Group 309
Ryan Zimmitti & George Kopf
rz6542@princeton.edu & gk3946@princeton.edu

For speed control, navigation, and the final project, your team should upload two files to Canvas (one report per group):

- Lab report (PDF)
- A .zip file containing all code & designs (including .cydsn for PSoC, .ino for Arduino, etc.)

Statement of Objective:

Statement of Objective (5 pts.)
Speed Control: Your car must travel along: (i) a straight path, (ii) uphill, and (iii) downhill at approximately 4 ft/sec. We allow a 2% margin of error for the straight path and 10% for uphill/downhill parts. As a result, for completion (25% of your overall grade for this part), your time needs to be within [9.8, 10.2] seconds for the straight path and [8.1, 9.9] seconds for both uphill and downhill sections.

Overview of Key Subsystems & Components:

Overview and brief explanation of key sub-systems and components (30 pts.)

- Terminal Block - The terminal block is the main hub of our incoming battery wires. It serves as a physical organizer, taking in the 9.6 and 7.2 V inputs and allowing easy distribution to other parts of the car. It also serves as the main physical hub of the car's common ground.
    - Having the grounds of the two batteries hooked together in the terminal block keeps their electric potentials in reference to the same point, which avoids voltage differences that could damage components. This ground also acts as an easy anchor for every other ground on the car, so they're all connected in one place.
    - The 7.2 V battery ground wire goes through a single pole, double throw switch into the terminal block. When the switch is turned on, the 7.2 V connects to the Power MOSFET Board and is sent through to the motor (as dictated by the MOSFET, which we'll get into later). This allows the motor to run at a pulse width modulated 7.2 V.
    - The 9.6 V battery is directly connected to PSoC's 9 V connector. Thus, our PSoC is continually powered unless the 9 V snap connector is unplugged (which is our simple but effective method of turning the PSoC off).
        - We make sure to unplug the snap connector whenever the PSoC is plugged into a computer through USB, because that could fry the USB port.

    - The 9.6 V battery also sends voltage through a double pole double throw switch that connects it to the Voltage Regulator Board. When the switch is on, the Voltage Regulator Board powers the Hall-Effect Board and the Servo Motor.

- Hall-Effect Sensor and Board - The purpose of this board is to read in the rotation rate of the wheel in terms of how fast the embedded magnets pass the Hall-Effect sensor.
    - The Hall-Effect sensor is glued into the casing attached to the wheel, where it directly senses the magnetic fields of the five embedded magnets. It has a 5 V in wire, signal wire, and ground wire all going up to the board.
    - There is a pull-up resistor between the Hall signal out and the 5 V power line on the Hall-Effect Board. This resistor means the system avoids floating voltage values when the Hall-Effect sensor is not registering any magnetism, instead pulling our signal line up to 5 V (because there is no current across the resistor). When the Hall-Effect sensor *does* pick up a magnet, the output signal line drops to 0 V.
        - The PSoC has a digital NOT gate reading in these values, meaning our code sees rising edges in the binary input signal as the appearance of a magnet.

- Voltage Regulator Board - The voltage regulator board takes the incoming 9.6 V from the battery and drops it to lower voltages for the Hall-Effect Board (5 V) and the Servo Motor (6 V). This serves two purposes. The first is that it lowers the voltage to the specifications required for the specific components. The Hall-Effect sensor is designed to run at 5 V, not 9.6 V, so it's important to drop the incoming voltage. The second purpose has to do with the battery losing charge, as its output voltage will change over time. In fact, the 9.6 V battery fluctuates from over 11 V down to under 8 V before it's time to recharge it. Using a higher voltage battery to run lower voltage systems gives us some breathing room. Even as the battery loses charge, we'll still send a steady voltage supply to our components.
    - The LM7805 & LM7806 voltage regulators are the specific regulators that set our voltages to 5 V and 6 V respectively. We have two LM7805 (one for the Hall-Effect Board and one for the video board (not attached yet)). And one LM7806 (for the Servo Motor).
        - Each voltage regulator has two capacitors attached to it, one 0.33 uF capacitor from input to ground and one 100 nF capacitor from output to ground. These capacitors smooth the voltage output and help avoid oscillations.
    - The servo motor signal wire also attaches to the Voltage Regulator Board. This signal runs to our wireless transmitter that connects to the remote control, allowing for manual steering during the Speed Control demo. In the future, we'll

replace this with a signal fromPSoC, allowing the car to steer itself via servo control.

- Power MOSFET Board - The Power MOSFET Board handles the control of the driving motors. It takes PWM signals in from the PSoC, which control the activation of the MOSFET. The MOSFET toggles on and off quickly, allowing current to flow through the 7.2 V → battery → ground circuit in a PWM fashion. This modulating signal then powers the motor (at various speeds depending on the duty cycle).
    - We used an IRLZ44N MOSFET due to its very low voltage threshold of 1-2 V. This is great, because our 5 V PWM signal coming in from the PSoC will strongly activate the MOSFET, causing full switching which results in a full PWM signal being passed to the motor. This also minimizes energy loss through heat as the MOSFET does not get partially switched on.
        - We use two resistors between our signal line and the MOSFET: a 10k ohm resistor from signal to source and a 100 ohm resistor from signal to gate. When the signal line is active (charging or discharging the gate capacitor), it runs through the 100 ohm resistor to gate. When the signal line is empty, the 10k resistor pulls gate down to zero at source (ground) to avoid floating voltages.
    - The motor has an inherent inductance, and when the 7.2 V in signal stops, a current can be induced in the wires that pushes back to the MOSFET due to the collapsing magnetic field. This is not good for our MOSFET and can burn it out. The one-direction 1n4005 flyback diode redirects this “backflow” of energy back into the motor windings where it dissipates as heat in that wiring and the diode, keeping the rest of the car safe.
    - The amount of current coming into a motor determines the speed at which it spins. This is of course controlled by the voltage set across the motor and the motor’s resistance (V/R=I). However, implementing variable voltages is very difficult and involves power electronics. To circumvent this, we use PWM. Due to the inherent inductance of the motor, it continues to be powered between peaks on the incoming PWM signal as the inductor discharges. Thus, the motor receives the equivalent of an average voltage value. Changing the duty cycle of the PWM signal changes what this average works out to, causing the motor to change speed.

