ESE 2050 Final Paper: System Performance of the PiCar Motor and Control Systems

Paul Pham
Department of Electrical and Systems Engineering
Washington University in St. Louis
St. Louis, USA
p.h.pham@wustl.edu

***Abstract*—This case study evaluates the system motor performance and PID control implementation of a Raspberry Pi-controlled vehicle (PiCar). First, the open-loop gain is characterized by mapping motor duty cycles to rotational speed (RPS). Next, the system's analog-to-digital (AD) and RPS sampling rates are optimized for real-time control, with accuracy verified by Fast Fourier Transform (FFT) analysis. The report then details the tuning of the proportional (K_p), integral (K_i), and derivative (K_d) parameters to minimize overshoot, achieve quick response times, and eliminate steady-state error under two objectives: suspended and driving. Finally, this report outlines the design strategies and sensor implementations used to achieve autonomous navigation objectives such as target-seeking and a traffic light simulation, concluding with future system improvements.**

I. Introduction

In this study, I evaluated the autonomous navigation systems and motor performance of a Raspberry Pi-controlled vehicle (PiCar). Because raw motor inputs—governed by Pulse Width Modulation (PWM)—are highly susceptible to environmental disturbances, battery fluctuations, and physical load changes, relying on open-loop commands is not enough for precise navigation and to achieve an accurate, steady velocity. This paper details the design, tuning, and application of a Proportional-Integral-Derivative (PID) controller for the PiCar's drive system.

The study begins by establishing the baseline open-loop gain of the motor, mapping PWM duty cycles to Rotational Speed (RPS). I then explored the correct selection of Analog-to-Digital (AD) and velocity sampling rates, and validated the accuracy of our real-time calculations using Fast Fourier Transform (FFT) analysis. With reliable RPS estimation established, the report examines the tuning of the K_p, K_i, and K_d control parameters. The system's step response is evaluated on two distinct states: a suspended condition and a ground-driving condition. Rise time, percent overshoot, and steady-state error are analyzed to determine the optimal control gains. Furthermore, the behaviors of the system are observed by isolating and scaling the proportional gain to observe its direct impact on stability, overshoot, and steady-state error.

Finally, the optimized control system serves as the foundation for complex autonomous tasks. The latter half of this report documents the algorithmic design strategies and sensor implementations, utilizing computer vision via Pi camera, ultrasonic distance sensors, and MPU-6050 accelerometers, all integrated to successfully execute dynamic target seeking, traffic light recognition, and physical environment challenges.

II. Open-Loop Characterization

I began with an analysis of the motor's open-loop gain by mapping the PWM duty cycle to the resulting rotational speed (RPS). To establish an accurate initial estimation (feedforward ratio) for our control loop, I first had to characterize the specific physical constraints of our hardware. In an ideal, frictionless environment, plotting the duty cycle against RPS would yield a perfectly linear relationship originating from $(0, 0)$. However, physical hardware constraints such as friction and motor limits will expose a less ideal graph.

At low duty cycles, the average voltage delivered to the motor is insufficient to overcome the static friction of the gearbox, causing the RPS to remain at zero. Once the duty cycle crosses this frictional threshold, the RPS scales in a nearly linear fashion. Conversely, at high duty cycles, the motor approaches its physical voltage and speed limits, causing the velocity to cap out and the graph to flatten. To quantify this, the vehicle was operated in an open-loop state ($K_p = K_i = K_d = 0$) across duty cycles ranging from 30% to 100%. The average RPS was calculated for each interval and plotted.

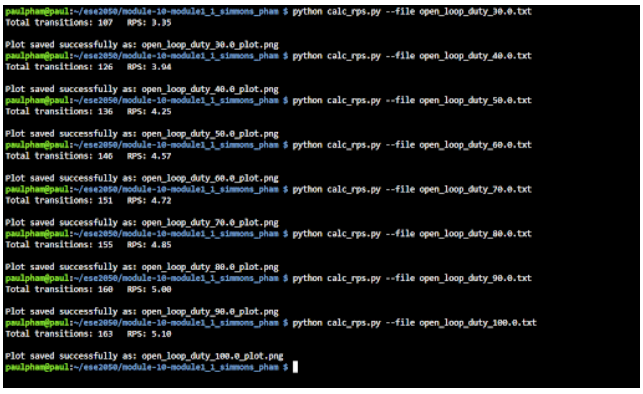


Fig. 1: Console output of average RPS across 30%-100% duty cycles.

