consists of a center body and two rimless wheels, each with 10 dampened spokes. Each wheel is axially connected to the body through a shaft. Each motor is connected to a shaft through two pulleys and a belt, resulting in a reduction of 5.4:1 from the motor to the wheel. The body houses the sensor, computer, microcontroller, motors and encoders, motor controller, and batteries. An inertial measurement unit is used to measure the angle of the body with respect to the vertical axis. The robot is controlled hierarchically; on the top is the Raspberry Pi, which reads sensor data, communicates with the microcontroller, and collects data. In the middle is the Teensy microcontroller, which collects the sensor data and calculates an output using a proportional-integral-derivative controller. On the bottom is the motor controller, which receives the output from the Teensy and moves the motors accordingly. The controller on microprocessor ensures the body maintains a steady pitch. The steady pitch angle propels the robot forward. To achieve turning, a differential current was applied, either added or subtracted, to each motor. The robot is able to achieve a steady state speed of 1 m/s with an energy usage (power per unit weight per unit velocity) of 0.13, about half the energy consumption of a human walking. The robot is also capable of sharp turns with a radius of approximately 0.5 meters.

TABLE OF CONTENTS

Acknowledgements . . . . . **iii**

Abstract . . . . . **iv**

List of Tables . . . . . **vii**

List of Figures . . . . . **viii**

Chapter 1: Introduction **1**

1.1 Literature Review 2

Chapter 2: Design **3**

2.1 Mechanical 3

2.1.1 Rimless Wheels 3

2.1.2 Torso 4

2.2 Electrical 6

2.2.1 Computation 6

2.2.2 Motor and Motor Controller 7

2.2.3 Sensors 8

2.2.4 Joystick and Batteries 8

2.2.5 Miscellaneous 9

2.3 Controller 9

2.3.1 Torso Pitch Controller 9

2.3.2 Turning 12

2.4 Software 13

2.4.1 Raspberry Pi 14

2.4.2 Teensy 15

Chapter 3: Testing and Results 17

3.1 Pitch Control 17

3.1.1 Testing Approach 17

3.1.2 Results 17

3.2 Turning 18

3.2.1 Testing Approach 18

3.2.2 Results 19

3.3 Cost of Transport 20

3.3.1 Testing Approach 20

3.3.2 Results 20

3.4 Maximum Speed 21

3.4.1 Testing Approach 21

3.4.2 Results 21

3.5 Future Work 22

Chapter 4: Conclusion 23

Appendix A: Code 24

Bibliography 25

Vita

LIST OF TABLES

Table 2.1 PID gains 12

Table 3.1 Cost of Transport and Velocities on Different Surfaces 20

Table 3.2 Cost of Transport Breakdown for Concrete 21

LIST OF FIGURES

Figure 1.1 Rimless wheel on a slope 1

Figure 2.1 Rowdy Runner II 3
Figure 2.2 Assembled rimless wheel 3
Figure 2.3 Rimless wheel hub 3
Figure 2.4 Rowdy Runner II torso 4
Figure 2.5 Isolated power transfer mechanism 4
Figure 2.6 Closeup of motor assembly 5
Figure 2.7 Raspberry Pi 6
Figure 2.8 Teensy 3.2 microcontroller 6
Figure 2.9 ODrive 3.5 and motors 7
Figure 2.10 BNO055 absolute orientation sensor 8
Figure 2.11 Dualshock 3 controller 9
Figure 2.12 LiPo battery 9
Figure 2.13 Free body diagram of torso (only one leg shown) 9
Figure 2.14 PID control diagram (discrete time) 10
Figure 2.15 Graphical view of the turning method 12
Figure 2.16 Software hierarchy 13
Figure 2.17 Communication diagram 14
Figure 2.18 Raspberry Pi software diagram 14
Figure 2.19 Teensy software diagram 15

Figure 3.1 Physical Rowdy Runner II 17
Figure 3.2 Pitch and setpoint over time 18
Figure 3.3 Velocity (radians) over time 18
Figure 3.4 Rowdy Runner turning radius 19

Figure 3.5 Rowdy Runner trial trajectory 19

CHAPTER 1: INTRODUCTION

Wheeled robots have been around since the late 1940's [1]. They offer plenty of advantages, such as many commercial options and detailed control literature [2]. However, wheeled robots have one downfall: terrain. They cannot traverse uneven terrain, such as rocky outcrops or hills, without using large, heavy wheels. Another solution for traversing terrain is legged robots. Boston Dynamics' Big Dog [3] and Little Dog [4] are quadruped robots that are robust, have multiple locomotion gaits, and can walk in multiple environments. These robots are unfortunately power hungry: Big Dog has a combustion engine onboard, and Little Dog has a walking time of just 30 minutes [4].

A rimless wheel is a similar to a wagon wheel, with the circular rim removed (Figure 1.1). Instead of having a rolling surface that contacts the ground, the robot has many radial legs that touch the ground as it rotates (Figure 1.1). This makes its behavior with the ground like taking steps, rather than rolling, allowing the robot to traverse rough terrain. Because legs are arranged in such a way that they emulate a wheel, a rimless wheel can evade the energy costs from traditional legged walking.

Passive rimless wheels were first explored by McGeer [5] when examining passive bipedal locomotion. In the early work, the rimless wheel robot was just simple links connected and coupled together [6] that was rolled down an incline. There are now powered rimless wheeled robots

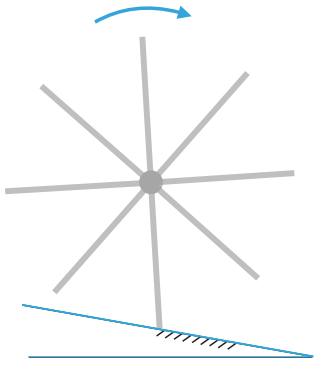

Figure 1.1: Rimless wheel on a slope

capable of stepping over large obstacles, such as curbs [7], and even accompanying traditional rovers on missions to Mars [8, 9].

