Development of an Inertially-Actuated Passive Dynamic Technique to Enable Single-Step Climbing by Wheeled Robots

John C. Humphreys

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Mechanical Engineering

Dr. Dennis Hong
Committee Chair
Mechanical Engineering

Dr. Marty Johnson
Committee Member
Mechanical Engineering

Dr. Mary Kasarda
Committee Member
Mechanical Engineering

Dr. Andrew Kurdila
Committee Member
Mechanical Engineering

April 28, 2008
Blacksburg, Virginia

Keywords: Passive dynamics, Inertially actuated, Wheeled robot,
Step climbing, Sliding mass, Mobility

Development of an Inertially-Actuated Passive Dynamic Technique to Enable Single-Step Climbing by Wheeled Robots

John C. Humphreys

Abstract

For their inherent stability and simple dynamics of motion, wheeled robots are very common in robotics applications. Many highly complex robots are being developed in research laboratories, but wheeled robots remain the most used robot in real-world situations. One of the most significant downfalls of wheeled robots is their inability to navigate over large obstacles or steps without assistance. A wheeled robot is capable of climbing steps that are no larger than the radius of the robot's tires, but steps larger than this are impassable by simply rolling over the object. Active systems that have been designed for use on wheeled robots to lift the robot over a step are effective, but are generally not easily implemented on a range of robotic platforms. Also, the additional size, cost, and power required for the additional actuators is a major drawback to these options.

A solution to these problems is a novel, passive dynamic system that is inertially excited by the motion of the robot to allow the robot to rotate on each axle and "hop" over the step. The system that was investigated for this project is a sliding mass-spring that shifts forward and backward based on the acceleration of the base robot. With high acceleration, the mass is pushed towards the rear wheel from an inertial force and compresses a spring that creates a moment on the body to induce rotation. This torque can cause the robot to "pop a wheelie", lifting its front wheels off the ground. To pull the rear wheels up, the inertial force from a large deceleration of the robot shifts the mass forward and extends a spring. These effects result in a moment acting in the opposite direction that can rotate the robot on its front axle and pull the rear wheels up. By coordinating the acceleration and deceleration of the robot, the front wheels can lift over a step and the rear wheels can be pulled up afterward – both actions being a product of inertial actuation. This passive system does not need additional actuators or direct control of the sliding mass, so it can be more durable over a robot's lifetime. Other advantages of this system are that the design is simple, cost-effective, and can be adjusted and retrofit to a different wheeled robot in the future with little effort.

By deriving the equations of motion of this inertially actuated sliding mass, the dynamics show how design parameters of the system can be tuned to better optimize the overall step-climbing process. A computer simulation was created to visualize the robotic step-climbing process and demonstrate the effects of changing design parameters. An implementation of this sliding mass system was added to a wheeled robot, and the results from experiments were compared to simulated trials. This research has shown that an inertially actuated sliding mass can effectively enable a wheeled robot to climb a step that was previously impassable and that the system can be tuned for other wheeled robots using an understanding of the system dynamics.

Acknowledgements

Much as it drives me nuts when people make an exaggerated display of this, I would like to take this chance to thank my God who has blessed me with a good life so far. Nothing would be possible without His grace and I am truly thankful for the guidance that I have received. I also recognize that my parents knew enough about life to raise me well and apparently teach me a thing or two. My entire childhood has steered me on the path I've taken, so I thank everyone who was involved in that for their contributions in making my life what it is.

As for my educational experiences, I would first like to thank my advisor, Dr. Hong, for his leadership during my project and also for allowing me to join the Robotics and Mechanisms Laboratory (RoMeLa) at Virginia Tech. RoMeLa is quickly becoming the best robotics research lab in the country; although I probably did more harm than good in the lab, I'm still proud to have had the chance to work there. I also thank all the members of RoMeLa for welcoming me in and making the lab a fun place to be – even late on a Friday night. From my undergraduate program, I greatly thank Dr. Reinholtz (who has since moved on) for being the best teacher I've ever had and for showing me the bridge between education and excitement.

Finally, I appreciate the tolerance that my amazing girlfriend, Samantha, has had with me and my mood swings during the last couple months of this project. Any other guy would be absolutely thrilled to be with her (and maybe treat her a little better) but she's decided to put up with me for several years anyway.

Table of Contents

List of Tables and Figures

Nomenclature

Lift-off – the process of step climbing where the robot rotates on its rear axle and lifts the front wheels off the ground
Pop-up – the process of step climbing where the robot rotates on its front axle and lifts the rear wheels off the ground

α_1 – angular acceleration of robot about the rear axle
α_2 – angular acceleration of robot about front axle
a – acceleration of the pivot axle of the robot
a_x – horizontal distance to base COG from rear axle (positive up)
a_y – vertical distance to base COG from rear axle (positive toward front axle)
b – spring damping
b_x – horizontal distance to sliding mass from rear axle (positive toward front axle)
b_y – vertical distance to sliding mass from rear axle (positive up)
c_x – horizontal distance to base COG from front axle (positive toward rear axle)
c_y – vertical distance to base COG from front axle (positive up)
d_x – horizontal distance to sliding mass from front axle (positive toward rear axle)
d_y – vertical distance to sliding mass from front axle (positive up)
δ – deflection of spring (positive in compression)
g – gravitational acceleration
h – height of linear bearing where contact between slider and robot occurs
I_1 – moment of inertia of robot about the rear axle
I_2 – moment of inertia of robot about front axle
k – spring stiffness
L – wheelbase of robot
L_M – total distance to base COG from rear axle
$L_{M'}$ – total distance to base COG from front axle
M - mass of base robot
m – mass of slider
R_{1y} – vertical reaction at rear tire
R_{2y}- vertical reaction at front tire
T_w – wheel torque acting on robot from motion
θ – rotation of robot (positive with front wheels higher than the rear wheels)
μ – kinetic coefficient of friction between slider and platform
ω_1 – angular velocity of robot about the rear axle
ω_2 – angular velocity of robot about front axle

Chapter 1: Introduction

The need for wheeled robots to climb stairs is an often-studied topic, and this section presents prior work in the field. After discussing the limitations of active methods that enable step climbing, a novel, inertially actuated, sliding mass-spring system is introduced that is studied as a technique to assist in step-climbing. The sliding spring-mass system is discussed with respect to its ability to increase mobility of wheeled robots. A brief analysis of how this passive method can enable step climbing by wheeled robots shows the general theory behind this approach.

1.1 Background: Wheeled robots are common in robotics applications, likely due to their inherent stability and ease of motion when at least three wheels are present. Unlike robots that rely on a coordinated effort of legs to move, robots on wheels require very little design effort in order to maintain stability. Many wheeled robots [1-3] have the need to climb a step: wheeled airport security robots (like the one shown in Figure 1.1) that patrol the facility are limited to moving along flat and inclined surfaces. However, if the security robot needs to climb a curb and a ramp is not nearby, a possible security risk could go undetected. For situations like this where a robot needs the ability to climb a step immediately, some type of design is required to allow this maneuver.

Figure 1.1: A wheeled security robot that is restricted to flat surfaces and ramps [2].

While other robotic platforms [4-8], such as bipedal robots and snake-like robots, are able to surmount large steps by manipulating themselves to rise up, wheeled robots are limited to flat surfaces and small steps. A wheeled robot can climb steps that are lower than the radius of the robot's tires [9], but larger steps are insurmountable by basic friction contact of the wheels on the step. Robotic platforms have been designed to allow wheeled robots to climb steps [10-19], but these platforms rely on additional actuators, such as motors or hydraulic systems. For these options, extra power and added processing for control and coordination of these actuators are needed for the extra degrees of freedom.

We present a novel approach to climbing large steps with a wheeled robot utilizing a passive dynamic, inertially actuated sliding mass. This technique involves attaching a passive system that is inertially actuated to allow the robot to climb a step without needing active control. Before discussing the dynamics involved in this system, a proper introduction to the concept of *passive dynamics* and what this term generally applies to is required. Passive dynamics were first realized [20] as a more efficient walking pattern that takes advantage of gravity and the natural swing of legs instead of spending effort to actively manipulate legs for walking. From this initial application of passive dynamics, many robotic systems [21-27], in addition to two-legged walking, took advantage of this concept. In contrast to this original idea of passive dynamics that generally applies to a more efficient walking process, this research presents an idea of inertially actuated passive dynamics – which means a system is actuated through accelerations of a separate entity and is not directly controlled.

A simple implementation of inertially actuated passive dynamics is a spring-mass slider, which will be the main platform under investigation. By fixing one end of a spring to the robot and the other end to a mass that is constrained to linear motion in the direction of the robot's travel, the mass can be inertially actuated by the accelerations of the robot and will deflect a spring. The effect of the sliding mass and spring deflection can cause the robot to "pop a wheelie" and be able to hop over a step. This process will be described in detail later in the thesis. The idea of using a wheelie, when one set of the robot's wheels are off the ground, to climb over a step is not new, but the implementations of these systems [28-29] utilized active control of the robot to pull the front wheels up. A sliding mass system is also not a novel concept in engineering [30], though typical applications [31-32] involve pendulum control. One application of a moving mass has been studied [33] to steer a mathematical model of a rigid body

balanced on a blade (termed the Chaplygin sleigh), though the use of a sliding mass in robotics has been limited.

To complete this project, two relatively significant simplifications were made to the wheeled robot: the tires can be modeled as spring-damper systems and the terrain around the robot is known. While much more complex models of rubber tires have been designed [34-36], the simplification to model the tires as spring-dampers will be sufficient to analyze the dynamics of the overall system. In addition, systems to detect a robot's environment [37-40] and use this information to find a travel path were considered to be an unnecessary complexity for this phase of the project. By assuming the world around the robot to be known and unvarying, the main focus can rest on determining how an inertially actuated system can enable step-climbing by wheeled robots.

1.2 Inertially actuated step climbing process: While greater detail is reserved for later, this section will briefly describe how an inertially actuated sliding mass can be used to increase mobility of a wheeled robot. In short, a sliding mass on top of a robot is pushed backward, during the robot's acceleration, from inertial forces and compresses a spring that is fixed to the sliding mass and the robot. This shift in the center of gravity location and coupled spring force can help lift the front wheels of the robot off the ground, essentially forcing the robot into a "wheelie". Once the front wheels contact the top of the step, braking causes a change in momentum that will slide the mass forward and stretch the spring. If done appropriately, this action can pop up the rear wheels over the step and allow the robot to have successfully climbed a step. A detailed analysis of these effects will be provided, showing how these coupled reactions will allow a robot to pass its wheels over a step without actively interacting with the step. While the wheels of the robot must be turned for acceleration, and brakes must be applied for deceleration, this does not eliminate the concept of passive dynamics. The wheels and brakes must be used for basic motion, so the power required for the wheel motors and brakes is not an additional requirement on the system. Also, the sliding mass is not directly controlled and truly is inertially actuated, so the step-climbing process in enabled by an implementation of passive dynamics.

There are several advantages to this passive dynamic approach to stair-climbing that allow this technique to be extended to the general field of wheeled robots. There are a few

configurable components of the system that can be scaled to meet the specific demands of a robot. Based on a robot's mass, geometry, and acceleration capabilities, the sliding mass's design parameters can be tuned, including mass, spring stiffness, and platform height. If done properly, a selection of these parameters will enable many other wheeled robots to climb over a single stair using passive dynamics.

This project is generally applicable to robots with four wheels that are arranged near the corners of the robot's base. A robot with three wheels arranged in a triangle (or fewer tires) will require significant effort to remain stable during the phase of step-climbing that would balance the robot on a single tire. The use of a six-wheeled robot would not be a detriment to this inertial actuation technique, though the middle set of tires will not contribute to the "wheelie" actions. Also, these extra wheels may reduce the step-climbing ability of the robot by providing an interference with the step; when the front tires settle on the step and the rear wheels are still below the step, this middle set of tires may hit the step and affect the overall system.

Due to the simplicity of the design, components of the platform can be easily exchanged to better optimize the system in certain situations. The sliding mass is the only moving part that is added, so there is less of a chance to break important components versus other, more complicated, designs. Also, the components are very inexpensive, so replacement costs for a part failure will not be too costly.

1.3 Thesis outline: This paper will summarize the efforts to study how an inertially actuated, passive dynamic system can be used and optimized to enable wheeled robots to climb a step. First, a derivation of the dynamics of a wheeled robot using inertial forces for step-climbing is provided. Simulations of the system are discussed in order to prove that the equations can effectively allow a wheeled robot to climb over large steps and to design the system parameters for testing. The results of an experiment to further validate the derived equations are included and show that a wheeled robot can climb a step using passive dynamics. Finally, possible future work and recommendations for further research are given.

Chapter 2: Dynamic model

This section will present derivations of the equations of motion needed to model how the wheeled robot climbs a step under inertial actuation. First, a thorough description of the novel step climbing process will be given, including the parameters that will be used throughout the derivation. Two discrete step-climbing actions are performed: a *lift-off* process where the front wheels lift off the ground as the robot rotates about its rear axle and a *pop-up* process where the rear wheels pop up as the robot rotates about its front axle – which is now on top of the step. The dynamics in the 2-D sagittal plane of the sliding mass and the robot rotation during each of these actions will be discussed in detail in the following sections.

2.1 Assumptions: To limit the scope of this project, several assumptions were used pertaining to the dynamic model of the system. The derivations provided here assume that the robot has 4-wheel drive, although reducing the equations for a 2-wheel drive robot is trivial. The only complication involved in switching to 2-wheel drive in the equations of motion is ensuring that traction is not being applied to the ground when the powered tires are not in contact with the ground. Also, the suspension of the robot will be ignored for simplicity – this is not an inappropriate assumption since the suspension of a wheeled robot can be bypassed with little effort. A more significant assumption on the system is that the tires can be modeled as spring-damper systems. This assumption is not entirely valid but can be used to increase the accuracy of the overall dynamics without adding extreme complexity to the model. By assuming the robot drives a straight path (perpendicular to the step), the dynamic model can be reduced to be two-dimensional for the step-climbing process.

2.2 Dynamics of step-climbing with a basic wheeled robot: The general approach to accomplish step-climbing by a basic wheeled robot (without a sliding mass) via inertial forces is shown in Figure 2.1. A rapid acceleration will produce inertial forces on the robot. If the robot's center of gravity is above the rear axle and the robot's acceleration is sufficiently high, the robot will lift off its front wheels because of the torque generated about the rear axle from the inertial force on the robot (see Fig 2.1b). When the front wheels are on top of the step (see Fig 2.1c), a large deceleration will reverse the inertial forces and help the rear wheels to pop up (see Fig 2.1d). This action will only occur if the center of gravity is still above the front axle when the

wheels hit the step. With some forward velocity remaining, the rear wheels will fall onto the step (see Fig 2.1e) and the robot will be able to continue moving.

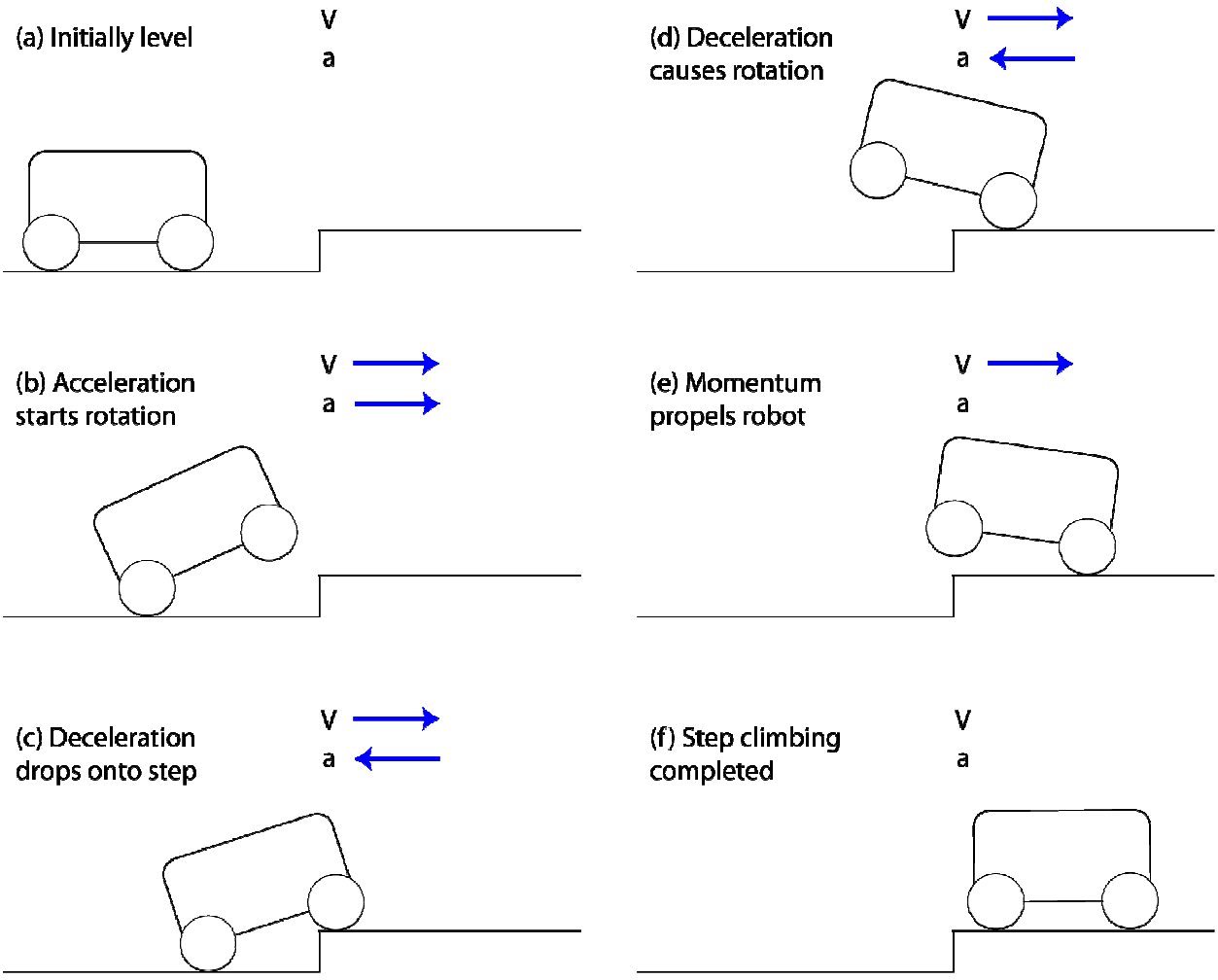

Figure 2.1: Diagram showing how a basic wheeled robot can hop over a step from inertial forces.

However, for a basic wheeled robot, rotation about the rear wheel without additional actuation is unlikely, except at high accelerations or awkward geometries. As shown in the free-body diagram in Figure 2.2, the main reaction forces on a rigid robot stem from the inertial force on the robot, reaction forces at the wheels, and forces due to gravity. The rotation of the robot can be modeled using Equation 2.1,

$$I_1\alpha_1 = Ma\left(a_x \sin\theta + a_y \cos\theta\right) - Mg\left(a_x \cos\theta - a_y \sin\theta\right) + T_w + R_{2y}L \qquad (2.1)$$

where I_1 and α_1 are the moment of inertia and angular acceleration of the robot about the rear axle, respectively and θ is the rotation of the robot above horizontal. The term M is the mass of the robot, a is the horizontal acceleration of the robot, and g is the gravitational acceleration. The motor torque is expressed as T_w and the reaction force at the front wheel is given as R_{2y}, at a distance of L from the rear wheel. The center of mass location of the robot is represented by a_y and a_x in the vertical and horizontal directions, respectively.

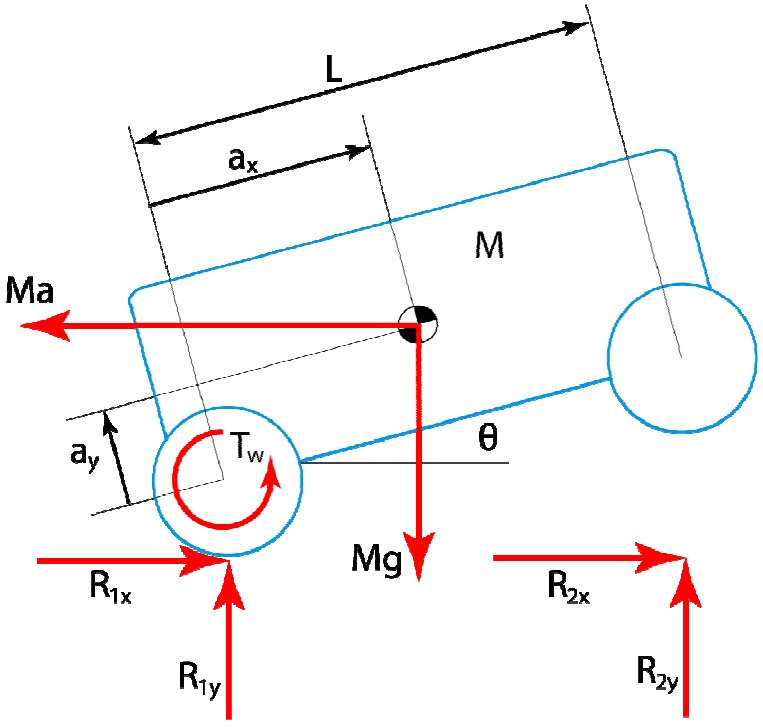

Figure 2.2: Free body diagram of a wheeled robot while accelerating. The inertial force must create a torque greater than the torque from gravity in order to rotate about the robot's rear wheel.

