MARVIN:
Multimodally Advantaged Robotic Vehicle for Improved Navigation

Adam Fisch[1] and Maksimilian Shatkhin[2], '15

Submitted to the
Department of Mechanical & Aerospace Engineering[1]
and Department of Electrical Engineering[2]
Princeton University
in partial fulfillment of the requirements of
Undergraduate Independent Work.

Final Report

May 4, 2015

Professor Andrew Houck
Professor Clarence Rowley
ELE 497-498
97 pages

Demo and test videos:
http://bit.ly/1bI3IYS

Abstract

Hybrid robots leverage the advantages of multiple types of locomotion. More specifically, wheel-legged hybrid robots aim to capture the speed, stability, and power efficiency of wheeled robots as well as the ability to traverse robust natural terrain that legged robots provide. Effective hybrid designs are able to capitalize on both sets of advantages without compromising the overall effectiveness of the machine. Here, we present a design and implementation of MARVIN, a wheel-legged hybrid robot that emphasizes three key features: a quick transition mechanism, a well-defined wheel and leg mode, and the capacity for flexible control through continuously variable leg length. We demonstrate how the two clearly defined modes of legs/wheels in MARVIN capitalize on their respective advantages. Furthermore, in realizing the tradeoff between modes specific to this robot, we derive a hybrid path-planning algorithm using an empirically driven cost function, which we found by collecting data in real-terrain experiments. We discuss our mechanical, electronic, and software design approaches in building a prototype of the proposed design. We also review our experimental methods. Lastly, we point out lessons learned from the operation of our prototype robot, identifying directions for future upgrades.

Acknowledgements

This work is funded by the Princeton University School of Engineering and Applied Science, the Department of Electrical Engineering and the Department of Mechanical and Aerospace Engineering. The authors would like to thank Professors Andrew Houck and Clarence Rowley for acting as advisers throughout the process. We would also like to thank David Radcliffe, Glenn Northey, Gabriel Mann, and Spencer Nichols for their help in troubleshooting our issues and discussing design considerations, as well as numerous other staff, graduate students, and professors in the Electrical Engineering Department and Mechanical Engineering Department for their contributions to this project.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Hybrid Modes: An Optimization Approach

For ground robots, mobility is a key, defining feature. In fact, even in the age of rapid advances in artificial intelligence and cutting-edge robotic sensors, robot mobility is still considered an area of predominant importance, and a major limitation in current robotic technology [20, 43]. With respect to locomotion, robots are generally classified as either legged or wheeled. As a consequence, this distinction creates a sharp dichotomy in the applications for which these types of robots are well-suited.

Wheels are widely known to be smooth, quick, and energy efficient. However, they are designed for the artificially paved world, and quickly lose effectiveness on rough, natural terrain. In contrast, legs are capable of traversing tough environments, and often mimic biological behaviors that perform well in nature. But they come at a significant cost. Legged robots are typically slow, unstable, and power-hungry.

In certain domains, this dichotomy does not produce a noticeable disadvantage. Clearly, legs would not provide any use for factory robots intended for flat, indoor facilities, nor would wheels appreciably aid a climbing robot. Today, however, there is an increasing demand for robotic vehicles suitable for high-risk, diverse terrain missions [21]. One example, in particular, of intense interest is disaster-zone rescue efforts. In both the 9/11 and Fukushima-Daiichi catastrophes, robots were deployed in an attempt to search for trapped victims, as well as report feedback on the status of humanly inaccessible areas [43]. In such scenarios, especially when the exploration region is large, speed and efficiency are paramount. Likewise, robustness against unforeseen obstacles is not just desirable — it is a fundamental necessity.

In response to these demands, there are hybrid designs seeking to combine legs and wheels in a single design in an attempt to capture the advantages of both modes. The hybrid model is a far less developed concept than either solely wheeled or legged systems, however now it is receiving intense research interest [5].

In this thesis, we propose an improvement upon existing hybrid approaches by introducing a novel geometrical structure and control strategy that allow for new flexibilities and performance characteristics. Specifically:

1. We developed a design that has smooth and fast transitions between distinctive wheel and leg modes.

2. We introduced the capability of what we term "flexible control." By flexible control we mean that the robot has the ability to modulate the extent to which the leg mode is emphasized in an effort to select the best leg dynamics[1].

3. Given our design, we introduced a novel cost model that allows us to algorithmically estimate the most efficient, hybrid strategy to move from point A to point B given information about the current terrain. This model is developed from experimental data of power consumption and stability, which are translated into costs that the robot incurs during each mode.

Our contributions are implemented through the design, construction, operation, and analysis of a multimodal, robotic vehicle prototype called MARVIN, that seeks to enhance navigational ability in diverse, unstructured environments. While we have only produced a basic prototype, we envision that with further development MARVIN could eventually be suitable for major applications both on earth and in space. While deployed on remote celestial bodies, such as the Moon or Mars, where energy is limited, efficient locomotion over unstructured environments is important. A planetary rover might face anything from flat crust, to sharp rocks that would puncture wheels, to uneven boulder-strewn landscapes. On Earth, MARVIN's terrain robustness and path-planning capabilities give it potential as an Urban Search and Rescue (USAR) robot, or in multiple other roles in remote exploration or the military.

[1]Flexible control is proposed and developed as a design feature in this report. However, due to time constraints, MARVIN's prototypical, physical implementation does not take advantage of this feature, and it remains a rich area for future, real-world experimentation.

1.2 The Legs vs. Wheels Tradeoff

Prior to delving into hybridization methods, we first explore the mechanical attributes of legs and wheels in greater depth to gain a better understanding of the tradeoffs. In the design of dynamical systems — and in robotics in particular — naturally evolved, biological systems often serve as sources of inspiration. Nature favors legs as a means of locomotion in most organisms, but the motorized wheel is a human invention that dominates on flat surfaces [40]. The main benefit of wheels is their ability to roll and coast. When driving on suitable terrain, the main energy losses for a wheeled robot come only from friction in the motor shafts and wheel axles, as well as limited aerodynamic drag at higher speeds. This is significant: consequently, on flat surfaces wheeled locomotion can be up to two orders of magnitude more efficient than legged locomotion [40].

Rolling motion has several other desirable qualities in addition to power efficiency. The movement of the robot is smooth; the wheel's rotation is transferred directly into translation, and oscillations of the robot's center of gravity are avoided. This creates less wear and tear on mechanical parts. Moreover, acceleration of a conventional wheel is a simple process to actuate and large accelerations can be generated without unreasonable effort. Given their efficiency and ability to accelerate easily, wheeled robots can achieve and maintain high speeds, even when carrying heavier payloads.

However, the effectiveness of wheels deteriorates rapidly with increasing terrain roughness. On soft, readily deformable surfaces, rolling friction begins to have a pronounced effect. When the ground surface is depressed, asymmetrical pressure distributions form on the wheel, producing counter-moments that impede the rolling motion. Loose terrain, such as sand, is also difficult for wheeled robots to traverse [17]. For example, in 2009, the Mars rover Spirit became stuck in a sand trap and never recovered. With loose substrates, wheels no longer maintain a no-slip condition at the surface. Additionally, since wheels maintain continuous contact with the ground, they tend to have a plowing effect on the ground material in front of them. Eventually, this buildup can become severe enough to completely stop the robot.

Wheels are also not particularly proficient at maneuvering over obstacles. Standard wheels are typically only capable of surmounting steps of heights less than their radius. Furthermore, wheels have to always maintain contact with the ground to move. Thus, discontinuous terrain, where there are gaps in the path, is very challenging for wheeled robots to navigate.

Legged robot designs succeed in many of the situations where wheeled robots

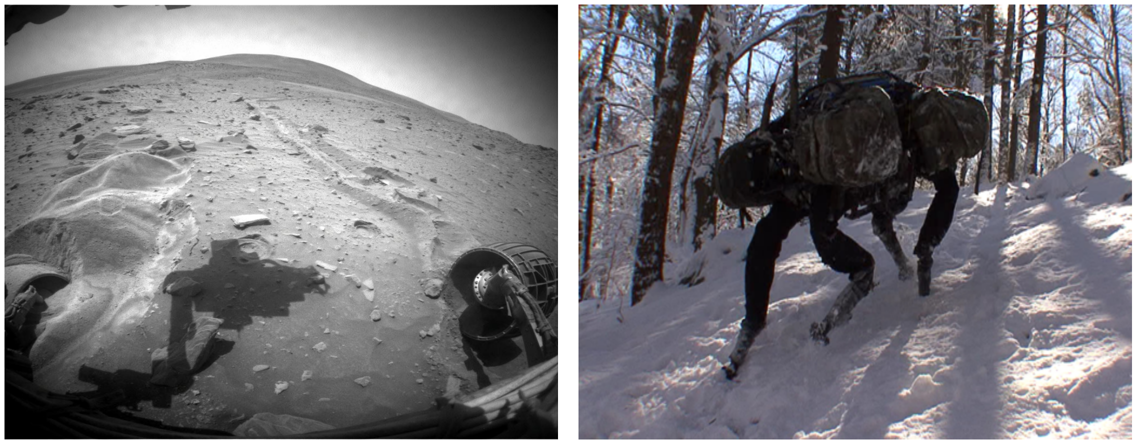

(a) (b)

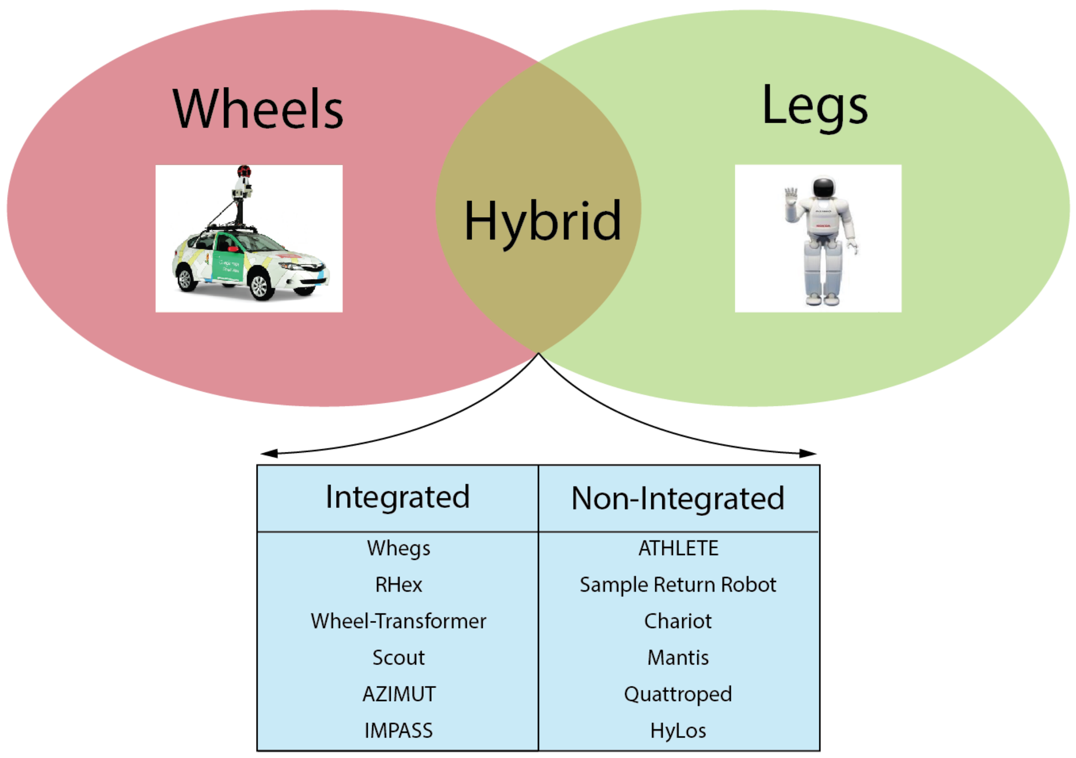
Figure 1.1: (a) The Mars rover Spirit stuck in sand in 2009 [34]. (b) BigDog performs well on rugged terrain, but is limited to slow speeds [15].

fail. With legs, a robot only maintains a discrete number of contact points with the ground. As long as the robot has adequate leg clearance over obstacles, the quality of the ground between each contact point along the path does not matter as much [40]. This lends itself to a large boost in maneuverability in rough environments. Furthermore, legs can be manipulated strategically to allow the robot to surmount taller barriers. For example, a robot might be able to lift its legs up to directly step on or over a vertical obstacle — an action that cannot be done with wheels.

Still, legged motion comes at a steep price. While leg designs vary widely in their implementation, they all have inherent inefficiencies. In most designs, energy is lost in each leg collision with the ground [8]. Moreover, dynamic instabilities may be introduced at high velocities. Finally, the legs must be capable of raising, sustaining, and lowering the robot's total weight, a task made more difficult in view of only a few contact points with the ground.

1.3 Prior Work

As previously mentioned, hybridized robotics is now an area of rapid development and active experimentation [1, 5, 42]. Creating a robust and adaptive robot encourages creativity. As a result, the existing body of research is remarkably diverse. At the highest level, however, many of the designs can be sorted into two fundamental categories: integrated versus non-integrated leg-wheels [26]. Non-integrated leg-wheel designs have individually distinguishable leg and wheel components. In contrast,



Integrated	Non-Integrated
Whegs	ATHLETE
RHex	Sample Return Robot
Wheel-Transformer	Chariot
Scout	Mantis
AZIMUT	Quattroped
IMPASS	HyLos

Figure 1.2: Examples from the hybridized mobile robot design space.

integrated leg-wheels seek to incorporate leg-like performance characteristics into their design by using novel wheels with specially altered physical structures.

Both approaches and their variations have advantages as well as drawbacks. Non-integrated designs typically are able to carry a heavier payload. However, their mechanisms can become very complex, requiring intensive control efforts and maintenance [40]. Additionally, robots that transform their shape often experience a dead-time period as the change takes place. Meanwhile, integrated wheel-leg robots often minimize design complexity and mode switching latency. Still, these approaches have to compromise between the idealized performances of wheels and legs. Often the resulting wheel-leg is neither great at rolling nor particularly impressive at walking. Furthermore, hybrid wheels generally have some aspect of disjoint geometry. This can cause robot stability to suffer at high speeds.

In order to obtain a deeper appreciation of the existing research, explain the origins of MARVIN's structure, and better define where our contributions align with the current literature, we now address a few of the more compelling and successful designs referenced in Fig. 1.2.

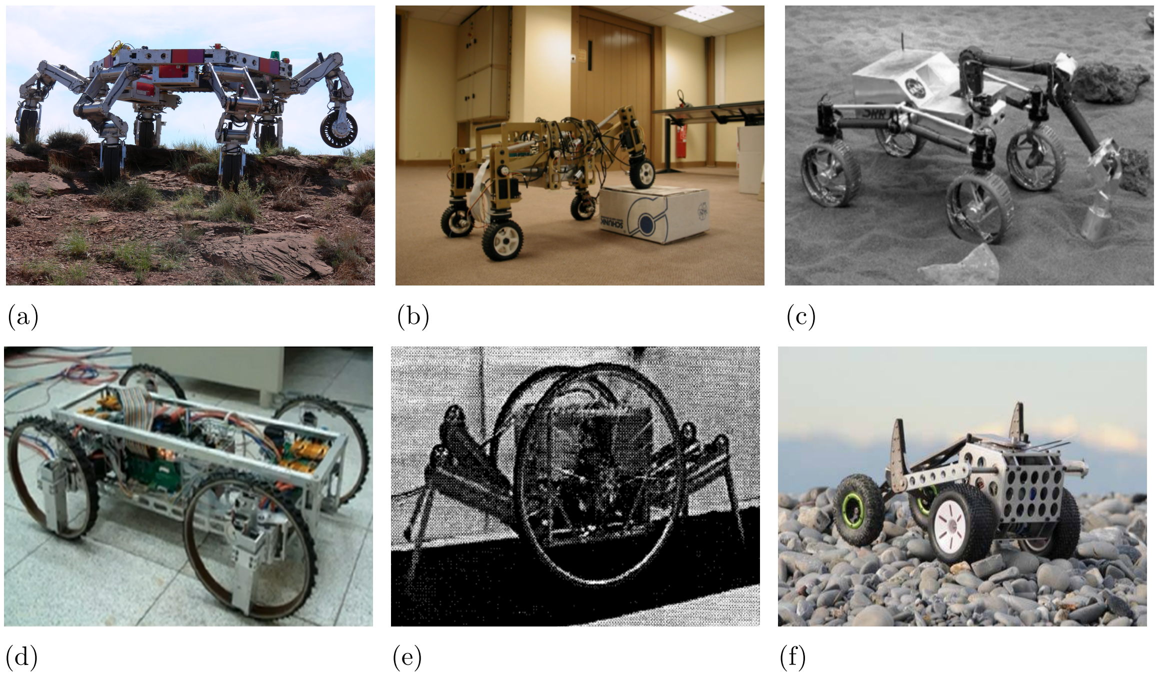
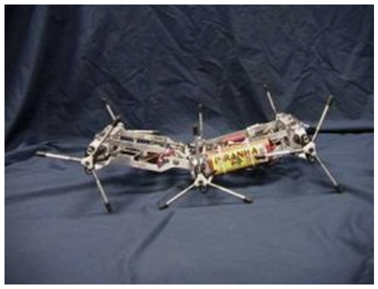


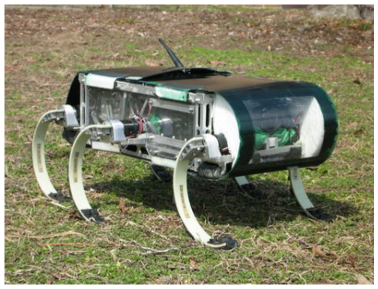
Figure 1.3: Non-Integrated hybrid robot designs: (a) ATHLETE [47], (b) HyLos [18], (c) Sample Return Robot [21], (d) Quattroped [7], (e) Chariot [10], and (f) Mantis [6].

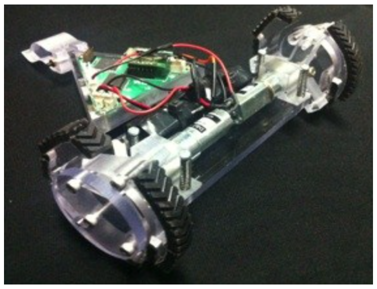
1.3.1 Non-Integrated Designs

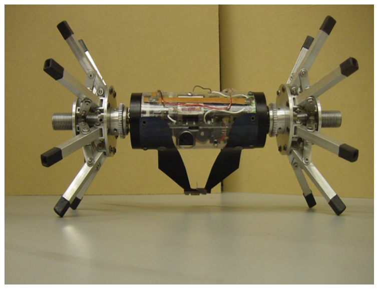
The Jet Propulsion Laboratory (JPL) and NASA have been at the forefront of the field, given the interest in hybrid robotics' promising potential in space applications. Both JPL and NASA have developed many leading designs, especially in non-integrated designs. The NASA ATHLETE robot for planetary exploration is an example of an approach that uses wheels attached to the bottom of several leg-like appendages [47]. When necessary, each wheel can be locked and treated as a foot while the limbs are used to walk out of extreme terrain. JPL's Sample Return Robot achieves robust performance on rough terrain by manipulating its chassis joints to shift itself and accommodate for obstacles [21]. HyLos, a smaller-scale, well-popularized variant built by the Laboratoire de Robotique de Paris (LRP), is similar to the ATHLETE, except that it extends the functionality of the leg mode even further by giving the hybrid limbs 16 degrees of freedom [18].

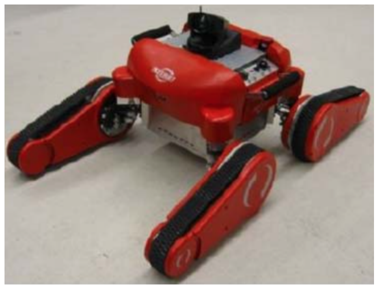
Different non-integrated designs avoid using wheeled appendages, because of the complex dynamics and often convoluted designs that they entail. Robots like the Chariot robot made at Tohoku University take a more straightforward approach, and have legs and wheels that are completely mechanically separated from each other [10].


(a)


(b)


(c)


(d)


(e)

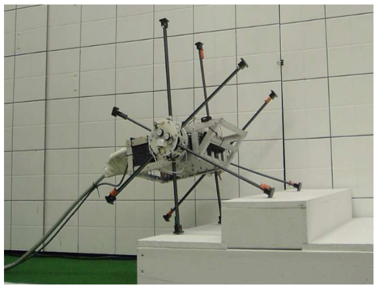

(f)

Figure 1.4: Integrated hybrid robot designs: (a) Whegs [35], (b) RHex [39], (c) Wheel-Transformer [25], (d) Scout [13], (e) AZIMUT [30], and (f) IMPASS [26].

The legs lift and push the wheeled system, similar in nature to a scooter or skateboard. The Mantis robot, a University of Genova design, employs related tactics with separate, rotating hooks that can be used to drag the robot up and over obstructions [6].

Other robots undergo extensive physical transformations when switching between modes. Quattroped, developed by the National Taiwan University, is such a design [7]. It stops, folds its wheels into semicircles, switches the rotation point to the outside of the wheel, and begins to operate as a four-legged robot.

1.3.2 Integrated Designs

While not yet deployed in important space exploration missions like some of their non-integrated wheel cousins produced by NASA, integrated-wheel designs are rising in popularity. In general, the integrated-wheel robots are small and low-weight. The Actuating Wheel Scout robot is a notable example of a diminutive, cheap, no frills robot designed to be used as part of a distributed reconnaissance team in challenging environments [13]. When faced with an obstacle that might be too tall to climb over and too low to roll under, the Scout uses a single degree of freedom umbrella mechanism wheel to either increase or decrease its wheel diameter accordingly.

The AZIMUT robot is also able to change the orientation of its wheels. Here AZIMUT's "wheels" are actually four, independent, elongated leg-track-wheel articulations [30]. Depending on the scenario, AZIMUT can operate with its wheels pointed up, down, or straight. This feature makes AZIMUT capable of remarkably versatile motions and of negotiating difficult, three-dimensional obstacles like stairs.

Within the category of multi-wheeled and legged vehicles, RHex is a well-known hexapodal robot developed at the University of Michigan and then the University of Pennsylvania, as well as other robotics laboratories like Boston Dynamics [39]. A variant on the RHex model was also studied in 2013 as part of a senior thesis at Princeton [28]. The RHex design consists of six rotary, compliant, curved leg blades. It is the unique nature of the wheel-like legs that enables RHex to achieve fast and robust forward locomotion.

Related to RHex, the Whegs robot, first conceived at Case Western University, is another prominent example of a successful integrated wheel-leg design. Biologically inspired by insects, Whegs uses rigid, rimless "spoke wheels" for excellent speed and mobility over varied terrain [35]. In many ways, RHex is the single-spoked version of Whegs [26]. Furthermore, since its popularization, the Whegs robot has inspired many spin-offs, including the IMPASS [20, 26] and Wheel-Transformer [25] designs. IMPASS, the Intelligent Mobility Platform with Active Spoke System, uses a Whegs-based model, however the legs are individually actuated to change the length of the leg touching the ground — allowing the robot to generate a more stable gait.

Last but not least, as mentioned in the introduction, the Wheel-Transformer uses a passive mechanism that allows the robot to morph between wheeled and "Whegged" modes [25]. In its wheeled mode, the Wheel-Transformers curved, thick legs roughly join end-to-end to make up the perimeter of a wheel. In leg mode, these legs are unfurled into a star-like configuration. Interestingly, the robot does not cause the transformation by choice. Rather, as it collides with a step or similar obstacle, friction drives the legs out. This passive actuation allows the design to be implemented on a small robot with limited power consumption. This design lies somewhere at the intersection of integrated and non-integrated design, and serves as a primary motivator for MARVIN's own hybrid design.