- PSoC - The PSoC is the brains of the operation, but the actual hardware here is relatively simple.
    - The PSoC runs directly on 9.6 V from the battery.
    - There are three pins wired into the PSoC.
        - The Hall-Effect sensor signal input comes from the Hall-Effect Board. It delivers a pulse to the PSoC whose period is determined by how fast the

magnets in the wheel move past the Hall-Effect sensor. The PSoC uses the time between magnet triggers, along with the hard-coded diameter of the wheels, to calculate the amount of distance covered per unit time, giving us our speed in ft/sec.

- The motor PWM signal output comes from the PSoC to the Power MOSFET Board, where it controls the speed of the motors. We control the duty cycle of the PWM signal coming from the PSoC using PI-control to increase the motor speed when the car is below 4 ft/s and decrease it when the car is above 4 ft/s.
- The servo-controller signal output would come from the PSoC to the Voltage Regulator Board, to connect to the servo motor and control the turning of the car. However, the Speed Control demo uses manually operated turning, so this wire was unused (and disconnected) for the demo.

- The XBee came pre-wired into the car, and allowed us to print serial output to our computers for easy data collection. We had it print out several key variables, such as our PWM value and our PI-variables, which allowed for more efficient tuning.

Calculations:

- Wheel speed:
  Cp: $previous\ clock$, Cn: $current\ clock$, d: $wheel\ diameter$, n_m: $number\ of\ magnets$

  PSOC timer counts down, we need $(C_p - C_n)$ for change in time

  $\triangle time\ (s),\ time\ between\ each\ hall\ measurement\ dt = (Cp - Cn)\ ticks\ /(1 * 10^4\ ticks/sec)$
  Convert clock time to seconds → Clock is 10 KHz → we divide dt / 10^4 to get seconds

  $Distance\ (ft)\ traveled\ by\ wheel\ for\ each\ hall\ measurement$: $\frac{\pi * d}{n_m}$

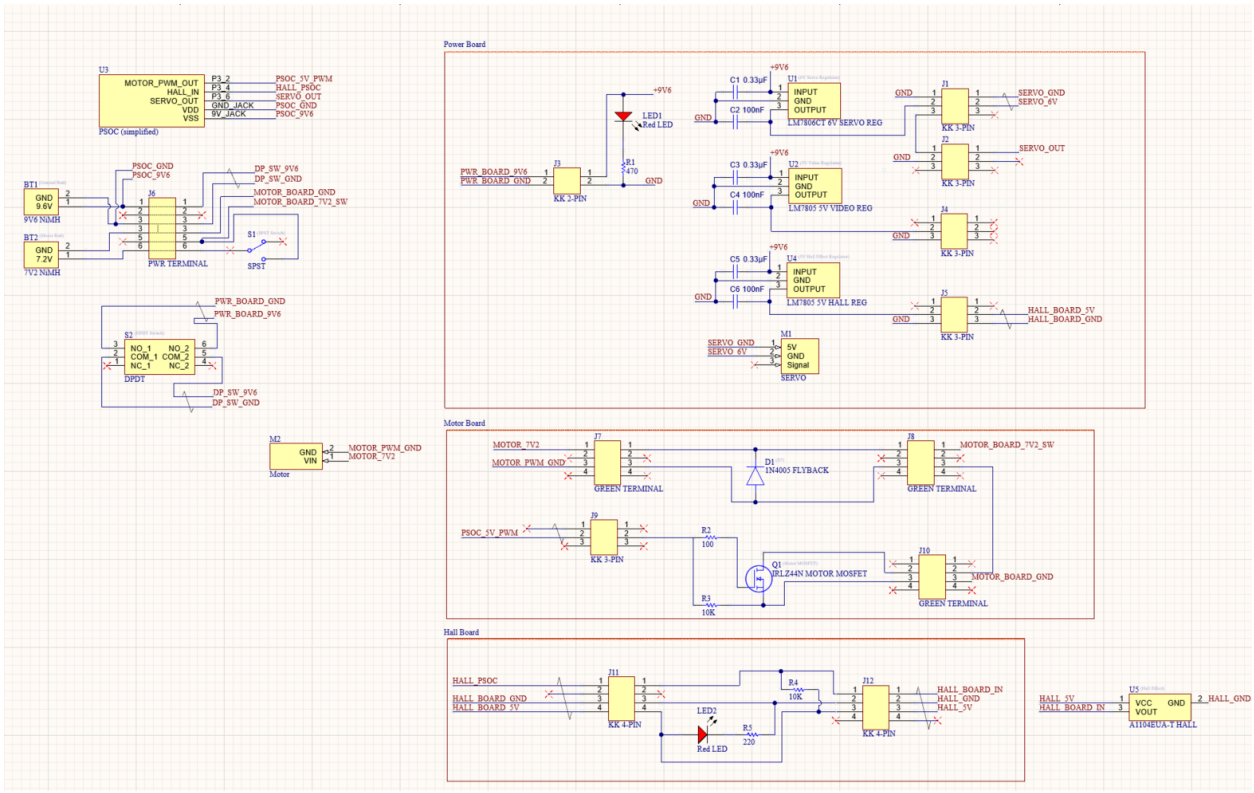
  $Speed\ (ft/s)$: $\triangle distance/\triangle time = \frac{\pi * d}{n_m} / dt$

Schematic Diagrams:

Schematic diagram of circuits (computer-drawn, try CircuitLab, etc.) (15 pts.)