At the moment the robot begins to rotate ($\alpha_1 \geq 0$), the front wheel reaction force, R_2, will be zero. The motor torque, T_w, is an important term and should be included in a full dynamic model, but the most significant parts of Equation 2.1 for designing the system are the first two terms – as will be validated later. For this reason, the motor torque will be ignored for this portion of the thesis. With the robot initially level, θ is equal to zero. Simplification of Equation 2.1 yields an expression for the limit when rotation about the rear axle will start with the assumption of small motor torque, shown in Equation 2.2,

$$0 \leq aa_y - ga_x \tag{2.2}$$

where a is the robot's acceleration, g is the gravitational acceleration, and a_x and a_y are the horizontal and vertical distances to the center of gravity from the rear axle. From this relationship, it is apparent that to assist rotation about the rear axle, three options are possible: the robot can be accelerated more quickly (higher a), the center of gravity can be shifted higher (higher a_y), or the center of gravity can be shifted farther backward towards the rear axle (lower a_x).

The first option to help a robot rotate about its rear axle is to increase acceleration. The required acceleration to lift the front wheel without changing other aspects of the robot is often quite high. For example, assume the robot shown in Figure 2.2 is being used, where the center of mass is twice as far from the rear axle horizontally as it is vertically. Use of Equation 2.2 shows the required acceleration to lift the front wheels of this robot must be twice the gravitational acceleration. An acceleration of 2 g's is not practical for many robots, especially when the robot operates in public environments. While this example is a random selection, the principle holds that unless the robot is configured in a very convenient arrangement, the required acceleration to lift off the front wheel is very high.

Another option to induce rotation is to heighten the robot's center of mass. Without changing other parameters and requiring only moderate accelerations, the center of gravity would need to be raised very high. Assuming a reasonable acceleration for a robot in public places to be 0.5 g's (which is typical and justified in a later section), the vertical distance to the center of gravity must be twice as large as the horizontal distance. With such a top-heavy robot, the stability and driving characteristics of the robot during other actions are negatively affected.

The final option to aide in rotation of the rear axle is to shift the center of gravity backward. While the effect of this seems analogous to moving the center of gravity higher (since the ratio of these distances is the important value), analysis of the second phase of step climbing will show why this shift is ineffective at helping a basic robot to climb a step.

The pop-up action is accomplished using similar dynamics to the lift-off process. Figure 2.3 shows a robot with its front wheels on a step as it decelerates in an attempt to pop up the rear wheels. The dynamics of rotation of the robot about its front axle are shown in Equation 2.3,

$$I_2\alpha_2 = Ma\left(c_y\cos\theta - c_x\sin\theta\right) + Mg\left(c_x\cos\theta + c_y\sin\theta\right) + T_w - R_{1y}L \qquad (2.3)$$

where the newly introduced terms include c_y and c_x, the vertical and horizontal distances from the front wheel to the center of mass, respectively. The terms I_2 and α_2 are the moment of inertia and angular acceleration of the robot about the front wheel and R_{1y} is the vertical reaction force at the rear wheel. The wheel torque, T_w, is now affecting the front axle – assuming the robot is powered on all four wheels. If rear-wheel drive is the only power, this torque term would need to be zero and there should be no acceleration of the robot; coasting of the robot would be the only dynamic effect occurring.

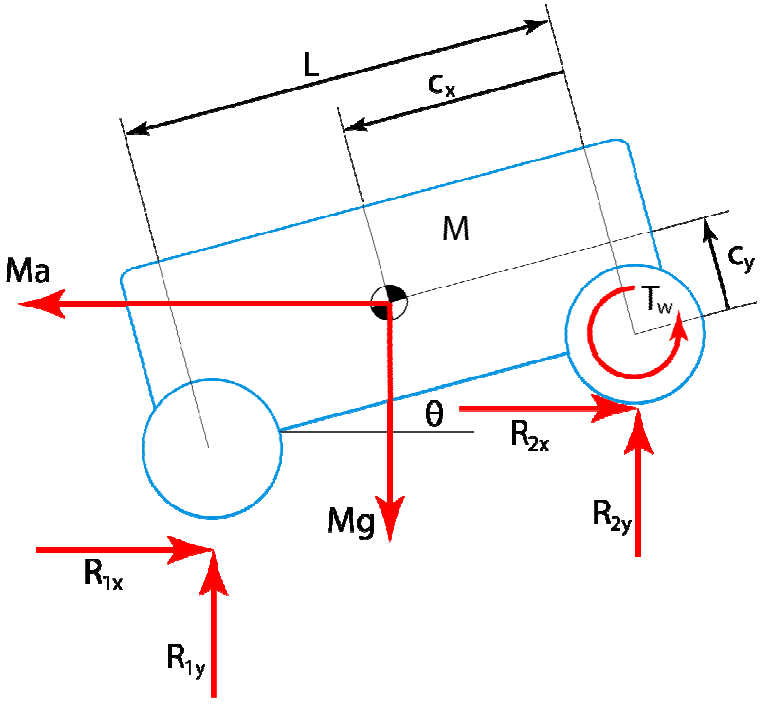

Figure 2.3: Free body diagram of a rigid robot while decelerating for the pop-up phase. The inertial force must create a torque greater than the torque from gravity in order to rotate about the robot's rear wheel.

Similar to the lift-off process, the wheel torque and reaction force are relatively insignificant for a design discussion, but now θ has an initial value that is nonzero and dependent on the step size. The hardest part of the pop-up action for a rigid robot is getting started (which will be justified later), so analysis at this initial value of θ is appropriate. Simplification of Equation 2.3 for the stated assumptions (which have not yet been justified) shows that to pop up the rear wheels ($\alpha_2 < 0$), the relationship shown in Equation 2.4 must be satisfied,

$$0 \geq a\left(c_y \cos\theta - c_x \sin\theta\right) + g\left(c_x \cos\theta + c_y \sin\theta\right) , \tag{2.4}$$

where variables are defined similarly from Equation 2.3. Considering that *a* is negative for deceleration, three options again *appear* available to assist in popping up the rear wheels: decelerate quicker, shift the center of mass higher, or shift the center of mass farther forward. The solution here is not so simple, though, because increasing the height of the robot is advantageous according to the first part of the equation but is detrimental in the second part. This effect will be discussed in greater detail in later sections, but it is worth noting.

According to Equation 2.4, the first option to induce rotation about the robot's front wheel is to decelerate quicker. To do this without changing other parameters of the robot is very hard – even harder than during the lift-off stage. Since the robot is rotated, the inertial force caused by deceleration has a smaller moment-arm to act on the front wheel. For example, if the robot from the earlier case study is used where the horizontal location of the center of gravity is twice that of the vertical location and the robot is rotated at 15 degrees, the deceleration required to pop the rear wheels up is nearly 5 times the gravitational acceleration. This deceleration is not a reasonable expectation, so clearly simply braking harder is not a valid approach.

The second option to aide in popping up the rear wheels during step-climbing is to increase the height of the center of gravity. Again, because of the rotation of the robot, the required height for this is much higher than for the lift-off process. Assuming a reasonable deceleration rate of 0.5 g's for the robot (which is justified by testing – shown in a later section), the required height to start the pop-up action from an angle of 15 degrees is nearly 4 times the horizontal offset of the center of gravity. Similar to the discussion about the lift-off process, a robot that is so top-heavy is likely to have worse stability limits and driving characteristics.

The third option in helping the pop-action is to shift the center of gravity forward. Recalling the requirement from the lift-off phase that the center of gravity needs to be shifted backward, these two demands are in conflict. While a balance between the two limits can be reached so as to provide equal benefit to each phase of step-climbing, moving the center of mass horizontally to a fixed position is unable to fully assist in the overall step-climbing process.

For the reasons shown above, changing parameters of a rigid wheeled robot is either insufficient or impractical in order to allow the robot to hop over a step using inertial forces.

2.3 Step-climbing process using an inertially actuated sliding mass: Using the insight gained from the analysis of a basic wheeled robot climbing a step via inertial forces, a passive system is developed that will facilitate rotation of the robot about both wheels without demanding excessive accelerations or awkward geometry. A diagram of the proposed design is shown in Figure 2.4. By adding a mass-spring slider on top of the robot at a predetermined height, the required accelerations to climb a step can be significantly reduced for both lift-off and pop-up phases of step climbing. The center of gravity (COG) that is shown is for the base robot and is fixed. The overall COG of the system will change as the sliding mass moves, but the COG shown in Figure 2.4 is not affected by this motion and is at a constant position.

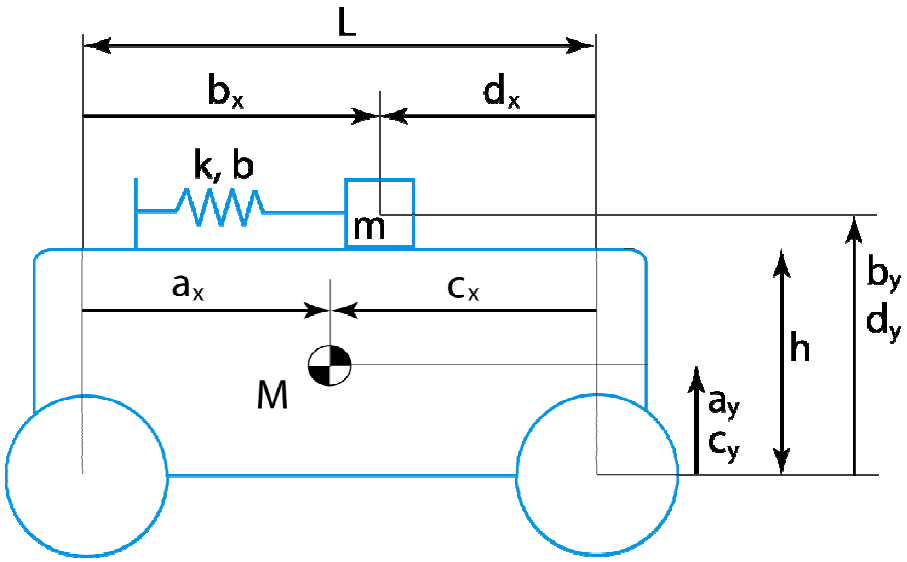

Figure 2.4: Diagram of a robot with a mass-spring slider attached to facilitate step-climbing.

While the mass addition changes the dynamics, an appropriate selection of mass, height, and spring stiffness will increase the robot's ability to rotate on both axles. Since the mass-slider will be added above the base robot, the effective center of mass will be raised. By using a spring to allow the mass to slide, the effective center of mass can shift forward and backward with the sliding mass. This satisfies two of the three options to hop a step with a basic robot and will limit the required acceleration needed. In addition to these effects, the deflection of the spring adds an extra torque on the robot to assist in the robot's rotation. A diagram of the step-climbing process for a wheeled robot with an inertially actuated sliding mass is shown in Figure 2.5.

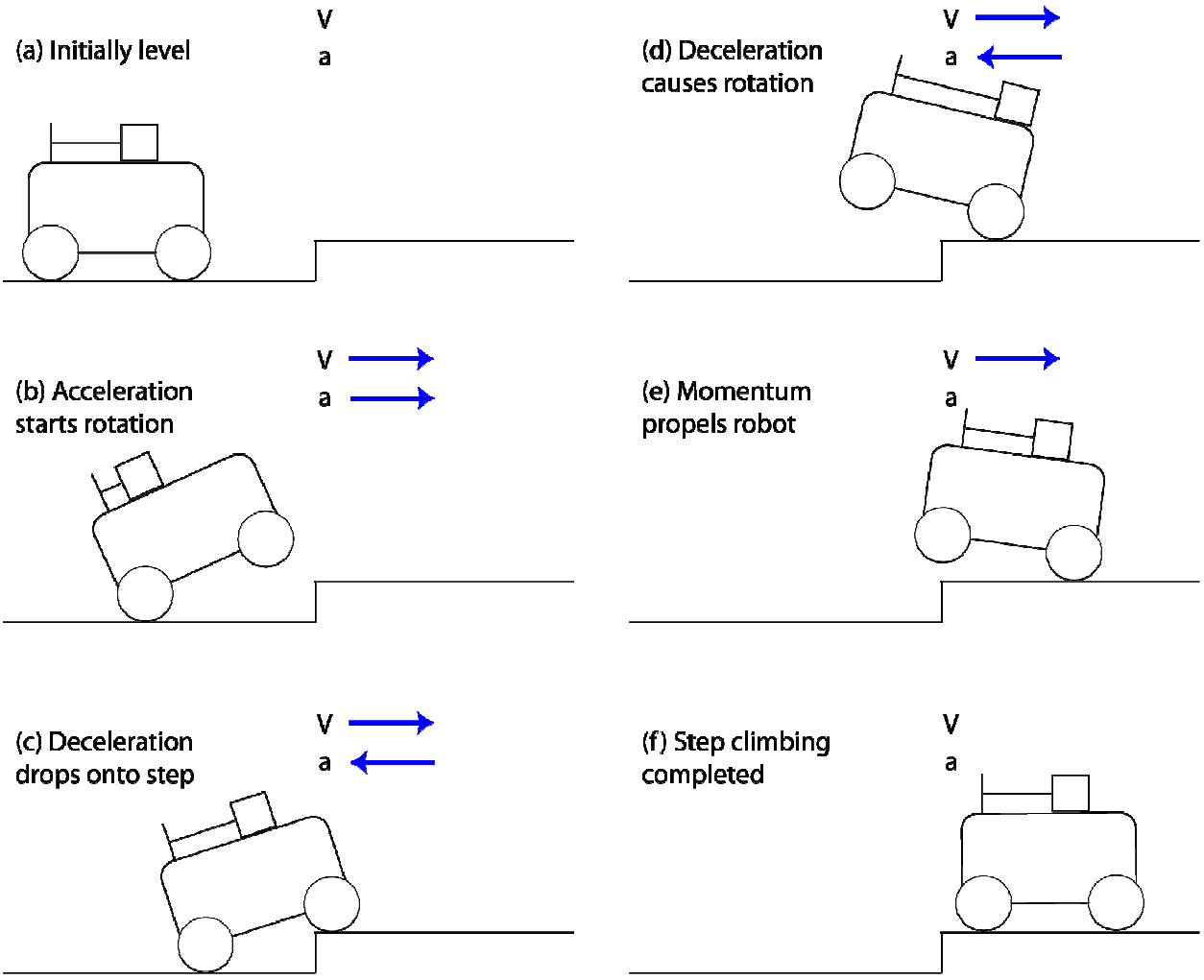

Figure 2.5: Diagram showing how an inertially actuated sliding mass can help a robot hop over a step. The straight line connecting the sliding mass to the base robot represents a linear spring.

For this inertially actuated step climbing process, the robot's acceleration causes the mass to slide back and compress the spring (see Fig 2.5b). This accomplishes two things – the center of gravity slides backward and an inertially induced reaction force is created in the spring that pushes back on the robot. When the robot is near the step (see Fig 2.5c), deceleration of the robot propels the mass forward and reverses the effects created during the lift-off. Now, the center of gravity is shifted forward and the inertial forces stretch the spring, which pulls forward on the robot. After the rear wheels are lifted as a result of this torque (see Fig 2.5d), the robot will use any forward momentum remaining and roll over the step before the rear wheels fall back down (see Fig 2.5e).

With the proper design and operational parameters, the spring-mass slider can both raise and shift the center of mass in order to aide in rotating the robot as well as create an additional reaction force from the spring compression. These changes allow for moderate accelerations to be used in both phases of the step-climbing process. The following sections will discuss the dynamics of the robotic system in each phase of step-climbing. These derivations will assume the properties of the base robot are fixed, so only parameters of the passive dynamic platform can be tuned to enable step climbing.

2.4 Displacement of sliding mass during lift-off: Focusing on the mass-slider attached to the robot, this section will derive the equations of motion of the mass as it slides horizontally with respect to the robot base. This discussion pertains to the lift-off phase of step-climbing – the pop-up equations are derived later. A close-up view of the mass-slider and its associated reaction forces are shown in Figure 2.6. The mass is rotated at an angle of θ because of the robot's rotation, though the equation to compute this rotation has not been shown, yet. The initial condition for θ is 0^o, since the robot will be assumed to start on a flat surface. Lines are drawn to the rear axle, point O.

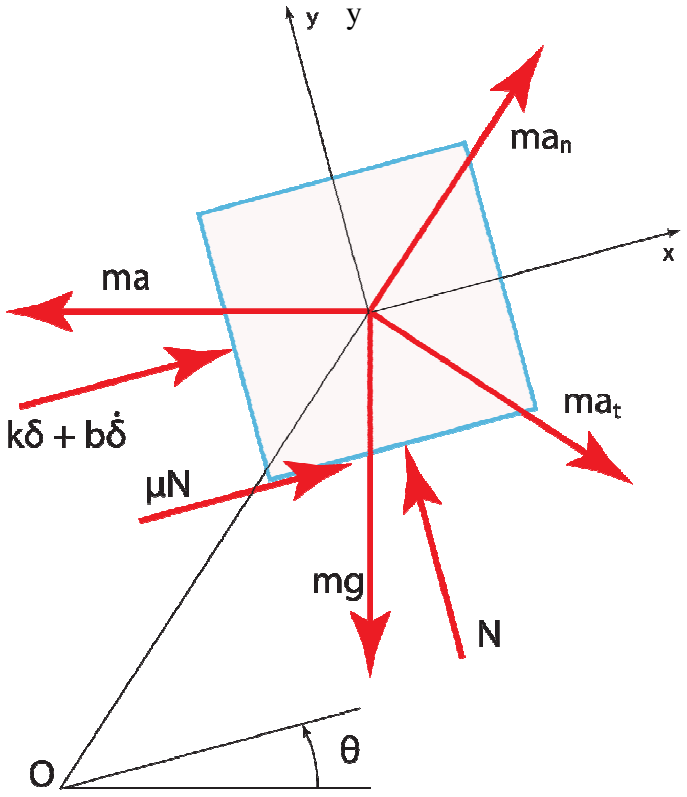

Figure 2.6: Free body diagram of the mass-slider during the lift-off phase

The reaction forces from the spring are dependent on the spring deflection, δ, which is positive in compression. The spring force causes a torque about the rear axle, helping the robot to lift its front wheels if the spring is compressed (δ > 0). Also, a positive deflection reduces the negative effect of the slider's mass on the total torque at the rear axle, which will be discussed in the next subsection. From these two effects, a larger, positive spring deflection is advantageous to helping the robot rotate on its rear axle. A summation of forces in the local-horizontal direction yields the equation of motion of the mass-slider, shown in Equation 2.5,

$$\begin{aligned} m\ddot{\delta} = {} & mg\sin\theta + ma\cos\theta - k\delta - b\dot{\delta} - \mu\left(sign\,\dot{\delta}\right)\left(mg\cos\theta - ma\sin\theta\right) \\ & - m\alpha\left(b_y + b_x\mu\left(sign\,\dot{\delta}\right)\right) + m\omega^2\left(b_y\mu\left(sign\,\dot{\delta}\right) - b_x\right) \end{aligned} \tag{2.5}$$

where variables can be defined using Figure 2.4 or the Nomenclature index on page viii. Minor details such as spring damping, b, and friction between the sliding mass and the base robot, μ, are shown for completeness. The assumption is being made that the friction of the sliding mass is low enough that the slider does indeed move and is not held by friction. From the equation of motion for the mass-slider and knowing that a larger deflection value, δ, is helpful for the lift-off process, several design parameters can be adjusted. The first four components of Equation 2.5 are largely the most significant, while the last three are smaller details that will be ignored for this design discussion – this simplification is justified in Appendix A. These effects are important and are included in the full dynamic model but are not significant when tuning design values. Using these assumptions, Equation 2.6,

$$\ddot{\delta} \approx g\sin\theta + a\cos\theta - \frac{k}{m}\delta - \frac{b}{m}\dot{\delta} \tag{2.6}$$

shows a simplified form of Equation 2.5 that is useful for designing the spring-mass system. This simplification should not be used as an approximation of the system dynamics; the only functional use of this equation is to investigate how changing certain design values *generally* changes the deflection of the mass-spring system. Equation 2.6 shows that increasing the slider mass will tend to increase deflection of the spring so long as the extra mass does not reduce the acceleration available. Also, a lower spring constant and damping value will lead to larger

spring deflections. These spring properties are completely independent of other parameters and can be varied without concern for worsening the effects from other values.

An important advantage to using a spring in this system is that the reaction force on the robot can actually be larger than the inertial forces on the sliding mass. If the sliding mass were fixed to the robot, the force acting on the robot is merely the product of mass and acceleration of the slider. However, since the mass is free to oscillate and compress a spring, the maximum deflection of the spring will create a force greater than the inertial force of the mass. This means that a lower acceleration is required to start the lift-off phase than if the mass were fixed to the robot.

For the values of robot rotation ($\theta < 45^o$) and acceleration ($a < g$) that are used in the step-climbing process, larger rotation values lead to larger spring deflections. This shows that spring compression – which assists in lifting the front wheels – is encouraged by the actual rotation of the robot. This positive feedback indicates that once the lift-off process begins, less acceleration is needed to continue deflecting the spring; so, the challenge is solely to have enough acceleration to start lifting the front wheels.

While the selection of design parameters is not performed here, this rough analysis of how certain parameters affect the deflection of the spring will be useful when the final design is parameterized.

2.5 Rotation of robot during lift-off: With the dynamics of the mass-slider defined, the rotation of the robot during the lift-off phase can be analyzed. Removing the mass-slider, a free-body diagram of the robot is shown in Figure 2.7. The mass-slider has been replaced by the reaction forces that were created by its addition: a normal force from the mass contacting the robot base, a spring force where the spring connects to the robot base, and a friction force that is dependent on the mass velocity direction. Evaluation of the torque about the rear axle will show the robot's ability to lift its front wheel during acceleration.