1.4 MARVIN

A comprehensive evaluation of the existing hybrid designs led us to focus our efforts on an attempt to come up with an approach that attempted to address a few aspects that we perceived as systematic deficiencies in the field. We consider the speed and energy advantages of wheels to be extremely valuable, especially in areas where the majority of the terrain is flat. The main shortcoming in most integrated designs is that pure, efficient rolling is compromised. At the same time, when driving, non-integrated designs often have to waste effort locking or stabilizing the extra degrees of freedom fully articulated legs introduce. For MARVIN, we wanted to accomplish a design that preserved the integrity of the wheel mode. Additionally, we wanted to preserve certain desirable characteristics from integrated designs, such as rapid or no switches between modes, and design simplicity.

Consequently, MARVIN's final design, pictured in Fig. 1.6, lies on the intersection of integrated and non-integrated approaches. It leverages the geometry of the wheel to either conceal or extend spoke legs with a simple internal rotation. As illustrated in a simplified diagram in Fig. 1.5, MARVIN's wheels are comprised of two coaxial disks, one of radius r_1 and the other of radius r_2, connected by slotted legs. The conception of this layout was inspired by camera apertures that change size when turning the focusing dial. Here, when the disks move relative to each other, the slots allow the legs to rotate in and out. Folded in, the legs become less of protrusion, and more a part of the rolling wheel. At the right length, they are completely contained inside the wheel. Extended, they form a spoke wheel similar to the Whegs design.

Similar to IMPASS, MARVIN can change its behavior at the spoke level. In a feature unique to our design, MARVIN can control the relative extension of the

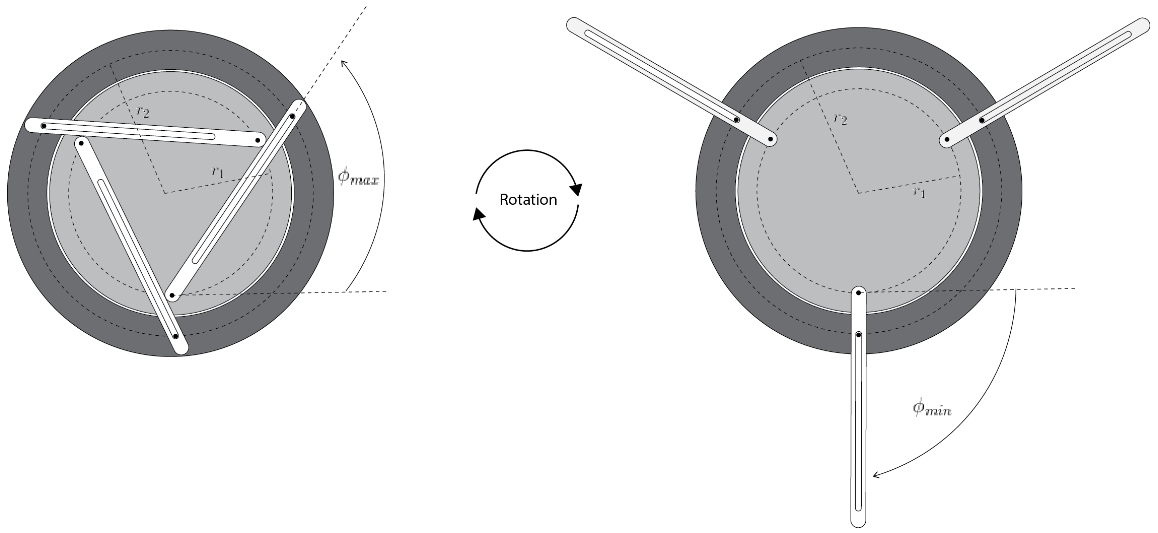


Figure 1.5: Internals of the leg extension mechanism. Spoke legs are retracted (l) or extended (r) via a relative rotation of the coaxial disks.

legs by changing the phase difference between the rotating disks. At the complete transformation level, the leg rotation angle, ϕ, is equal to ϕ_{min}. This maximizes the manifestation of leg-like characteristics. This is ideal for the roughest terrain. However, not all terrain demands the use of legs to the same extent. Furthermore, as we explore in Chapter 2, when spoke legs are more emphasized they incur increasingly expensive costs. Therefore, complete freedom over ϕ allows MARVIN to react to the terrain at an even higher level of specificity. A limited rotation can expose the legs just enough to traverse moderately rough ground while sacrificing minimal energy and stability.

Finally, MARVIN's design meets our specifications for a rapid transfer time between modes. In fact, there is no dead-time, as MARVIN can deploy its legs as it continues to move forward. This allows for fluid and frequent mode changes in response to intermittent obstacles, such as curbs or bumps in an otherwise smooth path. This is a significant improvement over designs like Quattroped that can take several seconds to complete a transformation.

It should be noted that the emphasis of this thesis is on MARVIN's theoretical design and potential for improved navigation through intelligent mode selection. As such, the physical implementation of MARVIN is a "Version 1.0" prototype, mainly intended for testing and design validation. Details on the construction process are given in Appendix B, while recommendations for implementation specific improvements are discussed in Chapter 7.

The following sections of this report conduct a rigorous analysis of MARVIN's design. First, we probe latent performance characteristics influenced by MARVIN's unique geometry and explore the complex dynamics of the leg mode. Then, both high and low level control laws for the leg-wheel mechanisms are developed and integrated into MARVIN's onboard software. This is followed by a discussion of MARVIN's electronics subsystem, including sensors for monitoring the robots energy consumption and dynamic stability during experimental tests. Findings from preliminary performance tests are subsequently presented and processed into the cost functions MARVIN uses for determining optimal, low cost, hybrid paths through varied terrain. In general, the material is presented in a more pedantic manner so that a future student or less experienced reader might be able to easily reproduce and continue research. Appendices are included for supplementary, detailed, technical information.

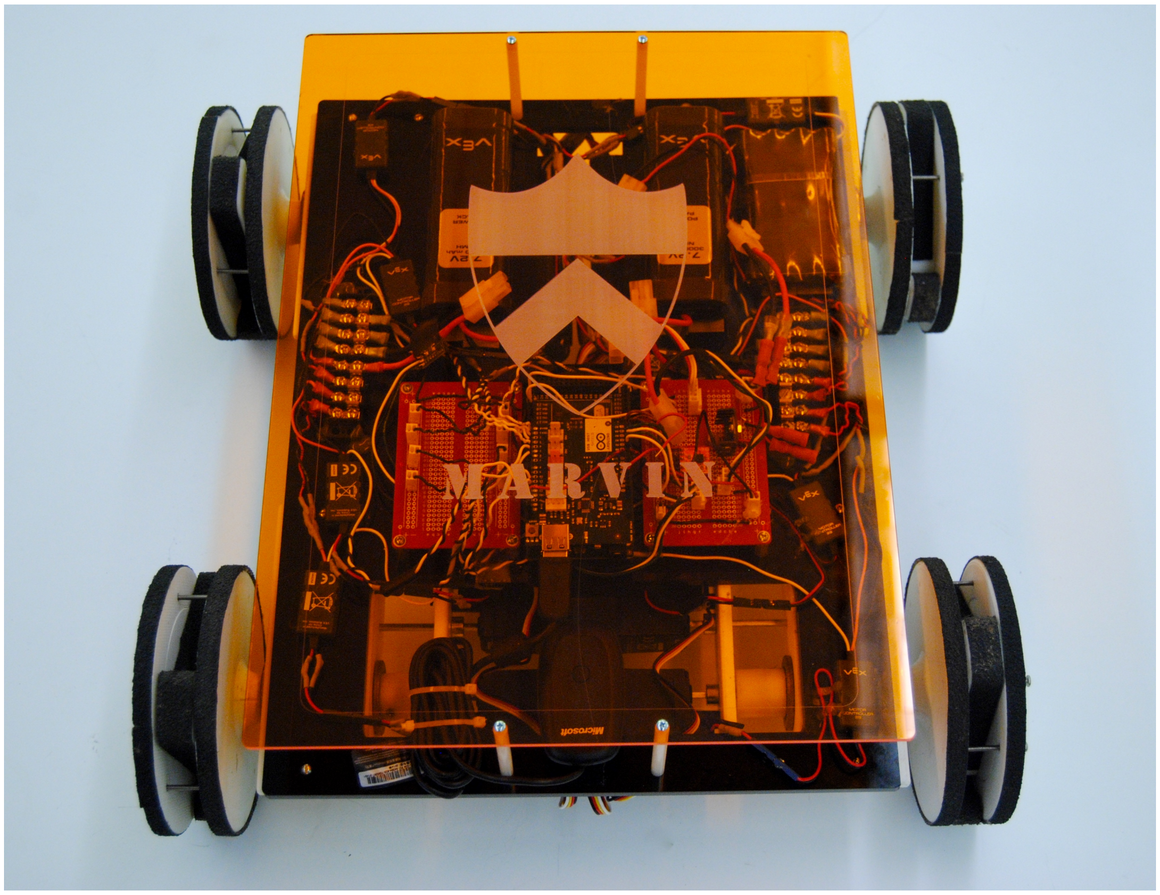


Figure 1.6: Top view of MARVIN.

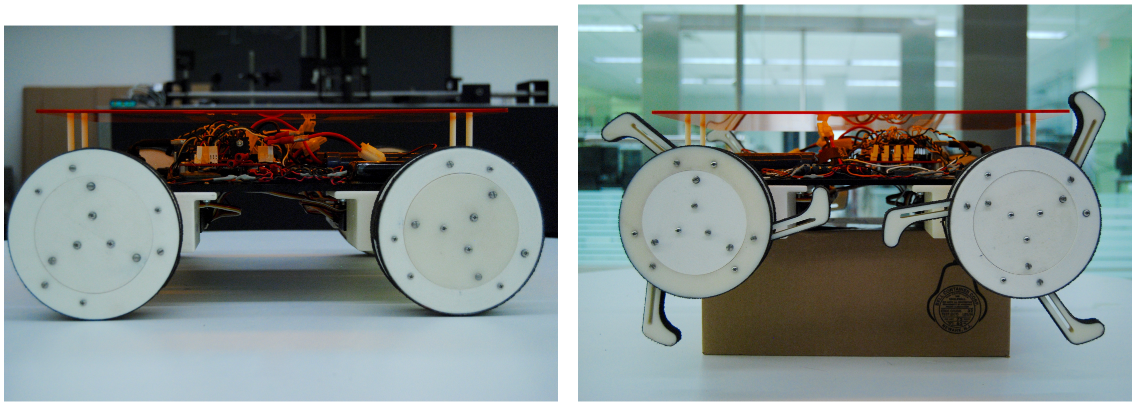


Figure 1.7: MARVIN side views.

Chapter 2

Mechanical Design Analysis

At its core, MARVIN's leg mode is a variant of the Whegs robot. Hence, a significant understanding of the methodology can be drawn from the prior research on Whegs models, such as [35], [31], or [46]. However, there are several new design challenges and considerations that are introduced due to the unique geometry of MARVIN's wheels and its active transition from wheels to legs.

In the following sections we examine the specifics of MARVIN's mechanical design in depth. In particular, we explore the effects of three primary, independent, free variables that determine much of the robot's performance:

1. The radius, r_1, of the inner disk.
2. The radius, r_2, of the outer disk.
3. The number of legs, N, used per wheel.

The selection of these dimensions has a widespread impact on important end behavior, such as climbing ability, dynamics, and torque requirements.

2.1 Retractable Leg Length

In order for MARVIN to roll smoothly in wheel mode, the legs must be able to retract far enough so that they fit perfectly within the outer perimeter of the wheel. Consequently, this condition places a constraint on the length of each leg. This dimension depends on a number of geometric interdependencies — primarily the radii of the two internal coaxial disks and the total number of legs per wheel.

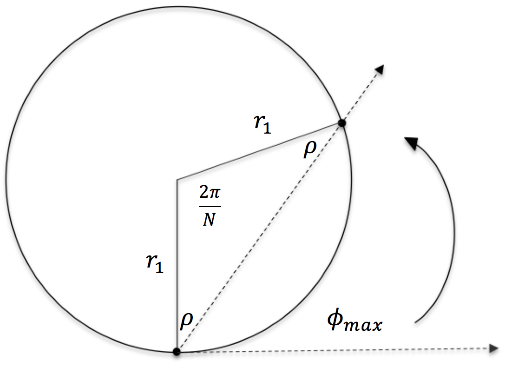


Figure 2.1: Determining the critical leg retraction angle.

The legs retract by pivoting inwards. However, since the legs occupy physical space inside the wheel, the legs are restricted from rotating past a critical retraction angle, ϕ_{max}. As shown in Fig. 2.1, ϕ_{max} for a leg is reached when it contacts the base pivot of the adjacent leg. The leg pivots are evenly distributed around the inner disk, thus as the number of legs increases, adjacent legs are positioned closer together. As a result, ϕ_{max} necessarily shrinks with increasing N.

Fig. 2.1 also depicts the pivot points of two adjacent legs, separated by an angle $\theta = 2\pi/N$. This angle is the vertex of an isosceles triangle, with base angle ρ. Using this observation, ϕ_{max} is calculated through basic trigonometry:

$$\begin{aligned} \pi &= \theta + 2\rho \\ \rho &= \frac{1}{2}\Big(\pi - \frac{2\pi}{N}\Big) \\ \phi_{max} &= \frac{\pi}{2} - \rho \\ \phi_{max} &= \frac{\pi}{N} \end{aligned} \tag{2.1.1}$$

In Eq. 2.1.1 we discount the effect of the thickness of the leg, assuming that it is small compared to the radii of the disks. It is important to note, however, that the true ϕ_{max} will be smaller — especially for thicker legs. Knowing ϕ_{max} then enables us to determine the maximum allowable leg length by locating the intersection point between a fully retracted leg and the outer disk radius. Fig. 2.2 shows that this point (x, y) is r_2 units from the disk origin, $(0, r_1)$. It follows that the length of leg contained within the wheel is simply the distance between (x, y) and the coordinate system origin, $(0, 0)$.

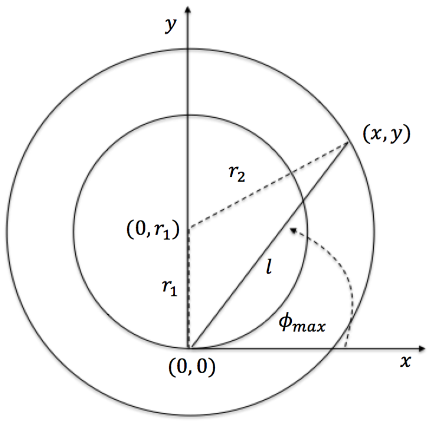


Figure 2.2: Determining the maximum leg length, l.

$$(x, y) = (l \cos \phi_{max},\ l \sin \phi_{max}) \tag{2.1.2}$$

$$r_2 = \sqrt{(l \cos \phi_{max})^2 + (l \sin \phi_{max} - r_1)^2} \tag{2.1.3}$$

$$r_2^2 = l^2(\cos^2 \phi_{max} + \sin^2 \phi_{max}) - 2r_1 l \sin \phi_{max} + r_1^2 \tag{2.1.4}$$

Nondimensionalizing by introducing $l' = l/r_2$ and $r' = r_1/r_2$ we have:

$$1 = l'^2 - 2r'l' \sin \phi_{max} + r'^2 \tag{2.1.5}$$

The positive solution of this quadratic form gives l' as a function of r' and ϕ_{max}:

$$\begin{aligned} l' &= r' \sin \phi_{max} + \sqrt{r'^2(\sin^2 \phi_{max} - 1) + 1} \\ &= r' \sin \phi_{max} + \sqrt{1 - (r' \cos \phi_{max})^2} \end{aligned} \tag{2.1.6}$$

Substituting Eq. 2.1.1 into Eq. 2.1.6 gives l' as a function of N and r'. This relationship is plotted in Fig. 2.3 for several values of r'. We see that smaller N and $r_2 \approx r_1$ results in longer legs. Additionally, it is important to examine not just the absolute leg length, but also the extension past the outer radius, r_2. By inspection, this relative extension is maximized for $\phi = -\pi/2$. For future use, we define this maximum relative extension as l_e:

$$l_e \equiv l - (r_2 - r_1) \tag{2.1.7}$$

As N increases, l approaches $(r_2 - r_1)$ and the l_e goes to 0.

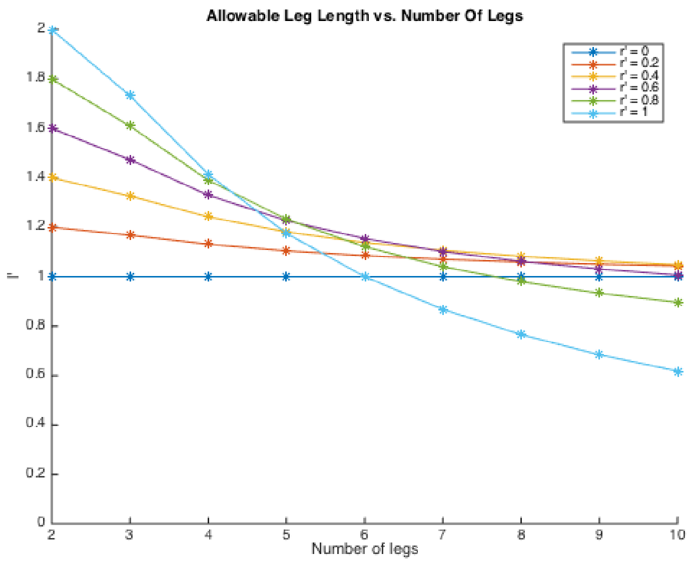


Figure 2.3: Nondimensional leg length $l' = l/r_2$ vs. number of legs.

2.2 Leg-Assisted Climbing

A significant advantage that spoke wheels have over conventional wheels is the ability to surmount taller obstacles. Obstacle traversal is largely limited by the amount of lifting force that the wheel or leg is able to generate on the vertical surface of the object. Normal wheels generally fail to produce a sufficient force when the obstacle height approaches the radius of the wheel. MARVIN's spoke design, however, gives a literal leg up to the robot, and allows it to generate force on higher points of contact not accessible to a solid wheel. This feature greatly improves climbing performance.

Fig. 2.4 illustrates quasi-static force balance conditions for a standard wheel of radius r approaching a step. Here, F_1 is the forward drive force applied to the robot body (possibly from multiple wheels), N_1 is the robot body weight, N_2 is the normal force from the step, F_2 is the frictional force applied to the step by the wheel motor torque τ, and μ is the coefficient of friction between the wheel and step surfaces.

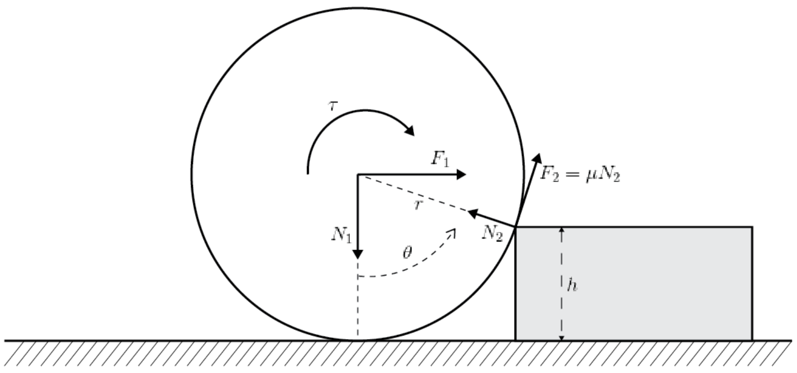


Figure 2.4: Forces on a regular wheel when climbing over a bump.

Previous work by [45] and [44] present a critical value for θ above which climbing is unlikely to be successful. This equation is arrived at through simple force balance in x and y:

$$\theta_{max} = \cot^{-1}\left(\frac{N_1 - \mu F_1}{\mu N_1 + F_1}\right) \tag{2.2.1}$$

The critical angle, θ_{max}, can be solved for given μ and the weight N_1. For $0 < \theta_{max} < \pi/2$, the corresponding step height h is then $r\cos\theta_{max}$. For $\theta_{max} \geq \pi/2$ where $h \geq r$, the wheel relies entirely on frictional force to climb vertically up the step surface. For that to be achieved, we must have $F_1 \geq N_1/\mu$ and $\tau \geq N_1 r$. However, for heavy robots or surfaces with poor to mediocre traction, these demands on F_1 or τ can often be unattainable.

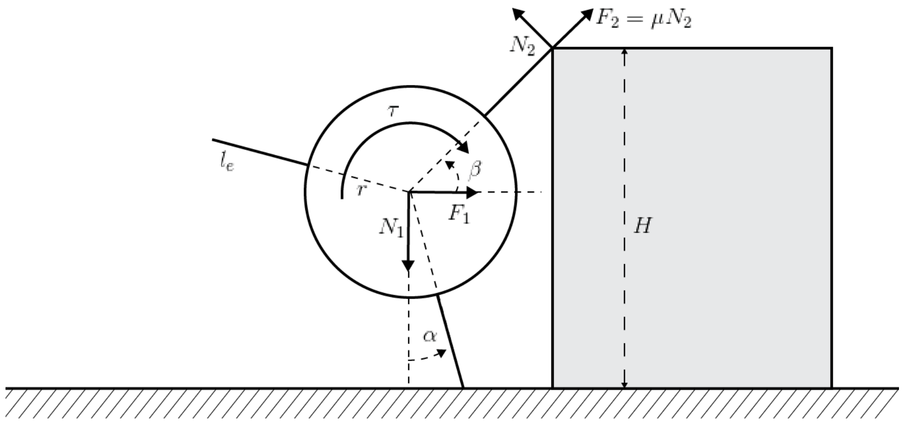


Figure 2.5: Forces on a wheel with spoke legs when climbing over a bump.

The capability to switch to a legged mode lends several benefits to step climbing. First, the effective radius of the wheel system increases from r_2 to $r_2 + l_e$. We can write the new effective radius in non-dimensional terms by scaling by $1/r_2$ as before:

$$\begin{aligned} r_e &= r_2 + l_e = l + r_1 \\ r'_e &= l' + r' \end{aligned} \tag{2.2.2}$$

Second, the geometry of the legs allows MARVIN to reach up at higher angles, as shown in Fig. 2.5. Here we measure α, the angle between the vertical and the supporting leg, and β, the angle between the stepping leg and the horizontal. We define the supporting leg as the leg in contact with the ground and the stepping leg as the leg closest to the step.

In an analysis of Whegs, Tantichattanont et al. [45] give an equation for β_{max}. Again, using static force balance in x and y, they get:

$$\beta_{max} = \cot^{-1}\left(\frac{N_1 - \mu F_1}{\mu N_1 + F_1}\right) \tag{2.2.3}$$

This is in the same form as Eq. 2.2.1, but the equivalent θ is shifted: equal to $\beta + \pi/2$. Thus the climbing ability of a spoke-legged wheel is automatically higher than its wheel radius [35]. The maximum step height H' (non-dimensionalized as H/r_2) for MARVIN's leg mode is given by:

$$H' = r'_e(\sin\beta + \cos\alpha) \tag{2.2.4}$$

As evident from Eq. 2.2.4, H' depends on the approach conditions α and β. Under the assumption that force requirements are met for all combinations of α and β, a spoke wheel will have optimal approach angles that maximize the step height. Examples of optimal and non-optimal wheel orientations for a three-spoke wheel are shown in Fig. 2.6.

Optimal configurations and the resulting climbing ability H' were calculated numerically for Eq. 2.2.4 for varying numbers of legs. Fig. 2.7(a) shows the isolated effects of improved geometry, where r'_e in Eq. 2.2.4 is replaced with a unit length. Fig. 2.7(b) combines the effect of increased r'_e, for a fixed r' of 0.6. We see that H' is higher for fewer legs, and as the number of legs grows, H' goes to 1 — which is consistent with a solid wheel. Again, it should be noted that Figs. 2.7(a)-(b) present theoretical limits to step heights that MARVIN can physically touch. While the trend will remain, the realizable steps heights are likely lower, and depend on the motor.