1.1 Literature Review

McGeer determined motion of a single rimless wheel going downhill as having an equilibrium speed that increased as the slope increased [5]. This is due to the collisions from each step taken reducing the energy of the system. The single wheel model was expanded into a three dimensional model by Coleman, Chatterjee, & Ruina [10]; the stability of a single rolling rimless wheel was not determined to be asymptotically stable in three dimensions as the energy dissipation was not enough to stabilize. Smith & Berkmemeier [11] then extended the scope from a single wheel to a model of two rimless wheels with a fixed width.

Recent work on rimless wheels has been on the previously mentioned rimless wheel, Axel, by Abad-Manterola [8] and Shankar & Burdick [11], IMPASS, by Jeans & Hong [7], and the Rowdy Runner, by Bhounsule et al. [12]. Shankar and Burdick developed a model for a differential drive rover and applied the model for motion planning. Bhounsule et al. developed a dead-beat controller for a torso actuated rimless wheel with the torso angle being the control input.

IMHC (Institute for Human and Machine Cognition) in Florida developed a large rimless wheeled robot called Hexrunner capable of achieving a speed of 22.9 mph or 36.8 km/h [13] [14]. Hexrunner had two three legged rimless wheels that were mounted together 60° out of phase with each other. In between the rimless wheels, a torso held the electronics and the motors that drove the wheel. This robot was limited to straight line walking and could neither turn nor stop. Hexrunner led to the development of of Outrunner [15] [16], a miniaturized version with turning capabilities. Outrunner's turning approach was tilting the articulated end of its torso to the side it wanted to turn. Changed the center of mass of the robot and caused the robot to tilt, allowing it to turn.

CHAPTER 2: DESIGN

2.1 Mechanical

The Rowdy Runner II, or RR II, (Figure 2.1) consists of two main sections. The two rimless wheels and the center body.

(a) Isometric view

(b) Front view

Figure 2.1: Rowdy Runner II

2.1.1 Rimless Wheels

The robot has two 3D printed rimless wheels (Figure 2.2), each axially attached to the center body. The dimensions are carried over from the first iteration of the robot [12]; the wheels have a radius,

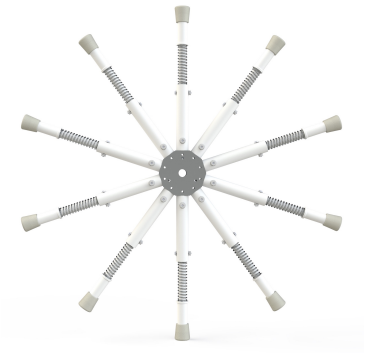

Figure 2.2: Assembled rimless wheel

Figure 2.3: Rimless wheel hub

or effective leg length, of 0.261 meters. Each wheel has 10 legs, each made up of three components: a tube that attaches to the hub, a spring, a rod that slides into the aforementioned tube, and a rubber foot. The rod has a lip which contacts the spring and compresses it whenever weight is placed on the leg. A slot designed into the rod allows for constrained movement by securing a screw through the rod and the tube.

After many failures from the first iteration of the robot, the hub was designed and milled from aluminum, with small cylinders coming from each of the 10 faces (Figure 2.3). Set screws placed into the hub prevent the cylinders from twisting of falling out. The cylinders are directly attached to the tube of the leg and fastened with a screw.

The wheels attach to the shafts protruding from the body with a keyed shaft collar. This collar is clamped onto the shaft; the clamping force exerted is enough to not need to use a key.

2.1.2 Torso

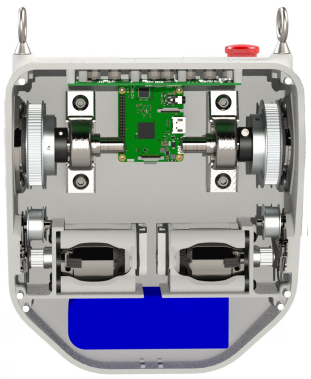

Figure 2.4: Rowdy Runner II torso

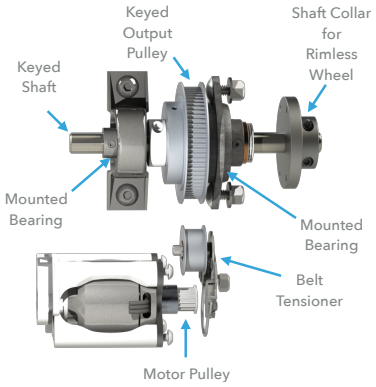

Figure 2.5: Isolated power transfer mechanism

The torso (Figure 2.4) of the Roadrunner II is 3D printed using fused deposition modeling (FDM), a process in which melted plastic extruded layer by layer to form the desired final object. The design was printed in order to create complex geometries within the body itself. This includes the bearing alignment holes, circuit board mounts, vent holes, battery compartment, and the motor attachment point.

A pitfall of FDM printed parts is their lack of dimensional accuracy. Thus, instead of pressing bearings directly into the plastic as is common if the material was metal, mounted bearings were purchased and then fitted into the body. The mounted bearings have slots to allow them to be placed as desired, allowing us to place them exactly where wanted, bypassing dimensional accuracy problems. Two bearings (Figure 2.5) are used for each shaft, one mounted to the sidewall of the body and one at the end of the shaft inside the body. Having two perpendicularly supported bearings prevents the shaft from moving around from the tension of the belt on the pulleys or from the robot walking.

Power transfer from the motors to the output shaft is done through two pulleys and a toothed belt, with a 5.4:1 reduction. The motor is mounted to a wall in the body with the face plate that was provided by the manufacturer (Figure 2.6). The encoder is attached to the motor with the help of a 3D printed bracket and second output shaft. This bracket holds four nuts which thread the screws coming through the wall and the face place. The pulley is attached to the output shaft of the motor with two set screws. The motor and output pulley are aligned with each other are rotated by a GT3 toothed belt.

Figure 2.6: Closeup of motor assembly

The output shaft is a half inch keyed shaft that engages onto the output pulley. This pulley is laterally held in place with a shaft collar and the side wall bearing surface. A small tensioner is installed on the side wall of the body to stretch the belt and prevent the belt from slipping.

All circuit boards are directly mounted onto be body's designed stand-offs. When mounting

directly into 3D printed plastic, the screws are able to self-tap themselves into the plastic, eliminating the need for a nut. The inertial measurement unit (IMU) circuit board is mounted directly to the bottom of the body using four screws. The motor controller and computer are also directly mounted to the top of the body using screws into the designed stand-offs.