As shown in Figure 2, the mid-range duty cycles illustrate a linear relationship, while the upper bounds clearly demonstrate

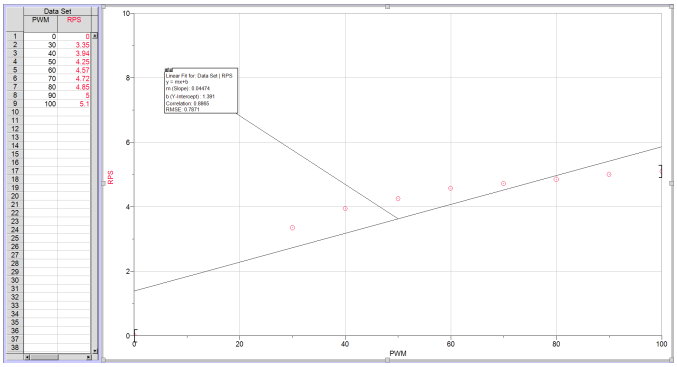


Fig. 2: Logger Pro Linear Fit of PWM Duty Cycle vs. RPS.

velocity cap. Applying a linear regression to the data yielded the following equation:

$$y = 0.04474x + 1.391 \tag{1}$$

Where y is the RPS and x is the PWM duty cycle. The observed slope of 0.04474 RPS/PWM indicates the open-loop gain. Taking the reciprocal of this slope provides a theoretical feedforward ratio of 22.35 PWM/RPS.

However, utilizing this raw reciprocal as a direct multiplier ignores the significant y-intercept (1.391) caused by the linear fit. If applied directly, the feedforward term would calculate a duty cycle nearly double what is actually required, guaranteeing massive system overshoot. Through empirical tuning, I determined that scaling this gain down to 33% (an effective ratio of ≈ 7.35) yielded the optimal baseline. This scaled ratio provides a good initial duty cycle, safely overcoming static friction while allowing the closed-loop PID parameters to smoothly correct the remaining error without aggressive overshoot.

III. Sampling Rates and Signal Processing

To achieve stable closed-loop control, accurate, real-time RPS estimation is needed. This requires a deliberate balance between the Analog-to-Digital (AD) sampling frequency and the RPS calculation window.

For our system, the AD sampling rate was set to 200 Hz (0.005 seconds per sample). This high-frequency ensures that the photo resistor captures every black-to-white transition on the wheel, even at maximum motor speeds. However, calculating the Rotational Speed (RPS) at this same 200 Hz frequency would result in highly sensitive, noisy data that depends on the rate of transitions found by the spinning motor. To fix this, the RPS calculation rate was buffered. By requiring a window of past samples before calculating the RPS, the system ensures enough transitions have occurred to compute a reliable average speed.

To mathematically validate the accuracy of our real-time RPS calculations, a Fast Fourier Transform (FFT) analysis was conducted. A subset of the PID-controlled feedback-loop data was isolated, specifically targeting the steady-state region where the motor was operating at an average constant RPS of 4.0.

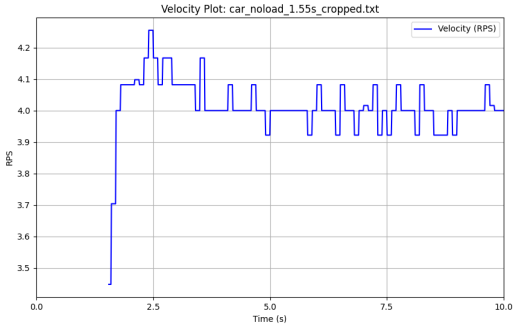


Fig. 3: Isolated portion of the steady state data for 4.0 rps.