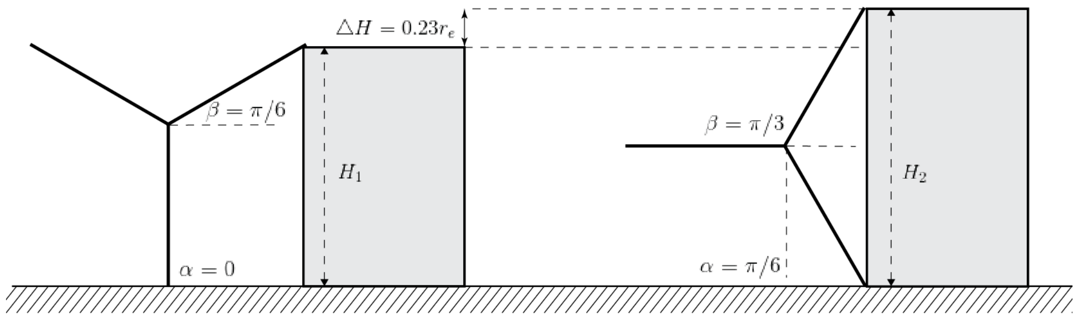
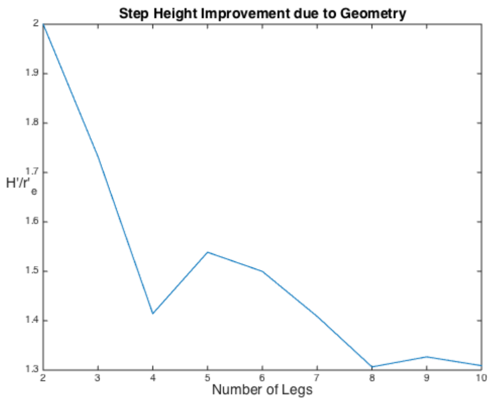
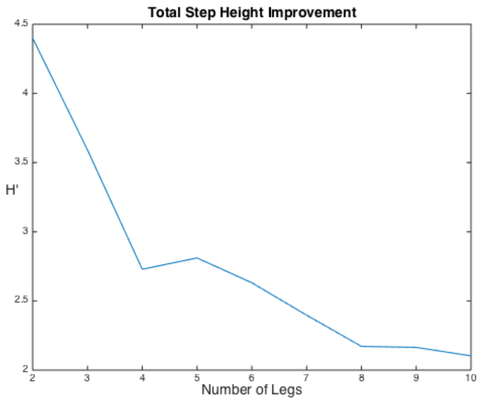


Figure 2.6: An example of a non-optimal (l) vs optimal (r) approach to a step.



(a)



(b)

Figure 2.7: Step height ratio H' vs. number of spoke legs. Graph (a) holds $r_e = 1$, while (b) combines both the effects of geometry and increased r_e.

2.3 Leg Mode Dynamics

2.3.1 "Walking"

As the spoke legs are deployed and MARVIN switches modes to a form of "walking", the dynamics change dramatically. When using wheels, the trajectory of the center of mass (COM) is smooth and stable, and the only torque requirements are due to overcoming friction in internal components. In leg mode, however, as shown in Fig. 2.8, the robot pivots from leg to leg — changing both the COM path and motor requirements.

These dynamics are closely related to those of rimless wheels and passive walkers[1] [29, 8]. Here we apply a similar, simple analysis to MARVIN's specific situation.

[1] *Engineering Dynamics* by Professor N. J. Kasdin [23] was particularly helpful for the derivation of this model. This solution is based on his example on the dynamics of passive walkers (Ex. 4.1).

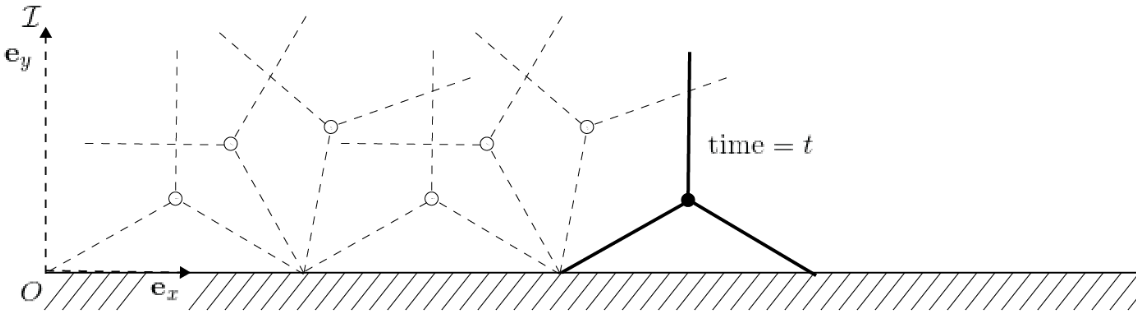


Figure 2.8: Wheel trajectory during leg mode.

Suppose that MARVIN's leg mode wheel system is modeled as single point mass with N massless legs of length $l = r_e$ extending radially outward. We can then show (see Eq. 2.3.5) that the intra-stride kinematics become that of an inverted pendulum. At the end of each pivot, the leg makes contact with the ground, which exerts a linear impulse that pushes the system into the next stride. Throughout its motion, the point mass is acted on by gravity, reaction forces with the ground, and a variable applied motor torque, τ.

2.3.2 Forces and Reference Frames

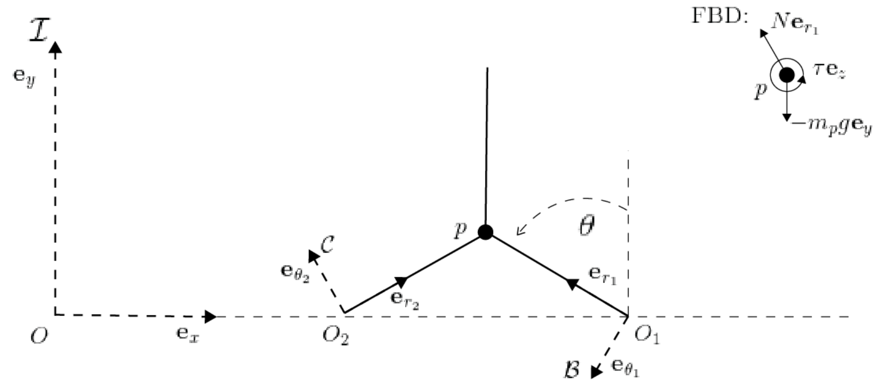


Figure 2.9: Reference frames and free body diagram for the leg mode.

Fig. 2.9 illustrates the free body diagram and set of coordinate systems that describe the wheel-leg system. $\mathcal{I} = (O, \mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$ is the inertial reference frame, while $\mathcal{B} = (O_1, \mathbf{e}_{r_1}, \mathbf{e}_{\theta_1}, \mathbf{e}_z)$ and $\mathcal{C} = (O_2, \mathbf{e}_{r_2}, \mathbf{e}_{\theta_2}, \mathbf{e}_z)$ are polar frames fixed to the two legs that come in contact with the ground. As shown in Fig. 2.9, $\mathcal{B}$ is related to $\mathcal{I}$ through a rotation θ. The reference frame transformation is shown in Table 2.1.

	$\mathbf{e}_x$	$\mathbf{e}_y$
$\mathbf{e}_{r_1}$	$-\sin\theta$	$\cos\theta$
$\mathbf{e}_{\theta_1}$	$-\cos\theta$	$-\sin\theta$

Table 2.1: Transformation from inertial frame $\mathcal{I}$ to polar frame $\mathcal{B}$.

Similarly, $\mathcal{C}$ is rotated from $\mathcal{B}$ by a fixed angle of $-2\pi/N$ given by the geometry of the legs. Thus, this reference frame transformation is given by Table 2.2.

	$\mathbf{e}_{r_1}$	$\mathbf{e}_{\theta_1}$
$\mathbf{e}_{r_2}$	$\cos(2\pi/N)$	$-\sin(2\pi/N)$
$\mathbf{e}_{\theta_2}$	$\sin(2\pi/N)$	$\cos(2\pi/N)$

Table 2.2: Transformation from polar frame $\mathcal{B}$ to polar frame $\mathcal{C}$.

2.3.3 Single Stride Dynamics

For conventional sign convenience, we now assume that the spoke wheel is rotating positively, traveling in the $-\mathbf{e}_x$ direction. During a pivot from O_1 to O_2, the system behaves like an inverted pendulum about O_1, with $-\pi/N < \theta < \pi/N$. This motion is illustrated in Fig. 2.10. From $t_0 \rightarrow t_1$ the kinematics of p are:

$$\mathbf{r}_{p/O_1} = l\mathbf{e}_{r_1} \tag{2.3.1}$$

$$^{\mathcal{I}}\mathbf{v}_{p/O_1} = l\dot{\theta}\mathbf{e}_{\theta_1} \tag{2.3.2}$$

$$^{\mathcal{I}}\mathbf{a}_{p/O_1} = l\ddot{\theta}\mathbf{e}_{\theta_1} - l\dot{\theta}^2\mathbf{e}_{r_1} \tag{2.3.3}$$

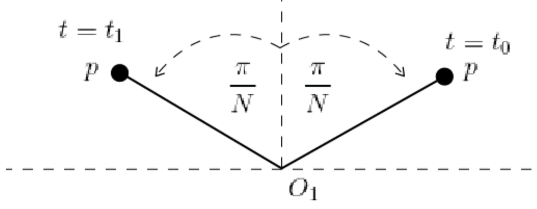


Figure 2.10: Inverted pendulum dynamics about O_1.

The angular momentum of p about O is written as:

$$ {}^{\mathcal{I}}\mathbf{h}_{p/O_1} = \mathbf{r}_{p/O_1} \times m_p^{\mathcal{I}}\mathbf{v}_{p/O_1} = m_p l^2 \dot{\theta}\mathbf{e}_z \tag{2.3.4} $$

Using Newton's second law for angular momentum and the moments from the FBD, we can calculate the dynamics:

$$ \begin{aligned} \frac{d}{dt}\left[{}^{\mathcal{I}}\mathbf{h}_{p/O_1}\right] &= \mathbf{M}_{p/O_1} \\ m_p l^2 \ddot{\theta}\mathbf{e}_z &= (\mathbf{r}_{p/O_1} \times -m_p g \mathbf{e}_y) + \tau \mathbf{e}_z \\ m_p l^2 \ddot{\theta}\mathbf{e}_z &= l\mathbf{e}_{r_1} \times -m_p g(\cos\theta \mathbf{e}_{r_1} - \sin\theta \mathbf{e}_{\theta_1}) + \tau\mathbf{e}_z \\ m_p l^2 \ddot{\theta}\mathbf{e}_z &= (m_p l g \sin\theta + \tau)\mathbf{e}_z \end{aligned} $$

$$ \ddot{\theta} = \frac{g}{l}\sin\theta + \frac{\tau}{m_p l^2} \tag{2.3.5} $$

From Eq. 2.3.5 we find that to maintain a constant angular speed, where $\ddot{\theta} = 0$, the motor supplied torque must equal $-m_p l g \sin\theta$ $(-\pi/N < \theta < \pi/N)$.

2.3.4 Pivot Leg Transition

During the stride phase, the COM traces out a circular arc around O_1 with a central angle of $2\pi/N$. After pivoting $2\pi/N$ radians, the next leg comes into contact with the ground and the current stride phase is over. At this moment the ground delivers an impulse, J_p, that drives the system into the next stride phase centered around a new pivot point, O_2. The displacement of O_2 relative to O_1 is determined by the stride length, which is equal to $2l\sin(\pi/N)$.

To calculate this impulse we examine the change in velocity of p. During the preceding stride phase from time $t_0 \to t_1$, Eq. 2.3.2 shows that p's velocity is entirely in $\mathbf{e}_{\theta_1}$. This is expected, given p's circular motion about O_1. After time t_1, p's velocity changes to be entirely in the $\mathbf{e}_{\theta_2}$ direction; it is tangential to the circular pivot about O_2. Using the definition for a change in linear momentum, we have:

$$ \mathbf{J}_p = m_p^{\mathcal{I}}\mathbf{v}_{p/O_2}(t_1) - m_p^{\mathcal{I}}\mathbf{v}_{p/O_2}(t_0) \tag{2.3.6} $$

Here ${}^{\mathcal{I}}\mathbf{v}_{p/O_2}(t_0)$ is equal to ${}^{\mathcal{I}}\mathbf{v}_{p/O_1}(t_0)$ as the relative velocity of O_1 and O_2 is 0 when both legs are touching the ground — which is when the transition occurs.

$$^{\mathcal{I}}\mathbf{v}_{p/O_2} = {}^{\mathcal{I}}\mathbf{v}_{O_1/O_2} + {}^{\mathcal{I}}\mathbf{v}_{p/O_1} = {}^{\mathcal{I}}\mathbf{v}_{p/O_1} \tag{2.3.7}$$

Substituting Eq. 2.3.2 for $\mathbf{v}$, we can solve for $\mathbf{J}_p$:

$$\begin{aligned}
\mathbf{J}_p &= m_p l\dot{\theta}_2(t_1)\mathbf{e}_{\theta_2} - m_p l\dot{\theta}_1(t_0)\mathbf{e}_{\theta_1} \\
&= m_p l\dot{\theta}_2(t_1)\mathbf{e}_{\theta_2} - m_p l\dot{\theta}_1(t_0)[-\sin(2\pi/N)\mathbf{e}_{r_2} + \cos(2\pi/N)\mathbf{e}_{\theta_2}] \\
&= m_p l[\dot{\theta}_2(t_1) - \dot{\theta}_1(t_0)\cos(2\pi/N)]\mathbf{e}_{\theta_2} + m_p l\dot{\theta}_1(t_0)\sin(2\pi/N)\mathbf{e}_{r_2}
\end{aligned} \tag{2.3.8}$$

If we assume that the legs are rigid and that there is conservation of momentum in $\mathbf{e}_{\theta_2}$, then $\dot{\theta}_2(t_1) = \dot{\theta}_1(t_0)\cos(2\pi/N)$. The impulse $\mathbf{J}_p$ is only in $\mathbf{e}_{r_2}$ with magnitude:

$$||\mathbf{J}_p|| = m_p l\dot{\theta}_1(t_0)\sin(2\pi/N) \tag{2.3.9}$$

As $2\pi/N \to 0$ for increasing numbers of legs, the linear impulse applied by the ground shrinks and the transfer of angular velocity becomes more complete. In the limit as $N \to \infty$, the leg mode becomes a standard wheel and rolls smoothly with $O_2 = O_1$ and $\dot{\theta}_2(t_1) = \dot{\theta}_1(t_0)$. Using fewer legs, however, results in greater impulse forces from each impact with the ground. The induced dynamic loads and stresses from these impulses may be problematic for weaker materials and overall hardware health.

2.3.5 Trajectory

Fig. 2.11 plots simulated trajectories of the COM of several different leg-wheel models with different numbers of legs. Each model is given a uniform unit length for r_e to isolate the effects of N. While it was shown that fewer legs give a large theoretical advantage in tasks such as obstacle climbing, we see now that having too few legs not only increases the impulses, but also results in inefficient trajectories. The inefficiency arises from the presumed practical inability to recover all the potential energy of a raised center of mass in another form of energy as the center of mass is lowered.

The COM of leg-wheels oscillates up and down significantly. As the number of legs increases, this vertical bouncing movement becomes less and less pronounced. The stride length of each step decreases, and the steps pick up in cadence. In the limit that the legs are dense enough to become a regular wheel, the trajectory of the COM is flat, and the number of steps per distance traveled is infinite.

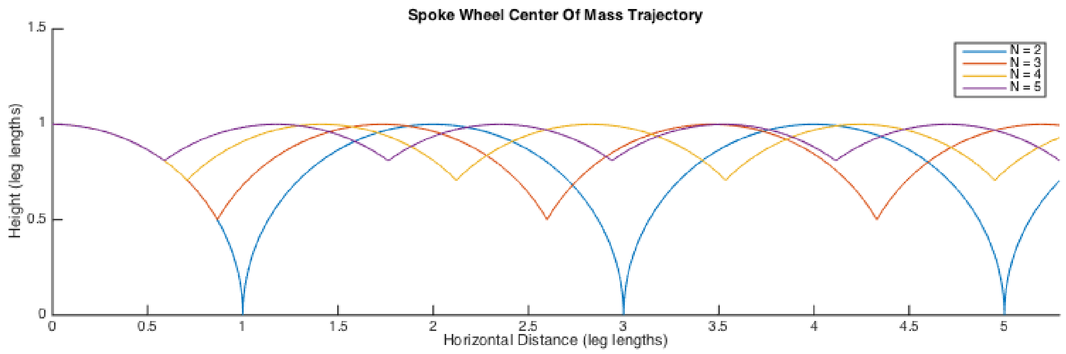


Figure 2.11: Trajectory of the wheel center of mass during leg mode operation.

2.4 Torque Requirements

In the derivation of the leg mode dynamics, we assumed that the extended legs formed a rigid body. This assumption would be valid if MARVIN's leg configuration were solid like the Whegs design, and the structure was fixed. However, in reality, the motors controlling the relative orientation of the inner and outer disks must supply internal torques to prevent the legs from collapsing. To get an estimate of the additional required torque, we treat the problem quasi-statically and balance moments about the inner and outer motor axels as the legs pivot.

Fig. 2.12 depicts the relevant coordinate systems and forces acting on the system for an arbitrary state during the pivot progression. We focus on the relative movement

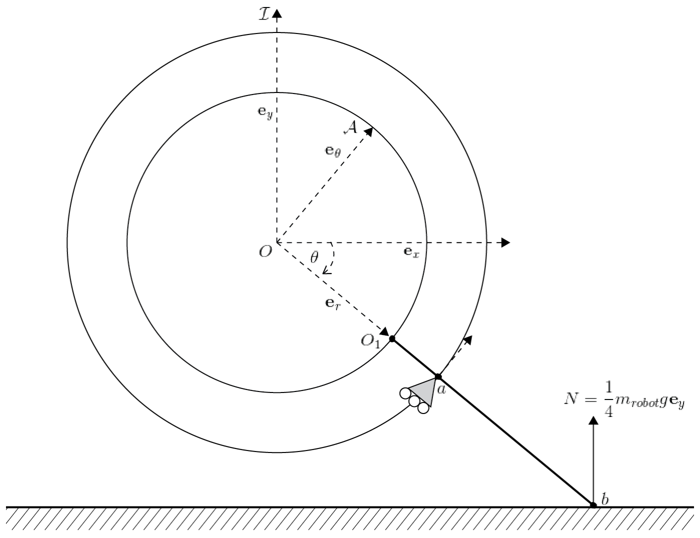


Figure 2.12: Quasi-static moment analysis on the inner axel.

of the two internal disks, and treat the outer disk as stationary with respect to the inner disk. Thus, the pin that rides in the leg slot is modeled as a roller support acting at point a. Additionally, we consider only the configuration where the leg is fully extended. The torque requirements do differ during the transformation period, or if the leg is intentionally only partially extended. This is a topic for further investigation, beyond the scope of this thesis. Here we present the results of the torque calculations for the inner and outer motors both quantitatively and qualitatively. The tedious specific steps of moment balance are included in slightly less detail.

2.4.1 Inner Motor

When a leg is in contact with the ground, it has to support one quarter of MARVIN's weight. This normal force, N, can be decomposed into components parallel and orthogonal to the leg coordinate system, $\mathcal{A}$. The orthogonal component is then leveraged about a, transmitting a reversed force magnified by $l_{\overline{ab}}/l_{\overline{O_1a}}$ at O_1. The force at O_1 then creates a moment about O:

$$\mathbf{M}_{O_1/O} = -\frac{l_e}{r_2 - r_1} N r_1 \cos\theta \mathbf{e}_z \tag{2.4.1}$$

The corresponding counter-torque, once again scaled by r_2, required from the inner motor, can be rewritten as:

$$\tau'_{inner} = \left(\frac{l'}{1-r'} - 1\right)\frac{m_{robot} g}{4} r' \cos\theta \tag{2.4.2}$$

By inspection, $|\tau'_{inner}|$ is maximized at the start and end at each pivot, where $\theta = -\pi/2 + \pi/N$ or $-\pi/2 - \pi/N$. Fig. 2.13(a) plots the maximum $|\tau'_{inner}|$ vs r' for a system of three legs with unit mass. We see from the graph, as well Eq. 2.4.4, that as $r' \to 1$ the torque requirement blows up.

2.4.2 Outer Motor

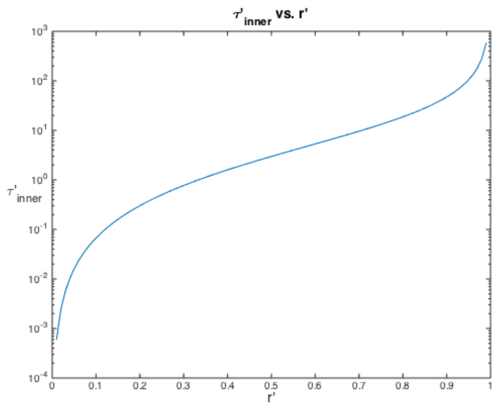
The outer motor has to supply the necessary support force acting at point a in $\mathbf{e}_\theta$, in addition to propulsive torque specified by Eq. 2.3.5 (and adapted to the current definition of θ). The total torque is:

$$\tau_{outer} = \frac{m_{robot}g}{4}\left[\left(\frac{l_e}{r_2 - r_1} - 1\right)r_1\cos\theta + (r_1 + l)\cos\theta\right] \tag{2.4.3}$$

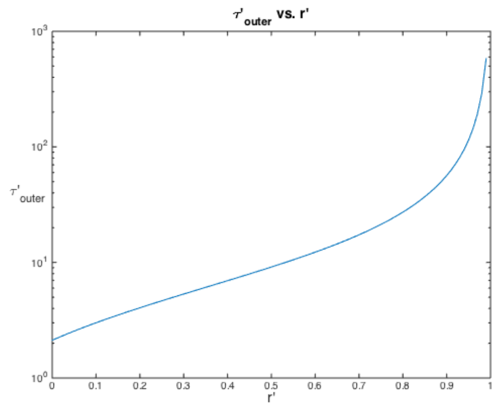
Dividing by r_2 and simplifying this becomes:

$$\begin{aligned}\tau'_{outer} &= \frac{m_{robot}g}{4}\left(r' + l' + \frac{r'l'}{1-r'}\right)\cos\theta \\ &= \frac{m_{robot}g}{4}\left(r' + \frac{l'}{1-r'}\right)\cos\theta \end{aligned} \tag{2.4.4}$$

Like $|\tau'_{inner}|$, $|\tau'_{outer}|$ is largest at the beginning and end of each pivot. Fig. 2.13(b) plots the magnitude of Eq. 2.4.4 vs. r' for three legs and unit robot mass, and exhibits similar singular torque requirements at $r' = 1$.



(a)



(b)

Figure 2.13: Maximal torque requirements on the inner motor (a) and outer motor (b) as a function of r'.

2.5 MARVIN Parameters

From the preceding analysis, it is clear that when choosing design parameters, there is distinct tradeoff between increased benefits from having more pronounced leg capabilities and the detrimental side effects that has on system stability and motor demands. Taking Figs. 2.3, 2.7, 2.11, and 2.13 into consideration, we identified a three-leg layout and $r' = 0.7$ as a desirable configuration for MARVIN.

In particular, MARVIN has radii of $r_1 = 3$ in and $r_2 = 4.25$ in, and its weight is 11.6 lbs. These values contribute to a slightly more aggressive design, and using four legs was a close runner-up decision. However, there is still an acceptable balance between longer leg lengths and improved climbing ability, while avoiding unrealistically large torque demands.

Chapter 3

Control System

MARVIN employs a mixture of automatic controls and user-inputs. Communication with the operator is established over a wireless connection with an Xbox controller. The user can specify the forward speed and turning rates, and choose between legs or wheels for the locomotion mode. At the same time, feedback laws in MARVIN's software track the given reference speeds and maintain the proper extensions of the legs according to the desired mode.

In order for MARVIN to achieve the desired speed and mode, the internal components of each wheel must be carefully and continuously controlled. The motors actuating the outer and inner disks must not only rotate at the correct speeds, but also maintain the proper phase difference for the current driving condition. In our design we chose to use classical proportional-integral-derivative (PID) controls to set motor speed and position. PID controllers are quite common in industry, and are useful for solving many control problems, from simple to complex [2].