To find the rotational dynamics of the robot about its rear axle, the torques created from the reaction forces must be analyzed. A summation of these torques, shown in Equation 2.7, indicate how capable the robot is of rotating,

$$\begin{aligned} I_1\alpha_1 = {} & \left(k\delta + b\dot{\delta}\right)b_y + Ma\left(a_x \sin\theta + a_y \cos\theta\right) - Mg\left(a_x \cos\theta - a_y \sin\theta\right) - M\alpha L_M^2 \\ & + \left(mg\cos\theta + m\alpha b_x - m\omega^2 b_y - ma\sin\theta\right)\left(\mu h\left(sign\,\dot{\delta}\right) - b_x\right) + R_{2y}L + T_w \end{aligned} \tag{2.7}$$

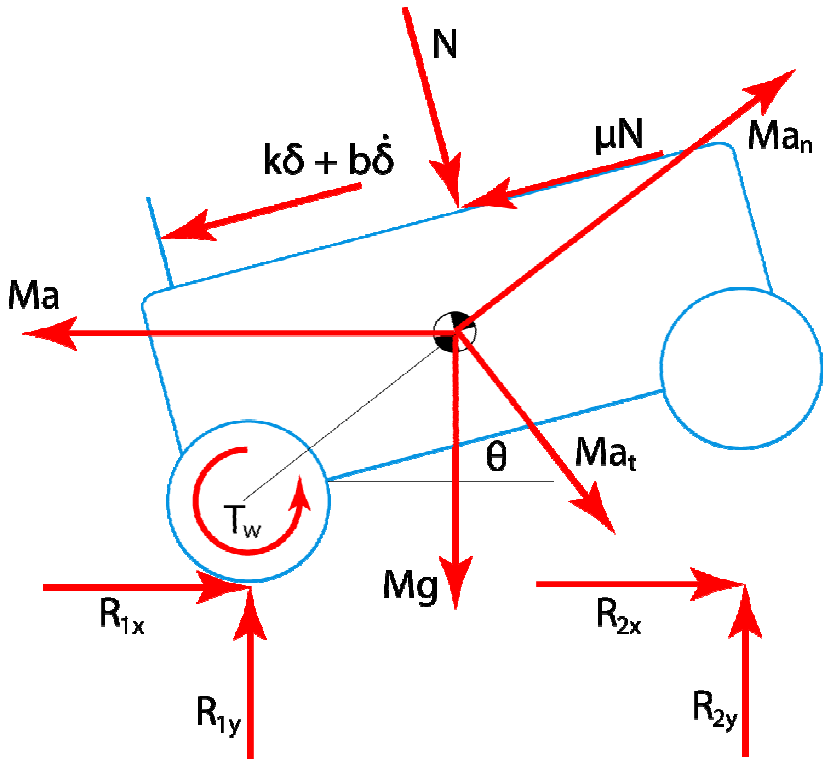

Figure 2.7: Free body diagram of the robot during the lift-off phase

where variables can be defined using Figure 2.4 or the Nomenclature index on page viii. The mass, M, and center of mass location of the robot's base, a_x and a_y, are essentially fixed, though they may vary depending on the rigid structure needed to add the mass-slider. The sliding mass moves and changes the effective center of gravity location for the entire robot, but the sliding mass location is denoted using b_x and b_y and this position doesn't affect the center of gravity of the base robot. The vertical reaction forces at the tires are not constant values. Rather, these reactions vary by assuming the tires to be spring-damper systems. The reaction at the front axle is coupled into the rotation equation – deflection of the tire occurs during negative rotation. The deflection at the rear axle is modeled as a separate system and will be discussed later.

Inspection of terms that involve the robot's rotation, θ, show that the rotation of the robot is advantageous to helping the robot to continue to rotate. For reasonable values of acceleration and geometrical configurations, the summation of all terms involving the robot's rotation increase as θ increases, which leads to a higher angular acceleration. As expected from analysis of the sliding mass, if the acceleration of the robot is enough to begin the lift-off process, then this acceleration is more than adequate to continue rotating the robot.

The rotational dynamics expressed in Equation 2.7 can be simplified for design parameter selection, as explained in Appendix A. Equation 2.8 shows this reduced equation,

$$I_1\alpha_1 = \left(k\delta + b\dot{\delta}\right)b_y + \left(-mg\cos\theta + ma\sin\theta\right)b_x + T' \tag{2.8}$$

where terms that are relatively small or consist of parameters of the base robot that are fixed are lumped into a temporary variable, *T'*. The configurable parameters of the design remain the spring stiffness (k), mass of the slider (m), and position of the slider (b_x and b_y). While it may seem that a higher spring constant would lead to more rotation, this is not necessarily true. A higher spring constant results in a lower deflection value, according to Equation 2.6. For this reason, the product of the spring constant and spring deflection (which is the important value in the rotation equation) does not necessarily rise as the spring constant increases. Since a differential equation is being solved, the true effect of a higher spring stiffness is dependent on the rest of the system, so an easy answer to how the spring stiffness should be modified is not available from Equation 2.8. Increasing the vertical offset of the mass-slider, however, does directly relate to a faster rotation of the robot.

The effects shown in Equation 2.8 indicate that increasing the mass of the slider negatively affects rotation, but this mass increase has been shown to increase spring deflection – a positive factor in helping the robot to rotate. This increase in spring deflection also lowers the b_x value, reducing the negative effects shown in Equation 2.8. Understanding of the full effects of increasing mass is required, but an appropriate increase in mass can assist in the lift-off process.

In summary of the lift-off process for step-climbing, rotation of the robot can be assisted by careful adjustment of three design parameters of the mass-slider: increasing mass of the slider, increasing height of the slider, and decreasing spring stiffness. These options will be weighed against the requirements to assist in the pop-up phase of step-climbing. Also, the limiting factor for lifting the front wheel of the robot is when the robot is still parallel to the ground. Once the robot begins to rotate, less strenuous acceleration is required to continue lift-off.

2.6 Displacement of sliding mass during pop-up: Once the wheels have reached the top of the step, the robot decelerates quickly to force the sliding mass forward and rotate the

robot about the front axle – pulling the rear wheels up over the step. This section will focus on the sliding mass as it is inertially pushed forward.

The dynamics of the pop-up action are very similar to those for the lift-off action, though the center of rotation is now at the front axle. A free body diagram of the sliding mass during pop-up is shown in Figure 2.8. Lines indicate the relationship between the sliding mass and the front axle at point Q. A force balance on the slider in the local-horizontal direction is shown in Equation 2.9,

$$\begin{aligned} m\ddot{\delta} = mg\sin\theta + ma\cos\theta - k\delta - b\dot{\delta} - \mu\left(sign\,\dot{\delta}\right)\left(mg\cos\theta - ma\sin\theta\right) \\ - m\alpha\left(d_y - d_x\mu\left(sign\,\dot{\delta}\right)\right) + m\omega^2\left(d_y\mu\left(sign\,\dot{\delta}\right) + d_x\right) \end{aligned} \tag{2.9}$$

where variables are defined from Figure 2.4 or the Nomenclature index on page viii. The main differences evident in the sliding mass equation during pop-up are that distances are now denoted using d_x and d_y variables, which are referenced to the front axle, and two sign changes in the rotational accelerations to account for reversed rotation.

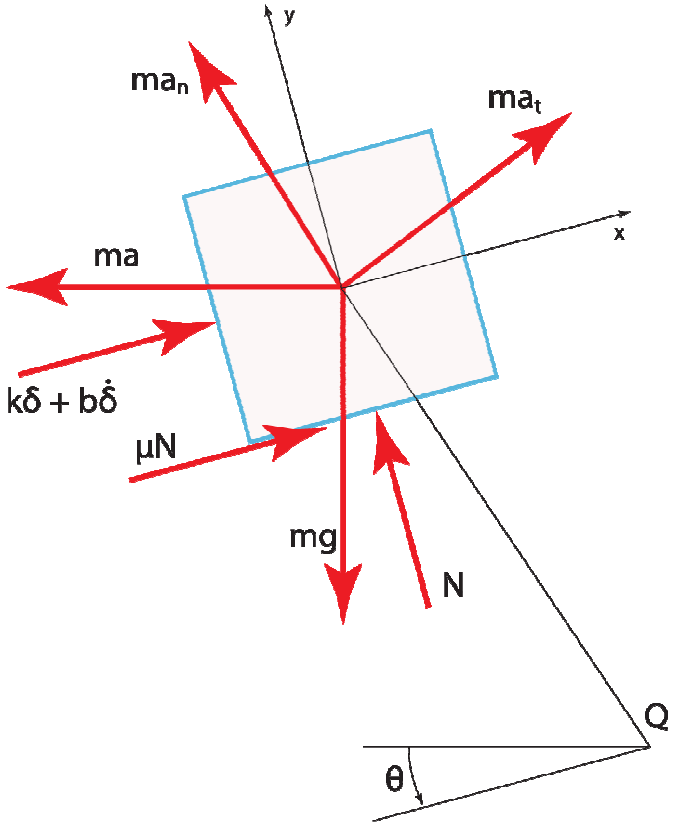

Figure 2.8: Free body diagram of the sliding mass during the pop-up phase

Making simplifications to the dynamics based on significance with respect to other values, as justified in Appendix A, Equation 2.10 was created,

$$\ddot{\delta} \approx g \sin\theta + a\cos\theta - \frac{k}{m}\delta - \frac{b}{m}\dot{\delta} \tag{2.10}$$

which shows how spring deflection depends on the most important terms from Equation 2.9. This reduced relationship is exactly the same as was derived for the sliding mass during lift-off. Since the equations are equivalent, the lessons available for the sliding mass during pop-up are similar to those from the lift-off process. While higher decelerations may be obtained without redesigning the robot – through the addition of the better braking – this discussion will continue to focus on tuning the parameters of the passive dynamic system.

Based on analysis of a basic robot, a viable option to assist pop-up is to shift the center of mass forward, which is accomplished here by stretching the spring in tension. This means that a negative deflection value, δ, is desired for the pop-up action. Equation 2.10 indicates that a looser spring constant will encourage the slider to deflect forward. While the spring deflection terms may seem to show that a larger spring constant will lead to more negative spring deflections, the value of δ will be negative for a majority of the pop-up action. Therefore, the advantage of a large spring constant during the brief period of spring compression ($\delta > 0$) is outweighed by the disadvantage during the longer period of spring tension ($\delta < 0$). This balance between compression and tension in the spring shows that increasing the slider's mass can also be beneficial to stretch the spring – so long as the extra mass doesn't limit the possible deceleration.

While not a design parameter, inspection of the effects of the robot's rotation, θ in Equation 2.10 is helpful to understand the system's limitations. Similar to the lift-off process, the rotation of the robot assists in sliding the mass in the desired direction – forward, for the pop-up process. As θ gets smaller and eventually goes negative, the sum of terms involving θ also decrease, helping to extend the spring. This shows that the limiting condition is when the robot is rotated at its maximum. If the deceleration and system parameters are able to adequately stretch the spring at the robot's maximum rotation, then once the robot does begin to rotate, less deceleration is required to maintain this spring extension.

2.7 Rotation of robot during pop-up: Having defined the dynamics of the sliding mass, the actual rotation of the robot during the pop-up phase can be established. Using a free body diagram of the robot when rotating about its front axle, shown in Figure 2.9, the effects of each reaction force on the pop-up action were derived for Equation 2.11,

$$I_2\alpha_2 = (k\delta + b\dot{\delta})d_y - Ma(c_x \sin\theta - c_y \cos\theta) + Mg(c_x \cos\theta + c_y \sin\theta) - M\alpha L_M^2 + (mg\cos\theta - m\alpha d_x - m\omega^2 d_y - ma\sin\theta)(\mu h(sign\,\dot{\delta}) + d_x) - R_{1y}L + T_w \quad (2.11)$$

where variables are defined via Figure 2.4 or the Nomenclature index on page viii. With the tires modeled as spring-dampers, the vertical reaction force at the rear axle, R_{1y}, varies based on the actual deflection of the tire. The main differences from the lift-off equation are sign changes to account for rotation about an elevated point (the front wheel on top of the step) and the use of variables *c* and *d* to reference distances to the front axle. The horizontal reaction force at the rear tires disappears once the rear wheels begin to pop up, so it doesn't warrant inclusion in the dynamics of the robot's rotation.

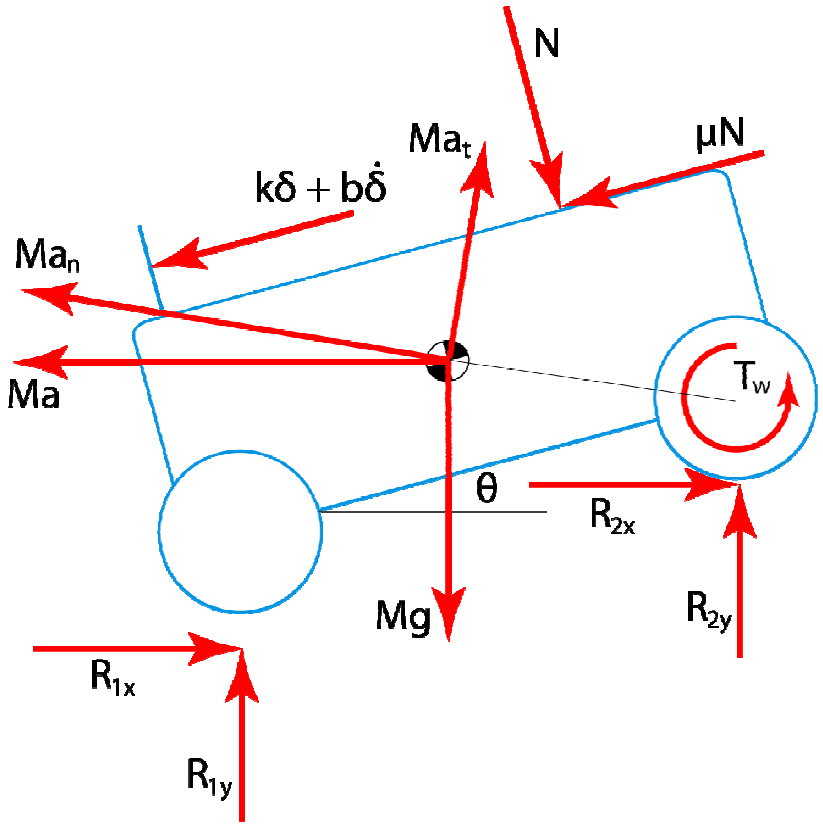

Figure 2.9: Free body diagram of the robot during the pop-up phase

By grouping terms from Equation 2.11 that are either fixed values or very small compared to other terms (as explained in Appendix A) into a temporary value, T', Equation 2.12 was created,

$$I_2\alpha_2 = \left(k\delta + b\dot{\delta}\right)d_y + \left(mg\cos\theta - ma\sin\theta\right)d_x + T' \tag{2.12}$$

in order to better understand the relationship between the design values and the robot's rotation. During pop-up, a negative angular acceleration is desired to pop the rear wheels up towards a horizontal, no-rotation condition. To achieve this, the spring stiffness (k), slider height (b_y), and slider mass (m) are again the terms able to be modified.

Like analysis of the lift-off phase, it appears that a larger spring constant, coupled with a negative spring deflection, would be conducive to rotating the robot. However, a larger spring stiffness reduces spring deflection, as discussed in the last section. The product of these two values, which is the value in the rotation equation, does not necessarily rise by only increasing the spring stiffness. Another design parameter is the height of the sliding mass and Equation 2.12 readily shows that a higher position will help in the pop-up rotation dynamics once the spring is in tension. While the spring is in compression, a higher slider position resists pop-up, so the slider should not be set so high that this effect is too great or the robot will fall over backward. The final design value that affects Equation 2.12 is the mass of the slider, but the effect of changing this mass is not as clear as for other parameters. When d_x is positive, meaning the slider is behind the front axle, increasing mass reduces the angular acceleration of the robot. However, the rotation of the robot can be sped up by increasing the mass if d_x is negative – when the slider is in front of the front axle. Depending on the center of oscillation of the mass, the slider may be on either side of the front axle for a longer time, so knowledge of the specific system is required before determining whether increasing slider mass will help the rotation of the robot.

Equation 2.11 also shows that as the robot completes the pop-up action – as θ gets smaller and goes negative – the process requires less acceleration to continue. The summation of terms involving the robot's rotation decreases as the rear wheels pop up, indicating a quicker angular acceleration. This shows that the limiting point for the pop-up action is at the initial rotation of the robot.

In summary of the pop-up phase, this part of the step-climbing process can be assisted by modification of two design parameters: decreasing spring stiffness (to shift slider position forward) and increasing the slider height. A third option, increasing the slider mass, is effective if done appropriately. A firm understanding of the effect of changing mass should be known before modifying the slider. Finally, if the robot is able to initially raise its rear wheels, then the system is fully capable of completing the pop-up action.

2.8 Deflection of tires on axle of rotation: The equations for the rotation of a robot during each phase of inertially actuated step climbing have been defined, including the tire deflection of half of the tires. By modeling the tires as spring-damper systems, the actual deflection of the tires can be simulated. This correction, however, only accounts for the tires that are not on the axle of rotation. The deflection of these tires is implemented in an independent set of equations. This section will present the deflection of the rear tire during lift-off. Modeling of the front wheel deflection during pop-up is very similar, so it is not shown.

The equilibrium force of the tire is first computed from a vertical force balance of the system in Figure 2.7, shown in Equation 2.12,

$$\begin{aligned} R_{1y,eq} = {} & \left(k\delta + b\dot{\delta}\right)\sin\theta + m\alpha\left(a_x\cos\theta - a_y\sin\theta\right) - m\omega^2\left(a_x\sin\theta + a_y\cos\theta\right) + Mg \\ & + \left(mg\cos\theta + m\alpha b_x - m\omega^2 b_y - ma\sin\theta\right)\left(\cos\theta + \mu\left(sign\,\dot{\delta}\right)\sin\theta\right) - R_{2y} \end{aligned} \tag{2.12}$$

where variables have been defined in previous figures and in the Nomenclature index on page viii. This force represents the force that would be in the tire if the tire was in equilibrium and its deflection was not changing. Relating this equilibrium force to the actual force absorbed in the tire based on the current deflection results in an unbalanced force in the tire deflection. This unbalance force is then used in Equation 2.13 to find the "deflection acceleration" of the tire, $\ddot{\delta}_T$,

$$(m+M)\ddot{\delta}_T = F_{unbalance} \tag{2.13}$$

where the total mass of the robot is used to compute how quickly the tire deflects. Integration of this acceleration yields a new tire deflection for the next time step. The tire deflection is generally very small and doesn't result in a significant robot rotation, so no analysis of this model is used to design the robot's geometrical parameters. This is a fairly simplified model of

tire deflection with some imperfect assumptions, but the small correction to the robot's dynamic model helps to capture all aspects of an actual robot.

2.9 Results and Discussion: Analysis of the two stages of inertially actuated step-climbing has shown that two design parameters of the sliding mass platform can be independently controlled with predictable results. Increasing the slider height and decreasing spring stiffness will result in easier rotation about each axle. Another option, increased slider mass, can have positive effects in certain situations. A fourth parameter which has not been discussed is the horizontal equilibrium position of the spring system. This subsection will discuss the findings from deriving the equations of motion for this system.

The dynamics of rotation for each phase of step-climbing show that increasing the slider height is helpful in rotating the robot around each axle. If increasing the slider height is advantageous to rotating the robot, why shouldn't we simply increase the height to a very large value? There are two concerns with increasing the height of the slider. First, a main goal of using passive dynamics is to reduce the required height of this additional platform. If height was not an issue, a rigid mass could simply be fixed very high and the robot could then use the equations derived for a basic robot to climb a step. By keeping the height of the slider at a moderate value, the robot does not turn into a geometrically awkward system. In addition, increasing the center of gravity of the robot too much can cause instability in other situations as the robot will become top-heavy.

More importantly, increasing the height of the slider actually reduces the height of the step that the robot can climb. When rotated about the rear wheel, increasing the height of the slider pushes the slider further behind the rear wheel. At substantially high rotation and slider height, the robot will flip backward on its own because the slider will push the effective center of mass behind the rear wheel. Increasing the height of the slider reduces the maximum rotation allowed before the robot flips over on its own. Therefore, raising the slider is certainly advantageous to rotation, but a limit on the height exists where switching from lift-off to pop-up to complete the step-climbing process becomes unrealistic.

The second design parameter that must be carefully chosen in the spring stiffness. Prior analysis has shown that a looser spring will help to rotate the robot on each axle. The question could then be asked, why not infinitely reduce the spring constant to zero and have a free slider?

The main reason a spring is needed, as opposed to a freely sliding mass, is to gradually increase the reaction force on the robot's base. As a spring deflects, the reaction force will raise accordingly. With this gradual increase, the step-climbing process is more easily controlled, since it happens at a slower pace. If the mass was free to slide, the reaction force on the robot that the spring induces would not exist until the mass hit the rear limit of its travel. At this moment, the momentum of the mass would cause a very large, immediate, and catastrophic reaction that would flip the robot over instantly.

Neglecting the option of removing the spring entirely, analysis of the robot's rotation still shows that lower spring constants are ideal for increasing rotation. So, should the spring constant simply be reduced to a very small value, while still having a spring technically in place? The problem with this approach lies in the dynamics of the slider oscillation. An oversimplified analysis of the slider shows that the frequency of oscillation is proportional to the square root of the spring constant. This means that a very low spring constant reduces the frequency of oscillation of the slider. In order to complete the pop-up action, the slider needs to be thrown forward very quickly or the rear wheels will hit the step before raising high enough to get over the step. A low frequency of oscillation will not push the slider forward fast enough to safely pop-up the rear wheels. Thus, a very low spring constant results in a low frequency of oscillation, which is detrimental to the overall step-climbing process.