The top of the torso has eyelets for a harness and a cutout for the computer ports. The bottom of the torso holds two batteries as well as the IMU.

2.2 Electrical

The Rowdy Runner II electrically consists of computation devices, motors, motor controller, sensors, a joystick, and batteries. All of these work together to allow the robot to be run remotely, monitored, and controlled.

Figure 2.7: Raspberry Pi

Figure 2.8: Teensy 3.2 microcontroller

2.2.1 Computation

The Rowdy Runner uses a computer, a microcontoller, and an orientation sensor to walk. The Raspberry Pi 3B (Figure 2.7) was chosen as the computer for the robot. This single board computer features a 4 core 1.2 GHz processor with 1 GB of RAM. The Pi is responsible for data collection, reading the sensors, and relaying data to a connected computer. A Teensy microcontroller (Figure 2.8) sends commands to the motor controller and runs a fast control loop. The microcontroller is connected to the Pi through a USB connection and to the motor controller via serial.

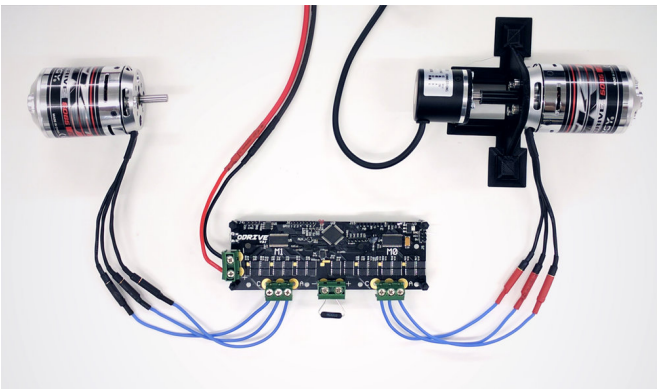

Figure 2.9: ODrive 3.5 and motors

2.2.2 Motor and Motor Controller

An ODrive 3.5 motor controller (Figure 2.9) is responsible for the robot's locomotion. This motor controller was chosen after trying several hobby motor controllers that did not meet our needs. The ODrive allows for accurate control of relatively inexpensive brushless direct current (BLDC) motors. BLDC motors are usually used for Radio-Controlled (R-C) cars, boats, and planes, making them easily available. However they lack a feedback mechanism, which is why an encoder is attached to the back of the BLDC motor (Figure 2.6). The motors and encoders are connected to the ODrive, allowing the motor controller to accurately control the output from the motors. The ODrive is connected to the Raspberry Pi through USB, as well as to the Teensy.

Two commercially available BLDC motors, with a KV rating of 280, were chosen to power the Rowdy Runner. KV, which is not an acronym, is the counter electromotive force that the motor would induce if it was spun at 1000 rotations per minute (RPM); i.e. the selected motor will produce a single volt of counter electromotive force if the motor was spun at 280 RPM. The motors were chosen with a low KV value as to limit the maximum physical speed of the motor. Low KV values also correlate to a higher torque output for the motor.

The encoders attached to the back of each motor are CUI AMT102 quadrature encoders recommended by the motor controller manufacturer. These encoders are counted by the motor controller and allow the controller to accurately control each motor.

2.2.3 Sensors

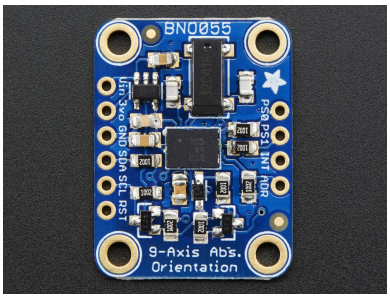

Figure 2.10: BNO055 absolute orientation sensor

In order to control the robot's walking, the robot's torso orientation in space needs to be sensed. A 9-Degree of Freedom (DOF) sensor from Adafruit was chosen. This sensor contains a BNO055 from Bosch, which has an on-board microcontroller that performs sensor fusion calculations. The sensor can then be queried for the angle values that are needed to control the robot. The orientation sensor is connected to the Raspberry Pi through a serial connection.

The ODrive is used as a sensor by querying for the position of the motors and the velocity, thanks to the connected encoders. The bus, or battery, voltage and the current drawn from the battery are also measured using the ODrive.

2.2.4 Joystick and Batteries

A Dualshock 3 Controller (Figure 2.11) is used as a remote joystick to control the robot. This joystick has a Bluetooth transmitter which is paired with the Raspberry Pi. Using Bluetooth, compared to an Xbee, another wireless transmitter radio module, based remote, removes the complexity of setting up and configuring transmitters.

The motor controller is rated for a maximum operating voltage of 24 volts. A suitable battery was chosen by taking into account the needs of both the motor controller and the BLDC motors. A 6S 30C 3000 mAh (milli-Amp-hour) Lithium Polymer (LiPo) battery (Figure 2.12) was chosen for the robot as its nominal voltage is 22.2 V. The 30C rating designates that this battery can output a constant current of 90 Amps (A). The 6S rating means that this battery has 6 battery cells in series,

Figure 2.11: Dualshock 3 controller

Figure 2.12: LiPo battery

totaling the 22.2 V nominal voltage.

A smaller battery is used to power the Pi. It is a double cell 3.7 V 4000 mAh LiPo battery. The battery is connected to a circuit board soldered to the Raspberry Pi that boosts the battery voltage to 5 V and prevents under voltage of the battery.

2.2.5 Miscellaneous

Because of the Raspberry Pi's mediocre internal WiFi antenna range and performance, a small wireless hotspot was attached to the robot for communication and data transfer. This hotspot is connected to the Pi through an Ethernet cable and is powered by a USB battery bank (5000 mAh).