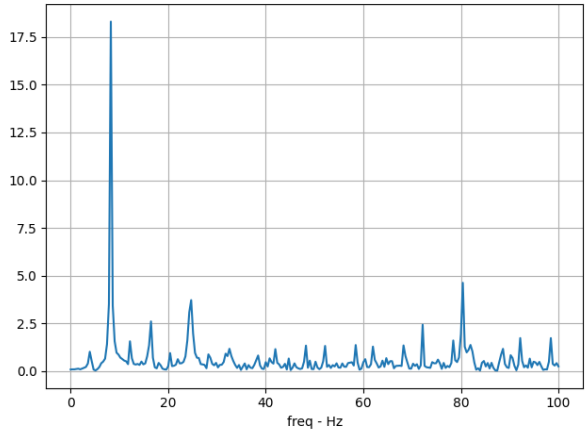


Fig. 4: FFT analysis of the steady-state data.

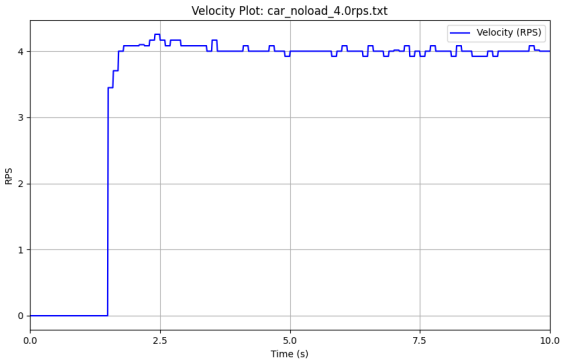
This FFT-derived velocity yields a dominant frequency of 7.94 Hz and a true speed of 3.97 RPS. Comparing this highly accurate, post-processed benchmark to the 4.016 RPS value calculated by our real-time Python script (objective1.py) during the same timeframe reveals a very minimal margin of error. This strong correlation justifies the accuracy of our real-time calculations and verifies the optimized sampling rates.

IV. PID Controller Tuning

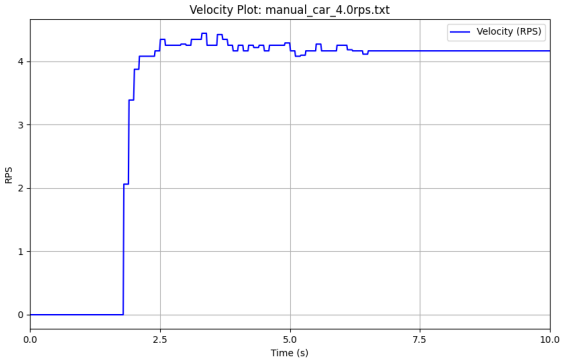
This section focuses on tuning the Proportional (K_p), Integral (K_i), and Derivative (K_d) parameters of the closed-loop controller. The objective was to minimize rise time and percent overshoot while also minimizing steady-state error at a target speed of 4.0 RPS. Because the physical dynamics of the PiCar change drastically depending on whether it is suspended (Objective 1: Control) or on the ground (Objective 2: Movement with Control), the system required two distinct sets of tuning parameters.

The optimal tuning parameters for the suspended state were determined to be $K_p = 0.03$, $K_i = 11.0$, and $K_d = 0.6$. As shown in Figure 6(a), this tuning yielded a rise time of 0.1050 seconds (Since the sampling rate is 0.005 and the script

requires 300 sample values to start calculating the RPS, it would be 1.6050 - 1.500), a percent overshoot of 6.375%, and a steady-state error of 0 RPS.



(a) System step response to a 4 RPS target (Suspended).



(b) System step response to a 4 RPS target (Ground-Driving).

Fig. 5: Comparison of optimized PID tuning under no-load and loaded conditions.

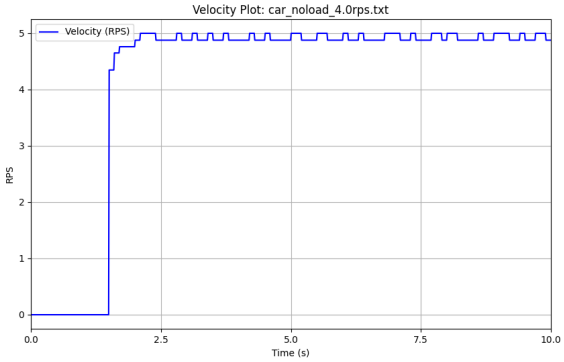
Driving on the ground introduces the full load of the PiCar's body and the static friction of the floor. To overcome this significant resistance and maintain the 4 RPS target, the controller required substantially more aggressive gains. The optimal parameters for ground driving were tuned to $K_p = 11$, $K_i = 7$, and $K_d = 0.1$. The aggressive proportional gain (11) provided the necessary power to overcome static ground friction, while the integral gain (7) helped eliminate the continuous steady-state error caused by rolling resistance. The small derivative term (0.1) was provided to prevent the aggressive K_p from causing severe overshoot. As shown in Figure 6(b), these loaded parameters resulted in a rise time of 0.607 seconds, a percent overshoot of 11%, and a steady-state error of 0.132 RPS.