The following sections cover derivations of the DC motor transfer functions, setup of feedback loops, and parameter tunings[1]. First the system is developed as a continuous model, and then it is discretized and implemented digitally on an Arduino microprocessor.

Due to the time constraints of the seven-month project, MARVIN currently only employs controls at a per-wheel level. Each wheel can be independently controlled, but inter-wheel phase differences are not calculated. As addressed in Section 7, we anticipate that the overall driving performance and stability in leg mode would be significantly enhanced by implementing system-wide gait control.

[1]Much of the following controls methodology was guided by homeworks, laboratory materials, and course notes [37] from Professor Clarence Rowley's MAE 433B Automatic Controls class.

3.1 DC Motor Model

3.1.1 Equation of Motion

To develop feedback laws, we first establish a dynamical model for the DC motors controlling the wheels. A simple equation of motion for a DC model can be written as [12, 37]:

$$\dot{\omega}_m + c_1\omega_m = c_2 v_a \tag{3.1.1}$$

Here, ω is the angular velocity of the motor shaft, v_a is the applied voltage, and c_1 and c_2 represent combinations of parameters specific to the motor-load subsystem such as the moment of inertia, efficiency, and coil resistance.

Certain adaptations of Eq. 3.1.1 are required for MARVIN. First, the motors used are housed in a gearbox with a gearing ratio of 156.8:1. Employing a change of variables from ω_m to $\omega_f = g_r\omega$, we see this simply has a scaling effect by a factor of the inverse gear ratio (g_r) on the constants in the equation of motion:

$$\dot{\omega}_f + \frac{1}{g_r}c_1\omega_f = \frac{1}{g_r}c_2 v_a(t) \tag{3.1.2}$$

Second, the motors used for MARVIN operate on a pulse width modification (PWM) signal instead of an analog voltage. A PWM signal regulates the speed of a motor by rapidly switching a full power supply, v_{source}, on and off. This eliminates the need for a DAC from the computer to the motor, and also minimizes power dissipation [24]. Varying the frequency, or duty cycle (D), of the PWM signal proportionally affects the average voltage applied (v_a') to the motor. This is related by the rule of thumb [49]:

$$v_a' \propto D v_{source} \tag{3.1.3}$$

In digital software implementation, the PWM signal D is constrained and linearly mapped to a signed int8 number space, $-255 \to 255$. Thus, without worrying about the meaning of specific constants, we simply absorb all these factors to rewrite the equation of motion in terms of final gearbox speed, applied PWM duty cycle, and new parameters c_1 and c_2:

$$\dot{\omega}_f + c_1\omega_f = c_2 D(t) \tag{3.1.4}$$

To solve for $\omega_f(t)$, we rescale the differential equation to be in dimensionless time units of $T = \frac{t}{\tau}$, where τ is selected to be $1/c_1$ and $c = c_2/c_1$.

$$\dot{\omega}_f + \omega_f = cD(T) \tag{3.1.5}$$

For a constant $D(T)$ such that $cD(T) = K$ and $\omega(0) = 0$, we find:

$$\omega_f(t) = K(1 - e^{-\frac{t}{\tau}}) \tag{3.1.6}$$

Appendix A explains how we determined the motor constants K and τ from experimental data. We estimated the parameters for the outer disk/motor system to be $\tau = K = 0.07$, and similarly $\tau = K = 0.09$ for the inner disk/motor.

3.1.2 Transfer Function

To find the plant transfer function, we return to the original, unscaled ODE in Eq. 3.1.4 and substitute τ and K divided by the PWM signal value used in the experiment, $K' = K/\mathrm{D_e}$.

$$\dot{\omega}_f + \frac{1}{\tau}\omega_f = K'D(t) \tag{3.1.7}$$

Taking the Laplace transform gives us the transfer function, as supported by [**?**]:

$$H(s) = \frac{W(s)}{U(s)} = \frac{C}{\tau s + 1} \tag{3.1.8}$$

Here we have substituted $C = K'\tau$.

3.2 Controller Design

3.2.1 Network Level Control

To operate the wheel-leg system as a whole, the motors controlling the two independent, internal disks must know how to cooperate. In order to monitor both angular speed and relative position, we chose to split each wheel into a master and slave follower subsystem, as illustrated in Fig. 3.1.

The outer motor is given reference commands for ω and determines the overall speed of the system. Meanwhile, the inner motor's input is latched onto the angular

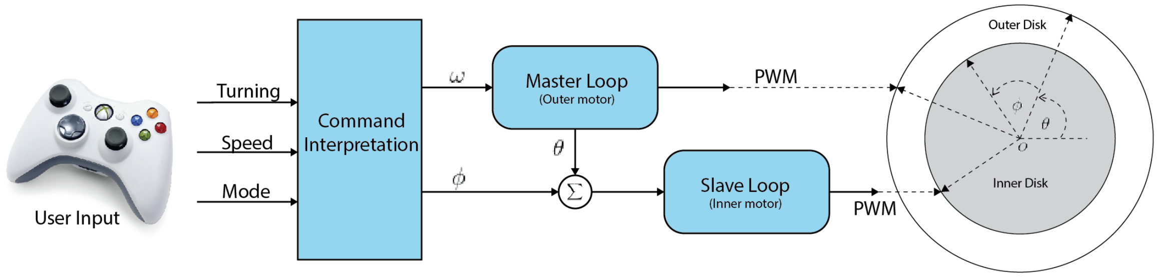


Figure 3.1: High level representation of the Master-Slave control network.

position reading, θ, of the outer motor. When the system operates in wheel mode the inner motor tracks the outer motor's position, and the resulting zero phase difference holds the legs in. Switching to leg mode simply involves the addition of a constant angular offset ϕ to the phase difference, and the inner motor adjusts to extend the legs. Finally, differential drive steering is accomplished by sending different ω references to the right and left wheels, depending on the desired turning direction and radius.

3.2.2 Motor Level Control

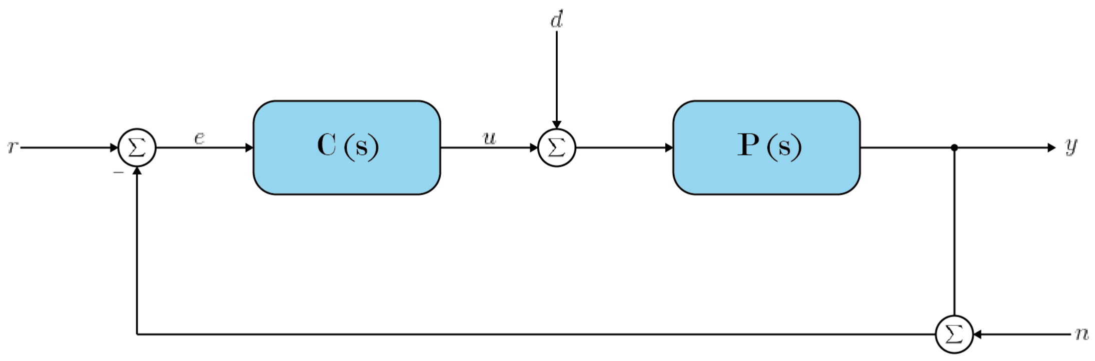


Figure 3.2: Feedback Loop

Fig. 3.2 depicts a standard controller-plant feedback system. $P(s)$ represents the plant transfer function of the DC motor (Eq. 3.1.8), while $C(s)$ represents the loop controller. Here we are interested in developing $C(s)$ such that we are able to robustly and quickly influence the motor output y, and achieve good tracking of the reference speed or angle, r.

For MARVIN's purposes, good performance is characterized by a fast and stable response from $r \rightarrow y$, in addition to reliable disturbance rejection. The tracking input for the motors is low-frequency, however periodic disturbances (d) will enter the system through various sources. Primarily, the motor will have to react to an increased demand for torque every time one of the legs hits the ground. When in leg mode, the motors are not anticipated to be driven higher than 70 rpm. At that speed, with three legs per wheel, ground collisions will occur at a frequency of 3.5 Hz. Additionally, aspects like non-uniform internal friction, changes in terrain, and changes in components will all contribute to time-varying demands on the motor.

However, while the feedback system must have good tracking and disturbance rejection, it also must be careful not to magnify noise. The output of the plant (y) is measured with real sensors. Thus, it is possible that some high frequency, random, low-amplitude sensor noise (n) can be introduced into the loop. Finally, the system must be stable, with acceptable gain and phase margins. Various sources [2, 37] view reasonable stability requirements as having a phase margin $\geq 30-60°$ and a gain margin $2-5$, or higher. These features motivate the following specifications for the feedback loop:

1. There must be good reference tracking at low frequencies, with a bandwidth at least on the order of, or greater, than 3.5 Hz.
2. The controller must be a low-pass filter, with good noise attenuation at high frequencies.
3. There must be zero steady-state error for both reference tracking of ω in the master controller and θ in the slave controller.
4. The loop must have good stability margins, with a phase margin of at least 60° and a gain margin greater than 2.

To meet our performance requirements we look at both the sensitivity function for reference tracking (t.f. from $r \rightarrow e$) and the complementary sensitivity function for noise magnification (t.f. from $n \rightarrow y$). These are written in terms of the loop gain, $L = PC$, as:

$$\text{Sensitivity Function} = \frac{1}{1+L} \tag{3.2.1}$$

$$\text{Complementary Sensitivity Function} = \frac{L}{1+L} \tag{3.2.2}$$

For satisfactory reference tracking error, the sensitivity function should be small for the appropriate frequency range. This means that the larger L is for frequencies below 3.5 Hz, the better the performance. Meanwhile, for noise attenuation at high frequencies, the complementary sensitivity function should be small. Thus, while L should be large at low frequencies, it must start to roll off sharply for frequencies above the bandwidth.

Outer Motor Controller

As the master component controlling the overall speed, the outer motor needs to track a constant angular speed reference, ω. For this situation we chose to use PI control:

$$C(s) = k_p + \frac{k_p}{s} \tag{3.2.3}$$

This type of control is guaranteed to produce zero steady-state error for a constant input by the Final Value Theorem [37]. For a system $y(t)$ with only left-half plane poles, the Final Value Theorem states:

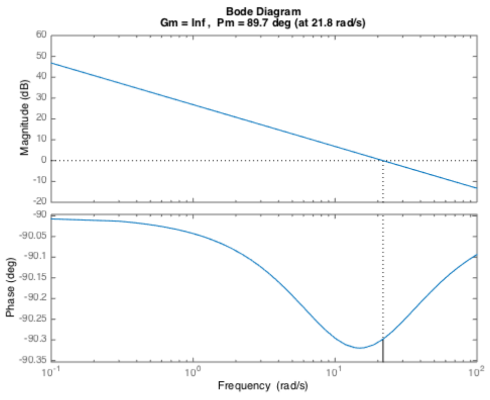
$$\lim_{t\to\infty} y(t) = \lim_{s\to 0} sY(s) \tag{3.2.4}$$

Applying the theorem to the sensitivity function, we see that the error, $e(t)$, goes to 0 for a step input ω_r:

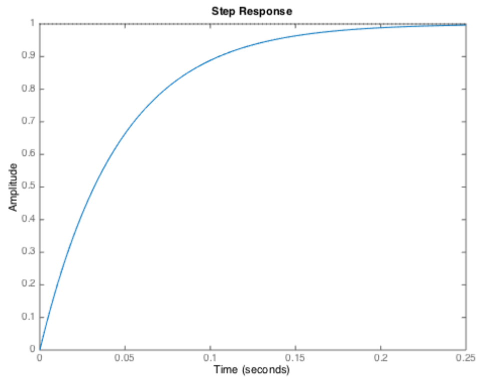
$$\begin{aligned}
E(s) &= \frac{1}{1+P(s)C(s)}U(s) && (3.2.5)\\
\lim_{t\to\infty} e(t) &= \lim_{s\to 0}(s)\left(\frac{1}{1+\frac{k_p s+k_i}{s}\frac{C}{\tau s+1}}\right)\left(\frac{\omega_r}{s}\right) && (3.2.6)\\
&= \lim_{s\to 0}\frac{\omega_r s(\tau s+1)}{\tau s^2+(Ck_p+1)s+Ck_i} && (3.2.7)\\
&= 0 && (3.2.8)
\end{aligned}$$

After loop-shaping, we determined a set of satisfactory gain parameters to be $k_p = 20$ and $k_i = 300$. This selection of k_p and k_i places a closed-loop pole near the moderately slow plant zero of $1/\tau$, which improves the bandwidth of the system. Fig. 3.3(a) shows the Bode plot of the open-loop transfer function L. We see that it has good stability margins with an infinite gain margin and $\phi_m = 89.7°$. Additionally, it has a bandwidth of approximately 21 rad/s. At high frequencies, the magnitude of L rolls off.

Fig. 3.3(b) shows a simulated step response. From this graph we see that the system has a quick rise time, no overshoot, and a settling time less than 0.25 seconds.



(a)



(b)

Figure 3.3: Bode plot (a) and a step response (b) for the PI ω controller.

Inner Motor Controller

As the slave, the inner motor follows the position of the outer motor. Since the input is now in terms of θ rather than ω, the motor plant transfer function must be integrated:

$$P(s) = \frac{C}{s(\tau s + 1)} \tag{3.2.9}$$

For a constant outer motor speed, the reference input to the inner motor will be a ramp function of the form $r(t) = \omega t$. Here, we chose to use full PID control, with the modification that we used low-pass filtered derivative control to avoid instabilities at high noise frequencies:

$$C(s) = k_p + \frac{k_i}{s} + \frac{k_d s}{s/50 + 1} \tag{3.2.10}$$

Once again, we use the Final Value Theorem to show that this controller achieves zero steady-state error for a ramp input with slope ω_r:

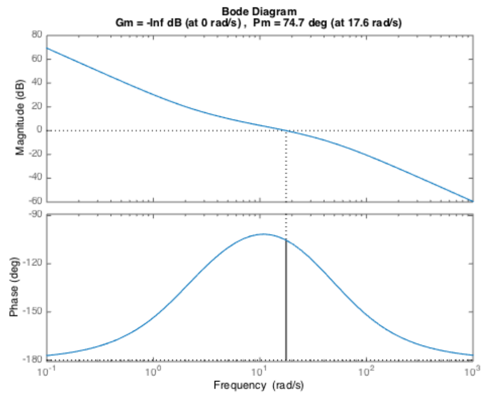
$$E(s) = \frac{1}{1 + P(s)C(s)} U(s) \tag{3.2.11}$$

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}(s)\left(\frac{1}{1 + \frac{k_p(s^2/50+s)+k_i(s/50+1)+k_d s^2}{s^2/50+s}\frac{C}{\tau s^2+s}}\right)\left(\frac{\omega_r}{s^2}\right) \tag{3.2.12}$$

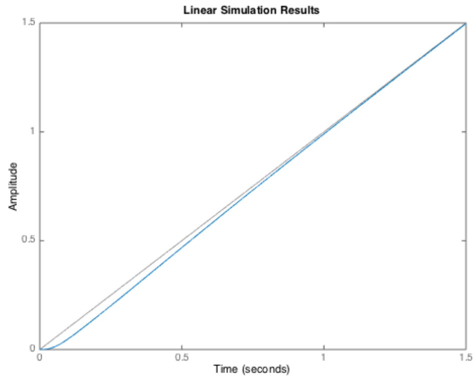
$$= \lim_{s\to 0} \frac{\omega_r s(s/50+1)(\tau s+1)}{\frac{\tau s^4}{50} + \frac{50\tau+1}{50}s^3 + (Ck_d + \frac{Ck_p}{50} + 1)s^2 + (Ck_p + \frac{Ck_i}{50})s + Ck_i} \tag{3.2.13}$$

$$= 0 \tag{3.2.14}$$

We designed our gains to be $k_p = 225$, $k_i = 400$, and $k_d = 15$. As for the outer motor controller, this set of parameters helpfully places a closed-loop pole near the slow motor plant zero. Figs. 3.4 (a) and (b) show the open-loop bode plot and closed-loop ramp response, respectively, for the system. The phase margin is stable at 74.7°, and the gain margin is infinite. Finally, the open-loop gain rolls off as desired for high frequencies. In (b) we see that the simulated system has a swift rise time, and attains close tracking in about a second.



(a)



(b)

Figure 3.4: Bode plot (a) and a linear input response (b) for the PID θ controller.

3.3 Real Systems Implementation

The preceding control laws were developed for continuous, linear systems. When transitioning from simulation to implementation on MARVIN's real components, other side effects are introduced that must be taken into consideration.

3.3.1 Motor Saturation and Integrator Anti-Windup

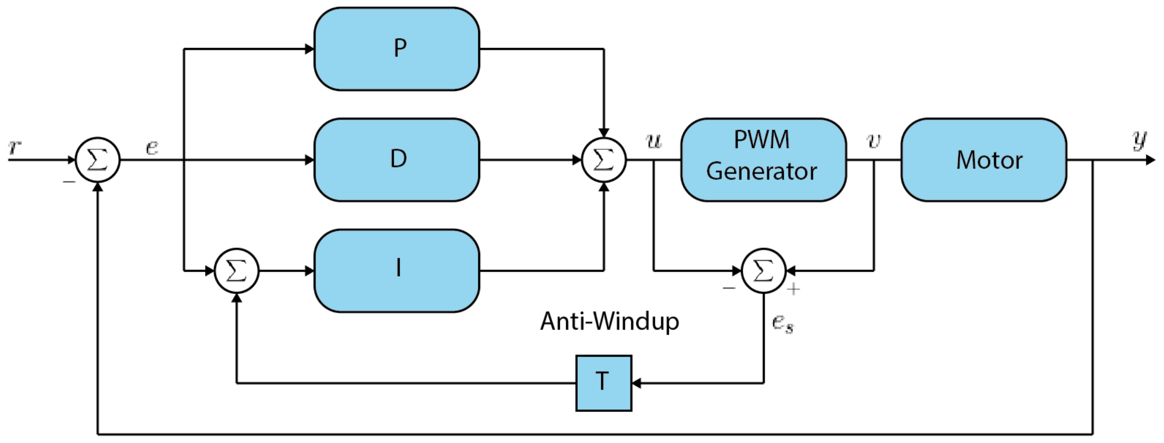


Figure 3.5: Feedback Loop

Both the inner and outer motor controllers assume that the plant can take an unbounded control input u. In reality, the PWM signal that is generated in the real system saturates. The viable operating range driven by the signal is between $-V_s$ and $+V_s$. When the control demand exceeds these limits, the feedback loop is broken, and the error will begin to integrate to unbounded values. When the system recovers, the accumulated error during the saturation period will cause a large overshoot. This undesirable behavior is called windup [2].

For MARVIN, integrator windup is an issue — as it is not particularly unusual for one wheel to stall until another wheel finds traction when attempting to traverse a tall obstacle. In order to alleviate windup effects, we use back-calculation and tracking to reset the integrator when the control output saturates, as shown in Fig. 3.5. The input command u is subtracted from the output, v, of the controller and fed back with a gain, T, to curb the integrating error. Since the actual PWM saturation limits of our controller are determined in the software, this is a straightforward adjustment to make. We chose T to be $1/k_p$, as suggested by [27]. If the controller is not saturated this has no effect on the loop. Of the many existing anti-windup techniques, this approach has been proven to be widely effective and does not exhibit detrimental artifacts, such as output chatter [33, 3, 27].

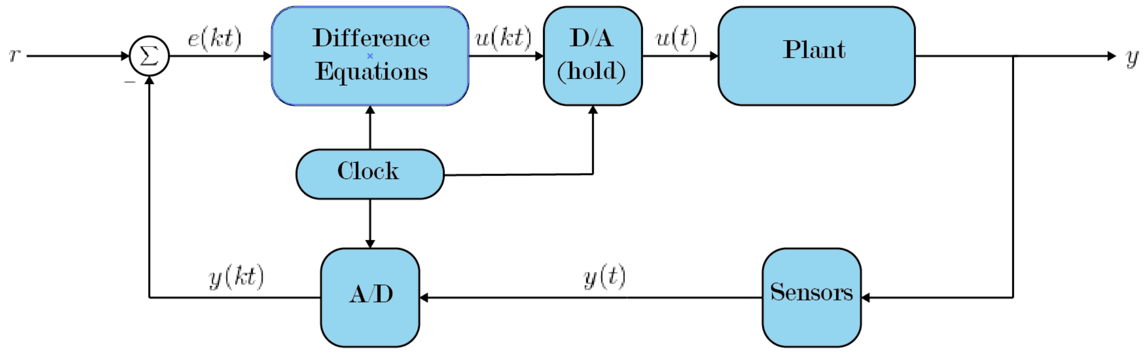


Figure 3.6: Digital feedback loop [16].

3.3.2 Digital Control

In order to implement MARVIN's controls on an Arduino, we converted the laws to operate on discrete signals, rather than continuous inputs. The structure of a digital control loop is shown in Fig. 3.6. The fundamental difference is that the computer uses a clock to sample the sensors and update the control effort at regular intervals. During the delay between loops the output is simply held constant — a behavior termed zero order hold.

Furthermore, the digital system has to approximate the transfer function for the controller with a set of difference equations [16]. We experimented with using Matlab's c2d and ss functions as well as Euler approximations to make this conversion. Matlab's discrete-time state-space form gives the following update relationship between our digital error input $e(kt)$ and control output $u(kt)$:

$$x_{n+1} = A_d x_n + B_d e_n \tag{3.3.1}$$
$$u_n = C x_n + D e_n \tag{3.3.2}$$

Alternatively, Euler's method [16, 2], estimates a continuous function $\dot{x}(t)$ as:

$$\dot{x}(t_k) \approx \frac{x(k+1) - x(k)}{T} \tag{3.3.3}$$

Where T is the sampling interval, $t_k = kT$, and k is an integer. By applying reverse Laplace transforms to the transfer functions, and then estimating the resulting differential equation with Euler's method, you can obtain a set of difference equations. While a rather primitive approach to discretization, it gave good empirical results.

The delay in a digital controller can have adverse effects if it is too large relative to the time scale of the system it is trying to control. The Arduino running MARVIN's software samples at 200 Hz, updating the states every 5 milliseconds. This is an order of magnitude faster than our system dynamics. Matlab simulations of step and ramp responses for discrete-time conversions of our controllers reveal little to no change in performance. As shown in Fig. 3.7, acceptable results were obtained for the real controllers. The system is slightly slow, but it is clear that the two signals move from being synchronized, to out of phase, and then back to synchrony.

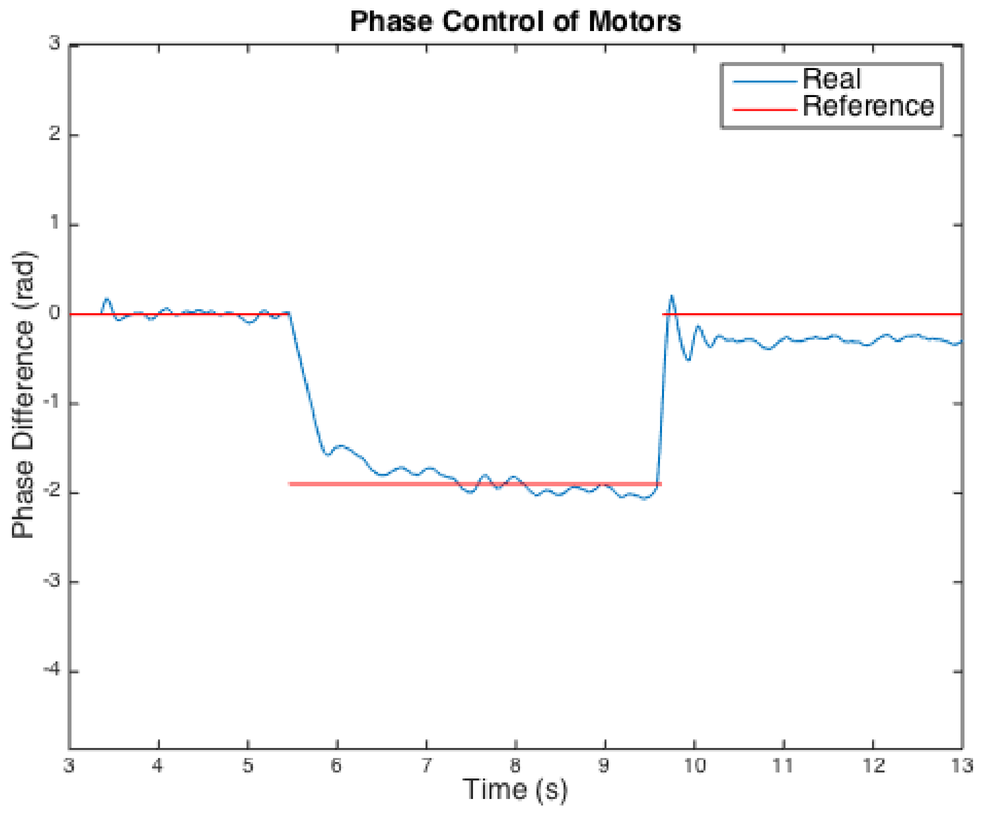


Figure 3.7: Experimental data of motor phase during a mode switch. From $t = 4 \rightarrow 6s$ the system is in wheel mode, from $t = 6 \rightarrow 10s$ it is using legs, and then it returns to wheel mode for $t > 10s$.