2.3 Controller

2.3.1 Torso Pitch Controller

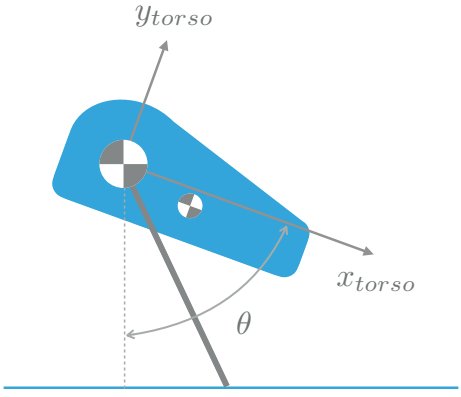

Figure 2.13: Free body diagram of torso (only one leg shown)

A proportional integral derivative (PID) controller is used to control the torso pitch, θ (Figure 2.13), of the robot. The pitch angle is correlated to a maximum speed that the robot can reach. This is due to the torso's center of mass position being in front of the axis of rotation, making the robot perpetually fall forward. However, due to having rimless wheels, the next leg on the wheel catches the robot, and so on. Then, commanding the motors to keep a certain θ propels the robot forward. At a certain speed, the impact forces are greater than the motors can compensate for while holding a desired pitch, even though the leg collisions are dampened with springs. Most commonly, the angle setpoint for the controller to maintain is 50°.

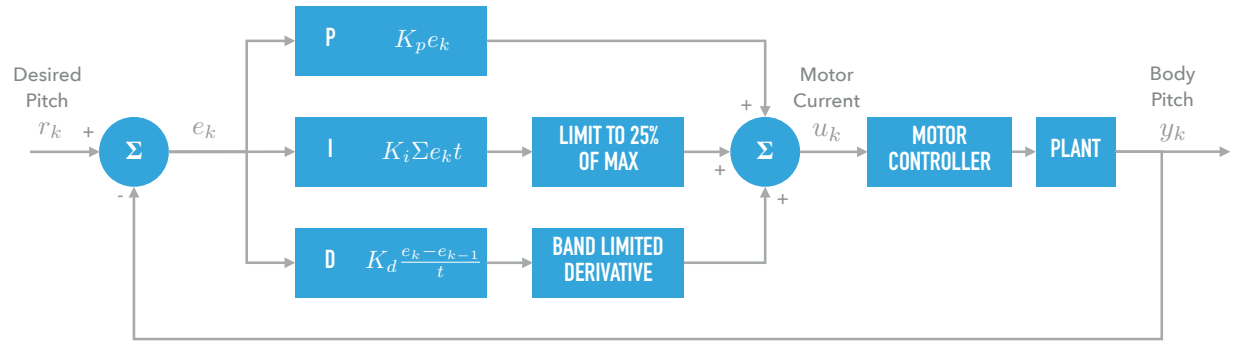

Figure 2.14: PID control diagram (discrete time)

A standard PID controller architecture is used, with the exceptions of limits on the integral term and derivative term. The discrete time form of the PID controller is used because the system is sampled with a predefined time step. The desired pitch is set by pushing up button on the Dualshock 3 joystick during operation. This setpoint (r_k) is then subtracted from the actual torso pitch (y_k) from the orientation sensor, resulting in the error (e_k). The error is then passed into the PID portion of the controller. The proportional term constitutes of the K_p gain multiplied by the error e_k. Because numerical integration cannot be done as this is a discrete system, the sum of the current and past errors are summed and multiplied by the time step. The summation is then multiplied by the K_i gain, resulting in the integral term (I-term). The error derivative is calculated by taking the difference of the current and previous error and dividing this by the time step. Multiplying this with the derivative gain K_d yields the derivative term (D-term). These are

then summed and turn into the input for the motor controller, which is electrical current in amperes. This causes the robot's body pitch to change, starting the controller again.

The P and D terms were modified as to give more accurate results with the physical robot. Without the modifications, the controller was prone to accumulating errors, causing the body pitch to oscillate wildly. Because the integral term is the summation of the previous errors, any inconsistencies in the pitch add up, causing accumulation error, or integral windup. The derivative term, being discrete, became very large as the robot was pushed. This caused the controller to compensate, resulting in the error derivative increasing. As this is a loop, this problem compounded and led to an uncontrollable robot. The solution for accumulating errors was capping the I-term contribution at a certain number, initially at the maximum desired current. This proved unsatisfactory, thus the contribution was capped to only $\pm 25\%$ of the maximum desired current. The trickier term to solve was the D-term. Because of its dependence on a discretized derivative, any large changes in pitch resulted in a large D-term, causing the torso to become unstable. A band-limited derivative (Equation 2.1) [17]was used to prevent the derivative term from increasing too rapidly.

$$d_k = K_d(1 - a_{dd})(\theta_k - xd_{k-1}) \tag{2.1}$$

$$xd_k = a_{dd} * xd_{k-1} + (1 - a_{dd})\theta_k \tag{2.2}$$

$$\tau_s = \frac{1}{F_s(1 - a_{dd})} \tag{2.3}$$

The equation weighs previous values of the D-term higher than the current by placing a contribution of a_{dd} to the previous values and a contribution of $1 - a_{dd}$ to the current value. a_{dd} was found using equation 2.3 with the desired settling time, τ_s, and PID frequency, F_s. The previous values of the D-term are stored in the variable xd_k denoted in Equation 2.2. If expanded, the xd_{k-1} term in 2.1 keeps track of the previous two values ($k - 1$ and $k - 2$) and the update of xd_k moves the tracking forward one step. Because of its large weight on previous values, the band limited derivative smooths out the noise in D-term from the discrete derivative performed by the PID controller. For the Rowdy Runner II controller, a D-term settling time of 0.1 seconds was chosen, and along

Table 2.1: PID gains

K_p	0.17
K_i	0.0005
K_d	16.0

with a PID frequency of 1000 Hz, the calculated a_{dd} term was 0.9.

The PID coefficients were tuned experimentally on the physical robot while stationary, using a pragmatic approach to tuning a PID controller detailed by Wescott [17]. K_d was tuned first. All gains start off at zero and K_p is set at an arbitrary number between 0 and 1. The starting point for K_d is $100 \times K_d$, and K_d is increased until excessive noise, oscillation, or overshoots is seen from the torso. The almost unstable K_d was then reduced by a factor of 2 to ensure a well behaved gain. Next, K_p was tuned by starting with a value between 1 and 100. The K_p gain is tuned to the point of oscillations, and then fine tuned by increasing/decreasing by a factor of 2. Lastly, K_i is tuned by setting the gain to a value between 0.0001 and 0.01. The value can then be fine tuned using the techniques for the previous gains.