V. Proportional Gain Scaling Analysis

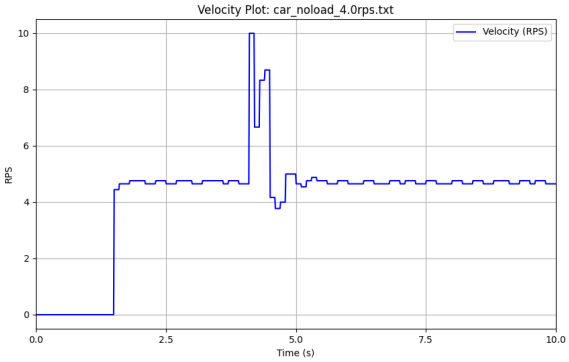
A proportional gain scaling test was conducted using the suspended car setup. The integral and derivative terms were disabled ($K_i = 0, K_d = 0$), and the system was tested at half ($0.5\times$) and ten times ($10\times$) the optimal K_p value.

When K_p was halved, the system exhibited practically no overshoot, and because there was no integral term to accumulate error, the system settled with a massive steady-state error, never actually reaching the 4 RPS target, and rather settled around 5.0 RPS.

When K_p was increased by a factor of ten, the system exhibited severe overshoot and oscillation. While the rise time was instantaneous, the massive gain resulted in extreme overshoot and continuous oscillation around the target velocity.



(a) Halved K_p system step response to a 4 RPS target.



(b) 10x K_p system step response to a 4 RPS target.

Fig. 6: Comparison of K_p PID tuning.

Ultimately, these tests confirm that while K_p is essential for decreasing rise time and providing the initial drive force, solely relying on it leads to high steady state error if too low, or extreme overshoot and oscillation if too high. It must be paired with an appropriately tuned K_i to smoothly eliminate the remaining steady-state error without inducing continuous oscillation.

VI. Objectives 3-5, Sensor Implementations, and Performance Analysis

With a solid PID control set up for the car, further implementations with the ultrasonic sensor, Pi camera, and the accelerometer were utilized in order to properly run Objectives 3 - 5: Seeker, Traffic Light, and Cruise Control, respectively.

A. Seeker

In Objective 3: Seeker, my task was to implement a script utilizing computer vision for the PiCar to autonomously seek out a blue target placed at least 10 feet away, and move as close to the target as quickly as possible without hitting it. This required a real-time continuous feedback loop between the visual data array and the steering servo, creating a dynamic adjustment loop to align the vehicle's trajectory with the center of the blue object.

1) Design Strategy and Sensor Implementation: I broke the control logic into four distinct states: Detecting, Alignment, Drive, and Arrived.

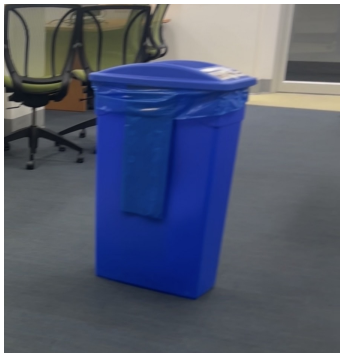
In the "Detecting" state, the vehicle utilized a step-controlled loop to sweep the swivel servo left and right to locate the blue object. The primary sensor utilized for this objective was the Raspberry Pi Camera Module. To isolate the target object from the background, images were captured at a frequency of 10 frames per second and converted from the standard RGB color space to HSV (Hue, Saturation, and Value). This allowed me to separate the color information from the lighting intensity, which assisted in creating a visual mask that narrowed down the primary blue color against shadows and ambient room lighting. Using `cv2.inRange`, upper and lower HSV thresholds were defined to create a binary mask that filtered out the blue target (white pixels) from the rest of the environment (black pixels).