Another geometrical concern when using very low spring constants is that the slider will need a much longer track to traverse. A lower spring constant will allow more deflection, but the required length for this deflection may make the robot excessively long and limit the maneuverability of the robot in smaller areas. Also, a robot with a long bar sticking out in the direction of the travel is not the safest option when operating in public environments.

While increasing the mass of the slider has been shown to increase deflection of the spring, this increase also negatively affects rotation of the robot when the slider is between the axles. The true effect of increasing mass will depend on the actual robot that this system is being used on, but it is possible that using a larger mass to increase spring deflection may end up resisting rotation because of the increased gravitational forces on the robot. This scenario is typical with very stiff springs, since the mass will not move much and will stay near the center of the robot. A lower spring constant will allow more deflection and reduce the impact of gravity

resisting rotation of the robot. Increasing mass may also limit the allowable accelerations of the robot, which will have a negative impact on step climbing.

The final design parameter is the equilibrium position for the sliding mass. While the spring deflection governs that actual position of the slider, shifting the equilibrium position forward or backward will directly assist or resist rotation about an axle. With an equilibrium position near the rear axle, lift-off will likely occur with little difficulty but pop-up will be nearly impossible since the sliding mass is so far behind the rear axle. The reverse is true, also, so the equilibrium position must be chosen so as to help each phase of step-climbing as much as possible.

The first step in designing a sliding mass platform is to choose design values to start with. If a simulation is available (which has been created and will be discussed in the next chapter), the step-climbing process should be simulated with these initial conditions. Based on knowledge of the dynamics of the equations and numerous trials with simulations and actual experiments, the following values are recommended as a general starting point. The equilibrium position of the sliding mass should be near the front axle. The pop-up action needs more help than lift-off, so starting the sliding mass more forward will be helpful in step-climbing. The mass of the slider is a hard decision, but a mass near 25% of the base robot's mass is reasonable. This will not too negatively affect the acceleration of the robot, but will be able to substantially move the center of gravity and create a reaction in the spring. The spring stiffness should be initially chosen so the maximum deflection (based on the available acceleration) of the sliding mass occurs around the rear axle. This will provide a large deflection to shift the center of gravity and create an adequate reaction force in the spring. The final design value to start with is the height of the sliding mass and this value can be chosen fairly arbitrarily. A reasonable choice is for this slider to be at least as high above the axles as the center of gravity of the base robot is between the axles. If there is a better, more convenient location for the slider, placing the slider there and tuning other parameters to allow step-climbing is a good approach.

After choosing the initial design values and attempting simulations or testing of the step-climbing process, tuning of the parameters is likely needed to achieve a smooth process. If the robot is not able to lift-off, decreasing the spring stiffness or raising the height of the slider will help. Increasing the mass of the slider in this case is also reasonable. If the robot can lift-off but will not pop-up, lowering the height of the slider or shifting the equilibrium position of the

spring system forward may encourage the system. Both of these options will reduce the likelihood that the robot will lift-off, so moderate adjustments should be made. Decreasing the mass of the slider may also help to start the pop-up process. If the robot can get its rear wheels up high enough but doesn't coast onto the step, an initial velocity of the robot will help to keep the robot moving forward. Finally, if the robot has no problem performing the step-climbing process but flips over forward after climbing the step, braking less hard or accelerating again once on top of the step may fix this problem without needing a system redesign. This situation is rare and is not a bad thing; this indicates the robot is fully capable of climbing a step but just needs to be controlled better.

Chapter 3: Simulations

After defining the equations of motion for step-climbing using an inertially actuated sliding mass on a wheeled robot, verification and simulation of the system is needed. Before implementing the system on an actual robot, much time and energy can be saved using a properly constructed simulation. This section will discuss a C++ simulation that was written to model the step-climbing process as well as the use of a commercially available software program used to compare results.

3.1 C++ simulation: To determine how the passive system can allow for step climbing by wheeled robots, a simulation was designed to model the system dynamics. Rather than using a software package, such as Matlab, that is already capable of simulating dynamic systems, a new simulation using C++ was created. By writing a custom simulation code, extra flexibility was allowed for modifying the simulation parameters and ensuring a firm understanding of the simulation process.

Implementing the equations derived in Ch. 2 and allowing for quick manipulation of data, an easy and effective simulation was created. The source code for the simulation is attached in Appendix B. The simulation uses a C++ graphics extension, OpenGL, to visualize the robot as it moves. This display shows a simplified model of the rotating robot, an indicator of the sliding mass position, and the location of the step. Extra tools, such as video rewind and zoom capabilities, were not added in order to maintain simplicity but could be implemented for future analysis.

Figure 3.1 shows an example of the two windows that are created during execution of the simulation. The top window mimics a camera fixed to global coordinates, focused on the step. The robot will move across the window and approach the step. The bottom window simulates a camera fixed to the local position of the robot and moves with the robot. The robot stays centered in the window and rotates as necessary as the "world" moves across the view. In each graphics window, the base of the robot is displayed as two wheels and a rectangular body. The center of gravity of the robot is shown as a black point inside the robot. Since this simulation is intended for a general wheeled robot, no further detail is necessary. A line extends from the base of the robot upwards to indicate the equilibrium height and horizontal position of the sliding

mass. The black point above the robot in the image shows the actual position of the sliding mass at the current time as the mass oscillates around the equilibrium point.

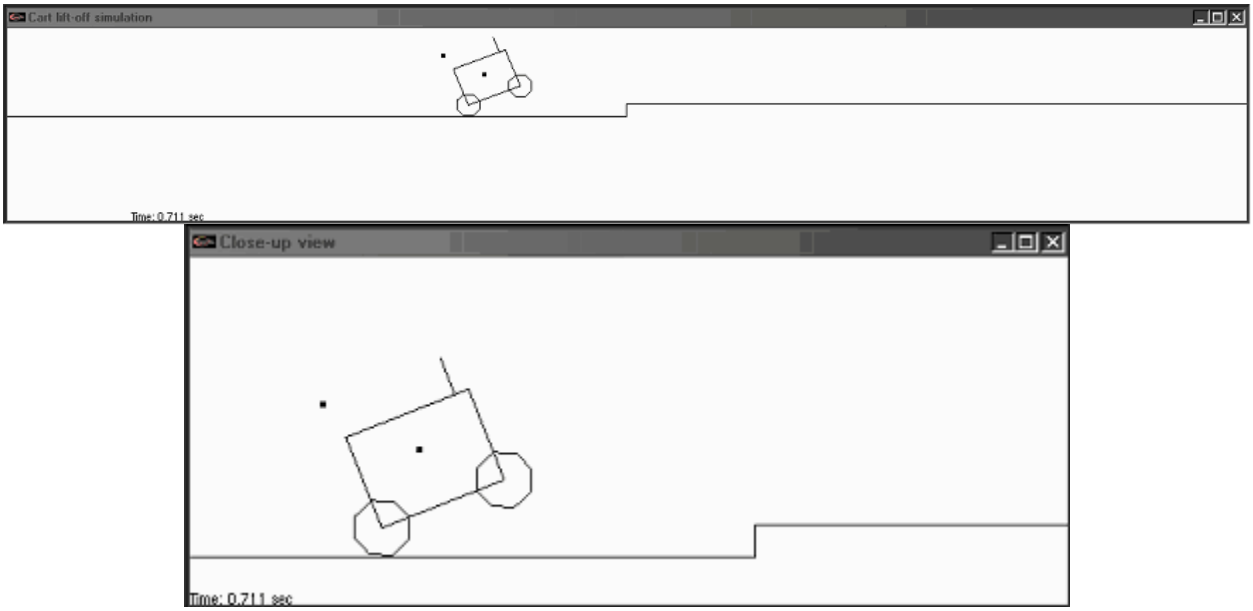

Figure 3.1: Example of the simulation display output. Two windows are created – one that is fixed on the step and one that is fixed on the robot.

By having these two windows operating simultaneously, full detail of the entire process is available. The globally fixed camera is able to show the motion of the robot with respect to the surroundings while the locally fixed camera gives a better indication of the rotation of the robot and location of the sliding mass. For a print-out of the system parameters during simulation, the simulation can write any data to file for later use. The inputs to the simulation include all variables in the dynamic models and an acceleration profile. The true input to a robotic system would be an electrical current or power draw, but this adjustment simplified the system and allowed the internal operation of the robot to be ignored. If a certain acceleration is attainable, the required power is inconsequential for this project. The code is fully documented to explain how the simulation is executed, so no further detail is given here.

3.2 Comparison to Working Model simulation: Using a commercially available software program that is designed to model dynamic systems, the C++ simulation was validated and also led to the discovery of a flaw in the software program itself, Working Model. After designing the custom simulation, a wheeled robot with a sliding mass was modeled in Working

Model to compare results. The main goal of this research is to climb a step, so the important value to monitor is the rotation of the robot as it hops over the step. Working Model provides an interface to input 2-dimensional shapes and add constraints to the system as needed to make a full model; this 2-dimensional approach matches how the system model was derived. The interface from Working Model is shown in Figure 3.2. Using a set of shapes, one robot that was tested with (which has not been discussed, yet) is modeled with the appropriate geometry and mass. The slider is attached to a platform above the base robot. Also shown are two indicators that make accessing system properties very easy – here, time and rotation are being tracked. The green bars connected to the wheels serve as the acceleration input for the system

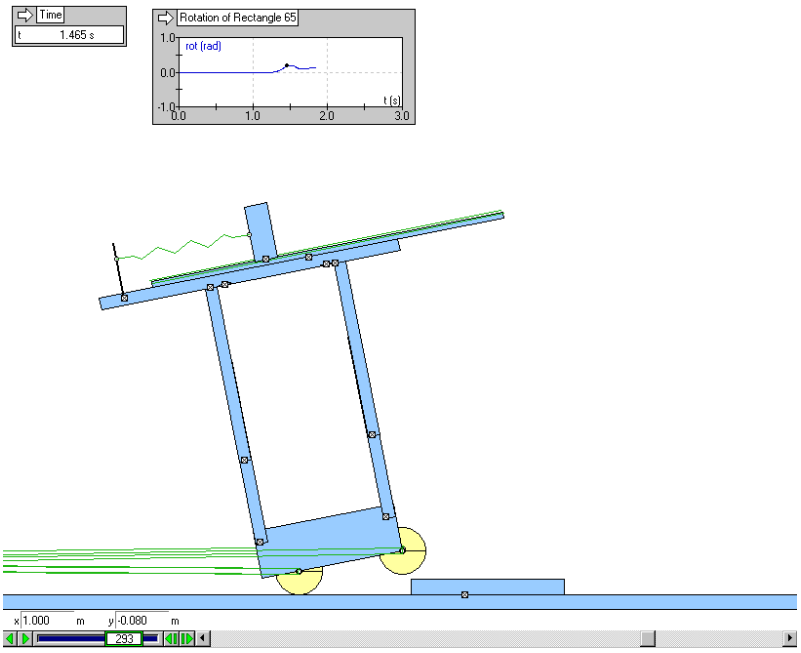

Figure 3.2: The interface for Working Model to create a dynamic system. The robot shown here is modeled after one that was used for testing.

The rationale for creating a separate simulation of the system instead of simply using this established simulation is to ensure a proper understanding of the system dynamics. If a custom simulation is created based on the derived equations of motion and this simulation can be confirmed through an external source, then the derivations can be assumed to be accurate. With a valid comprehension of the system dynamics, the effects of changing certain design values can be readily analyzed based on the dynamic equations. If Working Model was used by itself to create a step-climbing robot, the design selection would be largely an exercise in trial-and-error.

In theory, with a similarly modeled robot and a similar acceleration input, the two simulations should provide similar robot rotations throughout the simulations.

After inputting similar robots to each simulation, the rotation profiles of the two simulations were not in agreement. The C++ simulation was thoroughly inspected and debugged until it was determined that it was working as desired. For a test with constant acceleration, the two simulations were in agreement until the robot actually started to rotate. The time elapsed until rotation began and the deflection of the spring-mass slider were nearly exact between simulations until the front wheel lifted off. At this point, however, the simulations diverged as the Working Model simulation rotated the robot faster than the C++ simulation. Equation 2.7 is the dynamic model used in the C++ simulation and shows that rotation will begin once the total torque about the rear axle becomes positive. The magnitude of its angular acceleration then depends on the moment of inertia of the robot about the rear axle. For this reason, the moment of inertia was inspected for accuracy between the two simulations.

To investigate how the slider affected the moment of inertia of the system, the entire robot was considered to be massless except for the sliding mass. The C++ simulation then calculated the moment of inertia according to Equation 3.1,

$$I = mL_B^2 \tag{3.1}$$

where I is the moment of inertia, m is the mass of the slider, and L_B is the distance between the slider and the rear axle. Since the rigid parts of the robot were considered to be massless and the distance between the mass and the rear axle is sufficiently large (to ignore the moment of inertia of the sliding mass about its centroid), this equation is capable of representing the moment of inertia of this specific robot.

After ensuring this was the appropriate calculation, the moment of inertia used in Working Model needed investigation. Working Model does not provide a straight-forward method to access the moment of inertia of a system, so a "drop-test" was used to calculate the moment of inertia. By pinning the rear wheels above the ground and dropping the robot from horizontal to allow free rotation about the rear axle, the angular acceleration of the robot was found from Working Model. The total torque about the rear axle is easily calculated based on gravity: knowing the total torque and the angular acceleration of the robot provides an indirect method to calculate the moment of inertia.

Using the direct calculation based on geometry that was used in the C++ simulation, the moment of inertia for the robot being tested was 4.63 $kg \cdot m^2$. However, analysis of the Working Model data showed the moment of inertia to be 0.608 $kg \cdot m^2$. Since the reasoning for this error is not immediately obvious, one geometrical parameter at a time was varied in Working Model to see how the moment of inertia changed. Some parameters affected the moment of inertia as expected: doubling the mass of the slider, m, doubled the moment of inertia but spring stiffness, k, had no effect of the moment of inertia. However, changing the horizontal displacement of the slider, b_x, affected the moment of inertia more than expected.

Increasing the horizontal displacement increased the moment of inertia quadratically, as shown in Table 3.1. According to Equation 3.1, the moment of inertia should increase quadratically with the total distance between the slider and the rear axle, L_B. As evident in Table 3.1, the total distance changes very little as the horizontal displacement is varied – a result of a large vertical displacement of the slider from the rear axle. What this shows is that the moment of inertia in Working Model is quadratically proportional to the horizontal distance, rather than the total distance, between the slider and the axle. To verify that this explains the difference between the custom simulation and Working Model, Table 3.1 also shows the moment of inertia calculated directly in two ways: first using the total distance between the slider and the axle and second using the horizontal offset. The second approach matches the results from the Working Model simulation, which shows how the moment of inertia is being calculated and that Working Model is incorrect.

Table 3.1: Data used to find how Working Model computes the moment of inertia.

m (kg)	b_x (m)	L_B (m)	I from WM ($kg \cdot m^2$)	mL_B^2 – correct ($kg \cdot m^2$)	mb_x^2 – wrong ($kg \cdot m^2$)
2	0.1	2.00	0.0199	8.06	0.02
2	0.38	2.04	0.288	8.35	0.289
2	0.999	2.24	1.99	10.0	1.99

While a “drop-test” is useful to determine the moment of inertia during that test, the results may not necessarily translate directly to an accelerating wheeled robot equipped with a sliding mass. So, simulations were again setup to run at a constant acceleration to compare results. For this test, the simulations modeled an actual robot with no assumptions about massless components. The tires were made rigid to ensure all rotation was from the inertial forces on the system and to simplify the Working Model simulation. For a constant acceleration, the rotation of the robot is shown in Figure 3.3 for three simulations. The yellow plot, labeled

WM, is the rotation profile taken directly from Working Model. The pink plot, labeled Lb, is the rotation of the robot according to the C++ simulation using the correct calculations. The blue plot, labeled bx, is the rotation of the robot using a modified version of the C++ simulation that uses the incorrect calculations that are thought to be used by Working Model. This graph shows that by putting the error that was found from Working Model into the C++ simulation, the results of the two simulations are very similar. Since the moment of inertia calculation in the C++ simulation is still an estimation, the slight difference between the two similar profiles is acceptable and not surprising.

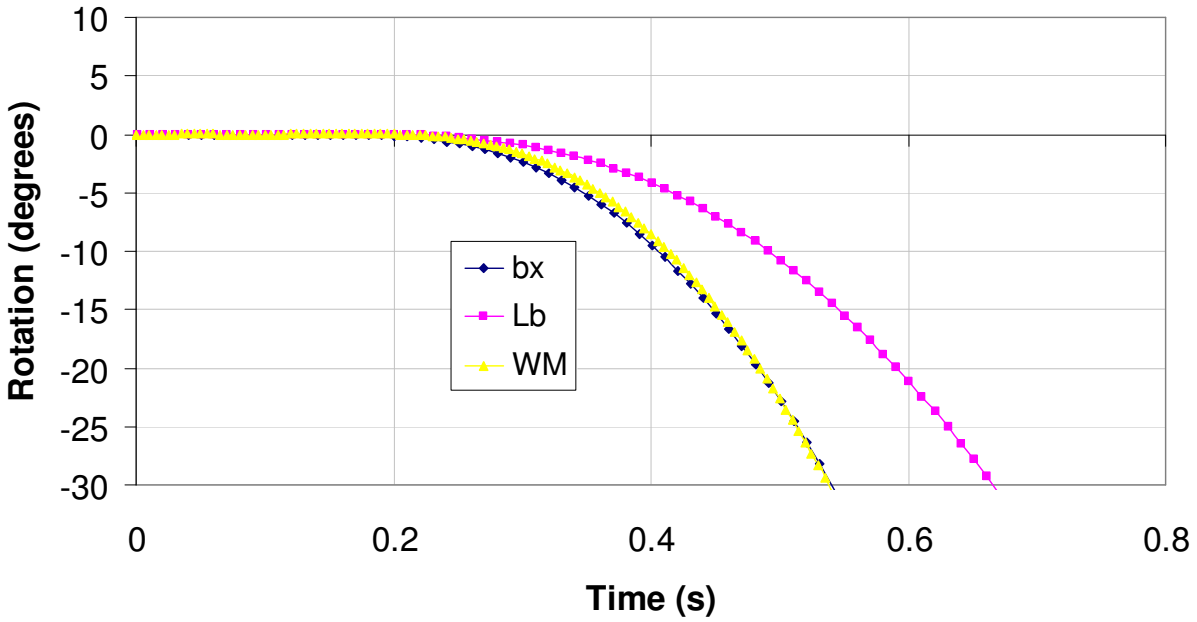

Figure 3.3: Rotation of the robot vs time for three simulations of constant acceleration. This shows that the incorrectly calculated moment of inertia explains the different in rotation profiles between simulations.

The results in Figure 3.3 show two important things. First, the error in the calculation of the moment of inertia in Working Model is confirmed. By putting that error into the C++ simulation, the simulations produced similar rotation profiles, so the error is apparent. Also, since the C++ simulation matched Working Model by temporarily modifying the logic, the comparison shows that the dynamics modeled are accurate. After taking the error out, the C++ simulation has shown that it has recreated the actual dynamics of this robot during acceleration.

With the moment of inertia properly accounted for during the lift-off portion of step-climbing, the simulations then verified the calculation of the moment of inertia during the pop-up phase. Shown in Figure 3.4, a plot similar to Figure 3.3 was made to compare the rotation

profiles of the Working Model simulation to results from the C++ simulation using the correct and incorrect calculations while the robot was decelerating and rotating on the front axle. This also verified that Working Model was incorrectly calculating the moment of inertia of the robot. Now that the simulation that was created has been verified, it can be used to design an inertially-actuated sliding mass to enable step climbing by wheeled robots.

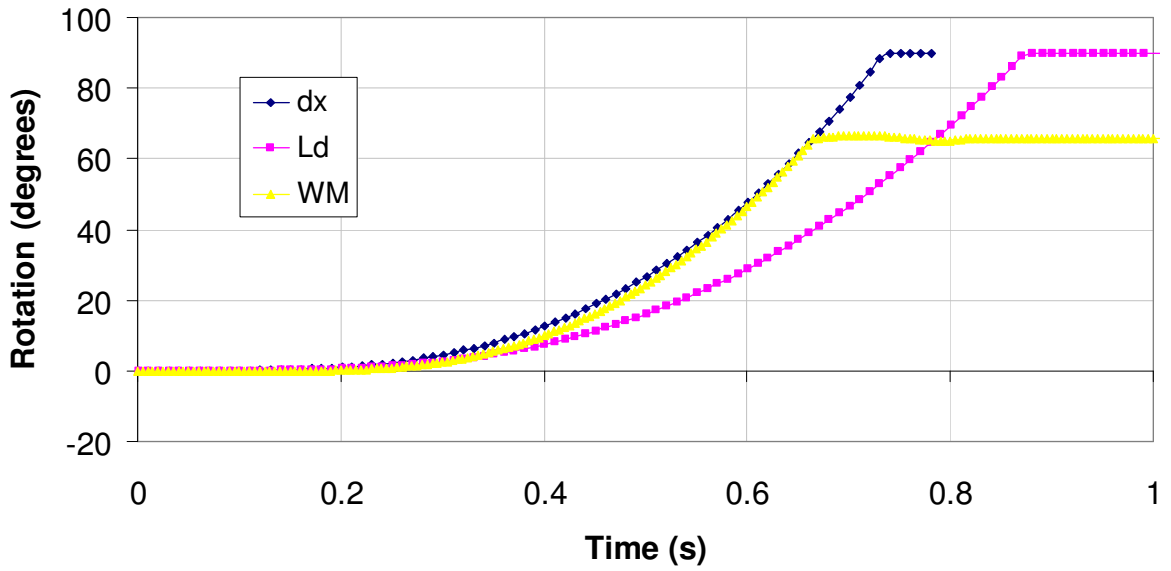

Figure 3.4: Rotation of the robot vs time for three simulations of constant deceleration. This shows that the incorrectly calculated moment of inertia explains the different in rotation profiles between simulations.