2.3.2 Turning

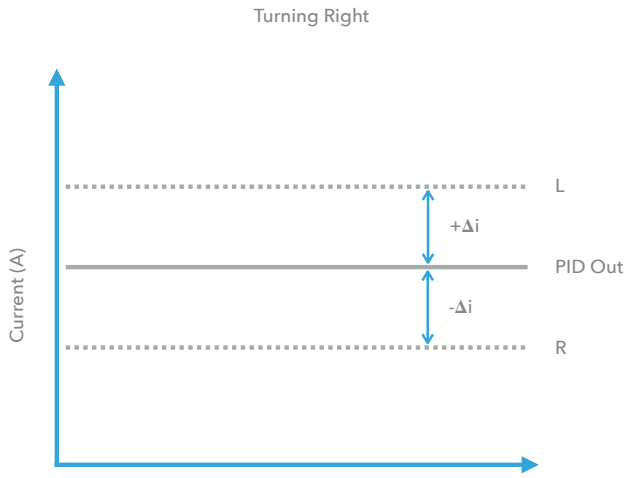

Figure 2.15: Graphical view of the turning method

The novel aspect of the Rowdy Runner II is that it is independently actuated and can turn. An intuitive turning approach was chosen: commanding a differential current to the motors. Because

the torso angle should remain at the setpoint as the robot is turning, the PID output must stay as the average value of commanded the motor currents. To do this, a $\Delta current$, or Δi, was added to the motor current to be biased. A Δi is then subtracted from the opposite motor. This can be visually seen in Figure 2.15, where the left motor current is biased to turn the robot to the right. The PID output remains the average current for the two motors, ensuring that the torso remains at the desired angle as the robot turns.

2.4 Software

The software is categorized into three levels: high level, mid level, and low level (Figure 2.16). Each corresponds to a different computing device in the robot.

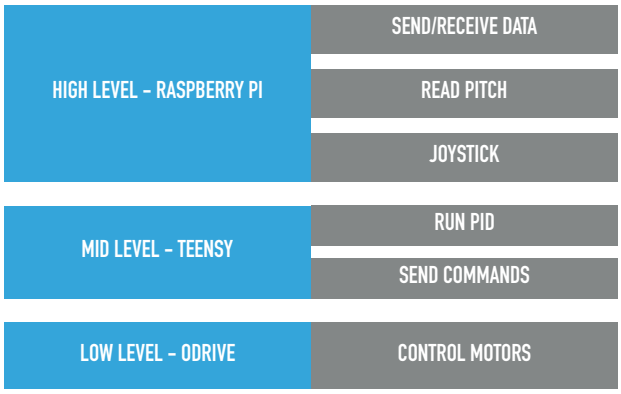

Figure 2.16: Software hierarchy

As the high level block, the Raspberry Pi is the system's scheduler and data logger. The Pi communicates bidirectionally with both the Teensy microcontroller and the ODrive at 100 Hz (Figure 2.17). The communication with the microcontroller is serial over USB. For the ODrive, the communication is through a library that directly communicates through USB. The Teensy calculates the PID loop at 1 kHz and communicates with the ODrive uni-directionally through physical serial. Lastly, the ODrive controls the motors at a frequency of around 4 kHz.

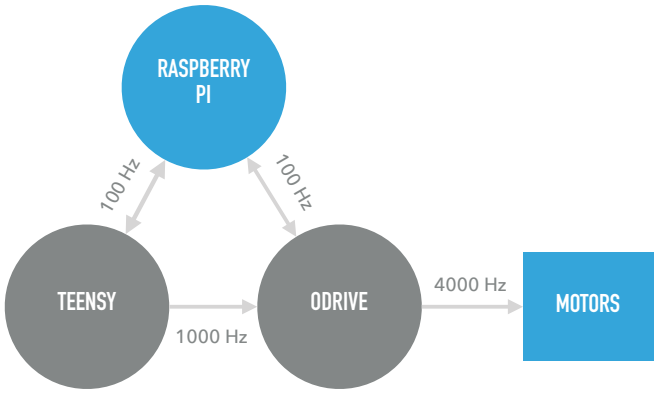

Figure 2.17: Communication diagram

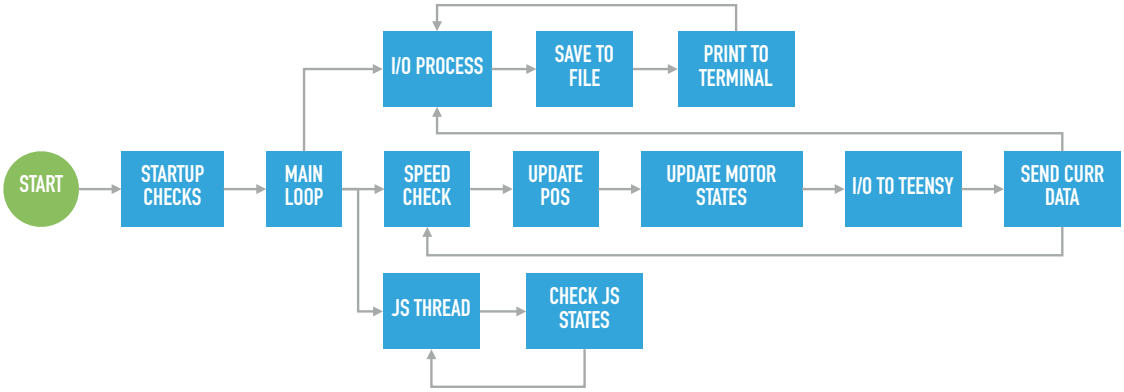

Figure 2.18: Raspberry Pi software diagram

2.4.1 Raspberry Pi

The Raspberry Pi software is written as a single script using Python 3, and uses both multi-threading and multi-processes to run functions simultaneously (Figure 2.18). Once the program is started, the Pi performs startup checks by connecting to the orientation sensor, the ODrive, and the Teensy. The main program consists of three loops: a main loop, an I/O process, and a joystick update thread. These are all run simultaneously to avoid the latency from writing acquired data to the disk. The I/O process collects data from the main loop, saves it to the disk, and prints desired information to the console. The data logged data includes: torso pitch, motor position, motor speed, PID output, pitch setpoint, battery voltage, and battery current. These are all saved for later analysis of walking trials for the robot. Data is collected at 100 Hz from the main loop, by passing all of the values in a Python dictionary. This is necessary when using multi-processes, as the

processes do not share memory space. A multi-process approach was taken with the I/O process because writing to disk is the slowest portion of the Pi code. The separate process can be run on a different core, alleviating bottlenecking from too many slow instructions in a single process.