Components:

- Power board:
  - 2x LM7805
  - 1x LM7806
  - 1x Red LED
  - 3x 0.33µF ceramic capacitor
  - 3x 100nF ceramic capacitor
  - 4x 3-pin kk connector
  - 1x 2-pin kk connector
- Motor board:
  - 1x IRLZ44N MOSFET
  - 1x 100 ohm
  - 1x 10k ohm
  - 1x 3-pin kk
  - 1n4005 diode
  - 4x green screw terminal block
- Hall effect:
  - 2x 4-pin kk connector
  - 1x 10k ohm resistor
  - 1x 220 ohm resistor
  - 1x Red LED
  - A1104 Hall Effect
- Other:
  - 1x PSOC
  - 1x Motor
  - 1x Servo Motor
  - 1x Single Pull Single Throw switch
  - 1x Double Pull Double Throw switch
  - 1x 7.2V Nimh
  - 1x 9.6V Nimh

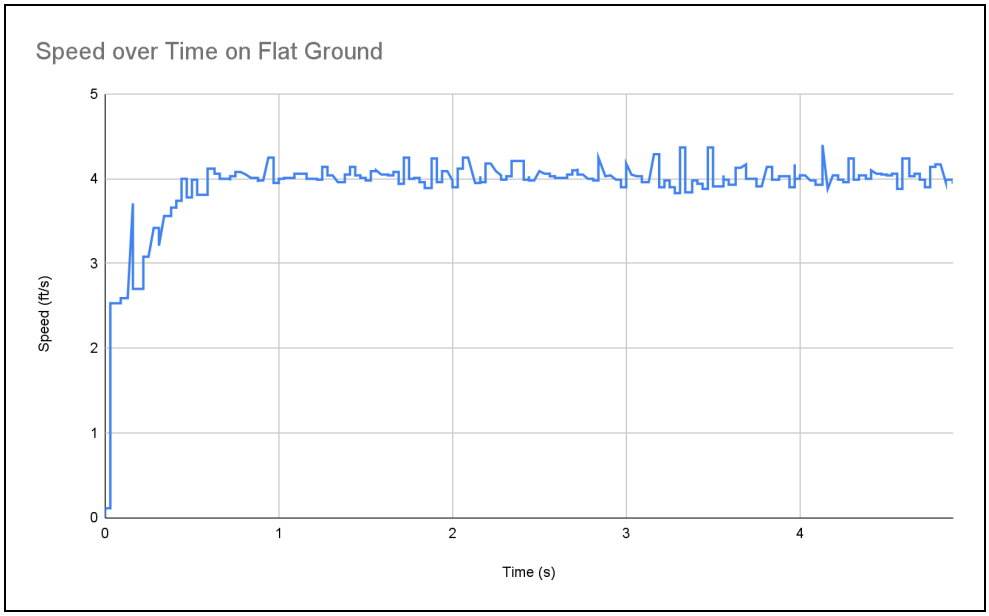


https://drive.google.com/file/d/1ifyV5jgB5bvW0BNDfbNNsRAMMTJ0o976/view?usp=sharing

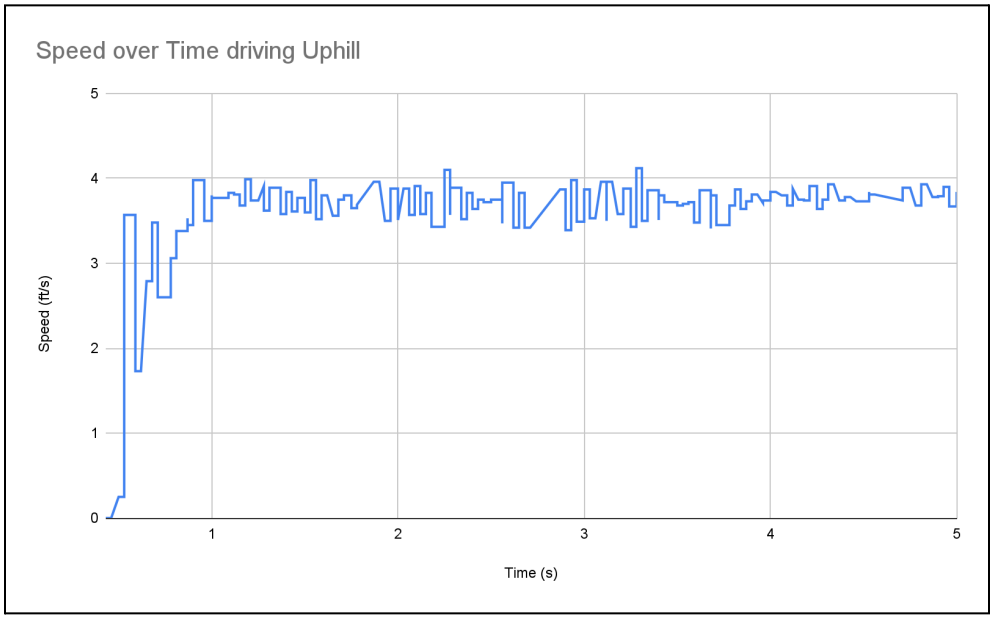
Data:

Data showing that it is working as you expect. Include explanations for this. (25 pts.)