Chapter 4: Experimental Results

To validate that the simulations of the system are useful in the real world, an experiment was designed to test the ability of an inertially actuated sliding mass to allow a wheeled robot to climb a step. This section will present the robotic platform that was used for this "proof-of-concept" testing as well as the design process for the sliding mass system. Test results will be discussed and compared to the expected results from simulations.

4.1 Base robotic platform: Instead of designing a wheeled robot that had no purpose other than to hop over steps, and since this research is intended to extend to any wheeled robot, it was decided to use a standard hobby-version remote-controlled (RC) car as the robotic platform. An RC car is essentially a wheeled robot with no task other than driving, which provides a suitable base robot. With the car already capable of driving, the only efforts required to begin testing were to design and attach a sliding mass platform. A brief look into available RC cars revealed the Hot Wheels Dune Devil Pro RC car, which is the base robot that was tested for this project. Figure 4.1 shows the Dune Devil as purchased and also a modified version with a few small modifications made to assist in testing. The modifications include removal of the plastic mold, replacement of the shocks with rigid bars, and the use of an external power supply (not shown) versus an on-board battery pack.

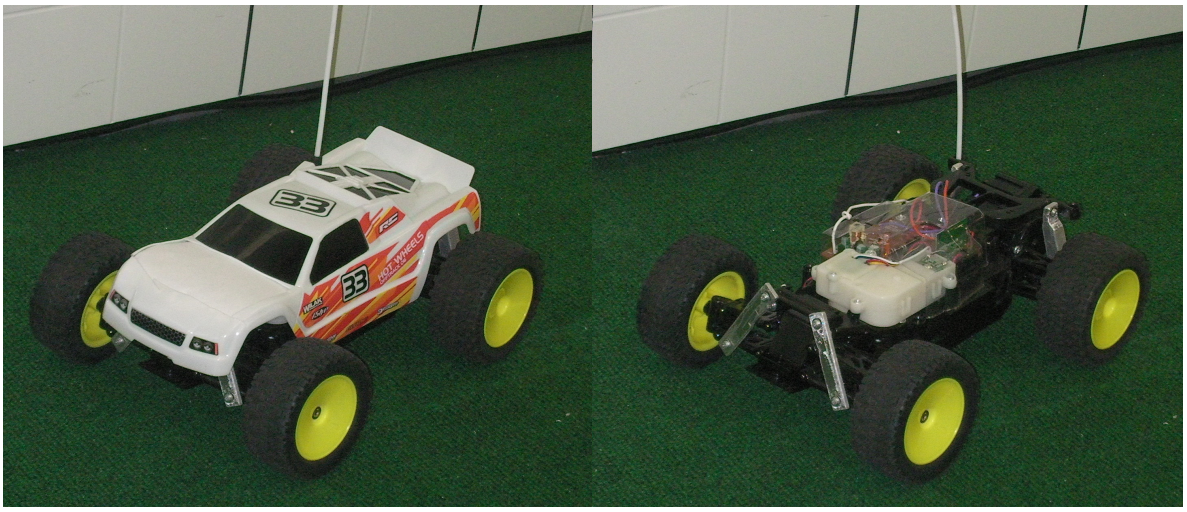

Figure 4.1: The Dune Devil RC car that was used for testing. Underneath the plastic cover is a rigid frame that was used to mount the sliding mass.

The main attribute that was desired in an RC car was high power, indicating the robot was capable of large accelerations during the step-climbing process. Active brakes would have been ideal, but no moderately priced RC car had brakes that would fit this project's needs. Control of all four tires would also have helped control the robot, but this again was an expensive option that was not absolutely necessary. The Dune Devil Pro was shown in an advertising video to have very high accelerations, based solely on appearance. Braking was accomplished by spinning the wheels in reverse, which should allow deceleration for the pop-up phase. This car is only powered on the rear wheels and modifications were planned to enable power on all wheels, but this did not turn out to be necessary.

A 27 MHz radio controller that was provided with the car was used to control the robot instead of making significant modifications to enable tethered control or the use of an on-board controller. The radio controller allowed for control of 4 levels of forward power and one level of reverse power. Using an accelerometer attached to the robot, the actual driving pattern of the robot can be tracked for comparison to simulations. By controlling the robot by hand and recording its acceleration, the internal dynamics of the robot (such as the required power draw and losses in the system) can be ignored and the dynamic simulation can be easily used.

To find the acceleration and deceleration limits of this robot, several acceleration tests of the system were performed. By applying full power during acceleration and deceleration and recording data from an accelerometer, a reasonable power limit can be estimated. With a small addition of mass (0.5 lbs) to simulate the added structure to support the sliding mass and a larger addition of weight (2.5 lbs) to simulate the sliding mass, a range of accelerations that can be expected while the sliding mass stretches the spring is apparent. While this technique is not exact, it will show the typical range of accelerations that the robot may experience during the step-climbing process. The extra mass was attached low on the robot so that the acceleration tests occurred with no rotation. With full power applied for each trial, the resulting acceleration was expected to show a relatively constant acceleration followed by a sharp change to a relatively constant deceleration.

Five tests were conducted for each mass and an example of the results is shown in Figure 4.2. At 0.5 seconds, full power was applied to the robot. After a short distance was traveled (at 0.8 seconds for the lighter robot and 1.1 seconds for the heavier robot), full reverse power was applied to brake the robot. The robot decelerated for about 0.5 seconds (to 1.3 seconds and 1.5

seconds for the lighter and heavier robots, respectively) and data after this point is irrelevant. These tests yielded predictable results; the test with a lower mass was able to accelerate higher than the test with more mass, though the deceleration values are very similar for each case. Equally important as the magnitudes of acceleration is the significant scatter in the data during deceleration. Either from the tires slipping, issues with the internal configuration of the accelerometer, or a possible harmonic in the system, the data during braking was very erratic and doesn't show a smooth braking pattern. Since, without any changes to the robot's dynamics, the braking is not relatively constant, we might expect the data during braking of the robot during the step-climbing process to be fairly unreliable. All tests yielded the same conclusion – the acceleration of the robot during the step-climbing process will likely be between 3 m/s^2 and 5 m/s^2 and the deceleration of the robot will average between 3 m/s^2 and 6 m/s^2. These values confirm the estimation of *reasonable* accelerations of a wheeled robot to be nearly 0.5 g's – which was used in an earlier section. However, the data of the deceleration should be used with caution as the data is much noisier than expected. Each test also showed a large spike in acceleration at the instant the robot starts moving, though this spike will be largely ignored since it is instantaneous and significantly higher than the rest of the data.

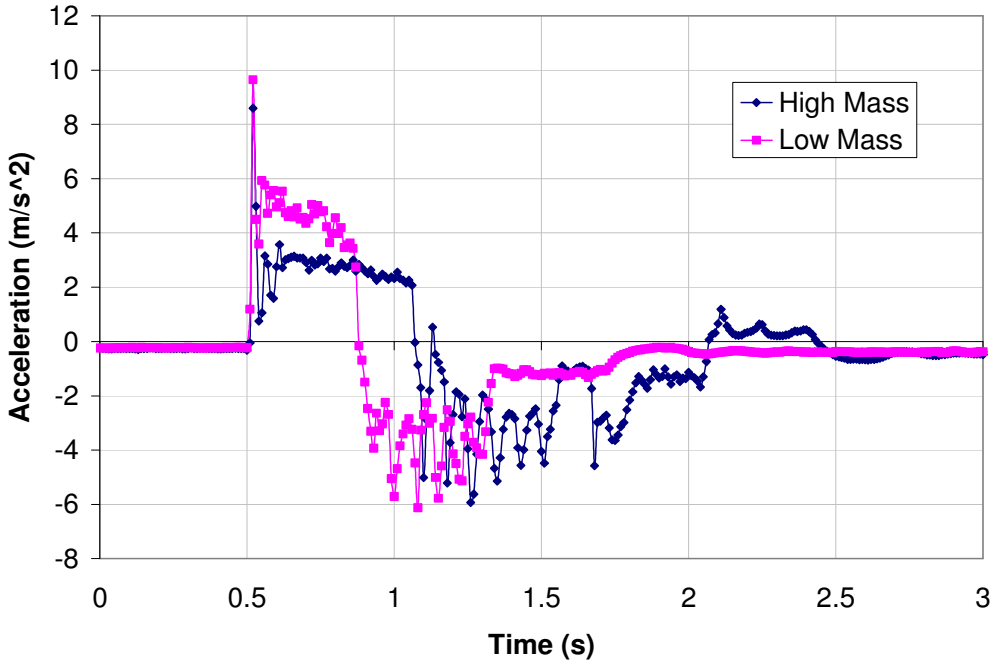

Figure 4.2: Acceleration profile of the RC car with masses added to simulate the additional platform. This shows the expected limits of acceleration of the robot throughout step climbing.

The base robot's specifications, such as mass and center of gravity location, are important but are not shown for the RC car by itself. Since a platform will be added to support the sliding mass, the base specifications will come from this more developed robot. This information will be provided in the following sections as required.

4.2 Preliminary design and testing: The first attempt at designing a step-climbing robot involved designing a robot using simulations, building the robot, and verifying that the simulations could match the actual rotation profile of the robot. This first step was done before the platform for the sliding mass had been designed, so the initial simulation was just an estimate of how the slider could affect an approximated robot. By modeling the base RC car and some extra mass to account for expected additions, a useful simulation was created. Knowing that a very high sliding mass meant the robot would have an easier time rotating around its axles, in general, the first robot design was very tall. The guidelines laid out in Section 3.7 were followed to design the remainder of the sliding mass system. A spring stiffness of 25 N/m and a slider mass of 1.5 lb (0.68 kg) were chosen. Since wheeled robots are able to climb over steps that are less than the tire radius of the robot, a step size of 1.625" – slightly larger than the tire radius of 1.5" – was used for the simulations. This was done to determine whether this passive system can not only climb a step, but also expand the step climbing range of wheeled robots. The base robot (without the sliding mass attached) was tested on this step and was unable to pass over the step. Once an acceleration profile was created in a simulation that showed this robot could climb the step, the actual sliding mass platform that would be installed on the base robot could be built.

The end result of this design is shown in Figure 4.3 – a wood block was formed to mount the top of the RC car and an aluminum platform was created to fit on top. A linear bearing was attached to the top of the platform and several blocks of aluminum were combined to form the mass of the slider. A spring is connected from the slider to a fixed bracket and is supported with a wood dowel that passes through the bracket. The platform was designed to be very simple but adjustable for easy design changes. The height of the platform, position of the spring bracket, and mass of slider are readily adjusted by removing a few machine screws and making the necessary changes. To cut weight, patterns were milled in the aluminum beams to minimize weight while ensuring structural stability. Since this design was being used to simply validate the simulations, the great height of the system was not a concern.

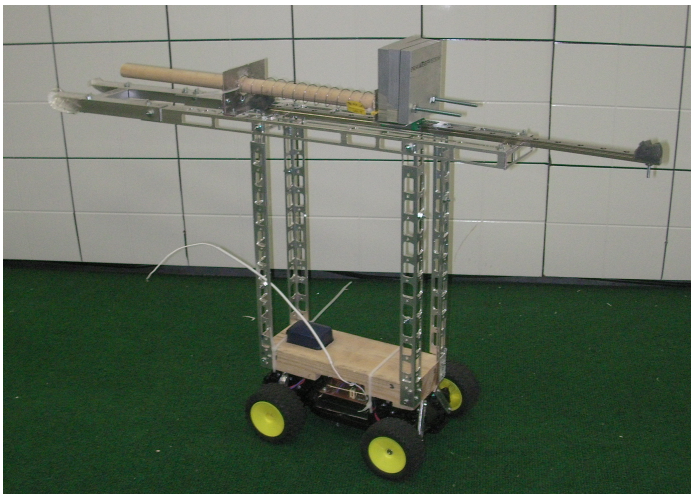

Figure 4.3: A robot equipped with a passive dynamic system to enable step-climbing. This robot was used to validate simulations and was designed to be excessively tall.

The design parameters of the actual robot that was created are shown in Table 4.1. Some values were modified based on what was available and some were modified based on testing and knowledge of the effects of changes. The spring stiffness was increased slightly, due to the available selection, and the mass of the slider increased as well. The vertical center of gravity of the base robot was raised significantly as a result of the aluminum near the top of the platform and the steel linear bar to guide the slider. None of these changes are overly significant because the original design was a crude attempt at modeling the system, so an exact match was not needed.

Table 4.1: Design parameters for the robot used to validate simulations

M	2.01 kg
a_x	8.25 cm
a_y	19.1 cm
L	17.1 cm
h	48.3 cm
m	0.77 kg
b_{xo}	17.8 cm
b_y	53.3 cm
k	26 N/m

Since the robot was being controlled through a radio controller by hand, numerous tests were required to achieve step-climbing, though the process did eventually work. The robot was able to lift its front wheels over the step and decelerate enough to pop the rear wheels above the step. Since acceleration is limited at higher velocities, the robot could not begin at a large enough initial velocity for the momentum to carry the robot over the step while in the air. This

limitation would likely not exist in a real robot that uses motors stronger than the default options for this RC car. Selections from a video that was taken of one successful test are shown in Figure 4.4. The process here follows the expectations outlined in Figure 3.1. A hand was kept near the robot at all times because of the difficulty controlling the system, but the actual contact was kept to a minimum. Frame (e) shows that the robot pulls its rear wheels high enough to get over the step, but Frame (f) indicates that the robot didn't continue forward quite enough to actually climb the step.

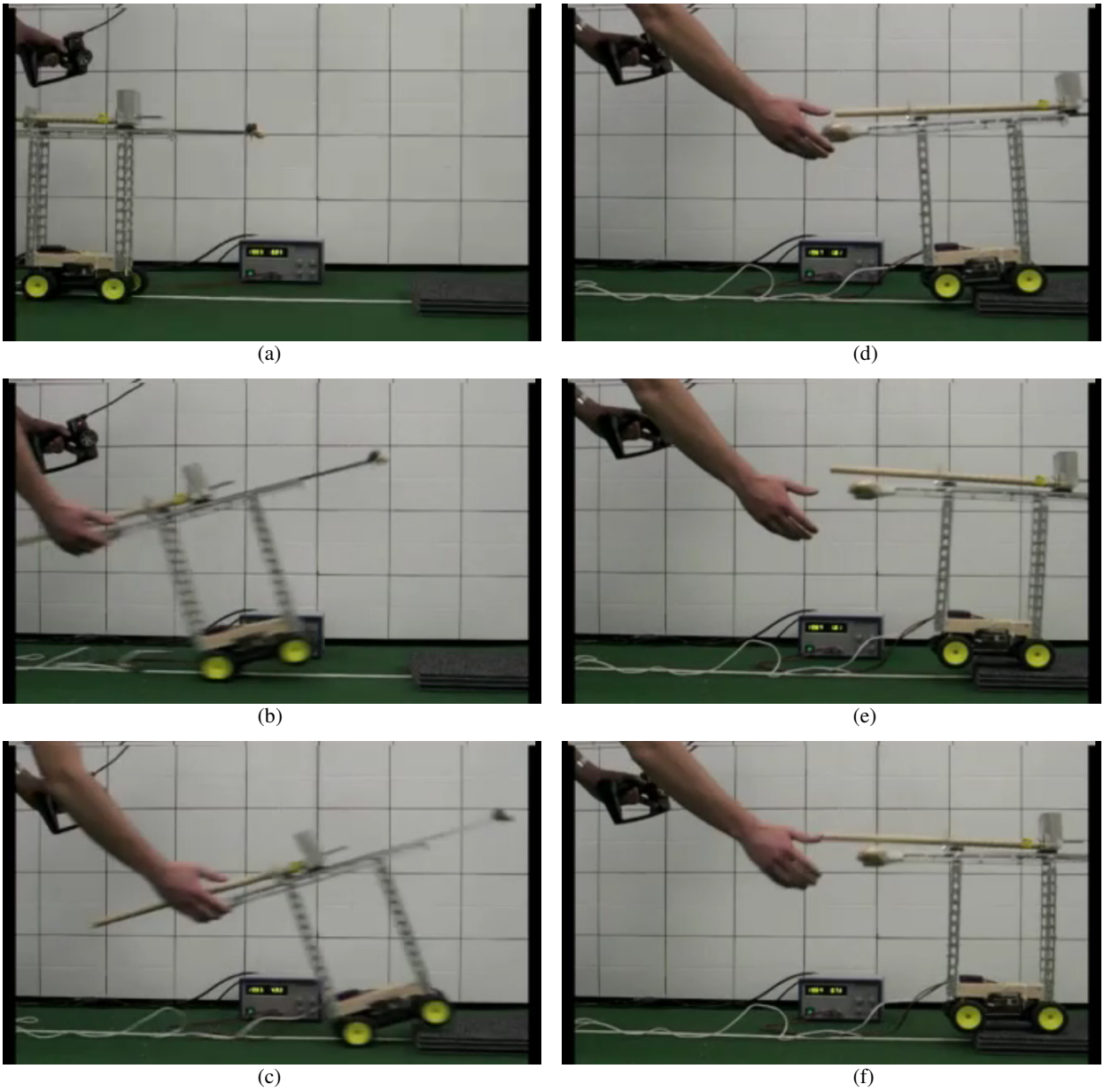

Figure 4.4: Selections from video of the step-climbing process for an initial test of the inertially actuated system.

With the acceleration data and actual structure of the robot from this test, these parameters were then reused in the C++ simulation to see if the profile of the robot could be matched using a similar acceleration input. Figure 4.5 shows the acceleration profiles for the actual test and the simulation. The maroon plot is the actual acceleration data taken from experimentation. The orange plot is the highly-simplified acceleration model that was put into the C++ simulation. This acceleration profile was created by keeping the geometrical parameters of the robot constant and varying the acceleration in order to best match the rotation data taken from the test.

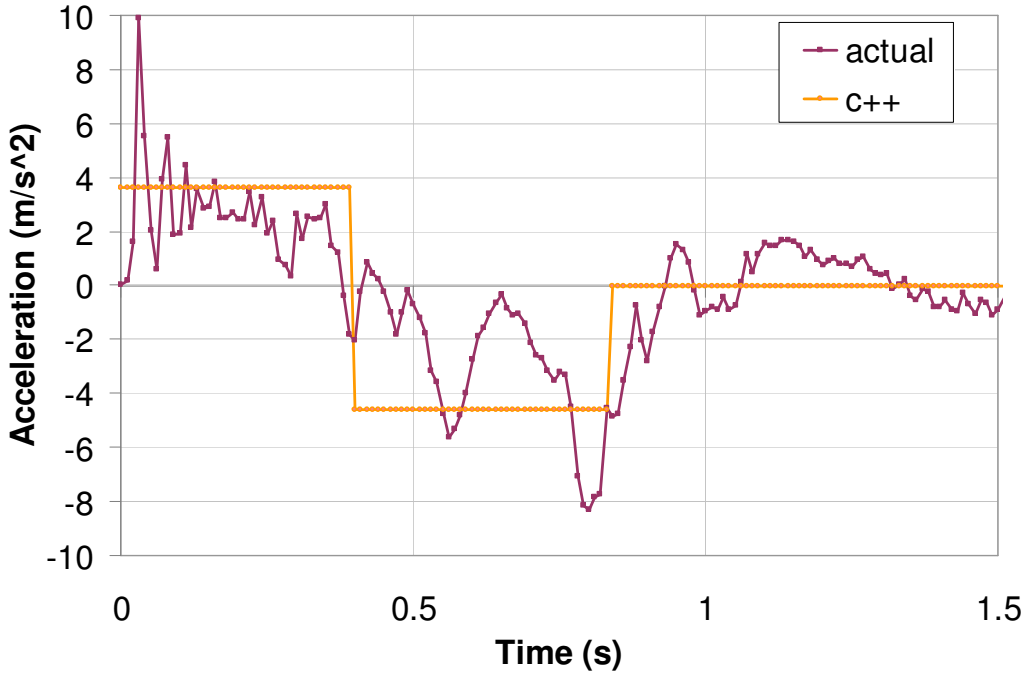

Figure 4.5: A plot of acceleration versus time for an actual test and a simulation.

There are two reasons for not using the actual acceleration profile taken from the test and using this as the input to the dynamic simulation. First, looking at Figure 4.5 shows the acceleration data is very noisy and doesn't seem to agree with the input method. To control this robot, full power was given to accelerate the robot until the robot was near the step. Then, the robot was immediately put in reverse at full power. This input should show a relatively constant acceleration (which is evident) followed by a relatively constant deceleration (which is not the case). In addition, the large oscillations and spikes in the acceleration data raise questions about its accuracy. Also, and for a worse reason, the acceleration data was not used as it was because simulations showed that this acceleration was not capable of sufficiently rotating the robot to

perform step-climbing. Since the actual acceleration data may need calibration and simulations indicate that the acceleration magnitude is inadequate, a similar acceleration profile was created that used reasonably similar values to the actual acceleration data.