Chapter 4

Electronics

4.1 Wireless Communication

MARVIN's human-in-the-loop control system is accomplished through an off-the-shelf Xbox 360 wireless gaming controller and an Xbox 360 wireless gaming receiver for Windows[1]. The receiver is connected to the Arduino Mega by USB connection through the USB Host Shield and mounted on to the chassis of the robot. The controller communicates to the receiver at 2.4GHz through radio transmitters. The controller has a range of up to 30 feet. Upon applying power, the wireless controller must be synched with the receiver. For first time connections, the manual synchronization process can be applied, however for recurring connections, the receiver remembers the controller and the two devices synch automatically.

The Arduino Mega translates commands sent to it by the wireless controller by using an open-source package for USB Host Shield connections. In our case, we made use of the XBOXRECV and USB libraries for establishing a connection and interpretting commands.

4.2 Power Analysis

4.2.1 Loads

1. **Motors:** MARVIN uses 2-wire 393 Motor from Vex Robotics[2]. These motors were

[1] http://support.xbox.com/en-US/xbox-on-other-devices/windows/xbox-360-wireless-gaming-receiver-windows

[2] http://www.vexrobotics.com/276-2177.html

rated at a .37A free-current current draw and a 4.8A current draw at stall current, at 7.2V. In practice, however, we noticed that the current draw from the motors never exceeded 4A despite being applied at a sufficiently high torque, possibly due to the batteries being charged beyond their 7.2V rating. Motors were powered by applying nominal power to the Motor Controller 29[3], which has a max current draw of 4A at 7.2V. Thus the maximum power draw from each motor is 28.8 W, which extends to a maximum power draw from the entire motor system to be 230.4 W.

2. **Arduino:** Throughout testing, the Arduino Mega was powered through two separate sources: remote laptop computer through USB connection and external power supply through Vin pin. The microcontroller has a nominal input power supply of 7-12 V but operates at a voltage of 5V through an on-board voltage regulator[4]. The DC current draw of the Arduino depends on the number of I/O pins in use, with each pin supplying a maximum of 40 mA. However, the maximum current that the device can put out is 500mA. Calculating based on maximum current output assumption, the Arduino draws 2.5W during normal operation.

3. **Encoders:** Initially, our motor encoders were supplied power by the 5V line from the Arduino, however we quickly realized that they were drawing too much current for the microcontroller to handle. We instead set up an independent 5V rail through the output of an LM7805c Voltage Regulator, powered by an independent 9.6V, 2000mAh battery. While testing, this battery only powered the encoders, which were drawing 300mA at 5V, or 1.5W. However, at times when data was not being collected, we also powered the Arduino through the 9.6V line.

4.2.2 Batteries

External power was chosen based off of the power requirements of the major subsystems on board the robot. As mentioned above, the three main systems that needed power management were the motor system, the Arduino, and the encoders. Because the Arduino is known to "brown out" at low input voltage levels, we made the decision to separate the power sources for the three sub-systems so that high loads for the motors would not cause erratic behavior for the robot. Both sets of batteries were charged using a standard Vex Smart Charger with a Tamiya Connector[5].

[3]http://www.vexrobotics.com/276-2193.html
[4]http://arduino.cc/en/Main/arduinoBoardMega2560
[5]http://www.vexrobotics.com/smart-charger-v2.html

The batteries selected for our Vex motors were Vex 7.2V NiMH 3000 mAh Robot Battery[6]. These batteries were chosen due to their known compatibility with our Vex motors as well as their high capacity and relatively flat discharge curve. Load discharge curves tests shown in Fig. 4.1 were conducted by Vex Robotics, and show the rate of voltage decay at loads of 12A, 8A, and 4A. Based on these graphs, we chose to split the power supply for the eight motors into two independent batteries (with four motors per battery). This allowed for a more tractable operating time for our maximum power ratings.

Empirical tests showed that the current draw from an individual motor did not exceed 4A, thus at continuous full power, each battery would output 16A of current, resulting in a maximum lifetime of around 10 minutes. However the average current draw per motor for wheeled mode is approximately 1.05A, or 4.2A of required power output per battery. The maximum lifetime in continuous wheeled mode operation is thus 43 minutes. The battery life of the robot that intelligently switches between both modes would will fall somewhere between these two bounds.

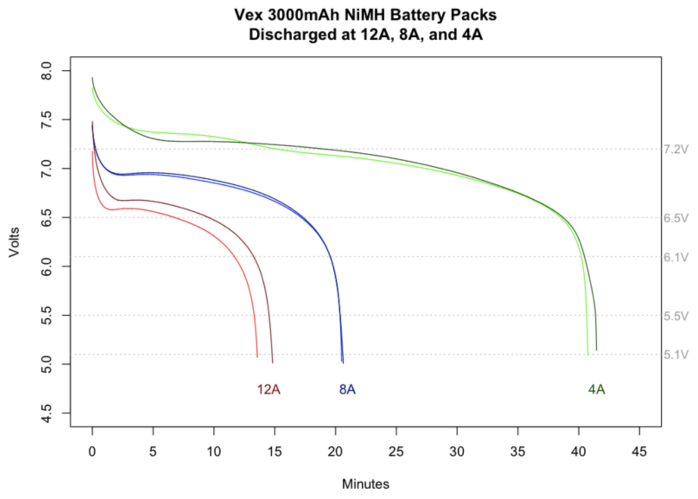


Figure 4.1: VEX Robotics battery load curves

As already mentioned, the Arduino requires much less power to operate, however we did not wish to contaminate its power source with the large power draw of the motor system. In order to ensure a long running time for the Arduino, we supplied it with a 9.6V 2000mAh NiMH Battery. The Arduino draws a maximum and fairly constant power draw of approximately 2.5W. Since we transitioned to also powering the encoders from this battery, an extra 1.5W was being drawn. At a maximum

[6]http://www.vexrobotics.com/wiki/7.2v_NiMH_3000mAh_Robot_Battery

capacity of 19.2 Wh, this battery can power the Arduino Mega and encoders for a minimum of 4.8 hours of continuous operation.

4.3 Motors

Motors were sized based on estimated torque requirements as well as other factors such as weight, price, and power consumption. Table 4.1 shows a comparison of the different motors we considered. Ultimately, we chose the Vex 2-wire 393 Motor due to its low price, power draw, and weight combination while maintaining a sufficient torque rating.

Motor	**Weight (kg)**	**Torque (oz-in)**	**Free Speed (rpm)**	**Cost ($)**	**Power (W)**	**Stall Curr. (A)**
AndyMark (0912)	.227	61	16000	14	179	64
AnyMark (0915)	.726	1209	198	69	45	22
BaneBots (M7-RS775-18)	.337	166	17040	17.50	273	130
CIM Motor	1.27	343	5310	28	337	133
Maxon DC Motor	Cust.	Cust.	Cust.	100-600	Cust.	Cust.
Vex 2-wire 393	.087	215	100	15	4	4

Table 4.1: Comparison of several commercial DC motor models.

Due to time constraints, we made an initial estimate for the required torque for each motor to be .61 $N \cdot m$, and chose motors based on this value. Upon more in-depth analysis, we found the required torque for motors in our original design to be quite higher, however by adapting the wheel design slightly, we were able to successfully operate the robot with the acquired Vex motors. Future designs would perform better by incorporating a motor with higher torque ratings.

The motor controller we chose for our motors was the Vex Motor Controller 29, the standard motor controller for Vex motors. The motor controller requires a maximum input voltage of 8.5V and an input PWM signal which it relays as an oscillating signal from 0V to Input voltage to the motor. The PWM input takes a 1-2ms duty cycle with 1.5ms being neutral or stand-still speed. Through empirical testing, we noticed that there is a “dead zone” around 1.5ms for which the motor will not turn due to a small effective voltage. This dead zone occurs around 1.5 $\pm$ .06 ms. The input PWM signal is supplied by the Arduino Mega.

In order to provide accurate closed-loop control for our motors, we implemented a feedback system for position and speed of our motors. This functionality was accomplished with the Motor 393 Integrated Encoder Module[7]. This encoder module was easily compatible with the Vex Motor 393, and acted as a simple attachment to the back of the motor module. The encoder module had two power ports, powered by a 5V input signal, and two digital communication ports using I2C. The encoders were rated at measuring ~ 630 ticks per revolution of the output shaft.

Communication to the Arduino was done through I2C protocol. The encoder defaulted to an address of 0x60 and is terminated upon power-up. In order to connect several devices to the same I2C port, we connected the encoders through a process known as "daisy-chaining," illustrated in Fig. 4.2. The encoder module itself allowed for an input and output set of ports to simplify the daisy-chaining process. In order for communication from the Arduino to make it past the first encoder module, each encoder in the sequence needed to be unterminated through software, to allow for the I2C clock and data lines to talk to devices down the chain. Finally, the last encoder in the chain needed to be terminated. The daisy-chaining process allowed for communicating with up to 8 devices and the order of the devices was represented by the order they were "attached" in the Arduino code.

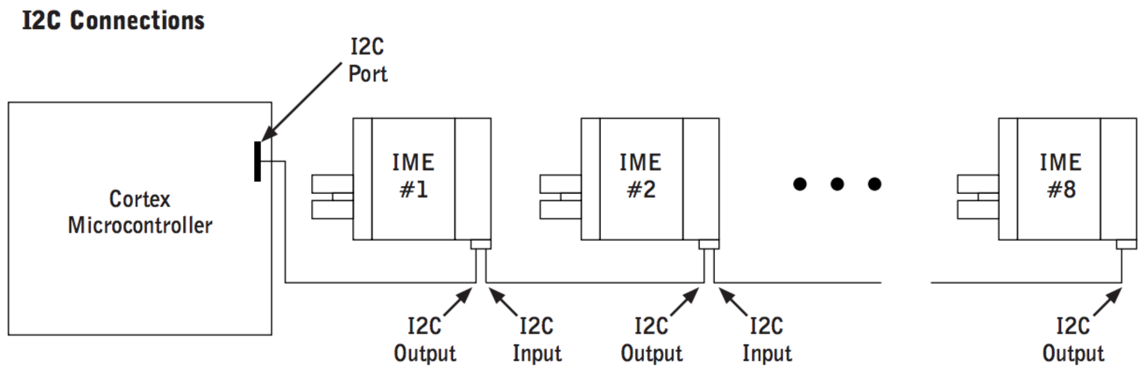


Figure 4.2: Daisy chained encoder modules on the I2C line (courtesy of VEX[7]).

4.4 Arduino Microprocessor

We used the Arduino Mega 2560 as the primary controller for MARVIN [9]. The Mega contains an ATmega2560 processor with 54 digital I/O pins, 16 analog inputs,

[7]http://www.vexrobotics.com/wiki/index.php/Intergrated_Motor_Encoders

and a USB connection and runs at a clock speed of 16 MHz. We powered the Arduino using an independent power supply (see Power-Batteries) as well as through a remote laptop during testing for Serial data collection. All of the control theory done for coordinating movement amongst the motors was programmed in Arduino language, a C-based coding language, and in the Arduino software environment. A variety of libraries were imported from open-source Arduino packages that can be seen in the appendix code section. On top of this code, we developed our own C-library in order to increase modularity in our design of the control system. The Host Shield was used as an attachment to the Mega Board in order to communicate with the wireless Xbox controller [10]. The Host Shield is based on the MAX3421E[8] and communicated to the mainframe board using the SPI bus. As the main controller for our entire system, the Arduino dealt with a variety of signals in controlling the sub-systems onboard.

- PWM: 8 of the 54 digital I/O pins were used to send a PWM signal to all 8 of the motor controllers used to control the speed of the motors.
- I2C: SDA and SCL ports were used to implement I2C protocol for communicating with the chain of motor encoders on board the robot. Each device was assigned an address by the Arduino and the data and clock lines could communicate to any of the 8 devices assuming they were un-terminated through software.
- USB: The USB Host Shield was used to connect to the wireless gaming receiver to communicate user-input controls.
- Analog: Several sensors on board the robot used analog signals to communicate data about stability and current/voltage levels of the batteries. These signals were read through analog I/O pins and converted to digital values by an internal ADC.

4.5 Sensors

To measure robot stability, we used an onboard SparkFun Triple Axis Accelerometer [13]. The accelerometer measured the variance in the z-axis in both wheeled and legged mode on different types of terrains. The accelerometer is powered by the 3.3V line from the Arduino, but operates at an extremely low power (drawing only 350 uA). More information can be seen in Section 5.

[8]http://www.maximintegrated.com/en/products/interface/controllers-expanders/MAX3421E.html

Our motivation for this study was rooted in the optimization of robotic performance over various types of terrain. One of the factors of this optimization is power consumption. As stated earlier, we assumed that wheels consume less power than legs. In order to validate and test the extent of this assumption, we used an AttoPilot Current/Voltage Sensor Breakout Board to test the current draw and voltage decay of our batteries as the robot performed in wheeled and legged mode [14]. Due to the high current draw from our motors, we chose a sensor rated up to 45A. Voltage levels are determined by the voltage drop across a pair of parallel shunt resistors and then converted to an analog 3.3V scale. Results of the tests using this sensor can be seen in the Section 5.

Chapter 5

Experimental Procedure

5.1 Methods

The theoretical understanding of the trade-offs between wheels and legs presented in the preceding chapters, however extensive, remains insufficient to determine an optimized design. What is needed is field test data. In order for MARVIN to make informed cost-based path decisions, we must have a quantitative sense of the actual toll different terrains have on MARVIN in the real world. Thus, we examined the effects of hybridization on MARVIN's performance through a set of systematic experiments.

As the advantages of multimodal locomotion are a function of the type of terrain being traversed, we made sure to test on a wide variety of surfaces as part of our experimental procedure. Both quantitative and qualitative results were recorded on natural terrain, as well as an artificially constructed testbed in which we could control the roughness of the landscape. Some examples of our natural terrain test field are depicted in Fig. 5.1. In total, we conducted trials on gravel, sand, tall grass, curbs, steps, and clumps of leaves. Our artificial testing arena is shown in Fig. 5.2.

In terms of collected data, quantitative metrics included real-time data of total current draw and stability. We also measured average speed. Qualitatively, we observed less measurable nuances in how the different modes handled each type of terrain. At the most basic level this entailed noting whether the robot failed or succeeded; at a more specific level we paid attention to overall perceived wear and tear or ease of maneuverability.

Current draw was measured on each of the 7.2V Vex Motors onboard by a battery

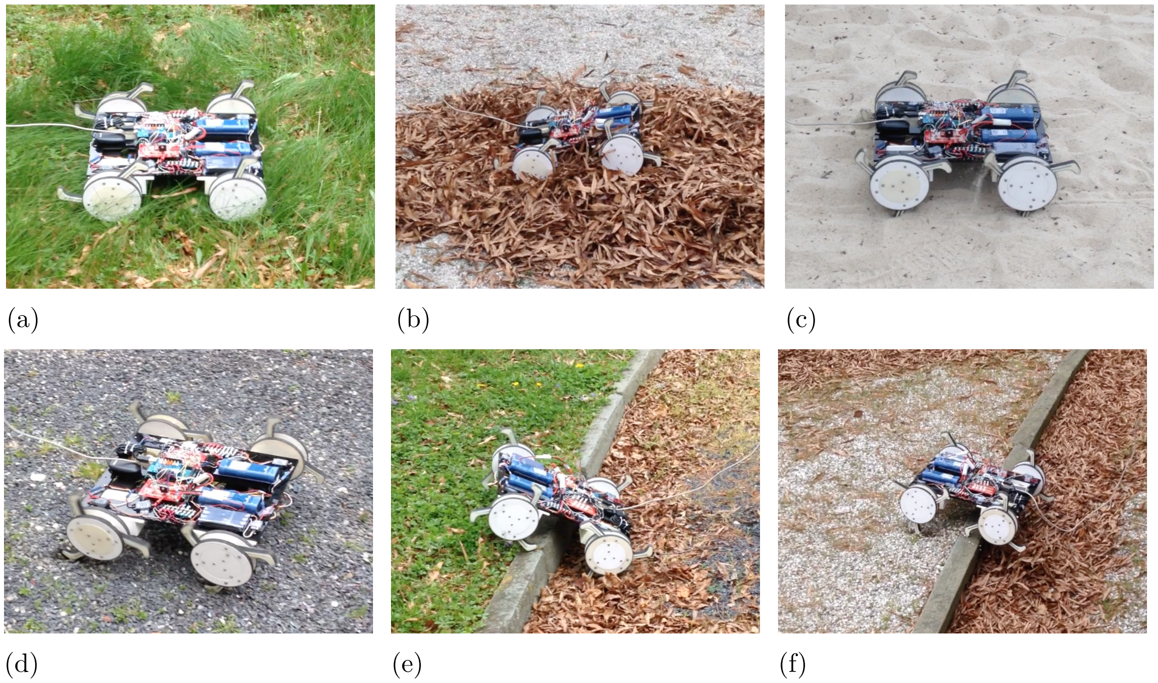


Figure 5.1: Examples of natural terrains used for MARVIN's performance tests.

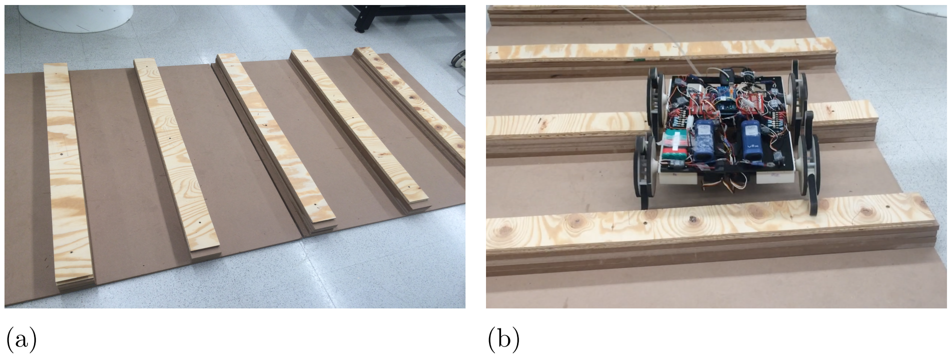


Figure 5.2: Artificial test field for controlled experimentation with MARVIN.

current and voltage sensor. Total current draw was monitored by taking the sum of the current in the right and left batteries. We discounted the current draw from the battery powering the encoders, as this is a steady power draw independent of the motor effort and terrain roughness. A metric for stability was obtained with a calibrated onboard three-axis accelerometer placed near the robot's center of mass. For our purposes, as discussed in Chapter 4, we defined stability as the variance of the acceleration on the z-axis. Performance-wise, this corresponds to any noticeable, vertical oscillations in the robot center of mass. Finally, average speed was taken from stopwatch times over a pre-measured distance.

Current draw and stability measurements were collected through the USB/Serial

Port of the Arduino Mega and fed into a Matlab script that read the information coming through the Serial line. The baud rate was set to 9600 for the Serial connection, however current and accelerometer data was sampled on the order of at 100 Hz due to precautions over not straining shared time resources with the control law software.

5.2 Controlled Environment

In order to study how wheels perform relative to legs as a function of terrain roughness, we constructed an artificial testing environment out of MDF. As shown in Fig. 5.3, the testbed consisted of periodic hurdles of adjustable height. These step heights were increased by increments of 0.5 inches, and were placed 1.5 feet apart evenly along the length of the 8-foot long pathway. In order to better standardize the type of terrain being traversed for each step height, the RMS height roughness was calculated for each step height.

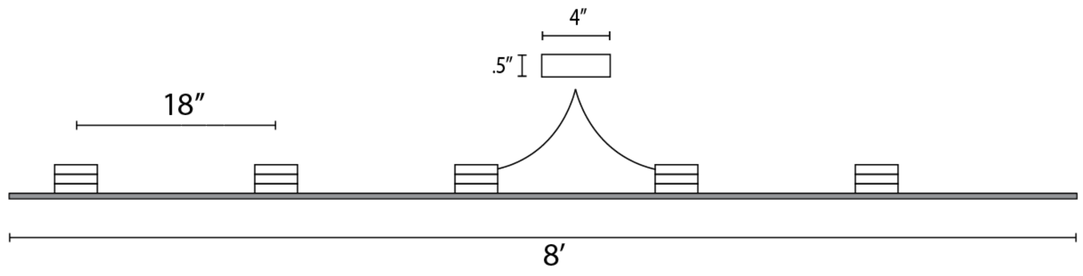


Figure 5.3: Artificial test bed schematic.

This metric is given by the Eq. 5.2.1, which computes the root mean square of the terrain elevation variance [11]:

$$R_{RMS} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{mean} - y_i)^2} \tag{5.2.1}$$

Here y_{mean} is the average ground height, and y_i is the elevation of data point i. We computed R_{RMS} by discretizing our testbed into inch long segments, such that the i^{th} point corresponds to the i^{th} inch along the course.

Each height to roughness value was traversed by both wheels and legs in order to compare data between the two modes. For the last two RMS roughness values, the wheels were unable to traverse the terrain at all — thus their power consumption, stability, and speed data were not counted and left as 0.

Due to time constraints, total data collection was slightly limited. Nevertheless, each mode was tested at least 3 times per height-roughness value (6 trials per testbed setup). Furthermore, given the nature of the experiment, the typical number of test runs is generally lower anyway in the robotics community. For example, our data volume is consistent with the order of RHex, where 10-20 samples were recorded per setup [39].

5.3 Data

5.3.1 Artificial Terrain

Here we present graphs of average current draw, average z-axis accelerations, and average speeds versus terrain roughness, as recorded over all trials of the experiments carried out on the artificial testbed. Error bars are included as well, to reflect a significant amount of uncertainty due to the limited number of total test runs. A discussion and interpretation of the data is left to Section 5.4.

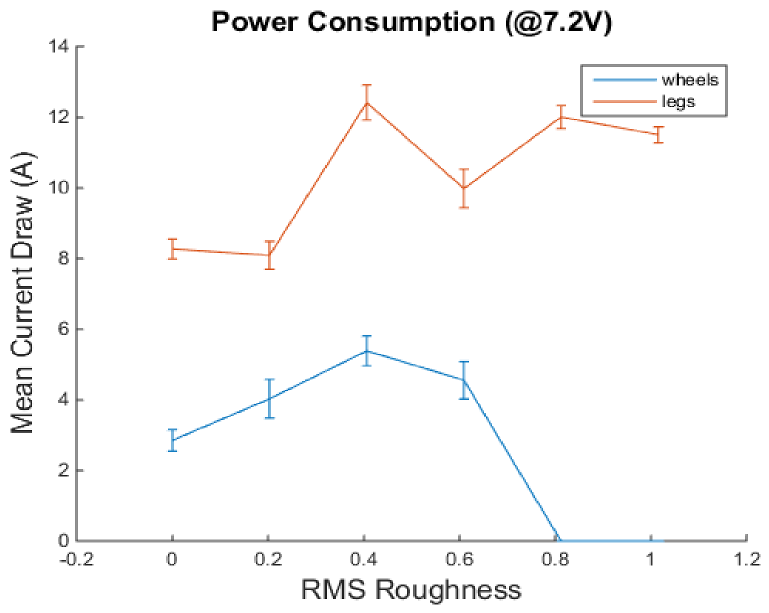


Figure 5.4: Mean current draw vs. RMS roughness.