The main loop starts a thread that monitors the Dualshock 3 joystick states. This is done using the built-in Linux `linux_js` application programming interface and a script that maps all of the buttons and axes into usable dictionaries. Since the joystick monitoring is event based, e.g. the loop waits until a button or axis value changes, the joystick function was placed in a thread, preventing it from stopping the main loop. The benefit of a thread, however, is that it shared memory space with the main loop, so no deliberate communication must happen between the thread and the main loop. All joystick related commands are placed within this thread. Pitch setpoint changes, motor calibrations, and motor on/off are all done in this thread.

The first action from the main loop is to check the motor speed as a safety precaution. The script queries the motors for their velocity, and if either one is greater than our speed limit, the Pi instructs the motor controller to turn off the motors immediately. Next, the torso pitch read from the position sensor at 100 Hz. The ODrive is then queried for the information from the power system: motor velocities and positions, bus (battery) voltage, and bus current. The current body pitch is then sent to the Teensy at 100 Hz, and, if available, the PID out is received from the Teensy. Lastly, the data is placed into a dictionary and that is sent to the I/O process to be saved and viewed.

2.4.2 Teensy

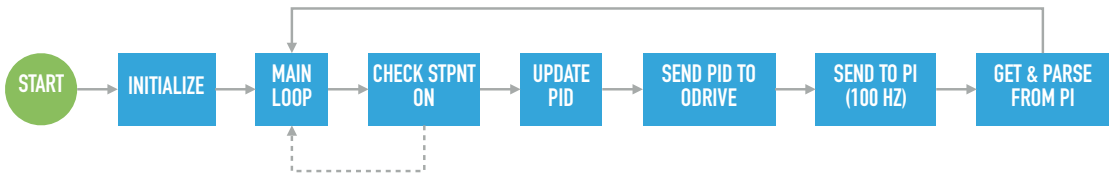

Figure 2.19: Teensy software diagram

The Teensy microcontroller was programmed using C++ and the Arduino IDE. The PID library for Arduino [18] was modified by enabling sub millisecond PID operation frequency, adding an I-term limit, and adding a band limited derivative term (Equations 2.1,2.2,2.3). The microncontroller

is reset each time the main program on the Raspberry Pi is run. The reset pin is pulled to low using the Pi's I/O pins. Once the Teensy initializes, the main loop starts by checking the value `setpoint_on` (Figure 2.19). This value is sent from the Pi when the start button is pressed, and if True, the PID controller starts. The PID is then updated with a built in function, completing the calculation. If the PID update function is called faster than the time step for 1 kHz (1 millisecond), the calculation is not performed. If the calculation has been performed, the PID output, in amps, is sent to the ODrive over serial with the correct formatting required by the ODrive. The PID output is then sent to the Pi over serial at 100 Hz. Lastly, the Teensy receives and parses the setpoint, the boolean `setpoint_on`, and the torso pitch. The information from the Pi must be parsed, because the message, encoded in ASCII, begins with a < and ends with a >. This prevents partial or corrupted messages from irregularities in communication.

CHAPTER 3: TESTING AND RESULTS

The Rowdy Runner II was evaluated on the following categories: pitch control, turning, maximum speed, and cost of transport.

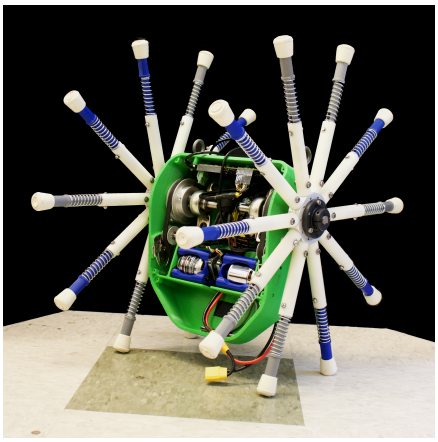

Figure 3.1: Physical Rowdy Runner II

3.1 Pitch Control

3.1.1 Testing Approach

The first approach was to have the robot walk forward, without turning. First, the program is started on the Raspberry Pi, and the motors are calibrated if needed. After calibration, the torso pitch of the robot is increased by changing the PID setpoint in increments of 5. Between each increase, the body pitch is allowed to settle. The setpoint is increased until the pitch hits 50°. The robot is then gently pushed forward on the topmost rimless wheel legs. The robot is allowed to walk forward until it is about to hit an obstacle. Once the run is complete, the walking data collected by the Pi is transferred to a local computer for analysis.

3.1.2 Results

Figure 3.2 shows a typical walking trial, one of eight, with extraneous data removed. By pushing the robot, the body torso dips from the setpoint to 40°. The PID controller is able to raise the pitch