Once the primary target was isolated, the PiCar entered the "Alignment" state, where the system processed the visual data to calculate the center of mass (COM) of the target. As illustrated in Figure 7, the system captures the original RGB frame (Figure 7a), applies the HSV thresholds to generate a binary mask isolating the blue pixels (Figure 7b), and calculates the geometric COM of the remaining white pixels, indicated by the red marker (Figure 7c). The horizontal angle of this COM relative to the center of the camera's field of view served as the primary error metric. This angle error was fed back into the alignment loop to calculate the necessary PWM adjustment for the steering servo to keep the target centered in the frame.

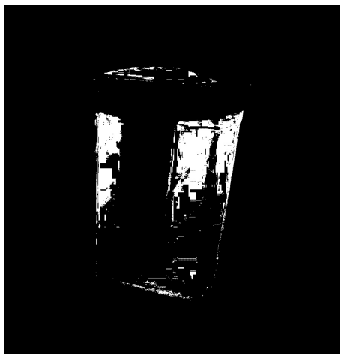
Once the car reached a minimal angle error threshold, it switched to the "Drive" state, setting the motor's PWM to the maximum duty cycle of 100%, allowing the car to move to the target as quickly as possible. Across the active driving states, a dynamic speed control loop was implemented based on the distance of the car from the object, read via the ultrasonic sensor.

As the car approached the object, the motor's PWM dynamically decreased. Upon crossing a minimal distance threshold, the system transitioned to the "Arrived" state. In this state, the motor's PWM was set to 0, safely stopping the car's movement to prevent a collision.

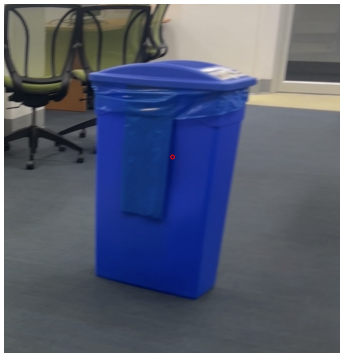
2) System Challenges and Performance Analysis: A significant challenge faced during this objective was the hardware delay between the high-frequency control loop and the physical frame rate of the camera. Operating the camera with `threaded=True` created a background data queue.



(a) Original RGB Capture



(b) Applied HSV Binary Mask



(c) Calculated Center of Mass (Red)

Fig. 7: Computer vision processing pipeline for target isolation and alignment.

However, the `while` loop executed faster than the camera could capture new frames, leading to empty values in the image variable and script failures. To resolve this, a safe-image wrapper function was implemented using a `try/except` block, allowing the loop to ignore dropped frames and continue executing the control logic without crashing.

3) Future Improvements: While the static HSV masking for isolating the blue target was effective under controlled lighting, it remains susceptible to inaccuracy when faced with changing lighting environments. A future improvement would be implementing dynamic HSV thresholding, allowing the system to actively widen or narrow its HSV bounds to combat

real-time changes in ambient lighting or surface glare.

B. Traffic Light

For Objective 4: Traffic Light, the task required the PiCar to autonomously drive in a straight line towards a simulated traffic light, interpreting its color to determine the appropriate driving behavior (Green to proceed, Yellow to slow down, and Red to stop) once it reached 3 feet from the light. This objective required the combination of three sensors: the Pi Camera for color recognition, the ultrasonic sensor for dynamic distance-based braking, and the MPU-6050 accelerometer to maintain a straight heading.

1) Design Strategy and Sensor Implementation: Similarly to Objective 3: Seeker, the control logic was divided into four primary states: Detecting, Driving, Object Close, and Arrived. To process the visual data, three distinct binary masks were created using `cv2.inRange` to isolate Green, Yellow, and Red pixels. Because the color red sits at both ends of the HSV spectrum, the red mask required a `cv2.bitwise_or` operation to combine the lower (0-10) and upper (160-180) hue ranges, as demonstrated in Figure 8. The system then calculated the total pixel count for each mask. If the highest pixel count exceeded a minimum threshold of 500 pixels, that color was chosen as the ”dominant” color.