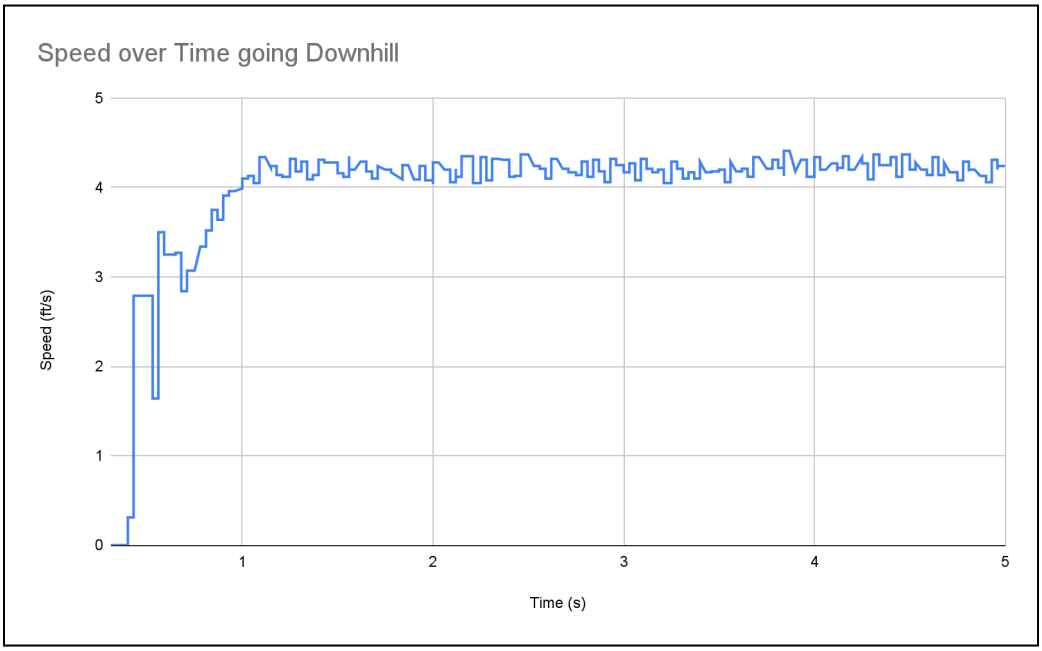
- Plot showing that your speed settles to around 4 ft/sec for straight, uphill, and downhill sections (use XBee to gather data).
    - Straight:



        - The straight drive graph looks pretty much as expected. The car jumps up to 4 ft/s in about half a second, then sits around that value, oscillating a bit but generally pretty stable. The car passed the demo with flying colors at 10.03 seconds.
    - Uphill:



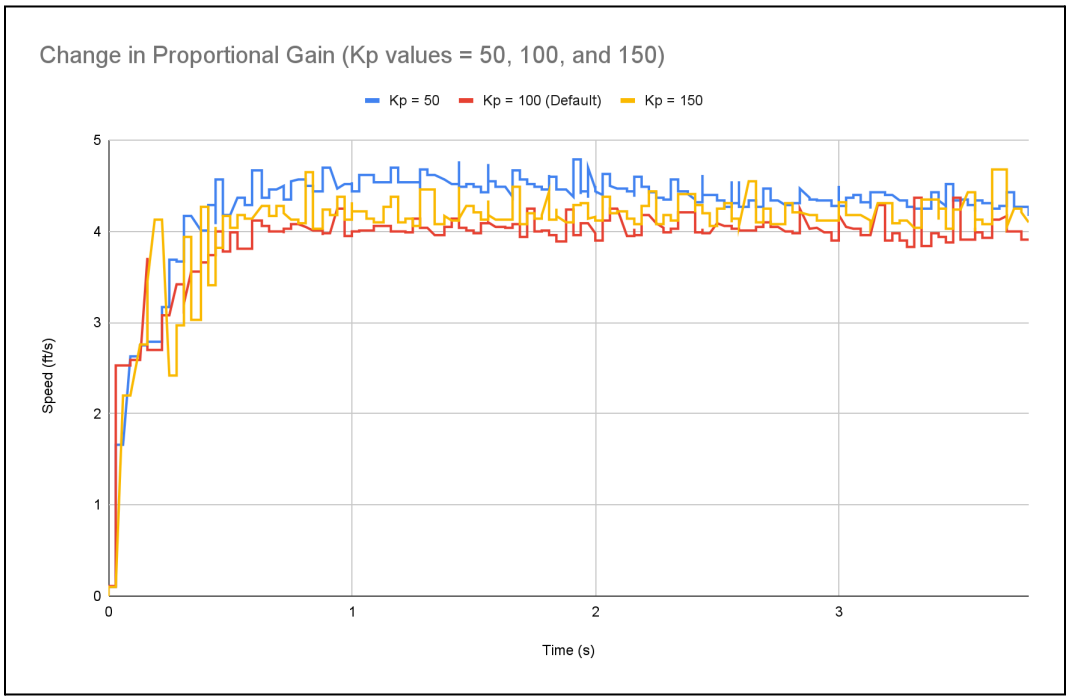
        - The uphill drive graph also looks good. The car struggles a bit at the start, over and undershooting a bit until it settles just under 4 ft/s. It spends the entire drive attempting to reach 4 ft/s, but every time it does, it slows down way faster than it "expects" to, meaning the car is continually playing catch up. Despite this, coming in at just under 4 ft/s was fast enough to pass the demo within the given time frame, at 9.81 seconds.
    - Downhill:



        - Similarly, the downhill graph does the same in the other direction. The car takes a little longer to reach 4 ft/s than in other scenarios, because it keeps slamming back the PWM value as it accelerates. However, it settles just above 4 ft/s, and continually attempts to slow down. Everytime it gets close, it adds a little gas and accelerates way quicker than it normally would. Driving just over 4 ft/s, the car managed to pass the demo within the time frame here too, completing the ramp in a comfortable 9.31 seconds. The longer acceleration time helped offset the higher speed.