Based on the acceleration tests that were performed on the base robot, the constant values of acceleration and deceleration used as the simplified input are within the ranges expected during the process. The actual acceleration profile, shown in Figure 4.5, also shows values within this range, though oscillations are significant in this data. These two factors mean that the magnitude of acceleration for the simplified model is reasonable. Figure 4.5 does indicate a point where deceleration starts for the experiment – the accelerations are largely negative after 0.4 seconds. This point is nearly exactly the same point as where deceleration starts in the simplified acceleration model, so the timing between the two models is similar.

The rotation profile of the robot from the actual test and from the simulation using the simplified acceleration input is shown in Figure 4.6. The actual rotation profile is again shown in the maroon plot and the C++ simulation data is the orange trend. The two plots match very well and share similar characteristics. A brief period of rotation lasts around 0.3 seconds, which is mostly due to tire deflection as the sliding mass pushes back. Once the slider has moved far enough, the rotations speed up and the two plots show similar maximum rotation values – 21^{o} for the simulation and 26^{o} for the actual test. After decelerating and landing the front wheels on the step, the front tire deflections and the rear tires pick up slightly from the angular momentum of the robot. When the angular momentum decreases and the slider hasn't slid forward enough to help pop the rear wheels up, the rotation of the robot levels off slightly and the robot may start to fall back down. This pause occurs at a similar time for each case, though at 2.5^{o} for the simulation and 5^{o} for the actual test. Once the slider has slid forward enough, the robot's rear wheels are pulled up over the step and the robot has successfully climbed a step. The two cases diverge somewhat during this process, but the overall goal of climbing a step is accomplished in both cases.

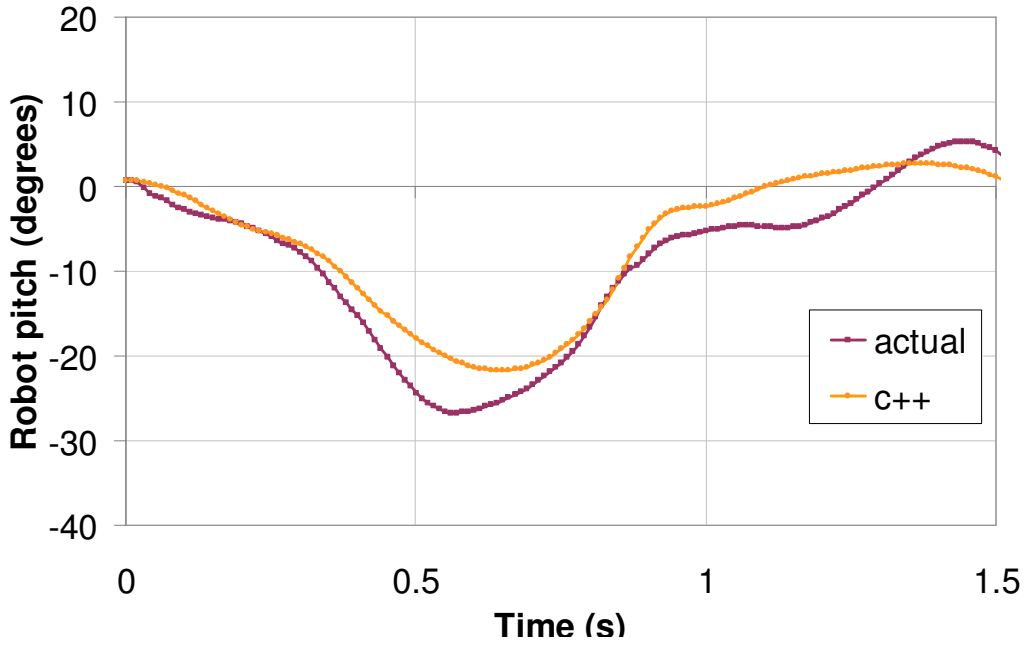

Figure 4.6: A plot of rotation versus time for the acceleration data in Figure 4.5.

Since the acceleration input to the C++ simulation was modified in order to match these rotation profiles, take caution to not take this plot to mean the simulations are accurate. The acceleration profile of the actual test is deemed to not be fully accurate, so the sole purpose of this test is to confirm that a reasonable acceleration input to the simulation could create a rotation profile similar to that of an actual experiment with a similar acceleration. The simulation is not expected to be an exact replica of the actual dynamics of a wheeled robot, so the errors and assumptions here are deemed to be acceptable since similar inputs to each system did result in a wheeled robot climbing a step larger than the radius of the tires in each case.

4.3 Final design and testing: Now that a successful test has been recreated in the dynamic simulator, the simulation can be used as intended – as a design tool. The step height was kept at 1.625" since this was a step that was impassable by the wheeled robot before the sliding mass was attached. Looking at the robot at hand and the platform that is attached, lowering the sliding mass from a height of 21" above the axles to a height of 7" was considered ideal. This leaves a couple inches between the platform and the main robot, which will allow access to the accelerometer and all connections, but also reduces the required height so an awkwardly tall robot is not created. After reducing the height of the model in the simulation to the desired height, only one relevant design change was needed. The equilibrium position of the

sliding mass was moved back towards the rear axle one inch to assist in lift-off. This shorter, slightly optimized robot is shown in Figure 4.7.

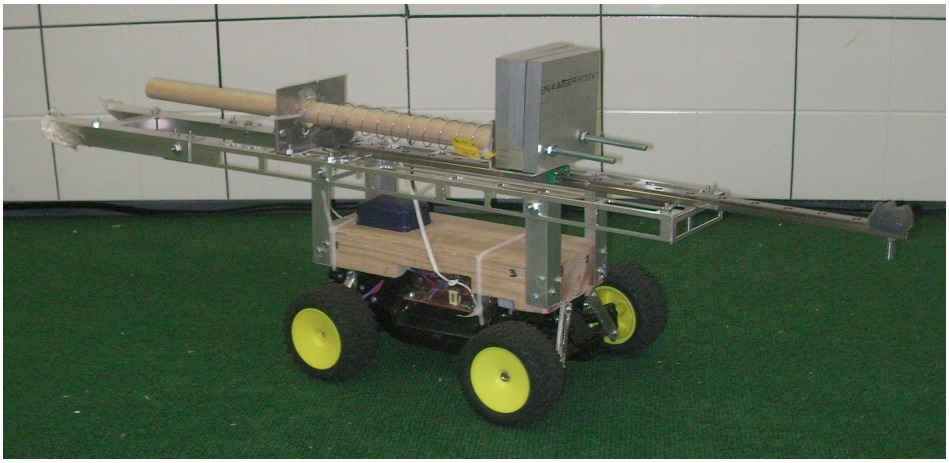

Figure 4.7: A robot designed using simulations to reduce the height of the slider for step-climbing.

Using the same size step (1 5/8”), the acceleration model required to enable step-climbing involved a longer period of acceleration but a similar deceleration length. The actual magnitudes of the accelerations are similar to those from testing of the preliminary robot: the new acceleration model for the shorter robot is shown in Figure 4.8. The orange plot shows this profile and was constructed based solely on feedback from the simulation. The maroon plot in Figure 4.8 is the acceleration model of an actual step-climbing test using the new parameters of the robot. This test was again done with full power at acceleration and deceleration; however, during tests with the shorter robot, either electrical interference or unknown factors caused the robot to not decelerate when the signal was given. This was confirmed by monitoring the power supply during a test and noting that the current draw while deceleration was expected was near zero. The robot would simply coast and not actively slow down. As a result, the robot would decelerate slightly and then bounce off the step, which caused a rapid deceleration by the robot. This explains a couple of the spikes in deceleration in the data shown, though some spikes are unaccounted for. This again indicates the acceleration data provided is not entirely reliable.

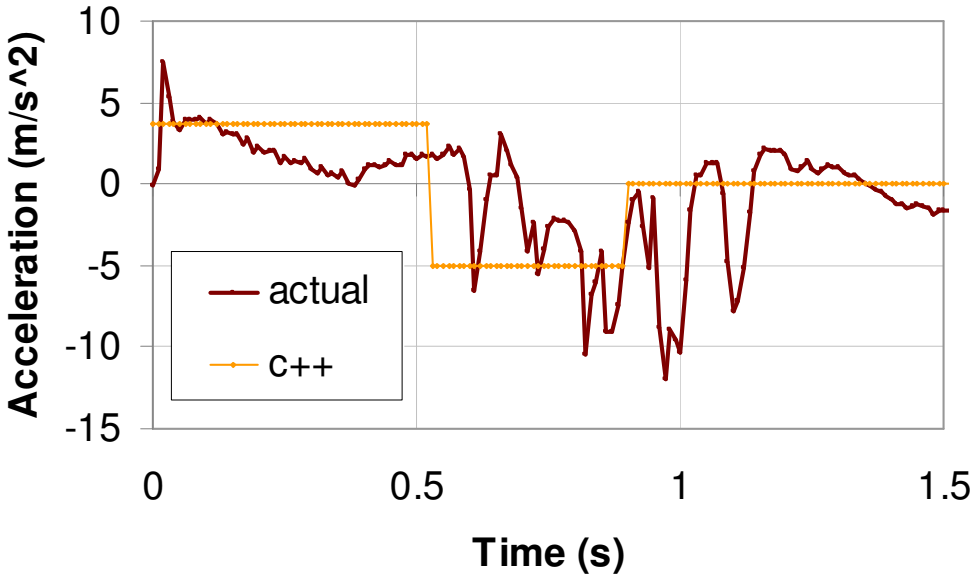

Figure 4.8: The acceleration input for a simulated trial and an actual trial based on that simulation.

Despite the concerns about accuracy, evident in Figure 4.8 is the fact that the magnitude of the accelerations between the two cases are in similar ranges. The acceleration of the actual test drops off, but the magnitude of the simulation input is agreeable to the initial acceleration of the robot and also to the range of expected accelerations based on prior experimentation. Since braking was minimal and most deceleration was done by step contact, a comparison of the deceleration values is difficult. Although the data is very scattered, the average deceleration during the process is similar between cases. Also apparent is that the switch from acceleration to braking occurred sooner for the simulation, indicating that the simulation will likely result in a lower maximum rotation.

The resulting rotation of the robot for each input is shown in Figure 4.9, where the maroon plot applies to the actual test and the orange line signifies the results from simulations. A spike in data occurs in the actual test around 0.6 seconds that is a result of a roll-bar contacting the ground. This bar prevented the robot from flipping over too far and damaging the sliding mass system. This bar was not incorporated into the simulations and is not evident in that data. While these rotation profiles do not match, two things are important to note. First, the dynamics occur in roughly the same timeframe and magnitudes. Second, both tests share a key similarity –

both models resulted in a robot successfully climbing a step, enabled by an inertially actuated system.

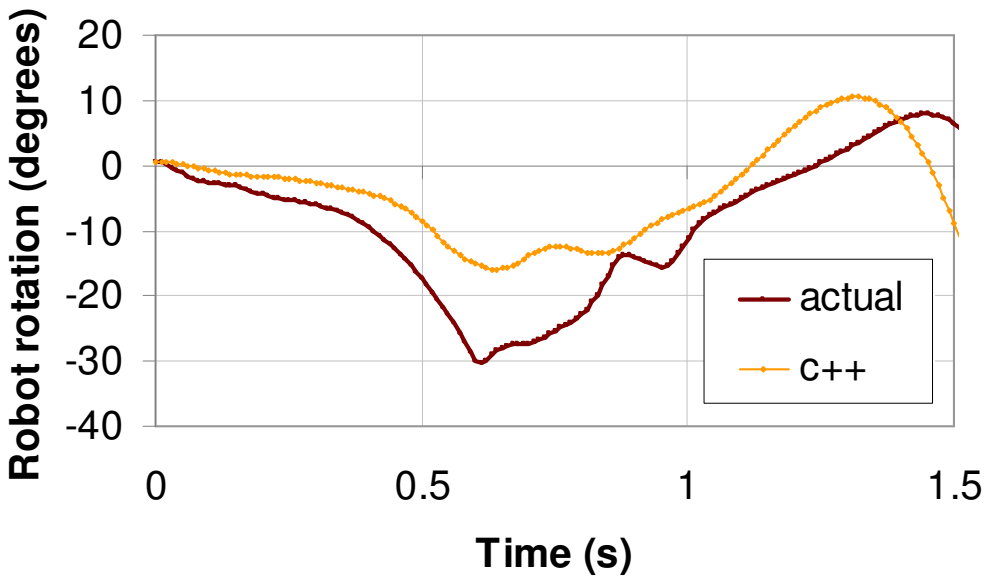

Figure 4.9: The resulting rotation of the robot from the inputs shown in Figure 4.8.

Although the rotation profiles of the simulation and an experiment do not match perfectly, the end goal of climbing a step was succeeded. Several factors could have led to the difference in the rotations – the most significant being that there was no attempt to match the acceleration profiles or the rotation profiles. The simulation served its purpose to help to design a sliding mass system that could enable step-climbing by wheeled robots. Experimentation of the robot that was designed has shown that the system can effectively pass a wheeled robot over a step. Selected images from video of this shorter robot as it climbs the step are included in Figure 4.10. This robot was able to successfully climb the step and allow its momentum to carry its rear wheels over the step.

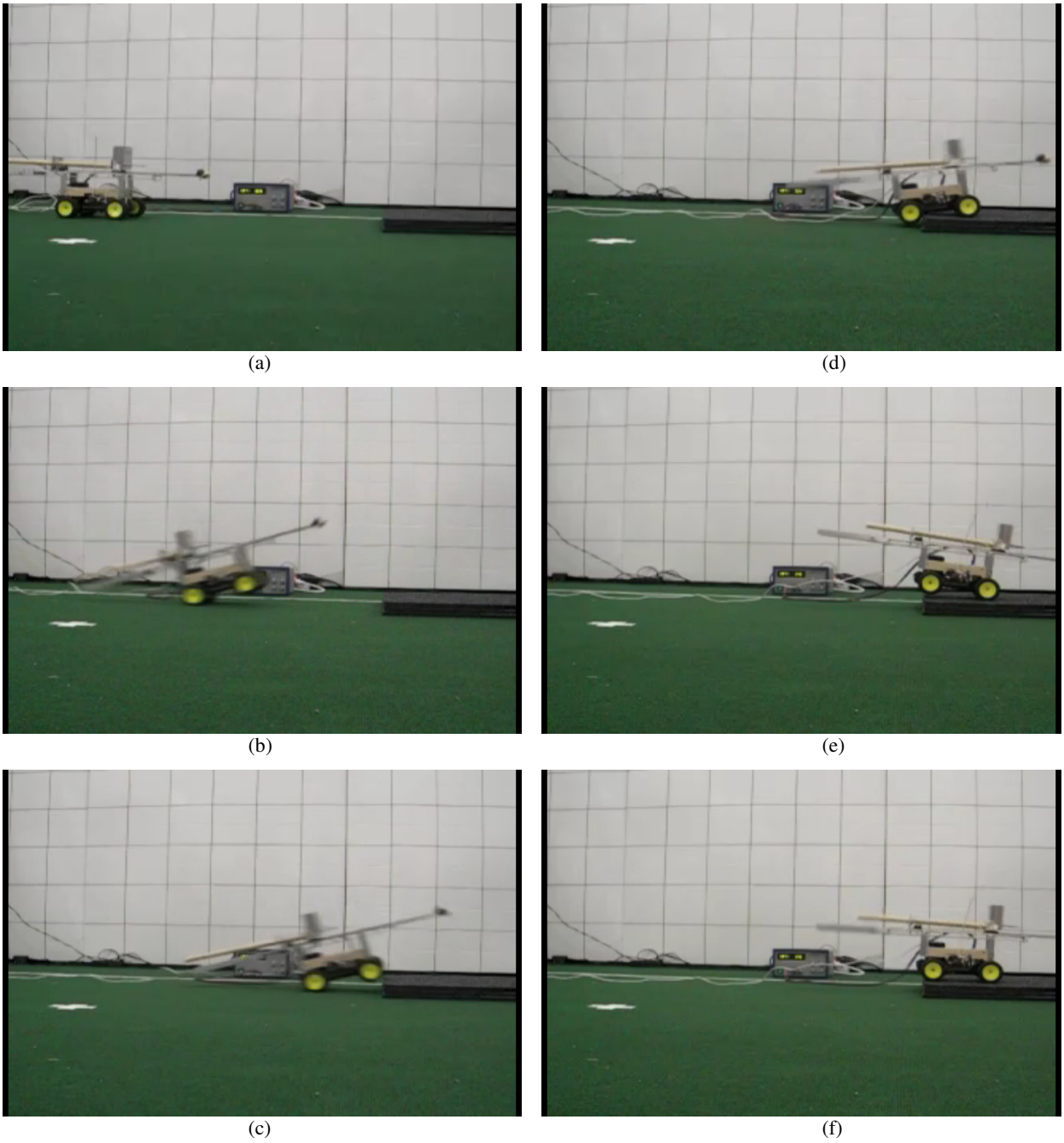

Figure 4.10: Selected images from video as the optimized, shorter robot climbs a step

4.4 Results and Discussion: Although the ultimate goal of this research project was to create a passive system that would allow wheeled robots to climb steps larger than the tire radius, equally important was gaining a full understanding of and insight to the dynamics of this process. To verify that the interpretation of the system was applicable to an actual robot, simulations of a robot were compared in two ways to an experiment to test a robot's ability to climb a step.

The first use of the simulation involved using a robot (designed through a crude use of the simulation) that was tested and shown to be capable of climbing a step and matching the rotation profile between the two tests. By adjusting the acceleration input to the simulation, the simulation was able to nearly match the rotation profile from the experiment. The purpose of this is to show that with a reasonably similar acceleration profile, the rotation profile of an actual robot can be matched using the derivations of the system dynamics.

The simulation was also used as a design tool – which is the true intent of having such a simulation. By slightly optimizing the robot from the first experiment using the simulation, a robot was created that was significantly shorter and was still capable of climbing a large step. The acceleration and rotation profiles for this situation were compared and did not match perfectly, but there was no attempt to improve the correlation between the tests. Since the simulation showed how the robot could be improved for step-climbing, the experiments were successful.

Chapter 5: Conclusions

This thesis has presented the work done to develop a passive dynamic system using a sliding mass to enable a wheeled robot to climb a step that is larger than the radius of the robot's tires. The following sections will summarize the results of those efforts and suggest improvements and possible future work if this project is continued.

5.1 Analysis of results: In order to expand the mobility of wheeled robots, we have introduced a novel approach to step-climbing that involves an inertially actuated sliding mass. The acceleration of the wheeled robot produces inertial forces on the sliding mass and can allow the robot to "hop" over a step. Wheeled robots are able to roll over steps smaller than the radius of the robot's tires, but this technique was done in an effort to allow a wheeled robot to climb steps larger than this limit.

After deriving the equations of motion for a sliding mass on a wheeled robot, the effects of changing parameters of the sliding mass were revealed. In general, increasing the height of the slider and decreasing the spring stiffness were found to increase a robot's ability to climb a step. There are limits to how far each of these parameters can be changed, but knowledge of these limits and moderate changes can assist in step climbing. Depending on what a robot is able to accomplish during the process, increasing or decreasing the mass of the slider can be beneficial, though the effects of this adjustment must be understood before significant changes are made. Shifting the equilibrium position of the slider forward or backward will help during one phase of step-climbing but hurt the other, so this should be done cautiously. A more in-depth discussion of how these factors affect a wheeled robot as it climbs a step using passive dynamics has been provided.

To verify that the dynamics have been fully understood, a simulation of the system was created and compared to a program that specializes in simulating two-dimensional dynamic system. In the process, a fatal flaw was discovered in the software program, Working Model. The moment of inertia of the robot was being calculated incorrectly in the software and this was confirmed by adding this error into the simulation for this research project and seeing that the resulting simulations matched nearly perfectly. This not only proved the error in Working Model but also verified that the derived dynamic model for this system was found correctly.

Use of the simulations and comparing the results to data from experiments showed that the simulations were useful, but not perfect models of a wheeled robot. A simulation was modified to match the rotation profile of a real test using a similar acceleration profile to show that the dynamics between the two cases are similar. By using the simulation as a design tool, that robot was then optimized with a lower sliding mass and was still shown to climb a large step in the simulation and in the actual test. While there are errors in this test when matching the actual driving pattern, the design tool was used and proved effective at being able to design an inertially actuated system for wheeled robots. A step size of 1.625", larger than the radius of the tires, was used to show that the climbing ability of a wheeled robot could be increased using this inertial actuation approach. Testing the robot before modifications were applied showed that the basic robot was unable to climb a step this tall without the inertially actuated sliding mass.

This research has proved that an inertially actuated sliding mass platform can be retrofit to a wheeled robot to enable step-climbing, but there are limitations to the extendibility of this system. The technique will work well for 4-wheeled robots but will likely be ineffective for other robots. A robot with fewer wheels will be unstable during the step-climbing process when only one wheel is on the ground. On the other hand, extra wheels on a robot may interfere with the step as the robot passes over it and resist step-climbing. The frame of the robot must not resist the robot from rotating on its axles or this step-climbing process is not possible. If the tires are recessed into the robot too far or guards are in place around the tires, the robot may drag on the ground and prevent rotation. Experimentation for this project involved a 2-wheel drive robot, although true step-climbing may require 4-wheel drive to allow the robot to continue forward when the rear wheels are in the air.

While this project has shown itself to be an interesting engineering problem, there are several limitations that prevent this system from being implemented as a step-climbing aide in the near future. As described in the next section, these limitations include that the process is not highly repeatable at this time and the dynamic model (especially the tire deflection model) may not fully encompass all dynamics of the system. With substantial improvement in the overall system, this inertially actuated technique to step-climbing has the potential to be very effective.