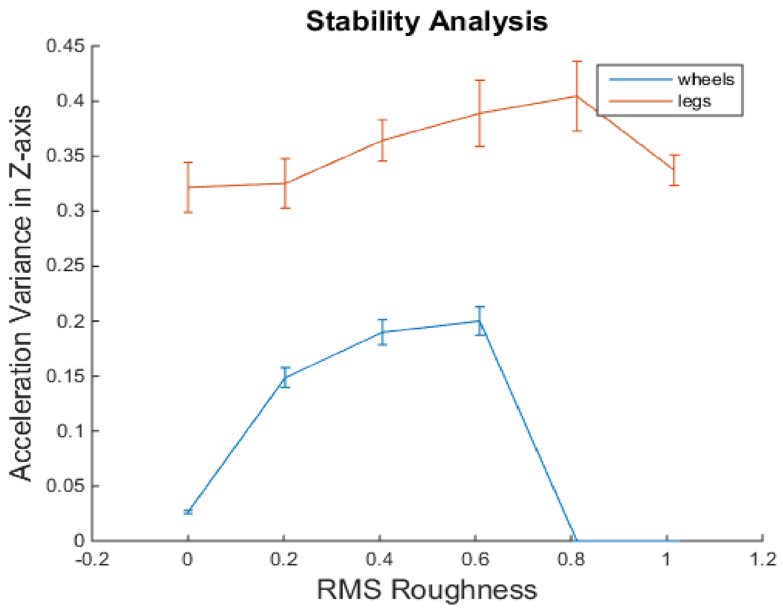


Figure 5.5: Mean z-axis acceleration variance vs. RMS roughness.

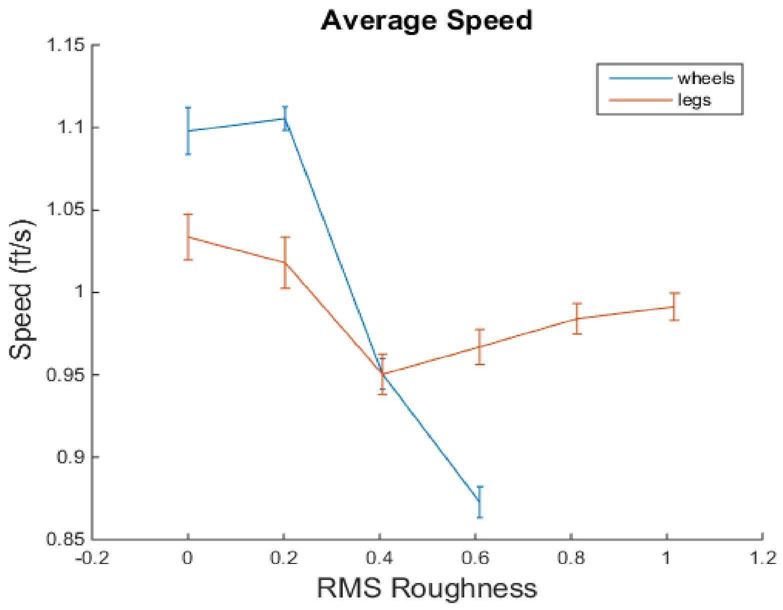
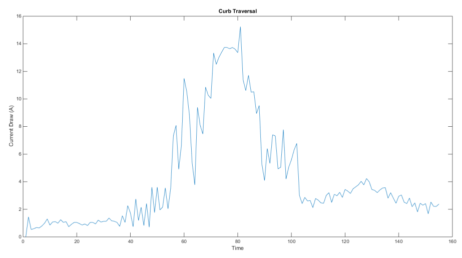


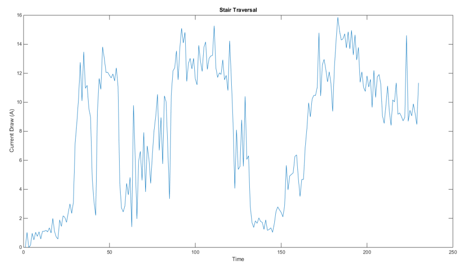
Figure 5.6: Mean speed vs. RMS roughness.

5.3.2 Natural Terrain

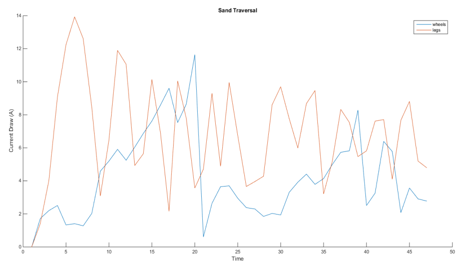
For the natural terrain experiments, we measured current draw and stability quantitatively, and left speed to qualitative observations. While the raw readings are too terrain dependent to give much general information, we compare the data of both wheels and legs to shed an informative light on the relative benefits of either design. Again, here the data is simply presented as is, while analysis is left to Section 5.4.



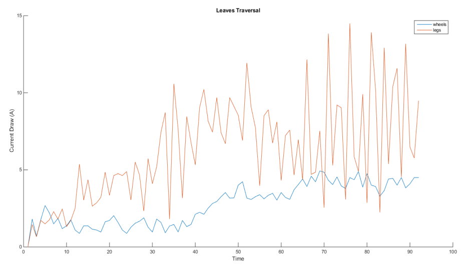
(a)



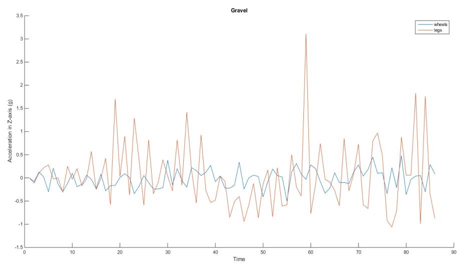
(b)



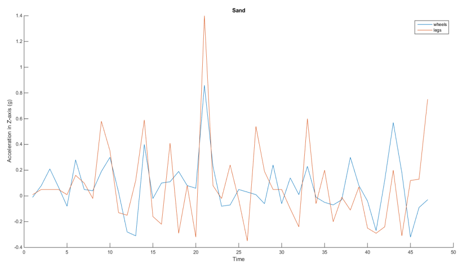
(c)



(d)



(e)



(f)

Figure 5.7: Comparative data from natural terrain test runs. (a) Current draw when going over a curb. (b) Current draw when going up long steps. (c) Current draw when operating in sand. (d) Current draw when transversing a deep leaf pile. (e) Accelerations in z on gravel. (f) Accelerations in z on sand.

5.4 Analysis

The data given in Sections 5.3.1 and 5.3.2 provide us with a good platform for which to base a quantitative juxtaposition between legs and wheels. Now we merge both the data from the controlled, artificial experiment and the more subjective natural terrain tests and investigate the comparative differences legs and wheels offer in terms of power consumption, stability, speed, and climbing ability.

5.4.1 Power Consumption

Experimentally, power consumption was the clearest gauge of the tradeoff between wheeled and legged modes. From the results of the artificial terrain experiments, we can see that current draw was always higher for legs than wheels. Therefore, by extension, if we reasonable assume a fixed voltage supply throughout the short trial run, the power, $P = IV$, is always higher.

Especially during the bumpy trajectories characteristic of the leg mode, current draw was far from constant. As a result, some spikes on natural terrain showed ratios of up to 8 : 1 for instantaneous power consumption values of legs to wheels. However, the average ratio in the artificial terrain experiments was approximately 2.3 : 1. This is the significant cost that we expected for using legs, and will indeed be a large contributor to the distinct tradeoff of the two modes.

5.4.2 Stability

Similar to the current data, stability measurements clearly showed an advantage for the wheeled-mode. In most instances in which both modes were able to successfully traverse the terrain, the acceleration variance was much lower for wheels while legs exhibited large amounts of jitter. In the artificial terrain experiments, the magnitude of acceleration in the z-axis had a legs to wheels ratio as high as 12 : 1 (on flat ground) and as low as 1.85 : 1 (in the roughest terrain over which wheels were still able to function). On natural terrain, we saw analogous effects. Although, overall the effect was slightly less pronounced as even flat natural terrain has inherent roughness that results in bumpy rides for the wheel mode.

5.4.3 Speed

Average speed was recorded only for the artificial terrain experiments where all factors that might affect performance could be kept in control. In Fig.5.6, the graph of speed vs. terrain roughness, we see a clear trade-off between the effect of wheels and legs. On flat ground, wheels were evidently faster than legs. This discrepancy is likely even *understated*, because speed was averaged only over a rather short distance (8 ft). Furthermore, the overall speed were kept low, at speeds that legs could still keep up with and remain stable. Yet, as the terrain became rougher, the advantage of legs became evident, and gradually overcame wheels as the RMS roughness index increased. Finally, the absolute, binary benefit of legs is seen for RMS roughness indexes past ≈ 0.6, where wheels are simply unable to traverse the terrain at all.

5.4.4 Climbing Ability

While not displayed in Section 5.3, we performed additional experiments on MARVIN's maximum step climbing abilities. The highest tested height that the robot was able to surmount was 7.75 inches — which is 3.1 times the radius of the retracted wheel. For comparison, MARVIN could not climb over 2 inches while in default wheel mode! This constitutes a significant advantage for obstacle traversal.

While MARVIN handles single steps with ease, flights of stairs remain more problematic. On stairs, a feature more commonly found in indoor environments, MARVIN's effectiveness is greatly dependent on the frequency of steps. For stairs with tall rises and short runs, MARVIN can get stuck between steps, and has difficulty finding purchase. A compliant spine or a refined gait control algorithm would likely improve this shortcoming.

5.5 Cost Based Mode Selection

The above results illustrate the empirical tradeoff specific to MARVIN's design, but really are inherent in any hybrid robot model. In order to fully capitalize on the advantages of hybridization, an intelligent method for switching between modes should be developed. At times this decision is an immediate binary — for example, when approaching stairs or curbs for which wheels are not a viable option. However, as the terrain decreases in roughness, the mode decision approaches a non-trivial, hard to estimate, threshold [32].

Unfortunately, we recognize that our data for MARVIN is sparse and lacking in resolution. Hence, predicting cost from our data will likely incur significant amounts of generalization error. Nevertheless, we still propose a preliminary cost analysis that serves to roughly guide the mode selection process. Given more data and experiments, this model can easily be improved.

Investigation of the data in Section 5.3 gives us the following average tradeoffs between legs and wheels for power, stability, and speed — the three most meaningful performance measurements:

- **Power** $\rightarrow$ average $= 2.3 : 1$, peak $= 8 : 1$
- **Stability** $\rightarrow$ average $= 4.7 : 1$, peak $= 12 : 1$
- **Speed** $\rightarrow$ average $= 1 : 1$, peak $= 1.1 : 1$

We then formulate the cost ratio of using legs as opposed to wheels as a function, G, where G is composed of a linear combination of our stated metrics. We define an application-specific vector of preference weights, $\mathbf{P} = [p_1, p_2, p_3]$ where $\sum_{i=1}^{3} p_i = 1$. Similar to the Q matrix in the LQR method for optimal control [2], P allows a user to indicate how much they value power relative to stability or speed, and so forth. Thus, this allows us to arrive at the following equation for G:

$$G = \mathbf{x}^T \mathbf{P} \tag{5.5.1}$$

where $\mathbf{x}$ is our vector of performance ratios: $[2.3, 4.7, 1]$.

Equipped with this definition of cost associated with the terrain, in the next section we leverage the hybridization tradeoffs with a proof of concept implementation of optimal, least cost path planning.

Chapter 6

Least-Cost Hybrid Paths

Path planning for mobile ground robots is an integral part of their ability to navigate various terrains. In essence, given a destination and a map, path planning is the task of autonomously identifying a trajectory that brings the robot to the goal [40]. Clearly, however, some trajectories are more desirable than others. In this section we demonstrate how hybridization allows allows robots to plan more efficient paths. This accomplished by both lifting restrictions that the terrain would otherwise place on wheeled robots, and also having to ability choosing the easier, drivable route where applicable — an opportunity unavailable to legged robots.

Despite the significant amount of attention now being focused on the design of various hybrid robots, there has been much less of a research effort towards optimizing their path planning processes [36], but some exist. One method proposes a algorithm that selects candidate paths based on an elevation map, and then analyzes these paths for switching conditions (from default wheels to legs). Finally, it evaluates the updated path with switching for performance characteristics such as energy consumption and travel time [36]. The reasoning does not have to be exact, either. Another method uses fuzzy logic at three separate scales of the path planning process to incorporate hybrid advantages given a "traversability index" [48].

6.1 Hybrid Global Path Planning

In a more holistic definition of path planning, *global* path planning is the process of optimizing an path from start to finish given a representation of the terrain, while also reconciling the chosen trajectory with unforeseen obstacles that arise along the way [40]. For the purpose of representing the robot's terrain, the most prominent methods include occupancy grids, configuration spaces, and potential field methods [4, 32]. Of these methods, the occupancy grid is the simplest, but its most common form is based on the assumption that occupied cells are to be avoided since they are non-traversable [32]. While this might be true for wheeled robots, hybrid robots might be able to overcome a subset of these "occupied" cells — resulting in a different set of conditions.

Furthermore, while there are a many algorithms that are adept at path planning [4], a fundamental feature common to all of them is that the calculations depend on the inherent costs assigned to each cell. As examined in Section 5.5, hybridization lends itself to a new form of cost function. A basic example is that of a robot trapped inside a fence. If the robot is wheeled, and the cost of climbing the fence is infinite, the robot will have to move on (possibly fruitlessly) in the hopes of finding an exit. A hybrid robot, however, can instead view the fence as a high, but finite cost, and weigh the effort of climbing it against the effort of searching all over for another way. More generally, any optimal path can now be reevaluated using this updated cost function, and lower cost paths in terms of distance or time might be found.

Furthermore, hybridization also allows for more robustness to errors in the occupancy grid. A mobile robot may need to react to unforeseen disturbances along its path. Hybridization affords the robot a larger margin of error in which to still be able to function — without having to continually update and recompute new paths.

6.2 A* with Hybrid Costs

Using our cost function from Section 5.5, we now implement a novel path planning algorithm for MARVIN, and show its effects. We represent the environment as an occupancy grid with three types of values: wheeled-terrain, legged-terrain, and insurmountable obstacles. Wheeled-terrain boxes are simply areas of terrain that are suitable for the default wheel mode (e.g. flat terrain), legged-terrain boxes represents intermediate, rough terrain, and finally obstacle boxes symbolize terrain that strictly cannot be traversed, even with legs. While eventually this process should seek to be autonomous, with terrain locally mapped by onboard robotic sensors, for now we assume a complete knowledge of the terrain, its costs, as well as the coordinates of both the robot and the goal.

The algorithm we chose to use for finding the minimum-cost path to the goal is a well-known algorithm called A*. This algorithm evaluates and chooses possible paths until it gets to the goal by summing $g(n)$, the cost to reach each node, with $h(n)$, the heuristic cost to get from each node to the goal node [38]. Formally, we have:

$$f(n) = g(n) + h(n) \tag{6.2.1}$$

Where $f(n)$ is the estimated cost of the cheapest path so far to node n. In our case, we used the Euclidian Distance to the goal node for our heuristic $h(n)$, and assigned cost functions to each node based on what type of terrain it represented. The cost of a wheeled-terrain node was set to a baseline value of 1. For our cost preference, we chose to emphasize only power, picking a $\mathbf{P}$ vector of $[1, 0, 0]$. Thus the single node cost of moving with legs is $G = 2.3$. In Fig. 6.1, we illustrate a few examples of our new hybrid paths.

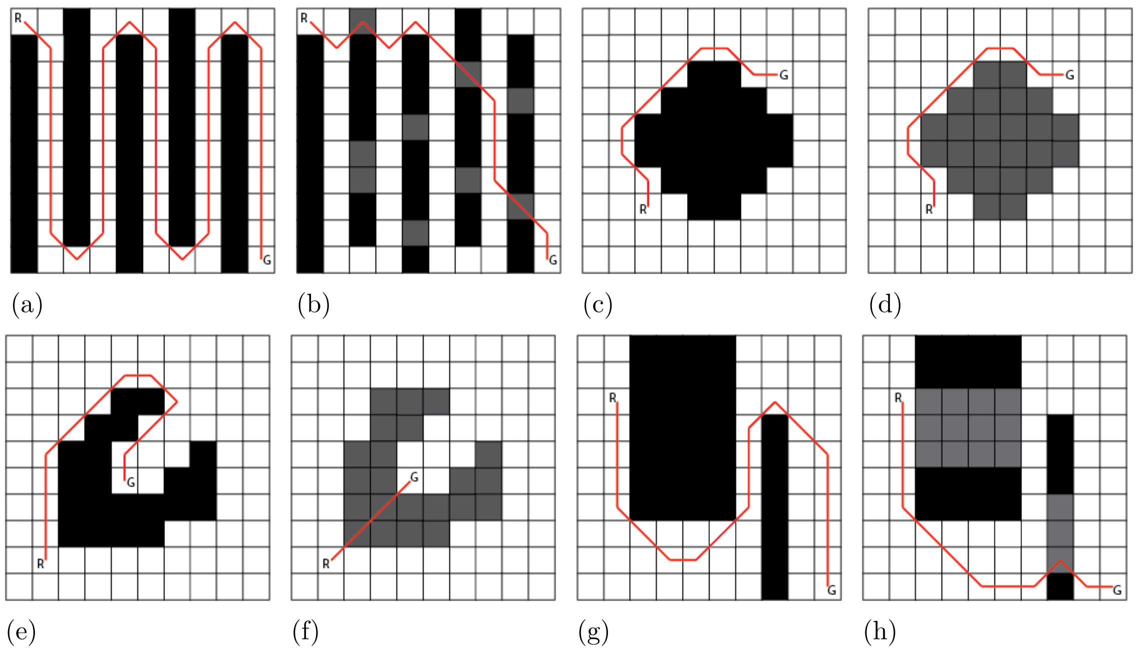


Figure 6.1: Comparison of standard vs hybridized optimal paths. The wheeled approach is on the left, while the hybrid approach is shown to the right

In Figs. 6.1 (a) and (b), we see the robot traversing a winding path in non-hybrid mode that results in it traversing the length of the terrain 5 times. By assigning only a few of these obstacle nodes to be legged-terrain nodes, the robot finds a much shorter and better path.

Figs. 6.1 (c) and (d) depict a situation in which hybridization does not affect the outcome of the path. In this environment, a dense obstacle blocks the robot from his goal, causing the robot to choose the default wheel-only path due to the large cost of using legs.

Figs. 6.1(e) and (f) show a similar situation to the one in (c)-(d), however this time the obstacle is not as dense and the goal node, closer. The robot deems the benefit of the shorter path to be greater than the cost of using legs to traverse the rough terrain.

Finally, Figs. 6.1 (g) and (h) depict a combination of using wheels and legs given two different types of obstacles. The first obstacle is too dense, thus the robot chooses to use wheel mode to get around it, however the second obstacle is small enough to switch to leg mode. The total path length is still shorter than in the case without hybridization.

6.3 Discussion

The method presented here is a simplification of a hybrid path planning system in an attempt to depict how hybridization provides advantages to the field of global path planning. In order to extend this method, one might incorporate other cost metrics other than power consumption, such as speed and stability or safety. However, even when only considering power consumption estimated from a cost metric derived from real experimental data, we have shown, to some extent, that MARVIN would benefit from a hybrid path planning system. In fact, MARVIN is particularly well suited for hybrid path planning given the ease and quickness with which mode switching is performed. In an improvement from the robot in [36], which could only switch modes under specific conditions and needed to evaluate whether switching was worth the time and energy, MARVIN can make this switch almost instantaneously. This avoids transition costs that would otherwise overshadow the benefits of making specific, low-level terrain adjustments.

Chapter 7

Future Work and Improvements

7.1 Design Improvements

Moving forward, we believe there are a number of improvements that can be made with our current design that would help optimize weight and power considerations of MARVIN. Given time constraints, we were unable to implement these improvements ourselves but realized their usefulness through the design process.

Firstly, we believe a more efficient design with fewer (but more powerful) motors is both plausible and desirable. The current design that uses one motor per axle was decided upon because of its simplicity over one-motor-per-wheel designs, and due to our confidence that we could build a rapid prototype in the time allotted given this method. Through experimentation, however, we have realized that the two-motor-per-wheel design as implemented in MARVIN, places too much stress on the individual motors in terms of torque requirements as well as control effort. We envision future designs to implement some version of a clutch or ratchet system to reduce the load on the motors, as well as allow for a one-motor-per-wheel design. Other additions such as suspension or torsion springs in the design of the wheel can further decrease the load placed on the motors and control algorithms as well as help to ensure a smoother ride.

From a systems-level view point, one of the first improvements to be made is to implement a version of gait coordination. This would not only improve the performance of the robot in climbing and rough terrain traversal tasks, but also help to better realize the true tradeoffs in power consumption, stability, and speed between wheels and legs. An open-loop gait control similar to the design in RHex [39] would be a tractable first step in this direction.

7.2 Scaling Considerations

Once efficient mechanical and electrical systems have been designed, there are a number of additions that need to be made to MARVIN in order for it to move toward our higher goal of an efficient autonomous hybrid robot with improved navigation. The most obvious fields in which improvements on this front could be made are an intelligent vision system and robust navigation techniques.

Vision systems are an integral part of autonomy in robotics. Complex vision systems has been accomplished on various mobile ground robots through a combination of laser and stereo camera sensor systems [19, 9]. These systems require reliable hardware that can work in a variety of conditions as well as robust software in order to create an accurate model of the environment. This accurate environment model would allow MARVIN to transition between modes autonomously based on the roughness of the surrounding terrain.

Improved navigation starts with a robust vision system to accurately represent the environment around the robot, however it extends beyond that. A great deal of research has been conducted on efficient path planning algorithms for exactly this purpose [41, 4]. Localization algorithms such as SLAM and integrated sensor systems such as high-precision GPS would ensure a knowledge of the robotÕs location within his environment that would undoubtedly lead to more robust navigation [14, 38, 22]. However, as mentioned before, path planning for hybrid robots is a relatively unexplored area. If MARVIN's hybrid ability is to be fully capitalized upon, a deep understanding of the cost tradeoffs between different modes of operation as a function of terrain roughness needs to be developed. The potential of such an understanding was shown in the previous section, however future work should look to refine the cost functions and tradeoffs between the two modes in order to better reflect the reality of hybridization tradeoffs.

Chapter 8

Conclusion

The design of hybrid robotics promises to continue to be a rich, exciting, and applicable area of research. Here we presented MARVIN, a contribution that is intended to improve upon existing hybrid robot designs by focusing on three fundamental features: a fast and smooth transition mechanism, good realizations of the major performance characteristics of both wheel and leg designs, and finally, a unique capacity for flexible control.

The elimination of an expensive transformation time allows for greater maneuverability in varied terrain — as well as a better realization of the advantages of a hybrid system. The transition to using spoke-legs acts as a natural extension of the wheel functionality, rather than as a cumbersome, disjoint appendage. However, when the spoke legs are not in use, MARVIN is able to stow them away completely — a feature not all hybrid design can claim. This allows the robot to make full use of the speed and stability inherent to wheels, while maintaining the option of switching to legs at any instant.

Finally, while not the main topic of this thesis, MARVIN was also designed to not just have control over which locomotion mode is used, but also the *extent* to which hybridization is realized. This capability stems from MARVIN's complete control over the degree to which the legs are extended. An exciting area of future research is how MARVIN might be able to leverage the idea of flexible control to better discriminate between types of terrain, and choose the leg extension that achieves the best stability and performance.