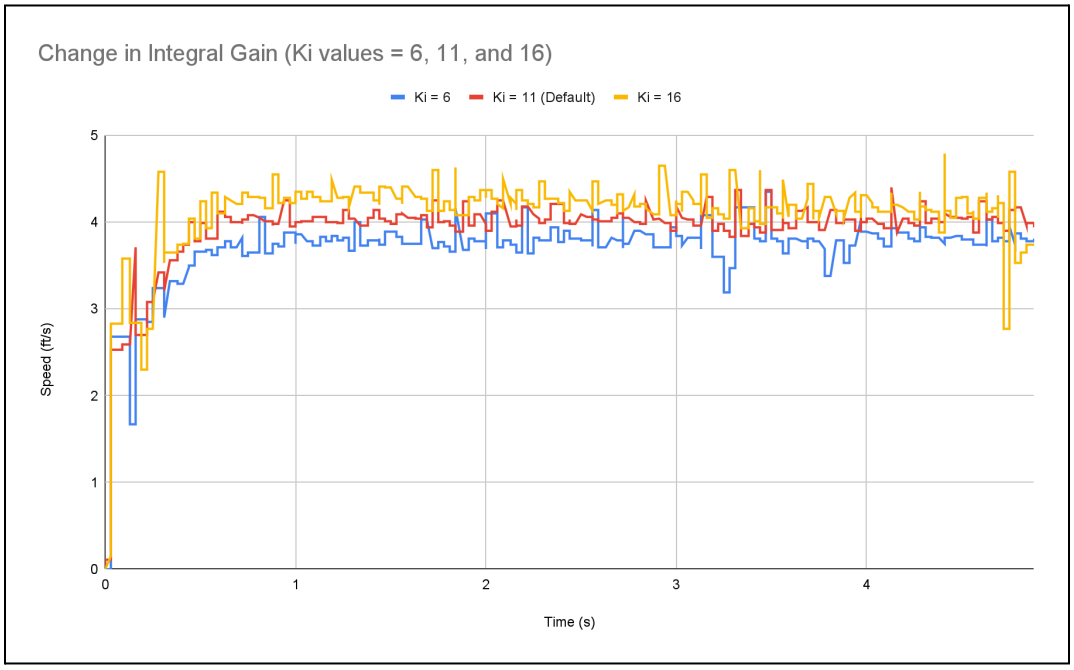
- Plot showing what happens as you increase/decrease your control parameters (e.g., kp, kd, ki, whichever you end up incorporating). For this part, you may place your car on the mount (free running) and acquire data (no need to test it on flat, uphill, and downhill sections).
    - Our car passed the speed demo using Kp = 100 and Ki = 11 for all tests. We tested each parameter at roughly 50% and 150% for comparison.

- Proportional Gain:



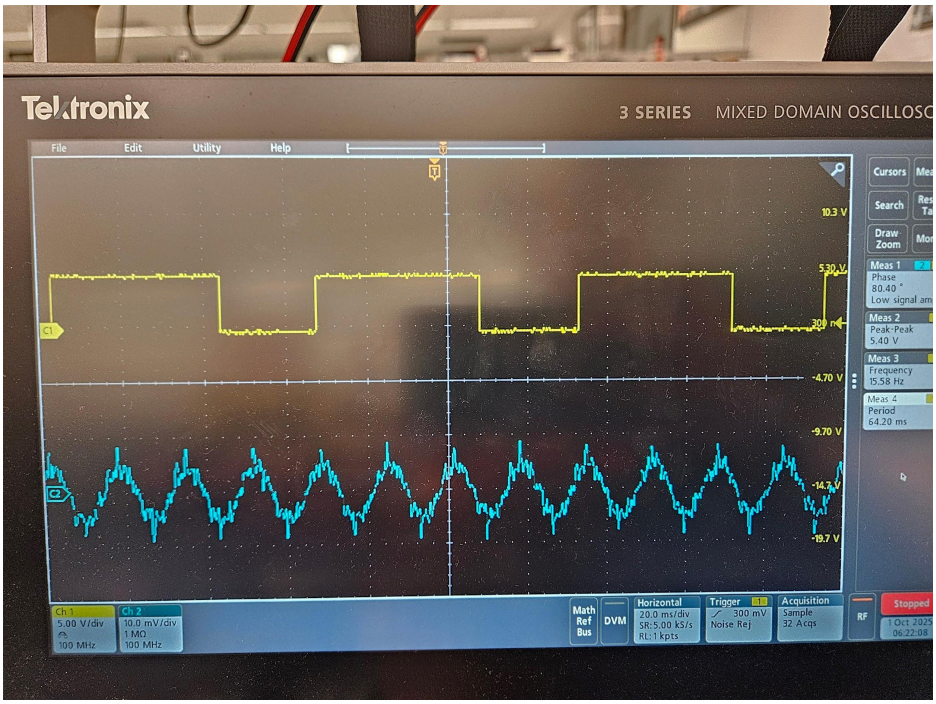
  - The proportional gain value is fairly straightforward. The bigger Kp is, the more aggressively the car will respond to any given amount of error. At Kp = 150, we can see this in the large oscillations as the PI system swings back and forth past its target. The car reacts very aggressively to slowing down, so we get these large spikes compared to other scenarios.
  - When Kp = 50, the car reacts more sluggishly to things. In fact, we can see that the car way overshoots the target of 4 ft/s and sits around 4.5 ft/s for several seconds. It slowly descends, but the reason we see this behavior is due to the now comparatively high Ki. The integral term generates a bunch of error and spikes our system to drive much faster than it needs to. However, once we overshoot, the integral term has to take time to offset the error it already iteratively built up. Our low proportional term means the system does not react aggressively to the raw error, so we get this result where the system oscillates around ~4.5 before slowly descending.
  - At Kp = 100 (the value we tuned our system to), our car avoids the worst of the overshooting of the high Kp value and the sluggishness of the high Ki value.