5.2 Future work: There were many lessons learned from this research project that should be accounted for in the future. Since the final experiments were technically successful but didn't

match the simulation perfectly, there is significant room for improvement in the derivation of equations as well as the experiments.

The dynamics of the system were modeled as best as possible, though a more detailed analysis could be done to improve the accuracy. While the derivations provided are believed to have been done properly, improvements and additions to the system could be made. An assumption that was made that is likely inappropriate is that the tires can be modeled as spring-damper systems. If a more accurate tire model is implemented, then the simulations could be more reliable. Other improvements to the system not listed could improve the completeness of the dynamic simulations.

Equally as beneficial as improving the dynamic model is removing sources of error in the experiments. The attached platform was not rigidly mounted – there was some flexure in the connection. While minimal, this may allow the top platform to rotate slightly more than the base robot and change the dynamics of the system. In addition, ensuring accurate acceleration data will help to validate the simulations that were created. Other aspects of the experiments, such as implementing a true braking system or upgrading to 4-wheeel control would help to complete the step-climbing process.

While these changes would help to make this research project more accurate, this project has shown there are many more research topics that relate to this passive approach to step climbing that could be investigated. One very helpful topic, which really should have been implemented for this project, is to develop a feedback controller to dynamically control the robot as it climbs a step. By eliminating the human involvement and adjusting appropriate gains, the step-climbing process could become highly repeatable and much more efficient. This project used a sliding mass on a linear bearing as the passive dynamic component, though more interesting systems could be analyzed. These alternatives include different travel profiles for the sliding mass (such as angles and curves) and different inertially actuated systems (ranging from a hanging pendulum to a mass fixed at the end of a cantilevered beam).

Appendix A: Justification for Design Simplifications

This appendix demonstrates that the simplifications made to the dynamic model derived in Chapter 2 for use in a design analysis are warranted. By removing the terms that were deemed to be "relatively small" with respect to other terms in the system equations – which included the rotational effects of the sliding mass and the wheel torque – simplified equations were created to facilitate designing the inertially actuated system. To ensure that these equations still capture the overall essence of the system, the simulations (as described in Chapter 3) were adjusted to model the step-climbing process while ignoring these smaller terms. The results of this change for one case are shown in Figure A-1; the blue plot shows the rotation of a robot as it climbs a step using the full dynamic model and the pink plot was created using the simplified design equations. The main difference here is created when the robot nears its maximum rotation, since the rotational inertia of the system is ignored. The robot rotates more before bringing the front wheels down and generates more energy – this then leads to a larger rotation as the rear wheels pass over the step. If this initial discrepancy did not occur, the two plots may agree more closely.

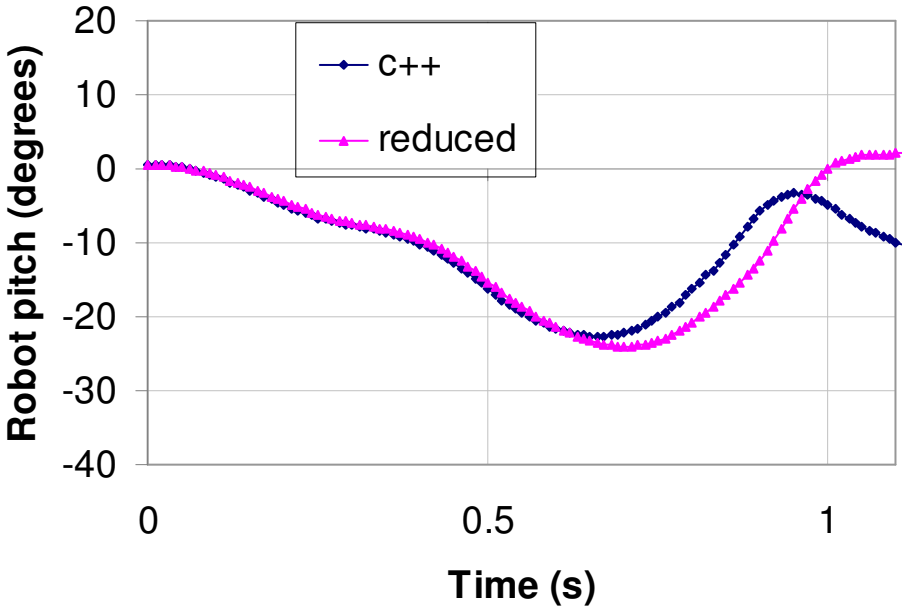

Figure A-1: A comparison of the simulated rotation profile between the full dynamic model and the simplified design equations

From this analysis, it is clear that simplifying the dynamic model of the system does change the resulting behavior. This indicates that the terms that are ignored for a design discussion are important and must be included in an accurate model. However, also evident is

that the wheeled robot follows a similar rotation trend using the simplified equations in this example. The full model shows that the robot will rotate less quickly than for the approximated system but the overall process is comparable for both scenarios. Since the simplified equations are being used to design the system and determine how changing design parameters of the sliding mass system will affect step-climbing, and the trend of the simplified equations is similar to that of the full dynamic model in this example, the design equations are assumed to be a valid and useful tool for investigating how to adjust design parameters of the system. The differences between the full model and the simplified model may change for different test cases, but this example is one where the reduced equations do not significantly affect the results. After using these simplified and less exact equations for design, the full dynamic model is then implemented for simulating the actual system.

Appendix B: C++ Simulation Code

Header file to store system inputs:

```
// main data container for simulation steps
struct data
{
	// current acceleration of wheel
	double a;
	// current velocity
	double v;
	// current displacement
	double x;
	// current time
	double time;
	// current rotation
	double theta;
	// current angular velocity
	double thetadot;
	// current angular acceleration
	double thetadotdot;
	// current bx value, distance from rear wheel to added weight,
horizontally
	double b_x;
	// current dx value, distance from front wheel to added weight,
horizontally
	double d_x;
	// current deflection in cart-local-horizontal direction of added
weight
	double deflect;
	// current local velocity of mass, used to correct deflection
	double mass_vel;
	// current local acceleration of mass, computed based on forces
	double mass_acc;
	// current by value, distance from rear wheel to added weight,
vertically
	double b_y;
	// current dy value, distance from front wheel to added weight,
vertically
	double d_y;
	// current value of L_b
	double L_b;
	// current value of L_d
	double L_d;
	// current value of J about rear wheel;
	double J1;
	// current reaction force on front wheel during lift-off actions
	double R2;
	// current value of J about front wheel
	double J2;
	// current reaction force on rear wheel during pop-up actions
	double R1;
	// vertical deflection of front tire, positive down
	double d_T2;
	// vertical deflection of rear tire, positive down
```

```
        double d_T1;
        // derivative of d_T1
        double d_T1_dot;
        // derivative of d_T1_dot;
        double d_T1_dotdot;
        // derivative of d_T2
        double d_T2_dot;
        // derivative of d_T2_dot;
        double d_T2_dotdot;
        // addition to theta based on tire deflection
        double theta_add;
};


// array to hold acceleration profile, will be interpolated linearly
// must start positive for lift-off, possibly go negative for pop-up, then go
to zero
double accel_list[] = {3.65, 3.65, -4.8, -4.8, 0};

// array to hold times for acceleration profile
double time_list[] = {0, 0.63, 0.631, 0.94, 0.941};

// number of items in acceleration profile list
double accel_num = 5;

// Mass of cart
double M = 4*0.4536;          // lb to kg
// Mass of added weight
double m = 1.69*0.4536;       // lb to kg
// Moment of inertia of base robot about rear wheel, calculated
double I_M1 = calc_I_M1_by7();            // kg m^2
// Moment of inertia of base robot about rear wheel, calculated
double I_M2 = calc_I_M2_by7();            // kg m^2
// Moment of inertia of mass slider about its centroid
double I_m = 0.000777;        // kg m^2
// spring constant
double K = 25;                // N/m
// spring damping
double B = 1;
// spring constant of tires
double K_tire = 2500;   // N/m
// damping of tires
double B_tire = 50;     // Ns/m
// coefficient of friction
double mu = 0.005;
// Axle-axle distance
double L = 6.75*0.0254;       // in to m
// Initial horizontal distance of added weight to rear wheel
double b_x0 = 6*0.0254;       // in to m
// Initial horizontal distance of added weight to front wheel
double d_x0 = L-b_x0;         // in to m
// Initial vertical distance of added weight to rear wheel
double b_y0 = 7*0.0254; // in to m
// Axle to top-of-cart
double h = 5*0.0254;          // in to m
// Initial vertical distance of added weight to front wheel
double d_y0 = b_y0;                 // in to m
```

```
// Horizontal distance of cart centroid to rear wheel
double a_x = 3.25*0.0254;           // in to m
// Horizontal distance of cart centroid to front wheel
double c_x = L-a_x;           // in to m
// Vertical distance of cart centroid to rear wheel
double a_y = 3*0.0254;        // in to m
// Vertical distance of cart centroid to front wheel
double c_y = a_y;       // in to m

// tire radius
double tire_rad = 1.5*0.0254;
// height of step
//double step_height = 2.2*0.0254;        // in to m
// x width of full simulation window
double track_x = 150*0.0254;        // in to m
// y height of full simulation window
double track_y = 25*0.0254;         // in to m
// x location of step on track
double track_step_x = 1.356;        // m
// angle to determine step-height
double track_pop_theta = 0.243;     // rad
// offset of track viewer
double track_x_offset = 15*0.0254;  // in to m
// offset of track viewer
double track_y_offset = 15*0.0254;  // in to m
// x width of close-up simulation window
double hold_x = 45*0.0254;                  // in to m
// y height of close-up simulation window
double hold_y = 40*0.0254;                  // in to m
// offset of close-up viewer
double hold_x_offset = 0.25;                // m
// offset of close-up viewer
double hold_y_offset = 0.1;         // m
#endif



// distance from rear wheel to cart centroid
double L_a = sqrt (a_x*a_x + a_y*a_y);
// angle from horizontal to cart centroid, radians
double phi_a = atan (a_y/a_x);
// distance from front wheel to cart centroid
double L_c = sqrt (c_x*c_x + c_y*c_y);
// angle from horizontal to cart centroid, radians
double phi_c = atan (c_y/c_x);
// distance from rear wheel to added mass
double L_b0 = sqrt (b_x0*b_x0 + b_y0*b_y0);
// angle from horizontal to added mass, radians
double phi_b0 = atan (b_y0/b_x0);
// distance from front wheel to added mass
double L_d0 = sqrt (d_x0*d_x0 + d_y0*d_y0);
// angle from horizontal to added mass, radians
double phi_d0 = atan (d_y0/d_x0);
// initial angle for pop-up
double pop_theta;
// local x-coordinate for rear wheel
double tire1_cen_x = 0.0;
```

```
// local y-coordinate for rear wheel
double tire1_cen_y = 0.0;
// local x-coordinate for front wheel
double tire2_cen_x = tire1_cen_x+L;
// local y-coordinate for front wheel
double tire2_cen_y = 0;
// used for making a tire circle
double half_ang = tire_rad*sqrt (2.0)/2;
```