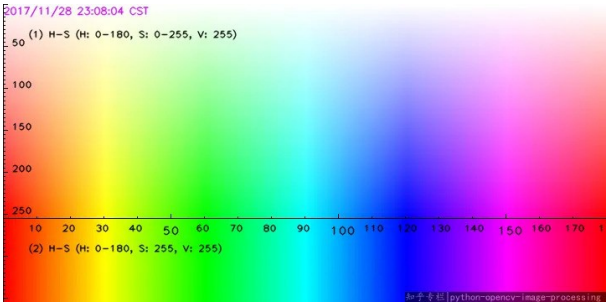
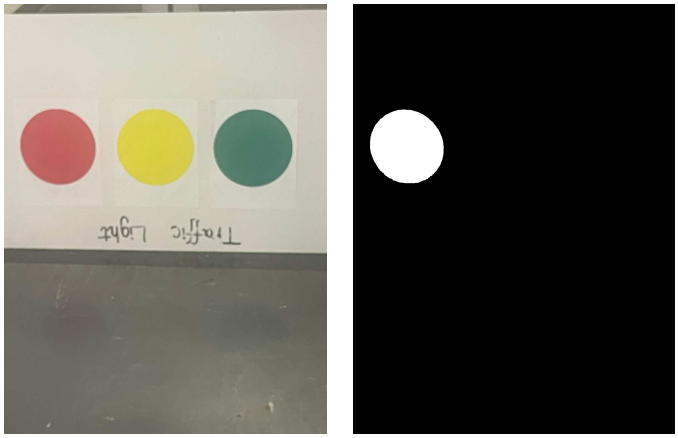


Fig. 8: OpenCV HSV Hue spectrum demonstrating the red color space split across the 0-10 and 160-180 bounds.

As shown in Figure 9, the system successfully isolates the printed colors from the background environment, allowing the vehicle to accurately count the pixel density of each color.

After a dominant color has been chosen, the car switches to the ”Driving” state. In this state, the car maintains a steady motor duty cycle towards the traffic light. If the camera detects that the dominant color is either Yellow or Red, and the ultrasonic sensor read distance is less than 200 cm, the system transitions to the ”Object Close” state. Here, a dynamic braking curve, similarly seen in Objective 3, was applied. If the light is read as Yellow, the motor’s duty cycle scales down proportionally as the distance decreases. If the light is read as Red, the vehicle halts to a complete stop once it reaches 170 cm and transitions to the ”Arrived” state until the light turns back to Green. As a cautionary fail-safe, once the vehicle reaches 80 cm or closer, it triggers an emergency stop regardless of the detected color. While these thresholds occur earlier (≈ 6.5 feet) than when the car reaches 3 feet, I



(a) Original Frame (b) Red Mask

(c) Yellow Mask (d) Green Mask

Fig. 9: Traffic light color isolation utilizing HSV thresholding.

accounted for inaccurate ultrasonic distance measurements as well as the car’s momentum.

To ensure that the vehicle drove straight toward the traffic light without drifting, a proportional steer controller was implemented using the accelerometer. By reading the raw Z-axis gyroscope and integrating it over time, the system calculates the current heading of the vehicle. The heading error and its rate of change were fed back into the steer controller to calculate readjustments for the steering servo, effectively allowing the vehicle to drive perfectly forward.

2) System Challenges and Performance Analysis: One significant challenge during the implementation of the HSV masks was the correct ranges for the yellow and green masks. The initial green mask failed to consistently detect the printed ”green” traffic light, and the initial yellow mask failed to narrow its mask down to only the printed ”yellow” traffic light, accounting for background noise such as the yellow carpet. Upon debugging the captured frames, I observed that the green mask was not detecting the ”green light”, as the printed ink had a heavy teal tint, and the room lighting desaturated the visual feed. To resolve this, the upper boundary was broadened for the green mask, and the upper boundary was narrowed

for the yellow mask, and the minimum Saturation and Value thresholds were lowered to accurately capture the colors in shadowed environments.

3) Future Improvements: The current vision system strictly relies on the pixel density of the current image. This means if a red, yellow, or green object exceeded 500 pixels for any of the three colors, it is assumed to be the traffic light. This makes the system susceptible to background objects with similar colors, such as a red shirt, or the yellow carpet on the floor. A future improvement could be implementing a shape analysis via OpenCV. This would require the vehicle to verify that the detected color is both dense in pixels *and* circular before executing any traffic light state transitions.