- Integral Gain:



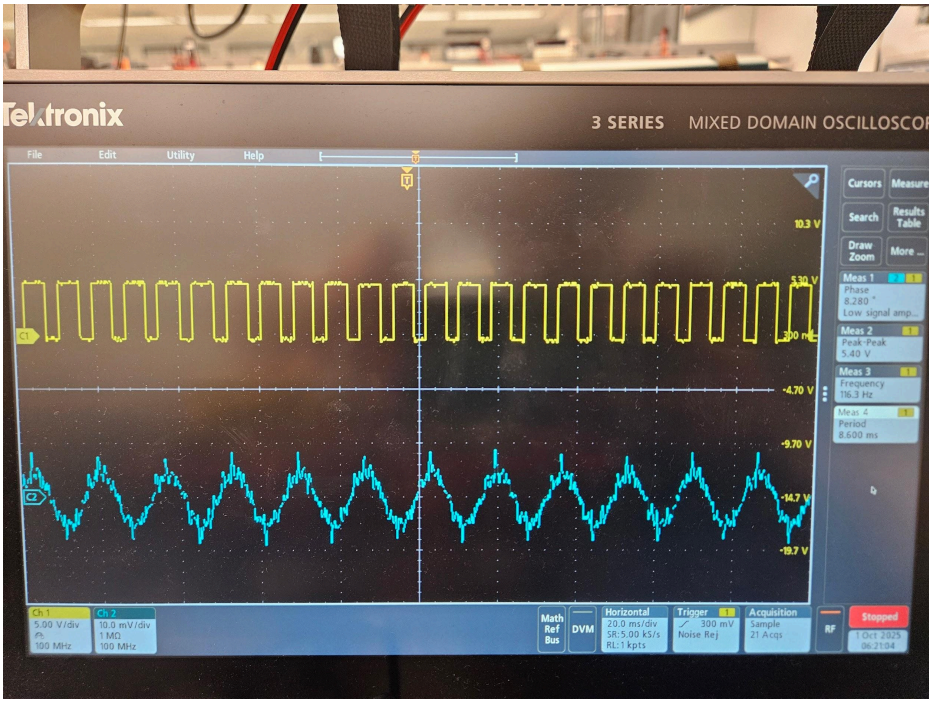
  - The integral term drives the system towards target, stacking cumulative error whenever the system is off. This eliminates steady state error, but can induce oscillations in the system. At Ki = 16, this large gain means our system is very responsive to this error. However, it also means we're slow to correct overshoots (because we need to cancel out the old error values before the integral term can start moving the other direction), which is why we get this large overshoot. If we continued to increase Ki, we would likely see low frequency oscillations induced in the system by large over and undershoots.
  - At Ki = 6, the low relative gain means our system doesn't respond to the cumulative error very effectively, causing it to sit at a consistent undershoot of the target speed.
  - At Ki = 11 (the value we tuned Ki to), things look good. Oscillations are minimum and the system effectively hovers around 4 ft/s.

- Scope trace from the hall-effect output for 3 speeds (low, moderate, high) when the car is free running.
    - **Slow:** PWM_WriteCompare(15);

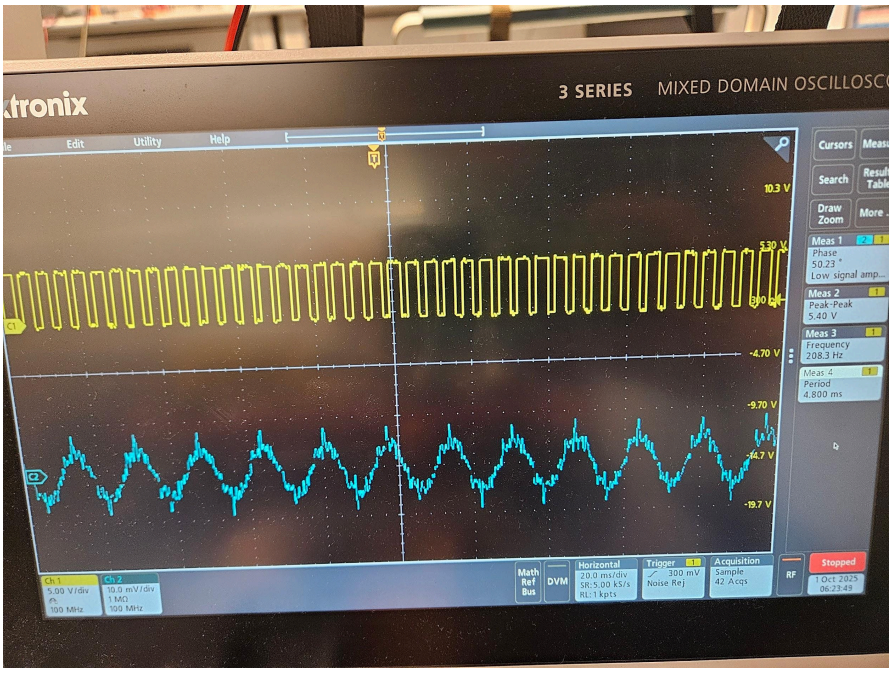


        - Period = 64.2 ms
        - Frequency = 15.58 Hz

    - **Medium:** PWM_WriteCompare(50);