These attributes are all pointed towards one common goal: improved navigation over varied, unstructured terrain. We showed that with better adaptability to terrain and a personalized cost function, MARVIN can theoretically generate lower cost paths from point A to point B on terrains with scattered obstacles.

While the thesis advanced here contains several innovations to hybrid locomotion, it comes with several caveats. MARVIN's implementation is just a prototype. As discussed in Section 7, certain design flaws were identified along with solutions that can be fixed with further iteration. Those improvements include motors better matched to the task requirements. The cost algorithm can be improved upon, in part with more data. MARVIN could be equipped with vision, in order to navigate terrain with path-finding autonomously updated.

Yet, in practice, MARVIN did perform well. Preliminary experiments with our constructed prototype do demonstrate, unmistakably, the intended advantages — described in this report and shown in videos on the referenced YouTube channel. MARVIN is able to traverse curbs and low-frequency stairs by approaching with wheels, deploying legs in order to overcome the hurdle, and then quickly returning to wheel mode. Furthermore, data gathered on power, stability, and speed showed that MARVIN's wheels do offer a large improvement over legs on flat terrain. Clearly, the two modes work together to reduce costs. Moreover, in deriving the rationale in detail for design parameters here, we managed to identify the directions and the potential for yet more efficient implementations of this approach.

Bibliography

[1] *A Roadmap for U.S. Robotics: From Internet to Robotics.* 2013.

[2] Karl Johan Astrom and Richard M. Murray. *Feedback systems : an introduction for scientists and engineers.* Princeton university press, Princeton, Oxford, 2008.

[3] C. Bohn and D.P. Atherton. An analysis package comparing pid anti-windup strategies. *Control Systems, IEEE*, 15(2):34–40, Apr 1995.

[4] Thomas Brunl. *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems.* Springer Publishing Company, Incorporated, 3rd ed. edition, 2008.

[5] L. Bruzzone and G. Quaglia. Review article: locomotion systems for ground mobile robots in unstructured environments. *Mechanical Sciences*, 3(2):49–62, 2012.

[6] Luca E. Bruzzone and Pietro Fanghella. Mantis: hybrid leg-wheel ground mobile robot. *Industrial Robot*, 41(1):26–36, 2014.

[7] Shen-Chiang Chen, Ke-Jung Huang, Wei-Hsi Chen, Shuan-Yu Shen, Cheng-Hsin Li, and Pei-Chun Lin. Quattroped: A leg–wheel transformable robot. *Mechatronics, IEEE/ASME Transactions on*, 19(2):730–742, April 2014.

[8] Michael J. Coleman, Anindya Chatterjee, and Andy Ruina. Motions of a rimless spoked wheel: a simple 3d system with impacts. pages 139–160, 1997.

[9] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski. Self-supervised monocular road detection in desert terrain. In *Proceedings of Robotics: Science and Systems*, Philadelphia, USA, August 2006.

[10] Yu-Jie Dai, E. Nakano, T. Takahashi, and H. Ookubo. Motion control of leg-wheel robot for an unexplored outdoor environment. In *Intelligent Robots and*

Systems '96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on, volume 2, pages 402–409 vol.2, Nov 1996.

[11] E. Paul (Ernest Paul) DeGarmo, J. Temple Black, and Ronald A Kohser. *Materials and processes in manufacturing.* New York ; Chichester : Wiley, 9th ed.,international ed edition, 2003. Previous ed.: London : Prentice-Hall International, 1997.

[12] Richard C. Dorf and Robert H. Bishop. *Modern Control Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 7th edition, 1994.

[13] A. Drenner, I. Burt, T. Dahlin, B. Kratochvil, C. McMillen, B. Nelson, N. Papanikolopoulos, P.E. Rybski, K. Stubbs, D. Waletzko, and K.B. Yesin. Mobility enhancements to the scout robot platform. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 1, pages 1069–1074 vol.1, 2002.

[14] H. Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *Robotics Automation Magazine, IEEE*, 13(2):99–110, June 2006.

[15] Boston Dynamics. Bigdog - the most advanced rough-terrain robot on earth. `http://www.bostondynamics.com/robot_bigdog.html`.

[16] G.F. Franklin, J. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems, Global Edition.* Pearson Education Limited, 2015.

[17] D. Goldman, H. Komsuoglu, and D. Koditschek. March of the sandbots. *IEEE Spectr.*, 46(4):30–35, April 2009.

[18] Christophe Grand, FaŢz Ben Amar, FrŐdŐric Plumet, and Philippe Bidaud. Stability and traction optimization of a reconfigurable wheel-legged robot. *I. J. Robotic Res.*, 23(10-11):1041–1058, 2004.

[19] Raia Hadsell, Pierre Sermanet, Jan Ben Ayse Erkan, and Marco Scoffier. Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, pages 120–144, 2009.

[20] Dennis Hong and Doug Laney. Preliminary design and kinematic analysis of a mobility platform with two actuated spoke wheels. In *US-Korea Conference on Science, Technology and Entrepreneurship (UKC2006), Mechanical Engineering & Robotics Symposium*, pages 03–03, 2005.

[21] Karl Iagnemma, Adam Rzepniewski, Steven Dubowsky, and Paul Schenker. Control of robotic vehicles with actively articulated suspensions in rough terrain. *Autonomous Robots*, 14(1):5–16, 2003.

[22] S. Jeschke, H. Liu, and D. Schilberg. *Intelligent Robotics and Applications: 4th International Conference, ICIRA 2011, Aachen, Germany, December 6-8, 2011, Proceedings*. ACM Digital Library. Springer, 2011.

[23] N.J. Kasdin and D.A. Paley. *Engineering Dynamics: A Comprehensive Introduction*. Princeton University Press, 2011.

[24] C.T. Kilian. *Modern Control Technology: Components and Systems*. Student Material TV Series. Delmar Thomson Learning, 2001.

[25] Yoo-Seok Kim, Gwang-Pil Jung, Haan Kim, Kyu-Jin Cho, and Chong-Nam Chu. Wheel transformer: A miniaturized terrain adaptive robot with passively transformed wheels. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5625–5630, May 2013.

[26] Shawn C Kimmel. Considerations for and implementations of deliberative and reactive motion planning strategies for the novel actuated rimless spoke wheel robot impass in the two-dimensional sagittal plane. Master's thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, May 2008.

[27] Xin-Lan Li, Jong-Gyu Park, and Hwi-Beom Shin. Comparison and evaluation of anti-windup pi controllers. *Journal of Power Electronics*, 11(1):45–50, 2011.

[28] Hayk Martirosyan, Gregory Hughes, Christopher Payne, Thomas Owlett, and Brendan O'Leary. xjus: A hexapedal robot with a passively flexible spine. Princeton Senior Thesis, 2013.

[29] Tad McGeer. Passive dynamic walking. *The International Journal of Robotics Research*, 9(2):62–82, 1990.

[30] FranÌĎois Michaud, Dominic LÌĽtourneau, Martin Arsenault, Yann Bergeron, Richard Cadrin, FrÌĽdÌĽric Gagnon, Marc-Antoine Legault, Mathieu Millette, Jean-FranÌĎois ParÌĽ, Marie-Christine Tremblay, Pierre Lepage, Yan Morin, Jonathan Bisson, and Serge Caron. Multi-modal locomotion robotic platform using leg-track-wheel articulations. *Autonomous Robots*, 18(2):137–156, 2005.

[31] J.M. Morrey, B. Lambrecht, Andrew D. Horchler, Roy E. Ritzmann, and R.D. Quinn. Highly mobile and robust small quadruped robots. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 82–87 vol.1, Oct 2003.

[32] Robin Murphy, R. Peter Bonasso, and David Kortenkamp. *Artificial intelligence and mobile robots : case studies of successful robot systems.* AAAI Cambridge, MA London, Menlo Park, 1998.

[33] Youbin Peng, D. Vrancic, and R. Hanus. Anti-windup, bumpless, and conditioned transfer techniques for pid controllers. *Control Systems, IEEE*, 16(4):48–57, Aug 1996.

[34] Tony Phillips. Spirit may never phone home again. `http://science.nasa.gov/science-news/science-at-nasa/2010/30jul_spirit2/`, 2010.

[35] R.D. Quinn, J.T. Offi, D.A. Kingsley, and Roy E. Ritzmann. Improved mobility through abstracted biological principles. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2652–2657 vol.3, 2002.

[36] Eric Rohmer, Giulio Reina, and Kazuya Yoshida. Dynamic simulation-based action planner for a reconfigurable hybrid leg-wheel planetary exploration rover. *Advanced Robotics*, 24(8-9):1219–1238, 2010.

[37] Clarence Rowley. Mae 433: Automatic controls systems, 2014. Lecture notes, homeworks, and lab materials from Professor C. Rowley's controls class taught at Princeton.

[38] Stuart Jonathan Russell and Peter Norvig. *Artificial intelligence: a modern approach (3rd edition).* Prentice Hall, 2009.

[39] Uluc Saranli, Martin Buehler, and Daniel E. Koditschek. Rhex: A simple and highly mobile hexapod robot. *The International Journal of Robotics Research*, 20(7):616–631, 2001.

[40] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots.* The MIT Press, 2nd edition, 2011.

[41] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317 vol.4, May 1994.

[42] Adam Stokes, Robert F. Shepherd, Stephen A. Morin, Filip Ilievski, and George M. Whitesides. A hybrid combining hard and soft robots. *Soft Robotics*, 1(1):70–74, 2013.

[43] John D. Sutter. How 9/11 inspired a new era of robotics. `http://www.cnn.com/2011/TECH/innovation/09/07/911.robots.disaster.response/`, 2011.

[44] M. Takahashi, K. Yoneda, and S. Hirose. Rough terrain locomotion of a leg-wheel hybrid quadruped robot. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1090–1095, May 2006.

[45] P. Tantichattanont, S. Songschon, and S. Laksanacharoen. Quasi-static analysis of a leg-wheel hybrid vehicle for enhancing stair climbing ability. In *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference on*, pages 1601–1605, Dec 2007.

[46] B. K. Taylor, S. Balakirsky, E. Messina, and R. D. Quinn. Design and validation of a whegs robot in usarsim. In *Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems*, PerMIS '07, pages 105–112, New York, NY, USA, 2007. ACM.

[47] Richard Volpe. The athete rover. `https://www-robotics.jpl.nasa.gov/systems/system.cfm?System=11`.

[48] Zhiying Wang and Xilun Ding. Path planning for mobile robots with leg/wheel hybrid locomotion system on outdoor terrain. In *Intelligent Control and Automation (WCICA), 2012 10th World Congress on*, pages 3669–3674, July 2012.

[49] Wei Zhan. Robust design of motor pwm control using modeling and simulation. In Sio-Iong Ao, Burghard Rieger, and Su-Shing Chen, editors, *Advances in Computational Algorithms and Data Analysis*, volume 14 of *Lecture Notes in Electrical Engineering*, pages 439–449. Springer Netherlands, 2009.

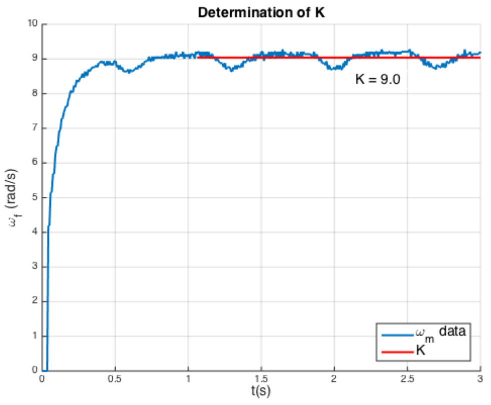
Appendix A

Motor Constants

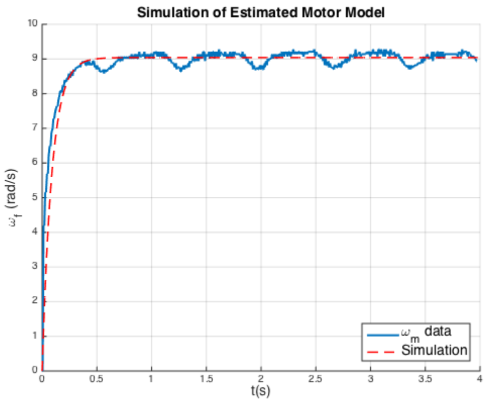
In order to find the parameters K and τ for the motor-wheel systems we took experimental data from a step response to a constant PWM signal, shown in Fig. A.1(a). K is found by taking the average of the steady state response. Next, by transforming Eq. 3.1.6 into the log space, we obtain a linear function with a slope of $\frac{1}{\tau}$:

$$\begin{aligned} \omega_f &= K - Ke^{-\frac{t}{\tau}} \\ \ln(K-\omega_f) &= \ln(Ke^{-\frac{t}{\tau}}) \\ -\ln(K-\omega_f) &= \frac{t}{\tau} + \ln K \end{aligned} \tag{A.0.1}$$

We use a best fit line in the well-behaved range of this equation to estimate τ. The reconstructed model for one of our motors is shown in Fig. A.1(b).



(a)



(b)

Figure A.1: Calibration of motor specific constants.

Appendix B

Manufacturing

The following sections detail the prototyping process that was taken to build MARVIN.

B.1 Wheel Module

There were several iterations in our construction of the 3D wheel module before the final structure was replicated and placed on the chassis. In the figure below you see three steps in the manufacturing process. Our first version was a simple, laser-cut 2D model used as a proof that the leg-extraction model worked. The second version of the wheel was 3D-printed and made significantly larger than the first iteration. The final version of the wheel was reduced in size and featured several improvements over the previous version. Firstly, hollow metal rods were inserted into the pins around screws in order to decrease friction in the mode-transition process. Also, through experimentation, we became worried that the torsion force on the inner axle was too great for the 3D printed material. We thus manufactured our own inner axle for all 4 motors using PVC so that the part would not shear off.

B.2 Gears and Gearbox

Due to the fact that both motors that actuated a single wheel needed to control coaxial rings, we needed to develop a system for independently turning each axle. We decided upon a gear system that connected the outer axle to its motor through a 1:1 gear ratio. This system was then implemented in two steps: designing and manufacturing

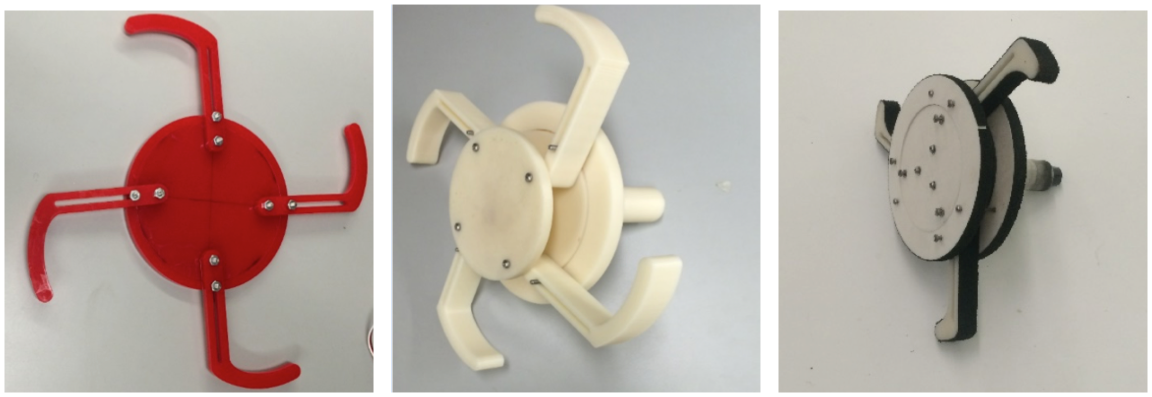


Figure B.1: Iterations of the spoke wheel.

the gears themselves, and developing the box which would mount both gears and motors to the same axis. The gears were designed in Creo and manufactured using a CNC machine. The gearbox was also designed in Creo as an integrated system with gears, motors, and wheel and then 3D printed.

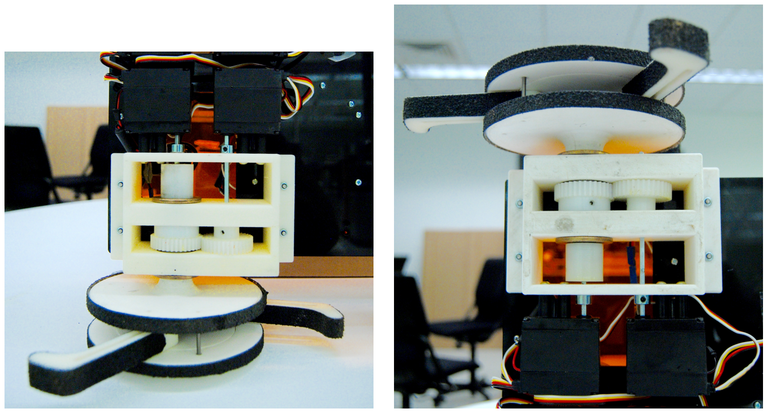


Figure B.2: The gearboxes.

B.3 Chassis

The chassis for MARVIN was designed in Adobe Illustrator and manufactured using Laser-cut Cast Acrylic material. The chassis itself went through two iterations in terms of layout and overall design. The first prototype was used as a rapid test platform to ensure MARVIN successfully was able to deploy legs and traverse rough terrain. This initial design however did not efficiently use the space on top of the chassis and was slightly long. The final design corrected these two deficiencies by decreasing the length such that there was just 1.5Ó of clearance between the legs when extracted, and more efficiently laid out systems on top of the chassis to better optimize stability.

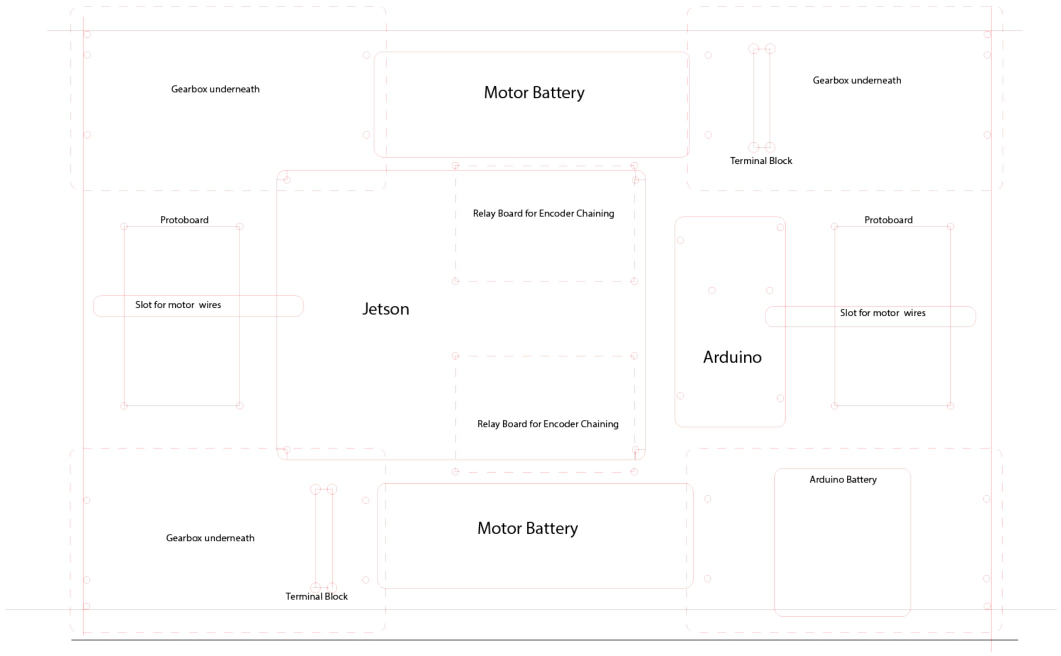


Figure B.3: Old chassis layout.

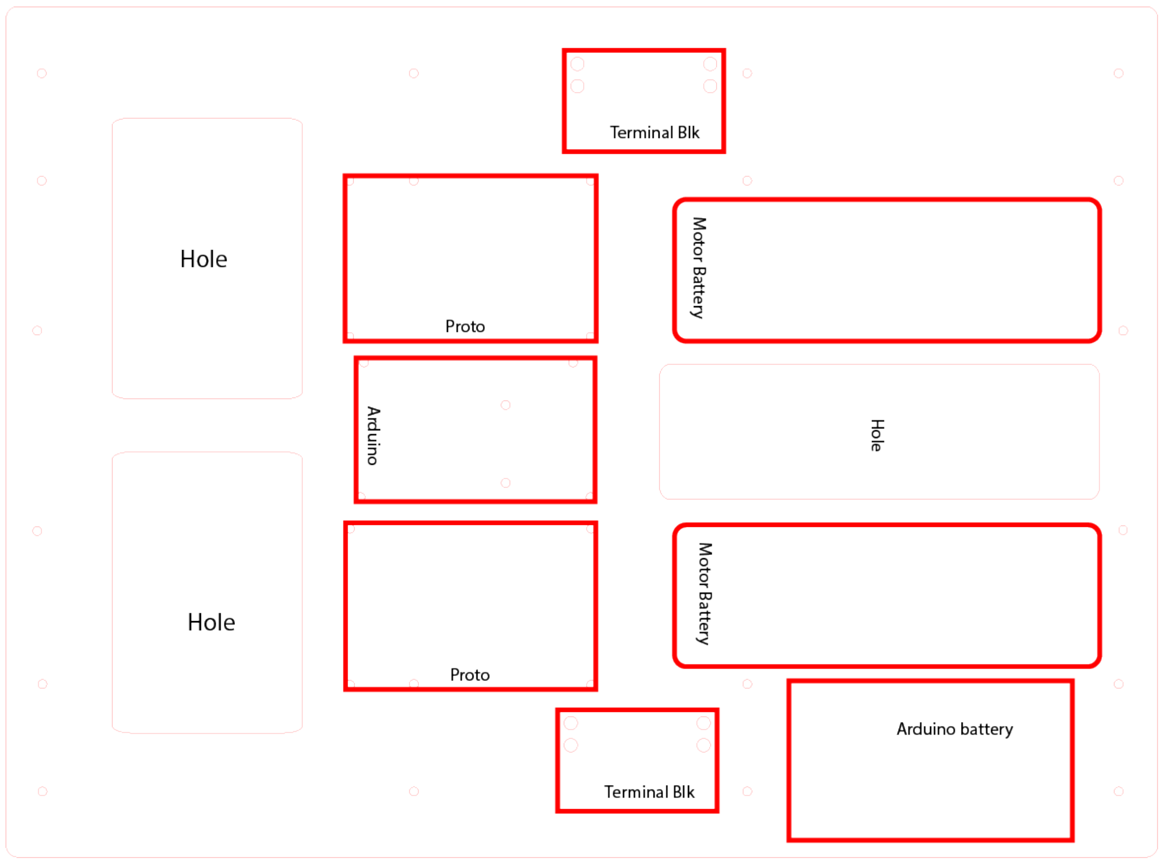


Figure B.4: The final chassis layout.