C. Cruise Control

The final objective, Cruise Control, served as the cumulative test of the PiCar's integrated hardware and software. The task was for the vehicle to autonomously drive up and down the hallway from Lopata under Sever. This hallway required the car to ascend a ramp, execute a 180-degree turn upon detecting a wall, and safely descend the ramp while actively maintaining a consistent desired speed.

1) Design Strategy and Sensor Implementation: This objective utilized the PID controller developed in earlier testing (Objective 2) to dynamically determine the motor's PWM output against physical resistance. Due to the new physical dynamics of Objective 5, new PID parameters were set in order to combat the gravity going up and down the ramp. While K_p remained the same, K_i and K_d were significantly raised (K_i increased from 7 to 15, and K_d increased from 0.1 to 0.8). The increase in K_i control helped the PiCar combat the slow RPS going up the hill, providing the push needed to gain the correct RPS, while the increase in the K_d control helped brake on the way down the ramp. To ensure the vehicle remained centered in the hallway and did not drift into the walls of the hall, the accelerometer was again utilized for continuous steering correction.

The ultrasonic sensor was utilized to determine when to turn 180 degrees. As the vehicle ascended the ramp and approached the back wall, the sensor continuously read back the distance. Once the distance threshold was reached, the main drive loop was temporarily suspended, and the vehicle entered a pre-programmed function to execute a three-point turn before resuming the PID-controlled driving. Once the vehicle reached the wall again, the variable `has_reversed` would be turned to true, allowing the car to stop and finish the script.

2) System Challenges and Data Analysis: In terms of the data logging, a gap in time occurred due to the execution of the three-point turn. The turning function relied on sequential `time.sleep()` hardware interrupts totaling approximately 14.8 seconds. Because these interruptions pause the execution of the main Python control loop, the data log is missing data from $\sim$14.3s to 28.8s.

Additionally, descending the ramp introduced physical challenges with the car's RPS overshooting. Gravity accelerated the vehicle beyond the PID's immediate braking capacity. To counteract this acceleration, an active brake block was implemented to slow down the vehicle and allow the PID values to retake control. If the RPS detected was greater than the desired RPS by 1.5 RPS, it would activate the active brakes, reversing the motor to slow the overall RPS.

As shown in Figure 10, the PID controller successfully managed the new physical challenges. During the uphill climb, the system exhibits slight expected undershoot as it fights gravity, and during the descent, it exhibits slight overshoot before the active braking stabilizes the vehicle, successfully maintaining a desired 3.5 RPS target throughout the entire run.

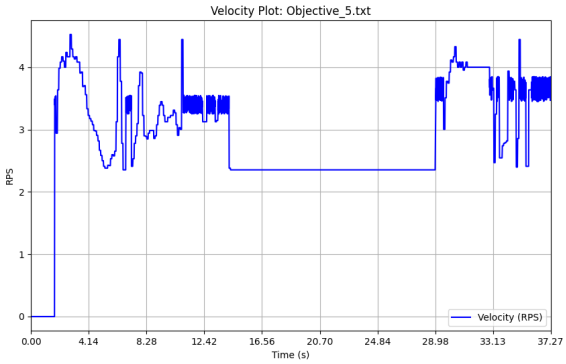


Fig. 10: Continuous velocity plot of the 3.5 RPS hallway run, exhibiting PID correction during uphill and downhill travel.

3) Future Improvements: A future improvement for the Cruise Control system would be a complete revision of the three-point turn function. While the open-loop turn functions effectively when the PiCar's batteries are fully charged, its accuracy degrades significantly as the battery depletes. Because the motor's mechanical output is directly dependent on the supply voltage, a lower charge means the same PWM duty cycle produces less rotational distance over the fixed time duration, resulting in an incomplete 180-degree turn. To improve the system's turning function, it should transition to a closed-loop maneuver, utilizing the accelerometer's integrated gyroscope to actively track the car's current heading and execute the turn until the necessary 180-degree threshold is reached.

VII. Conclusion

This study successfully demonstrates the integration of dynamic motor control with autonomous navigation. By first observing and analyzing the open-loop dynamics, a PID controller was developed and tuned to handle varying physical loads, friction, and environmental challenges such as slopes. The addition of computer vision via Pi Camera, ultrasonic distance tracking, and accelerometer-assisted steering control allowed the PiCar to become an autonomous system capable of dynamic target tracking, traffic light simulations, and tougher environmental terrain.