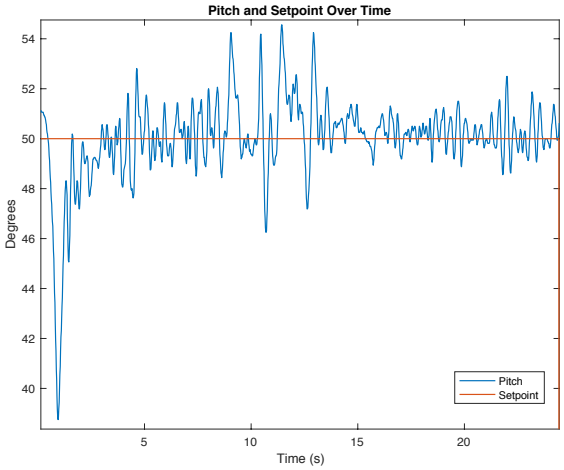

Figure 3.2: Pitch and setpoint over time

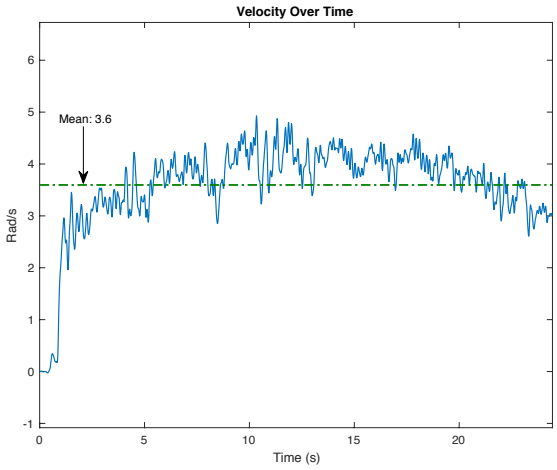

Figure 3.3: Velocity (radians) over time

to 10% of the setpoint in 0.216 seconds. The pitch is then kept steady, within ± 10 degrees, for the remainder of the trial, 23 seconds in this case. The accompanying velocity for the same trial can be seen in Figure 3.3. In both figures very regular spikes can be seen in the data. These spikes corresponds to each step taken by the robot. Each step slows the robot down, even with the passive damping from the springs, and disrupts the steady pitch. Another possible explanation is cogging from the motor. Because of the nature of brushless DC motors, there is resistance when one of the magnets moves from one coil to the next. The cogging resistance increases as the current into the motors increase. This results in the pitch movement looking like steps; or the space between the cogging.

3.2 Turning

3.2.1 Testing Approach

The turning capability of the robot is tested by using the same starting procedure as a forwards walking. After the robot stabilizes from the push, the joystick is used to turn the robot. The robot is then run for as long as possible; either until there is an obstacle or the torso becomes unstable.

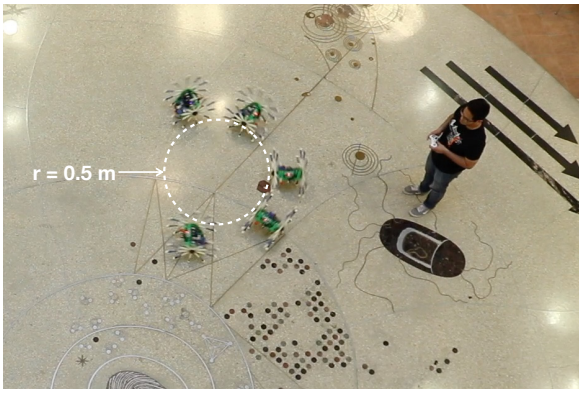

Figure 3.4: Rowdy Runner turning radius

3.2.2 Results

The plots of body pitch and speed for the turning trials are very similar to Figures 3.2 and 3.3. Thus, a much clearer example is a video still shot from top-down. Five frames were superimposed upon each other to clearly illustrate the turning capabilities of the robot (Figure 3.4). The turning radius was calculated to be approximately 0.5 meters by scaling the image to life size, overlaying a circle and measuring the radius of the circle. Thanks to the sharp turning radius, the Rowdy Runner was able to maneuver well inside the testing site, a building's atrium. The whole trial from Figure 3.4 is visible in Figure 3.5. A YouTube video of the robot in action can be seen here.

Figure 3.5: Rowdy Runner trial trajectory

3.3 Cost of Transport

3.3.1 Testing Approach

Total cost of transport (TCOT) is a dimensionless value that allows for a comparison of transport energy efficiency between different animals, robots, or vehicles. It is the ratio between the total power consumed over the weight times the velocity of the subject (Equation 3.1) [19].

$$TCOT = \frac{total\ power}{mass \times gravity \times velocity} \tag{3.1}$$

To calculate TCOT for the robot, the raw data saved from the robot is trimmed of the extraneous data, i.e. raising the pitch to 50° and empty data. Because the goal is to look at the TCOT as the robot is walking, the first few steps taken are also removed as the robot is recovering from the initial push. The average velocity and power are then calculated for the selected trial. These values, along with the power consumption of the Raspberry Pi and mass of the robot, are substituted into 3.1 and evaluated. The TCOT is then used to compare the efficiency of the Rowdy Runner II walking on different surface or to other legged robots.

3.3.2 Results

Table 3.1: Cost of Transport and Velocities on Different Surfaces

	COT	Avg. Velocity (m/s)
Polished Concrete	0.129	0.936
Asphalt	0.108	1.379
Polished Wood	0.121	1.197
Indoor Running Track	0.111	1.268
Outdoor Running Track	0.108	1.280
Average COT	0.115	

The Rowdy Runner was tested on a variety of surfaces. Total cost of transport (TCOT) was used to compare the results of walking on each surface (Table 3.1). A breakdown of the COT calculation for walking on polished concrete (Figures3.3 and 3.2) can be seen in Table 3.2.

Table 3.2: Cost of Transport Breakdown for Concrete

Power	RPi & Sensor	5 W
	Teensy	0.2 W
	ODrive	3 W
Mass	6.9 kg	
Velocity (m/s)	3.6 rad/s	0.936 m/s
COT	0.129	

Compared to similar rimless wheel robots, the Rowdy Runner II has a lower cost of transport, and thus is more energy efficient. HexRunner, the large rimless wheel robot from IHMC, has a theoretical TCOT of 0.13 [14]. Axel, the tethered Mars rover robot, has an average energy consumption per unit traveled of 180 J/m [8, 9], where as the Rowdy Runner II has an average of 7.86 J/m, calculated from Table 3.2. Overall, there are not many published rimless wheel robots with TCOT that can be compared against the RR II. The Rowdy Runner II is one of the few tested and working rimless wheel robots, and is, to our knowledge, the most energy efficient rimless wheel robot.

3.4 Maximum Speed

3.4.1 Testing Approach

Maximum speed was tested by increasing the pitch setpoint θ (Figure 2.13) above the standard 50°. The pitch increase started at 60° up to 85° at 5° increments. The robot was raised to the desired setpoint and then pushed to start the robot walking, exactly like the pitch control tests. The robot walked straight, as a differential current from turning would slow the robot down. The trial continued until an obstacle was reached and the motors were turned off.