Appendix C

Electronics Schematic

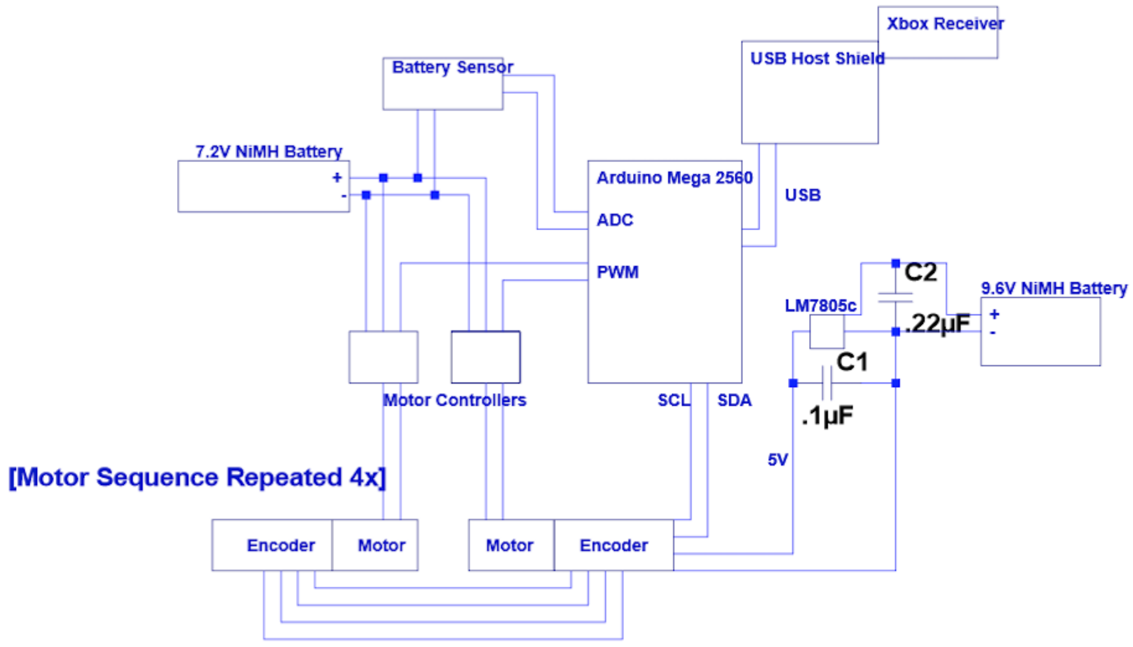


Figure C.1: Electronics schematic.

Appendix D

Parts List

Manufacturer	Part Name	Description	PartID
SparkFun	AttoPilot Voltage and Current Sense Breakout - 45A, 90A	Measures voltage and current coming from the battery and output to 3.3 V scale	SEN-10643, SEN-09028
SparkFun	Triple Axis Accelerometer Breakout - ADXL335	Measures acceleration in three axes up to +/- 3g and outputs as analog value	SEN-09269
Vex	2-wire Motor 393	Brushed Motor with 157:1 gearbox included in module	276-2177
Vex	Motor 393 Integrated Motor Encoder Module	Quadrature encoder that replaces back of 2-wire motor 393, communicates through I2C	276-1321
Vex	Motor Controller 29	Controller that connects to 2-wire motor 393, output PWM signal 1-2ms	276-2193
Arduino	Arduino Mega 2560 Rev3	Microcontroller based on ATmega2560, includes 54 digital I/O, 16 analog I/O, a USB connection and a clock speed of 16 MHz	A000067
Arduino	USB Host Shield	Peripheral device for Arduino Mega based on the MAX3421E	A000004
Microsoft	Xbox 360 Wireless Gaming Receiver	Interface between Xbox controller and the Arduino USB Host Shield	
SparkFun	LM7805c Voltage Regulator	Three terminal regulator that steps down input voltage to 5V	COM-00107
Vex	7.2V Robot Battery NiMH 3000mAh	Rechargeable NiMH battery with Tamiya Connector and large capacity or motors	276-1491
Tenergy	9.6V 2000mAh battery	Rechargeable NiMH battery with Tamiya Connector for Arduino/Encoders	11401-01

Appendix E

Arduino Code

The controls for MARVIN were implemented entirely on the AtMega. We have two main files: the control and sensor readings loop in the Arduino sketch file, and a VEX motor library that we wrote specifically for this project.

E.1 MotorController.ino

```
/******************************************************************/
/* MotorController.ino                                            */
/* Authors: Adam Fisch and Max Shatkhin                           */
/* use outer wheels for open/close
/******************************************************************/
#include <Wire.h>
#include <Servo.h>
#include <I2CEncoder.h>
#include <VexMotor.h>
#include <XBOXRECV.h>
#include <math.h>

/******************************************************************/
/*  Global driving constants.                                     */
/******************************************************************/
const double DT = 5;
const int numSpeeds = 7;
const double SPEEDS[] = {3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5};
const int RIGHT_SIDE = 0;
const int LEFT_SIDE = 1;
long t0 = millis();

/******************************************************************/
/*  Sensor constants.                                             */
/******************************************************************/
const bool PRINT = false;
const int SENSOR_SAMPLE_TIME = 150;
const float Iscale_45 = .0732;
const float Vscale_45 = .2415;
const float Iscale_90 = .0366;
const float Vscale_90 = .0630;
const float zeroG_x =  507.5;
const float zeroG_y =  507.0;
const float zeroG_z =  620.0;
const float scaleX = 102.5;
const float scaleY = 102.0;
const float scaleZ = 101.0;

/******************************************************************/
/*  Motor declarations.                                           */
/*  Motor paramaters: <Kp, Ki, Kd, Is outer wheel?>               */
/*  Inner axis motors: PID control for angle.                     */
/*  Outer axis motors: PI control for omega.                      */
/******************************************************************/
VexMotor mFrontRightO(20, 300, 0, true, 1);
VexMotor mFrontRightI(225, 400, 5, false, 1);

VexMotor mFrontLeftO(20, 300, 0, true, -1);
VexMotor mFrontLeftI(225, 400, 15, false, -1);

VexMotor mBackRightO(22, 300, 0, true, 1);
VexMotor mBackRightI(225, 400, 5, false, 1);

VexMotor mBackLeftO(20, 300, 0, true, -1);
VexMotor mBackLeftI(225, 400, 15, false, -1);

/******************************************************************/
/* PWM pin declarations. Maps Arduino AtMega 2560 output pins to   */
/*  motor controllers.                                            */
/******************************************************************/
int pinFRI = 2;
int pinFRO = 3;
int pinFLI = 4;
int pinFLO = 5;
int pinBRI = 6;
int pinBRO = 7;
int pinBLI = 12;
int pinBLO = 13;

/******************************************************************/
/*  Analog pin declarations for sensor readings.                  */
/******************************************************************/
int batt_l_i = A0;
int batt_l_v = A1;
int batt_r_i = A2;
int batt_r_v = A3;
int accel_x = A4;
int accel_y = A5;
int accel_z = A7;

/******************************************************************/
/*  USB and Xbox controller declarations.                         */
/*  Used for user input remote control.                           */
/******************************************************************/
USB Usb;
XBOXRECV Xbox(&Usb);

/******************************************************************/
/*  Initial setup code run once when the program is started.      */
/******************************************************************/
```

```
void setup() {
  // Begin I2C and Serial port communications.
  Wire.begin();
  Serial.begin(9600);

  // Attach motors to PWM output pins.
  mFrontRightO.attach(pinFRO, false);
  mFrontRightI.attach(pinFRI, false);

  mFrontLeftO.attach(pinFLO, false);
  mFrontLeftI.attach(pinFLI, false);

  mBackRightO.attach(pinBRO, false);
  mBackRightI.attach(pinBRI, false);

  mBackLeftO.attach(pinBLO, false);
  mBackLeftI.attach(pinBLI, true);

  // Initialize USB port
  if (Usb.Init() == -1) {
    Serial.print(("\r\nOSC did not start"));
    while (1); //halt
  }

  // The reference voltage for the analog signals is external
  analogReference(EXTERNAL);

  // Connect to the XBox receiver and wait for the start signal.
  while (1) {
    Usb.Task();
    if (Xbox.XboxReceiverConnected || Xbox.Xbox360Connected[0]) {
      if (Xbox.getButtonClick(Y, 0)) break;
    }
  }
}

/******************************************************************/
/*  This executes continuously while the Arduino is powered.      */
/*  Orchestrates control for the motors after reading desired     */
/*  output from sensors (Jetson, xBox controller).                */
/******************************************************************/
void loop() {
  static bool TURBO = false;
  if (Xbox.XboxReceiverConnected || Xbox.Xbox360Connected[0]) {
    if (Xbox.getButtonClick(A, 0)) {
      TURBO = !TURBO;
    }
  }

  if (millis() - t0 > SENSOR_SAMPLE_TIME && PRINT) {
    printData();
    t0 = millis();
  }

  // Set the locomotion mode to be used
  setMode();

  // Generate the smoothed reference speed
  double goalOmega = readInputSpeed();
  double rightOmega = constrain(goalOmega + steer(RIGHT_SIDE), 0.5, 9.5);
  double leftOmega = constrain(goalOmega + steer(LEFT_SIDE), 0.5, 9.5);
  Serial.println(mBackLeftO.getOmega());
  Serial.println(mBackLeftI.getOmega());
  Serial.println(mBackLeftO.getTheta());
  Serial.println(mBackLeftI.getTheta());

  if (TURBO) {
      mFrontRightO.write(255);
      mFrontRightI.write(-255);
      mFrontLeftO.write(-255);
      mFrontLeftI.write(255);

      mBackRightO.write(255);
      mBackRightI.write(-255);
      mBackLeftO.write(-255);
      mBackLeftI.write(255);
  }
  else {
    mFrontRightO.write(PIDControl(mFrontRightO, rightOmega));
    double refTheta = mFrontRightO.getTheta();
    mFrontRightI.write(PIDControl(mFrontRightI, refTheta));

    mFrontLeftO.write(PIDControl(mFrontLeftO, -leftOmega));
    refTheta = mFrontLeftO.getTheta();
    mFrontLeftI.write(PIDControl(mFrontLeftI, refTheta));

    mBackRightO.write(PIDControl(mBackRightO, rightOmega));
    refTheta = mBackRightO.getTheta();
    mBackRightI.write(PIDControl(mBackRightI, refTheta));

    mBackLeftO.write(PIDControl(mBackLeftO, -leftOmega));
    refTheta = mBackLeftO.getTheta();
    mBackLeftI.write(PIDControl(mBackLeftI, refTheta));
  }

  // Delay for a discrete timestep DT
  delay(DT);
}

/*****************************************************************/
/*  Read speed level input from the xBox controller.             */
/*  Returns a double that is the speed in rad/s.                 */
/*****************************************************************/
double readInputSpeed() {
  static int speedLevel = 3;
  Usb.Task();
  if (Xbox.XboxReceiverConnected || Xbox.Xbox360Connected[0]) {
    if (Xbox.getButtonClick(UP, 0) && speedLevel < numSpeeds - 1) {
      speedLevel++;
    }
    else if (Xbox.getButtonClick(DOWN, 0) && speedLevel > 0) {
      speedLevel--;
    }
  }
  return SPEEDS[speedLevel];
}

/*****************************************************************/
/*  Adjust the omega reference on a per side basis to allow for  */
/*  differential drive.                                          */
/*****************************************************************/
```

```
double steer(int side) {
  int differential = 0;
  Usb.Task();
  if (Xbox.XboxReceiverConnected || Xbox.Xbox360Connected[0]) {
    int right_value = Xbox.getButtonPress(R2, 0);
    int left_value = Xbox.getButtonPress(L2, 0);
    switch (side) {
      case LEFT_SIDE:
        differential = right_value - left_value;
        break;
      case RIGHT_SIDE:
        differential = left_value - right_value;
        break;
    }
    if (differential < 0) {
      differential = map(differential, -255, 0, -10, 0);
    }
    else {
      differential = map(differential, 0, 255, 0, 10);
    }
  }
  return differential;
}

/******************************************************************/
/*  Read instructions on whether to use legs or wheels.            */
/*  Returns true or false:                                         */
/*  (true) Open the wheel and use legs.                            */
/*  (false) Close the wheel and use standard drive.                */
/******************************************************************/
void setMode() {
  double offset;
  static bool useFrontLegs = false;
  static bool useBackLegs = false;

  Usb.Task();
  if (Xbox.XboxReceiverConnected || Xbox.Xbox360Connected[0]) {
    if (Xbox.getButtonClick(R1, 0)) {
      useFrontLegs = !useFrontLegs;
    }
    if (Xbox.getButtonClick(L1, 0)) {
      useBackLegs = !useBackLegs;
    }
  }

  if (useFrontLegs) {
    offset = OPEN_ANGLE;
    mFrontRightI.setOffset(offset);
    mFrontLeftI.setOffset(offset);
  }
  else {
    offset = -0.005;
    mFrontRightI.setOffset(offset);
    mFrontLeftI.setOffset(offset);
  }

  if (useBackLegs) {
    offset = OPEN_ANGLE;
    mBackRightI.setOffset(offset);
    mBackLeftI.setOffset(offset);
  }
  else {
    offset = -0.005;
    mBackRightI.setOffset(offset);
    mBackLeftI.setOffset(offset);
  }
}

/*****************************************************************/
/*  Classical PID control.                                       */
/*****************************************************************/
double PIDControl(VexMotor &motor, double reference) {
  // Calculate errors
  double e = motor.getError(reference);
  double proportionalE = motor.getEP(e);
  double integratedE = motor.getEI(e, DT);
  double derivativeE_lowP = motor.getED(e, DT);
  return (proportionalE + integratedE + derivativeE_lowP);
}

/*****************************************************************/
/*  Output data to the serial port for logging.                  */
/*****************************************************************/
void printData() {
  // battery left current
  float i_L = (float)analogRead(batt_l_i);
  float i_L2 = (i_L/1023)*3.3;
  float i_Lcal = (float)i_L2/Iscale_90;

  // battery left voltage
  float v_L = (float)analogRead(batt_l_v);
  float v_L2 = (v_L/1023)*3.3;
  float v_Lcal = (float)v_L2/Vscale_90;

  // battery right current
  float i_R = (float)analogRead(batt_r_i);
  float i_R2 = (i_R/1023)*3.3;
  float i_Rcal = (float)i_R2/Iscale_45;

  // battery right voltage
  float v_R = (float)analogRead(batt_r_v);
  float v_R2 = (v_R/1023)*3.3;
  float v_Rcal = (float)v_R2/Vscale_45;

  // accelerometer
  int x_val = analogRead(accel_x);
  int y_val = analogRead(accel_y);
  int z_val = analogRead(accel_z);

  float accelX_cal = ((float)x_val - zeroG_x)/scaleX;
  float accelY_cal = ((float)y_val - zeroG_y)/scaleY;
  float accelZ_cal = ((float)z_val - zeroG_z)/scaleZ;

  // Print voltage and current data to serial port
  Serial.println(i_Lcal);
  Serial.println(v_Lcal);
  Serial.println(i_Rcal);
  Serial.println(v_Rcal);

  // Print acceleration data to serial port
  Serial.println(accelX_cal);
  Serial.println(accelY_cal);
  Serial.println(accelZ_cal);

  // Print speed of the wheels
  Serial.println(mFrontRightO.getOmega());
  Serial.println(mFrontLeftO.getOmega());
  Serial.println(mBackRightO.getOmega());
  Serial.println(mBackLeftO.getOmega());
}
```

E.2 VexMotor.h

```
/****************************************************************************/
/* VexMotor.h                                                               */
/* Authors: Adam Fisch and Max Shatkhin                                     */
/****************************************************************************/
#ifndef VexMotor_h
#define VexMotor_h

#include "Arduino.h"
#include <Servo.h>
#include <I2CEncoder.h>
#include <math.h>

const double POSITIVE_DEAD_BAND = 20;
const double NEGATIVE_DEAD_BAND = -20;
const double ENCODER_OVERFLOW = 100; // True value is 104.5
const double OPEN_ANGLE = 1.8;

class VexMotor {
    public:
        VexMotor(double Kp, double Ki, double Kd, bool outer, int dir);
        int attach(int pin, bool last);
        void write(double value);
        void setOffset(double offset);
        double getTheta();
        double getOmega();
        double getError(double reference);
        double getEP(double error);
        double getEI(double error, double dt);
        double getED(double error, double dt);
        double thetaError(double reference);
        double omegaError(double reference);

    private:
        int _pin;
        int _sign;
        int _dir;
        bool _isOuterWheel;
        double _offset;
        double _overflow;
        double _kWeights[3];
        double _errorIntegral;
        double _errLast;
        double _uDLast;
        double _lastTheta;
        double _kt;
        Servo _motor;
        I2CEncoder _encoder;
};
```

```
#endif
```

E.3 VexMotor.cpp

```
/**************************************************************************/
/* VexMotor.cpp                                                           */
/* Authors: Adam Fisch and Max Shatkhin                                   */
/* Description: This is a class for a VEX motor object. The object        */
/* encapsulates several useful methods, such as reading positions and speeds */
/* from the encoders and storing instance specific constats.              */
/**************************************************************************/
#include "Arduino.h"
#include "VexMotor.h"
#include <Servo.h>
#include <I2CEncoder.h>
#include <math.h>

VexMotor::VexMotor(double Kp, double Ki, double Kd, bool outer, int dir) {
    // PID control weights
    _kWeights[0] = Kp;
    _kWeights[1] = Ki;
    _kWeights[2] = Kd;

    // The outer wheels direction is reversed due to the gear chain
    _isOuterWheel = outer;
    _sign = (outer) ? -1 : 1;

    // Determines whether the wheel is on the right or left side
    _dir = dir;

    // Keep running counters of the error state
    _errorIntegral = 0;
    _errLast = 0;
    _uDLast = 0;
    _offset = 0;

    // Keep encoder parameter to know when to reset counter and tell sign
    _overflow = 0;
    _lastTheta = 0;
    _kt = 1/_kWeights[1];

}

int VexMotor::attach(int pin, bool last) {
    _motor.attach(pin);
    _encoder.init(MOTOR_393_TORQUE_ROTATIONS, MOTOR_393_TIME_DELTA);
    if (!last) {
        _encoder.unTerminate();
    }
    else {
        _encoder.terminate();
    }
    return _encoder.getAddress();
}

void VexMotor::write(double value) {
    double constrained = value;
    if (constrained > 255) constrained = 255;
    else if (constrained < -255) constrained = -255;
   _motor.writeMicroseconds(map(constrained, -255, 255, 1000, 2000));
}

void VexMotor::setOffset(double offset) {
    _offset = offset;
}

double VexMotor::getEP(double error) {
    return _kWeights[0] * error;
}

double VexMotor::getEI(double error, double dt_ms) {
    double dt_s = dt_ms/1000;
    double prevEI = _errorIntegral;
    double uI = _kWeights[1] * prevEI;
    double sat = 0;
    if (uI > 255) {
        sat = uI - 255;
    }
    else if (uI < -255) {
        sat = uI + 255;
    }
    _errorIntegral += dt_s * error + _kt*sat;
    return _kWeights[1] * prevEI;
}

double VexMotor::getED(double error, double dt_ms) {
    double K = 15;
    double dt_s = dt_ms/1000;
    double uD = K * _kWeights[2] * (error - _errLast)
                + (1 - K * dt_s) * _uDLast;
    _errLast = error;
    _uDLast = uD;
    return uD;
}

double VexMotor::getTheta() {
    double revolutions = _encoder.getPosition();
    if (revolutions > ENCODER_OVERFLOW || revolutions < -ENCODER_OVERFLOW) {
        _overflow += revolutions;
        revolutions = 0;
        _encoder.zero();
    }
    revolutions = revolutions + _overflow;
    double theta = revolutions * M_PI;
    return theta;
}

double VexMotor::getOmega() {
    double currTheta = getTheta();
    int dir = (currTheta - _lastTheta >= 0) ? 1 : -1;
    _lastTheta = currTheta;
    double rpm = dir * _encoder.getSpeed();
    double omega = (rpm * 2 * M_PI) / 60;
    return omega;
}

double VexMotor::getError(double reference) {
    return (_isOuterWheel) ? omegaError(reference) : thetaError(reference);
```

```
}

double VexMotor::thetaError(double reference) {
    double diff = -_sign*reference - _dir*_offset - getTheta();
    return diff;
}

double VexMotor::omegaError(double reference) {
    double diff;
    diff = reference - getOmega();
    return diff;
}
```

Appendix F

A* Code

This is open-source code that implements a version of the A* algorithm in python. This code was modified to assign grid nodes a cost dependent on what type of terrain is represented by the node. By augmenting the code in this way, we were able to test the effects of hybridization on this well-researched path planning algorithm.

F.1 example_grid.py

```
# Copyright (C) 2012 Justin Poliey <justin.d.poliey@gmail.com>

# Permission is hereby granted, free of charge, to any person obtaining a
# copy of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom the Software
# is furnished to do so, subject to the following conditions:

# The above copyright notice and this permission notice shall be included
# in all copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
# EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
# OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
# IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
# DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
# TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
# OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


from astar_grid import AStarGrid, AStarGridNode
from itertools import product
#from random import randint

#obstacles = [];
```

```
def make_graph(width, height):
    nodes = [[AStarGridNode(x, y, 0) for y in range(height)] for x in range(width)]

## claw nodes
    nodes[2][4] = AStarGridNode(2,4,2)
    nodes[2][5] = AStarGridNode(2,5,2)

    nodes[3][3] = AStarGridNode(3,3,2)
    nodes[3][4] = AStarGridNode(3,4,2)

    nodes[4][2] = AStarGridNode(4,2,2)
    nodes[4][3] = AStarGridNode(4,3,2)
    nodes[4][7] = AStarGridNode(4,7,2)

    nodes[5][2] = AStarGridNode(5,2,2)
    nodes[5][3] = AStarGridNode(5,3,2)
    nodes[5][6] = AStarGridNode(5,6,2)
    nodes[5][7] = AStarGridNode(5,7,2)

    nodes[6][2] = AStarGridNode(6,2,2)
    nodes[6][3] = AStarGridNode(6,3,2)
    nodes[6][4] = AStarGridNode(6,4,2)
    nodes[6][5] = AStarGridNode(6,5,2)
    nodes[6][6] = AStarGridNode(6,6,2)
    nodes[6][7] = AStarGridNode(6,7,2)

    nodes[7][2] = AStarGridNode(7,2,2)
    nodes[7][3] = AStarGridNode(7,3,2)
    nodes[7][4] = AStarGridNode(7,4,2)
    nodes[7][5] = AStarGridNode(7,5,2)


    graph = {}
    for x, y in product(range(width), range(height)):
        node = nodes[x][y]
        graph[node] = []
        for i, j in product([-1, 0, 1], [-1, 0, 1]):
            if not (0 <= x + i < width):
                continue
            if not (0 <= y + j < height):
                continue
            graph[nodes[x][y]].append(nodes[x+i][y+j])
    return graph, nodes

graph, nodes = make_graph(10, 10)
paths = AStarGrid(graph)
start, end = nodes[8][1], nodes[5][4]
print 'start node: %s , %s' % (start.x, start.y)
print 'end node: %s , %s' % (end.x, end.y)
path = paths.search(start, end)
if path is None:
    print "No path found"
else:
    print "Path found:"
    for node in path:
        print node.x, node.y
```

F.2 astar_grid.py

```
from astar import AStar, AStarNode
from math import sqrt


class AStarGrid(AStar):
    def heuristic(self, node, start, end):
        return sqrt((end.x - node.x)**2 + (end.y - node.y)**2)


class AStarGridNode(AStarNode):
    def __init__(self, x, y, roughness):
        self.x, self.y, self.roughness = x, y, roughness
        super(AStarGridNode, self).__init__()

    def move_cost(self, other):
                if (other.roughness == 0):
                        return 1
                elif (other.roughness == 1):
                        return 2.3
                else:
                        return 100
```

F.3 astar.py

```
class AStar(object):
    def __init__(self, graph):
        self.graph = graph

    def heuristic(self, node, start, end):
        raise NotImplementedError

    def search(self, start, end):
        openset = set()
        closedset = set()
        current = start
        openset.add(current)
        while openset:
            current = min(openset, key=lambda o:o.g + o.h)
            if current == end:
                path = []
                while current.parent:
                    path.append(current)
                    current = current.parent
                path.append(current)
                return path[::-1]
            openset.remove(current)
            closedset.add(current)
            for node in self.graph[current]:
                if node in closedset:
                    continue
                if node in openset:
                    new_g = current.g + current.move_cost(node)
                    if node.g > new_g:
                        node.g = new_g
                        node.parent = current
                else:
                    node.g = current.g + current.move_cost(node)
                    node.h = self.heuristic(node, start, end)
                    node.parent = current
                    openset.add(node)
        return None


class AStarNode(object):
    def __init__(self):
        self.g = 0
        self.h = 0
        self.parent = None

    def move_cost(self, other):
        raise NotImplementedError
```