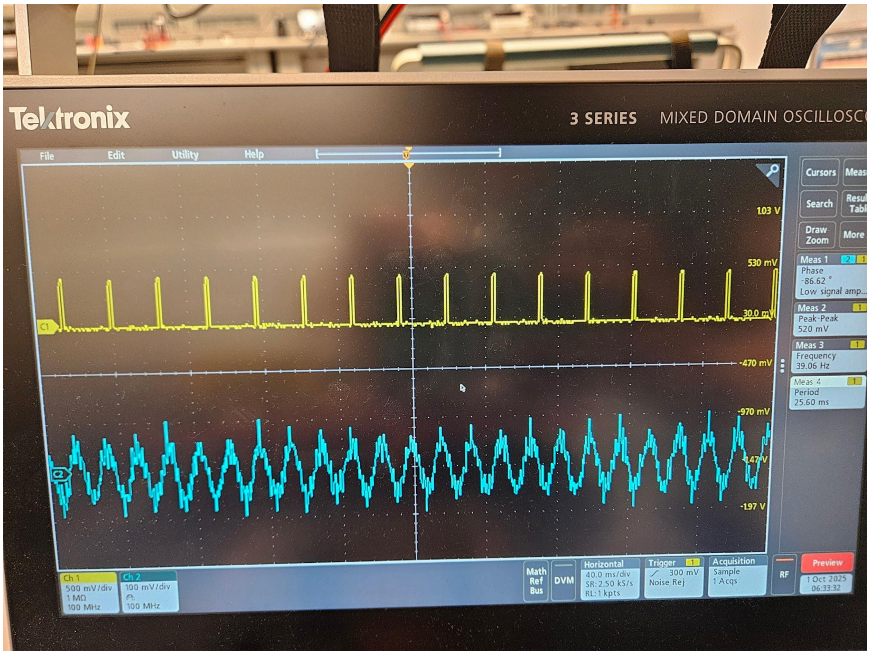


        - Period = 8.6 ms
        - Frequency = 116.3 Hz
    - **Fast:** PWM_WriteCompare(200);

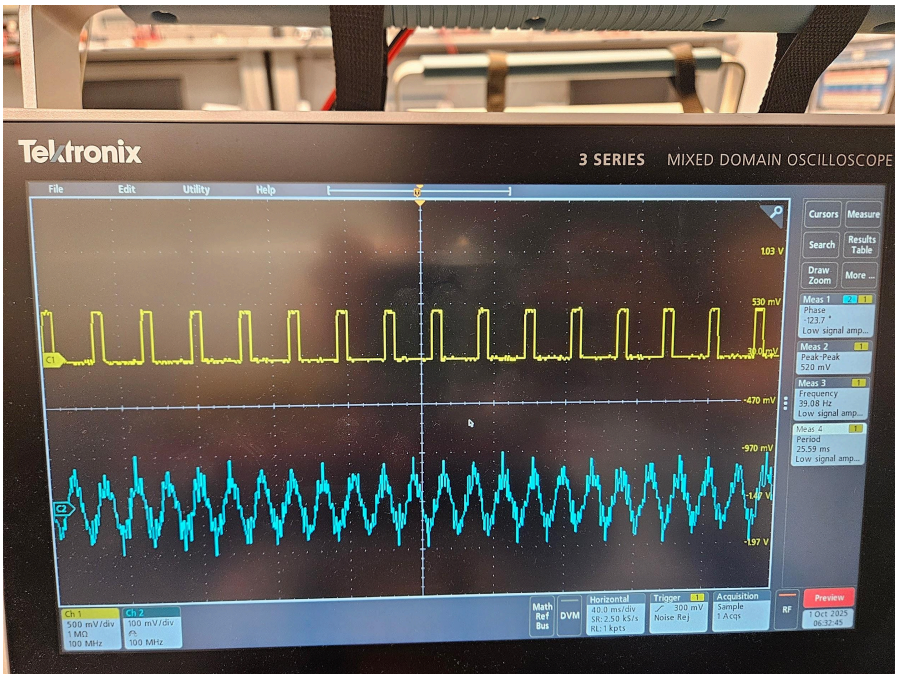


        - Period = 4.8 ms
        - Frequency = 208.3 Hz

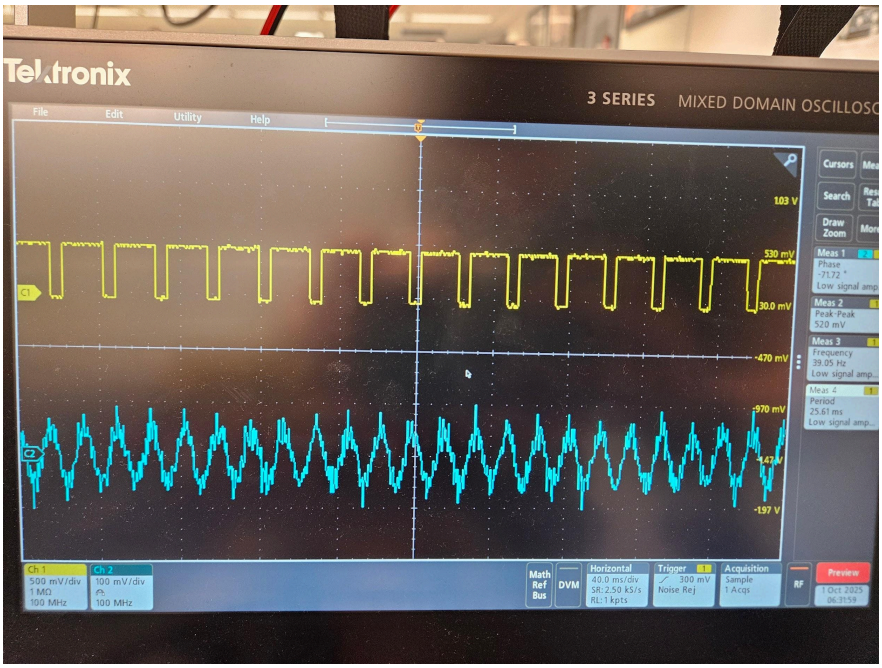
- Scope trace for the PWM signal that the PSoC is sending out for the 3 speeds above
    - **Slow:** PWM_WriteCompare(15);
        - Duty cycle = 15/255 = 5.9%



- **Medium:** PWM_WriteCompare(50);
  - Duty cycle = 50/255 = 19.6%



- **Fast:** PWM_WriteCompare(200);
  - Duty cycle = 200/255 = 78.4%



Discussion:

Discussion of challenges and design choices you made (10 pts.)