3.4.2 Results

Seven trials were run between 60° and 85°. The maximum speed was reached in the second trial at 85°. The robot reached a maximum speed of 17.04 radians/s or 4.32 m/s before running out of space and hitting an obstacle (smacking into a trash can). Increasing the pitch angle allows the

motor controller to apply more current to the motors, in turn increasing the speed while keeping the torso still. Using this correlation, the the maximum speed increased around three times from a 50° angle (Table 3.1).

3.5 Future Work

The next step for the robot is to overturn the minimum cost of transport record for legged robotics. The current record is held by Cargo, a monopod robot with a minimum COT_{mech} of 0.1069 [20]. From the COT breakdown, the largest power consumption is from the Raspberry Pi, at 5W. There are two possible approaches: to increase the weight of the robot or to remove the Raspberry Pi and solely use the Teensy microcontroller. Increasing the weight by just one kilogram, primarily around the point of rotation, would decrease the COT by 13% in the polished concrete trial. This would clearly reduce the RRII's cost of transport and beat the record.

The robot currently requires a push to initiate walking. The push is done by a human, so it is never consistent. Too hard of a push could easily render the robot unstable. Eliminating the need for a push could be done by using more complex control methods, such as dynamic programming, to start walking. This could be applied to the robot to make the robot launch much more reliable.

In this current work, simple PD controller was used to control the torso. However, a model-based controller such as Lyapunov-based controller [21] or PD sliding mode controller [22] that takes into account the dynamics of the walker is more suitable when the walker faces external disturbances. This approach could be used in the future to control the robot while going over complex terrain and slopes.

CHAPTER 4: CONCLUSION

In this thesis, an independently actuated rimless wheel robot, the Rowdy Runner II, was developed and controlled . The majority of the robot was 3D printed for ease fo manufacturing and to create complex geometries. A PID controller was used to control the pitch of the torso relative to the ground perpendicular. After reaching a pitch of 50°, for most tests, the robot was pushed and it walked forward without human intervention. Turning was achieved by applying differential current to bias one motor over the other. Our robot achieved an average cost of transport of 0.115, lower than similar rimless wheeled robots, and close to the record minimum for legged robots.

Ultimately, the Rowdy Runner II is a remote controlled, energy efficient, legged robot that is agile and can walk on varied surfaces.

APPENDIX A: CODE

The code for the Rowdy Runner II is made up of a main Python script, an Arduino program for the microcontroller, and a modified PID library for Arduino. The code can be seen in the following GitHub repo: https://github.com/s3basti/RowdyRunner2

BIBLIOGRAPHY

[1] Arthur Ed LeBouthillier. W. grey walter and his turtle robots. *The Robot Builder*, 11(5):1–3, May 1999.

[2] Ronald Ping Man Chan, Karl A Stol, and C Roger Halkyard. Review of modelling and control of two-wheeled robots. *Annual Reviews in Control*, 37(1):89–103, 2013.

[3] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. Bigdog, the rough-terrain quadruped robot. *IFAC Proceedings Volumes*, 41(2):10822–10825, 2008.

[4] Michael P Murphy, Aaron Saunders, Cassie Moreira, Alfred A Rizzi, and Marc Raibert. The littledog robot. *The International Journal of Robotics Research*, 30(2):145–149, 2011.

[5] Tad McGeer et al. Passive dynamic walking. *I. J. Robotic Res.*, 9(2):62–82, 1990.

[6] Tad McGeer. Dynamics and control of bipedal locomotion. *Journal of Theoretical Biology*, 163(3):277–314, 1993.

[7] Dennis Hong Blake Jeans. Impass: Intelligent mobility platform with active spoke system. In *IEEE International Conference on Robotics and Automation*, 2009.

[8] Pablo Abad-Manterola. *Axel rover tethered dynamics and motion planning on extreme planetary terrain*. California Institute of Technology, 2012.

[9] Krishna Shankar and Joel W Burdick. Motion planning and control for a tethered, rimless wheel differential drive vehicle. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4829–4836. IEEE, 2013.

[10] Andy Ruina Michael J. Coleman, Anindya Chatterjee. Motions of a rimless spoked wheel: a simple 3d system with impacts. *Dynamics and Stability of Systems*, 1997.

[11] Matthew D. Berkemeier Adam C. Smith. The motion of a finite-width rimless wheel in 3d. In *IEEE International Conference on Robotics and Automation*, 1998.

[12] et. al. Pranav Bhounsule. Dead-beat control of walking for a torso-actuated rimless wheel using an event-based, discrete, linear controller. In *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2016.

[13] TheIHMC. Ihmc hexrunner running robot, Jun 2014.

[14] IHMC Robotics Lab. *FastRunner*.

[15] et. al. Sebastien Cotton. *Multi-legged running robot*, us patent app. 14/596,514 edition, Jan 2014.

[16] Robotics Unlimited. Meet outrunner: The world's first remotely controlled running robot, May 2014.

[17] Tim Wescott. Pid without a phd. *Embedded Systems Programming*, October 2000.

[18] Brett Beuregard. Arduino pid library.

[19] Ali Zamani, Pranav A Bhounsule, and Ahmad Taha. Planning energy-efficient bipedal locomotion on patterned terrain. In *Unmanned Systems Technology XVIII*, volume 9837, page 98370A. International Society for Optics and Photonics, 2016.

[20] Fabian Guenther and Fumiya Iida. Energy-efficient monopod running with a large payload based on open-loop parallel elastic actuation. *IEEE Transactions on Robotics*, 33(1):102–113, 2017.

[21] Ali Zamani and Pranav Bhounsule. Control synergies for rapid stabilization and enlarged region of attraction for a model of hopping. *Biomimetics*, 3(3):25, 2018.

[22] S Ali A Moosavian, Mahdi Khorram, Ali Zamani, and Hamed Abedini. Pd regulated sliding mode control of a quadruped robot. In *Mechatronics and Automation (ICMA), 2011 International Conference on*, pages 2061–2066. IEEE, 2011.

VITA

Eric Sebastian Sanchez was born in Lima, Peru, and has been a San Antonio resident since 2005. He received his B.Sc. and is completing his M.Sc. at the University of Texas at San Antonio. He is passionate about mechanical design, rapid prototyping, and 3D printing. He is hopeful the Rowdy Runner II can beat the walking robot efficiency record.