Main program file:

```
/* Robotic step-climbing simulation
      John Humphreys
      RoMeLa, Virginia Tech*/

#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdlib>
#include <string>
#include <sstream>
#include <GL/glut.h>
#include <cmath>
#include <conio.h>
using namespace std;
#define M_PI 3.14159265358979323
#define g 9.80665
#define spring

#include "cartdata.h"

// time step for interation
const double time_step = 0.001;                  // s
// length of simulation
const double duration = 48;                // s
// number of time steps, duration/time_step + 1
const int steps = 48001;

// true once acceleration is negative to cause pop_up
bool pop_up = false;
// used to create strings
ostringstream ss;
// pointers for GL windows
int window1, window2;

// array of data to fill during simulation
data data_list [steps];
// counter used to track number of steps, must not exceed "steps" variable =
max steps
int t = 0;


// function to advance one time step, compute new parameters, and update
graphics
ofstream out ("log.txt");
void tick()
{
      // verify that desired "steps" value hasn't been exceeded
      if (t < steps-1)
      {
            t++;
            // Pull data from list
            data thisdata = data_list[t];
            data lastdata = data_list[t-1];
            // add time step
```

```
            thisdata.time = lastdata.time + time_step;

            // sign of mass local velocity, used for friction direction
            int sign_mv = (lastdata.mass_vel>0 ? 1 : -1);
            if (fabs(lastdata.mass_vel) < 0.001)
                  sign_mv = 0;
            // Compute deflection of added mass based on acceleration and
gravity
            if (!pop_up)
            {
                  thisdata.mass_acc = (1/m)*(m*g*sin(lastdata.theta) +
m*lastdata.a*cos(lastdata.theta)
                                                - K*lastdata.deflect -
B*lastdata.mass_vel
                                                -
mu*sign_mv*(m*g*cos(lastdata.theta) - m*lastdata.a*sin(lastdata.theta))
                                                -
m*lastdata.thetadotdot*(lastdata.b_y + lastdata.b_x*mu*sign_mv)
                                                + m*pow(lastdata.thetadot,
2.0)*(lastdata.b_y*mu*sign_mv - lastdata.b_x));
            }
            else
            {
                  thisdata.mass_acc = (1/m)*(m*g*sin(lastdata.theta) +
m*lastdata.a*cos(lastdata.theta)
                                                - K*lastdata.deflect -
B*lastdata.mass_vel
                                                -
mu*sign_mv*(m*g*cos(lastdata.theta) - m*lastdata.a*sin(lastdata.theta))
                                                -
m*lastdata.thetadotdot*(lastdata.d_y - lastdata.d_x*mu*sign_mv)
                                                + m*pow(lastdata.thetadot,
2.0)*(lastdata.d_y*mu*sign_mv + lastdata.d_x));
            }
            thisdata.mass_vel = lastdata.mass_vel +
thisdata.mass_acc*time_step;
            thisdata.deflect = lastdata.deflect +
(thisdata.mass_vel)*time_step;

            // Set mass position
            thisdata.b_x = b_x0 - thisdata.deflect;
            //Use the next two lines for rigid mass
            //thisdata.b_x = 6.5*0.0254;
            //thisdata.deflect = m*g*sin(lastdata.theta)/K +
m*lastdata.a*cos(lastdata.theta)/K;

            thisdata.d_x = d_x0 + thisdata.deflect;
            thisdata.b_y = b_y0;
            thisdata.d_y = d_y0;

            // search acceleration profile for current acceleration of rear
wheel
            for (int i = accel_num-1; i >= 0; i --)
            {
                  if (thisdata.time >= time_list[i])
                  {
                        if (i == accel_num-1)
```

```
                              thisdata.a = accel_list[i];
                        else
                              thisdata.a = (accel_list[i+1]-
accel_list[i])*(thisdata.time-time_list[i])/(time_list[i+1]-
time_list[i])+accel_list[i];
                        i = -1;

                        // Start pop up action, changes dynamics
                        if (!pop_up && fabs(lastdata.theta-
track_pop_theta)<0.01 && lastdata.thetadot <= 0.001)
                        {
                              pop_up = true;
                              pop_theta = lastdata.theta;
                              cout << "step at " << lastdata.x << " m" <<
endl;
                              cout << "pop_theta = " << pop_theta << " rad"
<< endl;
                        }
                  }
            }

            thisdata.L_b = sqrt(thisdata.b_x*thisdata.b_x +
thisdata.b_y*thisdata.b_y);
            thisdata.L_d = sqrt(thisdata.d_x*thisdata.d_x +
thisdata.d_y*thisdata.d_y);
            if (I_M1 < 0 && I_m < 0)
            {
                  thisdata.J1 = (M*L_a*L_a + m*thisdata.L_b*thisdata.L_b)*1 +
M*(L*L+h*h)/12;
                  thisdata.J2 = (M*L_c*L_c + m*thisdata.L_d*thisdata.L_d)*1 +
M*(L*L+h*h)/12;
            }
            else
            {
                  thisdata.J1 = (I_M1 + I_m + m*thisdata.L_b*thisdata.L_b);
                  thisdata.J2 = (I_M2 + I_m + m*thisdata.L_d*thisdata.L_d);
            }

            // Integrate position of rear wheel
            thisdata.v =                              // integrated from a
from last timestep
                  lastdata.v + lastdata.a*time_step;
            thisdata.x =                              // integrated from v
from last timestep
                  lastdata.x + lastdata.v*time_step;

            // Compute rotation dynamics
            if (!pop_up)
            {
                  thisdata.thetadot =                       // defined from
thetadotdot from last timestep
                        lastdata.thetadot + lastdata.thetadotdot*time_step;
                  thisdata.theta =                    // defined from
thetadot from current timestep
                        lastdata.theta + thisdata.thetadot*time_step;

                  // normal force from mass slider, perpindicular to cart top
```

```
                  double normal_force =
                        m*g*cos(thisdata.theta)
                        + m*lastdata.thetadotdot*thisdata.b_x
                        - m*pow(thisdata.thetadot, 2.0)*thisdata.b_y
                        - m*thisdata.a*sin(thisdata.theta);

                  // torque from wheel acceleration
                  double wheel_torque =
                        tire_rad*(M*thisdata.a + m*thisdata.a);

                  // If cart is rotated into ground, deflect front tire
                  if (thisdata.theta < asin(lastdata.d_T1/L))
                        thisdata.d_T2 = -L*sin(thisdata.theta) +
lastdata.d_T1;
                  thisdata.R2 = thisdata.d_T2*K_tire;

                  // value of R1 if tire deflection of rear wheel was
balanced
                  thisdata.R1 = K*thisdata.deflect*sin(thisdata.theta)
                                    + B*thisdata.mass_vel*sin(thisdata.theta)
                                    +
M*lastdata.thetadotdot*(a_x*cos(thisdata.theta) - a_y*sin(thisdata.theta))
                                    - M*pow(thisdata.thetadot,
2.0)*(a_x*sin(thisdata.theta) + a_y*cos(thisdata.theta))
                                    + M*g + normal_force*cos(thisdata.theta)
                                    +
mu*normal_force*sign_mv*sin(thisdata.theta)
                                    - thisdata.R2;
                  if (thisdata.R1 < 0)
                        thisdata.R1 = 0;
                  // Compress rear wheel based on vertical forces
                  thisdata.d_T1_dotdot = (1/(m+M))*(thisdata.R1-
lastdata.d_T1*K_tire-lastdata.d_T1_dot*B_tire);
                  thisdata.d_T1_dot = lastdata.d_T1_dot +
time_step*thisdata.d_T1_dotdot;
                  thisdata.d_T1 = lastdata.d_T1 +
time_step*thisdata.d_T1_dot;
                  //Use the next 6 lines to eliminate tire deflection
                  /*lastdata.d_T1 = 0;
                  lastdata.d_T2 = 0;
                  thisdata.d_T1 = 0;
                  thisdata.d_T2 = 0;
                  if (thisdata.theta < 0)
                  {     thisdata.theta = 0;
                        thisdata.thetadot = 0;
                  }*/

                  // update rotation based on deflection of rear wheel
                  thisdata.theta += asin(thisdata.d_T1/L) -
asin(lastdata.d_T1/L);

                  thisdata.thetadotdot =
                        (1/thisdata.J1)*
                        ((K*thisdata.deflect +
B*thisdata.mass_vel)*thisdata.b_y
                        - m*lastdata.thetadotdot*pow(L_a, 2.0)
```

```
                        + M*thisdata.a*(a_x*sin(thisdata.theta) +
a_y*cos(thisdata.theta))
                        - M*g*(-a_y*sin(thisdata.theta) +
a_x*cos(thisdata.theta))
                        + (mu*h*sign_mv - thisdata.b_x)*(normal_force)
                        + thisdata.R2*L + (thisdata.d_T2 -
lastdata.d_T2)*B_tire*L/time_step
                        + wheel_torque);
                  if ((t-1) % 10 == 0)
                        out << t/1000.0 << "\t" << thisdata.a << "\t" << -
thisdata.theta*180/M_PI
                        << "\t" << thisdata.d_T1 << "\t"
                        << thisdata.d_T2 << endl;

                  if (thisdata.theta > M_PI/2)
                  {
                        thisdata.theta = M_PI/2;
                        thisdata.thetadot = 0;
                        thisdata.thetadotdot = 0;
                  }
            }
            else
            {
                  // modified equations for pop up
                  thisdata.thetadot =                        // defined from
thetadotdot from current timestep
                        lastdata.thetadot + lastdata.thetadotdot*time_step;
                  thisdata.theta =                     // defined from
thetadot from current timestep
                        lastdata.theta + thisdata.thetadot*time_step;

                  double normal_force =
                        m*g*cos(thisdata.theta)
                        - m*lastdata.thetadotdot*thisdata.d_x
                        - m*pow(thisdata.thetadot, 2.0)*thisdata.d_y
                        - m*thisdata.a*sin(thisdata.theta);

                  double wheel_torque =
                        tire_rad*(M*thisdata.a + m*thisdata.a);

                  // If cart is rotated into ground, deflect rear tire
                  if (thisdata.theta > pop_theta - asin(lastdata.d_T2/L))
                        thisdata.d_T1 = L*sin(thisdata.theta - pop_theta) +
lastdata.d_T2;
                  thisdata.R1 = thisdata.d_T1*K_tire;

                  // value of R2 if tire deflection of front wheel was
balanced
                  thisdata.R2 = K*thisdata.deflect*sin(thisdata.theta)
                                    + B*thisdata.mass_vel*sin(thisdata.theta)
                                    -
M*lastdata.thetadotdot*(c_x*cos(thisdata.theta) - c_y*sin(thisdata.theta))
                                    + M*pow(thisdata.thetadot,
2.0)*(c_x*sin(thisdata.theta) - c_y*cos(thisdata.theta))
                                    + M*g + normal_force*cos(thisdata.theta)
                                    +
mu*normal_force*sign_mv*sin(thisdata.theta)
```

```
                                    - thisdata.R1;

                  if (thisdata.R2 < 0)
                        thisdata.R2 = 0;
                  // Compress rear wheel based on vertical forces
                  thisdata.d_T2_dotdot = (1/(m+M))*(thisdata.R2-
lastdata.d_T2*K_tire-lastdata.d_T2_dot*B_tire);
                  thisdata.d_T2_dot = lastdata.d_T2_dot +
time_step*thisdata.d_T2_dotdot;
                  thisdata.d_T2 = lastdata.d_T2 +
time_step*thisdata.d_T2_dot;
                  if (thisdata.d_T2 < 0)
                        thisdata.d_T2 = 0;
                  //Use the next 6 lines to eliminate tire deflection
                  /*lastdata.d_T1 = 0;
                  lastdata.d_T2 = 0;
                  thisdata.d_T1 = 0;
                  thisdata.d_T2 = 0;
                  if (thisdata.theta > pop_theta)
                  {     thisdata.theta = pop_theta;
                        thisdata.thetadot = 0;
                  }*/

                  // update rotation based on deflection of front wheel
                  thisdata.theta += -asin(thisdata.d_T2/L) +
asin(lastdata.d_T2/L);

                  thisdata.thetadotdot =
                        (1/thisdata.J2)*
                        ((K*thisdata.deflect +
B*thisdata.mass_vel)*thisdata.d_y
                        - M*lastdata.thetadotdot*pow(L_c, 2.0)
                        - M*thisdata.a*(c_x*sin(thisdata.theta) -
c_y*cos(thisdata.theta))
                        + M*g*(c_x*cos(thisdata.theta) +
c_y*sin(thisdata.theta))
                        + (mu*h*sign_mv + thisdata.d_x)*(normal_force)
                        - thisdata.R1*L - (thisdata.d_T1 -
lastdata.d_T1)*B_tire*L/time_step
                        + wheel_torque);

                  if ((t-1) % 10 == 0)
                        out << t/1000.0 << "\t" << thisdata.a << "\t" << -
thisdata.theta*180/M_PI
                        << "\t" << thisdata.d_T1 << "\t"
                        << thisdata.d_T2 << endl;

                  if (thisdata.theta < -M_PI/2)
                  {
                        thisdata.theta = -M_PI/2;
                        thisdata.thetadot = 0;
                        thisdata.thetadotdot = 0;
                  }
            }

            // Update data_list
            data_list[t] = thisdata;
```

```
            // Set modulus to 20 for laptop to run smoothly.  This will skip
a given number
            // of graphic refreshes to make simulation faster.
            if (t%1 == 0)
            {
                  glutPostWindowRedisplay (window1);
                  glutPostWindowRedisplay (window2);
            }
      }
}

// function to add graphics to a given OpenGL window
// "which_window" parameter defines which window is currently being updated
// this will change whether/where some procedures are called
void global_display (int which_window)
{
      glClear(GL_COLOR_BUFFER_BIT);
      glPushMatrix();
      // Pull current data from array.
      data thisdata = data_list[t];

      // Draw road profile, randomly adding step near the end for now
      // Shift road for close-up view
      glLineWidth (1);
      glPushMatrix();
      if (which_window == window2)
            glTranslatef (-thisdata.x, 0, 0);
      glBegin (GL_LINE_STRIP);
      glVertex2f (-track_x_offset, -tire_rad);
      glVertex2f (track_step_x+L*cos(track_pop_theta), -tire_rad);
      glVertex2f (track_step_x+L*cos(track_pop_theta), -
tire_rad+L*sin(track_pop_theta));
      glVertex2f (track_x - track_x_offset, -
tire_rad+L*sin(track_pop_theta));
      glEnd();
      if (which_window == window2)
            glTranslatef (thisdata.x, 0, 0);
      glPopMatrix();

      // Draw text on screen, showing time for now
      if (which_window == window1)
            glRasterPos2f (0, -track_y_offset);
      else
            glRasterPos2f (-hold_x_offset, -hold_y_offset);
      string text = "Time: ";
      ss << thisdata.time;
      text += ss.str() + " sec";
      ss.str("");
      for (int i = 0; i < text.length(); i ++)
            glutBitmapCharacter (GLUT_BITMAP_HELVETICA_10, text[i]);

      // Translate and rotate the cart in the screen as required
      // Shift cart for full window
      if (!pop_up)
      {
            if (which_window == window1)
                  glTranslatef (thisdata.x, 0, 0);
```

```
                  glRotatef ((thisdata.theta+thisdata.theta_add)*180/M_PI, 0,
0, 1);
      }
      else
      {
            // modified equations for pop up
            if (which_window == window1)
                  glTranslatef (thisdata.x, 0, 0);
            glTranslatef (L*cos(pop_theta), L*sin(pop_theta), 0);
            glRotatef ((thisdata.theta_add + thisdata.theta)*180/M_PI, 0, 0,
1);
            glTranslatef (-L, 0, 0);
      }

      // Draw the rigid cart
      glLineWidth (1);
      glBegin (GL_LINE_LOOP);
      glVertex2f (0, 0);
      glVertex2f (L, 0);
      glVertex2f (L, h);
      glVertex2f (b_x0, h);
      glVertex2f (b_x0, b_y0);
      glVertex2f (b_x0, h);
      glVertex2f (0, h);
      glEnd ();

      // Draw rear tire
      glPushMatrix();
      glBegin (GL_LINE_LOOP);
      glVertex2f (tire1_cen_x + tire_rad, tire1_cen_y);
      glVertex2f (tire1_cen_x+half_ang, tire1_cen_y+half_ang);
      glVertex2f (tire1_cen_x, tire1_cen_y + tire_rad);
      glVertex2f (tire1_cen_x-half_ang, tire1_cen_y+half_ang);
      glVertex2f (tire1_cen_x - tire_rad, tire1_cen_y);
      glVertex2f (tire1_cen_x-half_ang, tire1_cen_y-half_ang);
      glVertex2f (tire1_cen_x, tire1_cen_y - tire_rad);
      glVertex2f (tire1_cen_x+half_ang, tire1_cen_y-half_ang);
      glEnd ();
      glPopMatrix();

      // Draw front tire
      glPushMatrix();
      glBegin (GL_LINE_LOOP);
      glVertex2f (tire2_cen_x + tire_rad, tire2_cen_y);
      glVertex2f (tire2_cen_x+half_ang, tire2_cen_y+half_ang);
      glVertex2f (tire2_cen_x, tire2_cen_y + tire_rad);
      glVertex2f (tire2_cen_x-half_ang, tire2_cen_y+half_ang);
      glVertex2f (tire2_cen_x - tire_rad, tire2_cen_y);
      glVertex2f (tire2_cen_x-half_ang, tire2_cen_y-half_ang);
      glVertex2f (tire2_cen_x, tire2_cen_y - tire_rad);
      glVertex2f (tire2_cen_x+half_ang, tire2_cen_y-half_ang);
      glEnd ();
      glPopMatrix();

      // Mark centroid of cart and added mass with points
      glPointSize (4);
      glBegin (GL_POINTS);
```

```
	glVertex2f (a_x, a_y);
	glVertex2f (thisdata.b_x, thisdata.b_y);
	glEnd ();

	glPopMatrix();
	glutSwapBuffers();
}

// function to update graphics in close-up window
void display2 ()
{
	// call global_display
	global_display (window2);
}

// function to update graphics in full simulation window
void display()
{
	// call global display
	global_display (window1);
}

// main
int main(int argc, char **argv)
{
	// Check to verify "steps" value was input correctly.
	if (duration/time_step+1 != steps)
	{
		cout << "Error: Invalid # of steps." << endl;
		_getch();
		exit(1);
	}

	// Establish initial conditions.  data_list[1] is first time step, so
data_list[0]
	// serves as initial condition.
	data_list[0].a = accel_list[0];
	data_list[0].theta = 0;
	data_list[0].v = 0.2;
	data_list[0].x = 0;
	data_list[0].deflect = 0;
	data_list[0].b_x = b_x0 - data_list[0].deflect;
	data_list[0].d_x = d_x0 + data_list[0].deflect;
	data_list[0].b_y = b_y0;
	data_list[0].d_y = d_y0;
	data_list[0].L_b = sqrt(data_list[0].b_x*data_list[0].b_x +
data_list[0].b_y*data_list[0].b_y);
	data_list[0].L_d = sqrt(data_list[0].d_x*data_list[0].d_x +
data_list[0].d_y*data_list[0].d_y);
	if (I_M1 < 0 && I_m < 0)
	{
		data_list[0].J1 = M*L_a*L_a + m*data_list[0].L_b*data_list[0].L_b
+ M*(L*L+h*h)/12;
		data_list[0].J2 = M*L_c*L_c + m*data_list[0].L_d*data_list[0].L_d
+ M*(L*L+h*h)/12;
	}
	else
```

```
	{
		data_list[0].J1 = I_M1 + I_m +
m*data_list[0].L_b*data_list[0].L_b;
		data_list[0].J2 = I_M2 + I_m +
m*data_list[0].L_d*data_list[0].L_d;
	}

	double normal_force = m*g*cos(data_list[0].theta);
	data_list[0].R2 =                          // defined from a_mx and a_Mx
from current timestep
		(1/L)*(M*g*cos(data_list[0].theta)*a_x +
(data_list[0].b_x)*(normal_force));
	data_list[0].R1 = (m+M)*g - data_list[0].R2;
	if (data_list[0].R1 < 0)
		data_list[0].R1 = 0;
	if (data_list[0].R2 < 0)
		data_list[0].R2 = 0;
	data_list[0].d_T1 = data_list[0].R1/K_tire;
	data_list[0].d_T2 = data_list[0].R2/K_tire;
	data_list[0].theta += asin((data_list[0].d_T1-data_list[0].d_T2)/L);
	data_list[0].thetadotdot = 0;

	// Create window 1
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowSize(1200, 1200*track_y/track_x);
	window1 = glutCreateWindow("Cart lift-off simulation");
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(-track_x_offset, track_x - track_x_offset, -track_x_offset,
track_y - track_x_offset);
	glClearColor(0.0, 0.0, 0.0, 1.0);
	glColor3f(1.0, 1.0, 1.0);
	glutDisplayFunc(display);

	// Create window 2
	glutInitWindowSize(600, 600*hold_y/hold_x);
	glutInitWindowPosition(0, 1200*track_y/track_x+50);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	window2 = glutCreateWindow("Close-up view");
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(-hold_x_offset, hold_x - hold_x_offset, -hold_y_offset,
hold_y - hold_y_offset);
	glClearColor(0.0, 0.0, 0.0, 1.0);
	glColor3f(1.0, 1.0, 1.0);
	glutDisplayFunc(display2);

	// Begin simulation
	glutIdleFunc(tick);
	glutMainLoop();

	return 0;
}


double calc_I_M1_by7 ()
```

```
{
      return((
            0.05/2*pow(7.0,2.0) +
      // 7.0 - spring bracket to axle
            0.718*(pow(19.5,2.0)/12.0+pow(9.0,2.0)) +             // 9.0 -
slider bar to axle
            0.03/2*pow(5.0,2.0) +
      // 5.0 - rear slider-crossbar to axle
            0.03/2*pow(12.0,2.0) +
      // 12.0 - front slider-crossbar to axle
            2*0.16/2*(pow(5.5,2.0)/12.0+pow(5.5,2.0)) +                 //
5.5 - platform supports to axle
            2*0.02*(pow(3.25,2.0)/12.0+pow(4.0,2.0)) +                  //
4.0 - rear bot-vertical frame to axle
            2*0.02*(pow(3.25,2.0)/12.0+pow(7.5,2.0)) +                  //
7.5 - front bot-vertical frame to axle
            2.384*(pow(9.0,2.0)+pow(3.0,2.0))/12.0                // 3.0x9.0
- approximate dimensions of base
            + 2.384*(pow(c_x,2.0)+pow(c_y,2.0))
            )*0.00029264
      // convert lb-in^2 to kg-m^2
            );
}


double calc_I_M2_by7 ()
{
      return((
            0.05/2*pow(12.0,2.0) +
      // 12.0 - spring bracket to axle
            0.718*(pow(19.5,2.0)/12.0+pow(5.25,2.0)) +                  //
5.25 - slider bar to axle
            0.03/2*pow(9.5,2.0) +
      // 9.5 - rear slider-crossbar to axle
            0.03/2*pow(6.5,2.0) +
      // 6.5 - front slider-crossbar to axle
            2*0.16/2*(pow(7.25,2.0)/12.0+pow(7.25,2.0)) +         // 7.25 -
platform supports to axle
            2*0.02*(pow(11.75,2.0)/12.0+pow(9.0,2.0)) +                 //
11.0 - rear bot-vertical frame to axle
            2*0.02*(pow(11.75,2.0)/12.0+pow(3.5,2.0)) +                 //
8.0 - front bot-vertical frame to axle
            2.384*(pow(9.0,2.0)+pow(3.0,2.0))/12.0                // 3.0x9.0
- approximate dimensions of base
            + 2.384*(pow(c_x,2.0)+pow(c_y,2.0))
            )*0.00029264
      // convert lb-in^2 to kg-m^2
            );
}
```

References

[1] H. Schempf, W. Crowley, C. Gasior, D. Moreau, “Ultra-rugged Soldier-Robot for Urban Conflict Missions”, Unmanned Systems 2003 Conference - AUVSI 30th Annual Symposium and Exhibition, Baltimore, MD, July 15-17, 2003

[2] “KABOOM!! Setting off a bomb in the name of safety”, Feb 2008, National Research Council Canada <http://www.nrc-cnrc.gc.ca/eng/education/innovations/spotlight/kaboom.html>

[3] L. Grossman, “Maid to Order”, Inside Business, Sept 2002, <http://www.time.com/time/roomba/>

[4] S. Stoeter, and N.P. Papanikolopoulos, "Autonomous Stair-Climbing with Miniature Jumping Robots", IEEE Transactions on Systems, Man, and Cybernetics: Part B., Volume 35, No. 2, April 2005, pp 313-325.

[5] M. Yim, S. Homans, and K. Roufas, “Climbing With Snake-Like Robots,” IFAC Workshop on Mobile Robot Technology, May 2001.

[6] T. Mabuchi, T. Nagasawa, K. Awa, K. Shiraki and T. Yamada, “Development of a stair-climbing mobile robot with legs and wheels”, Artificial Life and Robotics, Volume 2, Number 4, December 1998

[7] G. Figliolini, M. Ceccarelli, ‘EP-WAR3 biped robot for climbing and descending stairs’, Robotica Internationa. Journal, Vol.22, 2004, pp. 405-417.

[8] C. Shih; C. Chiou; “The motion control of a statically stable biped robot on an uneven floor”; IEEE Transactions on Systems, Man and Cybernetics, Part B; vol.: 28 , Issue: 2; Apr. 1998; pp 244-249.

[9] Wilhelm, A., Melek, Sr., W., Huissoon, J., Clark, C., Hirzinger, Gerd., Sporer, N., Fuchs, M., “Dynamics of Step-climbing with Deformable Wheels and Applications for Mobile Robotics”, Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, November 2007, pp. 783-788.

[10] K.L. Moore, N.S. Flann, "A Six-Wheeled Omnidirectional Autonomous Mobile Robot," IEEE Control Systems Magazine, December 2000, pp. 53-66.

[11] D.R. Robinson, C. Wood, "Stair climbing capabilities of USU's T3 ODV mobile robot," Proceedings of SPIE, Vol. 4364 (2001).

[12] H. McGowen, “Navy omnidirectional vehicle (ODV) development and technology transfer opportunities,” Coastal Systems Station, Dahlgren Division, Naval Surface Warfare Division, unpublished report.

[13] M. Asama, M. Sato, H. Kaetsu, K. Ozaki, A. Matsumoto, and I. Endo, "Development of an omnidirectional mobile robot with 3 DOF decoupling drive mechanism," Journal of the Robotics Society of Japan, vol. 14, no. 2, pp. 95-100, 1997 (in Japanese).

[14] S. Kimmel, D. Hong, "Software Architecture and Motion Planning for Traversing Obstructed Terrain with a Rimless Wheel with Individually Actuated Spokes", KSEA US-Korean Conference on Science, Technology, and Entrepreneurship, Reston, VA, 2007

[15] D. Campbell, M. Buehler. "Stair Descent in the Simple Hexapod 'RHex'," Proceedings of the 2003 IEEE Int. Conf. on Robotics and Automation (ICRA).

[16] D.M. Helmick, S.I. Roumeliotis, M.C. McHenry, L. Matthies, "Multi-sensor, high speed autonomous stair climbing," Intelligent Robots and System, 2002. IEEE/RSJ International Conference on , vol.1, no., pp. 733-742 vol.1, 2002

[17] Y. Uchida, K. Furuichi, S. Hirose, "Consideration of stair-climbing performance of a six-wheeled off-road vehicle 'HELIOS-V'", Proc. 2nd Int. Symp. CLAWAR 99, 383-391(1999)

[18] W.A. Lewinger, M.S. Watson, R.D. Quinn, "Obstacle Avoidance Behavior for a Biologically-Inspired Mobile Robot Using Binaural Ultrasonic Sensors," Proceeding of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'06), Beijing, China, October 9-13, 2006.

[19] "Futuristic Unmanned Ground Combat Vehicle Revealed", Unmanned Systems Volume 24 No. 4, 2006

[20] T. McGeer, "Passive dynamic walking," Int. J. Robot. Res., vol. 9, no. 2, pp. 62–82, 1990.

[21] J. Heaston, D. W. Hong, "Design Optimization of a Novel Tripedal Locomotion Robot Through Simulation and Experiments for a Single Step Dynamic Gait", 31st ASME Mechanisms and Robotics Conference, Las Vegas, Nevada, September 4-7, 2007.

[22] S.H. Collins, M. Wisse, A. Ruina, "A 3-D Passive Dynamic Walking Robot with Two Legs and Knees", International Journal of Robotics Research, 20 (7):607-615, 2001

[23] S. Collins, A. Ruina, R. Tedrake, M. Wisse Efficient Bipedal Robots Based on Passive-Dynamic Walkers, *Science* 307, 1082, 2005

[24] Arai, Tanie, ,S. Tachi, "Dynamic control of a manipulator with passive joints in operational space", pp. 85-93.

[25] H. S. Seifert, “The lunar pogo stick,” J. Spacecraft, vol. 4, no. 7, pp. 941–943, Jul. 1967.

[26] M. Ahmadi, and M. Buehler (2006), "Controlled passive dynamic experiments with the ARL Monopod II," IEEE Transaction on Robotics **22**(5): 974-986

[27] M. Ahmadi, M. Buehler, "A control strategy for stable passive running," iros, p. 3152, International Conference on Intelligent Robots and Systems-Volume 3, 1995

[28] A. Deshpande, J. Luntz, “Decentralized Control for a Team of Physically Cooperating Robots”, Proceedings of the 2003 IEEBRSJ InU. Conference on Intelligent Robots and Systems Las Vegas, Nevada ’ October 2003, pp1757 – 1762

[29] T. Mongkol, S. Viboon, “Traction Control of a Rocker-Bogie Field Mobile Robot”, Proceedings. JSAE Annual Congress, VOL.;NO.38-05;PAGE.1-4, (2005)

[30] P. Song, P. Kraus, V. Kumar and P. Dupont, "Analysis of Rigid Body Dynamic Models for Simulation of Systems with Frictional Contacts," Journal of Applied Mechanics, Vol. 68, No. 1, pp. 118-128 (2001).

[31] J. Alberto, J. Bonitatibus, “Oscillation Control of a Pendulum with Sliding Mass and Perturbation in the pivot”,” 3rd International Conference Physics and Control, PhysCon 2007

[32] K. Yoshida, S. Okanouchi, H. Kawabe, "Vibration Suppression Control for a Variable Length Pendulum with a Pivot Movable in a Restricted Range," SICE-ICASE, 2006. International Joint Conference, vol., no., pp.4538-4544, Oct. 2006

[33] J. Osborne, D.V. Zenkov “Steering the Chaplygin Sleigh by a Moving Mass”, Proc. CDC 44, 1114–1118 (2005).

[34] E. Bakker, L. Nyborg, H. B. Pacejka, "Tyre Modelling for Use in Vehicle Dynamics Studies," SAE Paper No. 870421, (1987).

[35] H. Sakai, "Theoretical and Experimental Studies on the Dynamic Properties of Tyres III. Calculations of the Six Components of Force and Moment of a Tyre," International Journal of Vehicle Design, 2(3):182-226., (1981).

[36] Vehicle Dynamics Standards Committee, “Tire Normal Force/Deflection and Gross Footprint Dimension Test”, SAE, January 2005

[37] A. Discant, A. Rogozan, C. Rusu, A. Bensrhair, "Sensors for Obstacle Detection - A Survey," Electronics Technology, 30th International Spring Seminar on , vol., no., pp.100-105, 9-13 May 2007

[38] W. van der Mark, J.C. van den Heuvel, F.C.A. Groen, "Stereo based Obstacle Detection with Uncertainty in Rough Terrain," Intelligent Vehicles Symposium, 2007 IEEE , vol., no., pp.1005-1012, 13-15 June 2007

[39] R. Labayrade, D. Aubert, J.-P. Tarel, "Real time obstacle detection in stereovision on non flat road geometry through 'v-disparity' representation", Intelligent Vehicle Symposium, 2002. IEEE , vol.2, no., pp. 646-651 vol.2, 17-21 June 2002

[40] P.H. Batavia, S. Singh, "Obstacle detection using adaptive color segmentation and color stereo homography," Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on, vol.1, no., pp. 705-710 vol.1, 2001