As with all good projects, we faced a number of challenges that colored the design decisions made during the process. Overcoming or circumventing these issues were critical to the success of our car.

Choosing the appropriate power MOSFET was a difficult task. First, we had to properly identify the voltage range of signals coming out of the PSoC. Measuring test PWM impulses on our oscilloscope allowed us to determine the peak voltage to be 5 volts. Thus we needed a MOSFET with a gate threshold below 5 volts (and a max voltage well above 5, which essentially every MOSFET satisfied). Initially, we chose the M75 No3HDL, with a gate threshold of 2 to 3 volts. However, we did not realize that the range in gate threshold is due to manufacturer error, and that it was a random chance whether our MOSFET triggered closer to 2 or 3. Unfortunately, the MOSFET we used had too high of a gate threshold voltage, which meant our car's acceleration was very slow and the MOSFET generated a significant amount of heat (additionally, scans with the oscilloscope showed the PWM input we were sending to the motor through the MOSFET was very jagged, which is bad for accurate control). To rectify this, we switched to the IRLZ44N, with a gate threshold voltage of 1 to 2 volts. This lower threshold was much better for triggering precisely on PWM peaks, allowing our car to accelerate quickly. Oscilloscope scans showed clean PWM signals coming through, and the MOSFET stayed relatively cool throughout testing. The IRLZ44N was the way to go.

Another design choice on the car was using LEDs to indicate if a given board was powered. We included two red LEDs: one for the power board, and one for the hall effect board. These were used as a sanity check to help with debugging what could be wrong with the car. For instance, when our hall effect sensor seemed to be malfunctioning, we became very worried, but immediately noticed the LED was off, pointing us to realize we had just disconnected the power to the hall board.

On the choice of values for the various resistors and capacitors, it mostly came down to the recommendations of datasheets. Each voltage regulator has two capacitors as mentioned earlier, and the values of them were recommended by the datasheet. The resistor values for the LEDs were based mainly on what resistors we had around our bench under 1000 ohms, as this leads to a fairly bright illumination. The twisted pairs for most of the signal wires was a recommendation by the lab instructors. If the wires were carrying differential signals, the twisting would have minimized EMI, but the pairs in our car do not carry differential signals and were mainly for decorative purposes and routing. The cables were routed through the center notch in the car whenever possible to decrease clutter and avoid potentially unwanted wire contact. We also went out of our way to keep the battery cables under the boards and avoided looping higher voltage cables over the top of the car or around the side again to reduce clutter and avoid unwanted connections.

When initially creating the proportional control system, we were not sure where to clamp the motor's PWM range. The upper limit was pretty simple. Capping the duty cycle to about

80% left us with more than enough speed to complete the given challenge. The lower limit was more complicated. Initially, we wanted to keep our minimum duty cycle to around 10%, because the car tended to stall when first turned on. This worked great for flat driving, but when we tested on the downhill ramp we realized our PWM_control value was dropping to minimum (and we were still going too fast!). To fix this, we decided to allow the duty cycle to drop all the way to 0% (thus turning the motor completely off). This worked great for giving our car the capability to fully "hit the brakes" on the way down, but still left us with the stalling problem. We ended up solving this by increasing the P gain in our PI control, which gave our car the aggressive kick it needed to power through right off the get go.

Speaking of PI control, originally we only designed a proportional control system. We spent a long time tuning K_p in an attempt to pass the flat ground speed trial, and eventually got it working with relative consistency. However, when we attempted up- and downhill on the ramp it failed spectacularly. We could not increase K_P enough to power uphill without ruining our flat and downhill runs. Thus, we decided to add an integral term, making it a PI control. The integral term immediately made our car super sluggish, so we set K_I to a tiny number and began working our way up from there. By printing PI control values to UART, we were able to get through a lot of educated trial and error. This allowed us to narrow in on an integral gain value that properly brought error down to ~0. However, our car was sluggish to start again. To offset this, we doubled K_P, which of course made our integral response worse. We increased K_I a little bit, were happy with the results, then tested K_I and K_P values in that range until we found the gains that led to the most consistent results on the flat speed test. We took our car to the ramp, and it succeeded multiple times both uphill and downhill. Thus, happy with our control code, we passed the demo.

Improvements:

What could be improved (10 pts.)

Given the opportunity, there are a lot of things with the car that could be improved. First of all, attaching the 9.6 volt power in lines directly from the terminal block to the PSoC made it very annoying to program. We had to manually detach the 9v connector plug from the PSoC, then plug in the USB serial port every time we wanted to update the code (which was very annoying when we were tuning our PI control gains). In the class Slack, Divija announced that she figured out a method to directly update gain values through the XBee to the PSoC. It would be very nice to implement this in our code to expedite the tuning process.

Additionally, the plastic wheels on our car seemed to have less traction than some of the other cars' rubber wheels. Right at the start when our relatively high-gain PI control spikes, the car jumps forward, which can cause the car to slip a little. It's not a big deal, but it definitely affects our run times slightly.

The PI control works, but including a derivative term could make our system more responsive to big changes. Our car is not perfectly consistent with the ramp tests, and adding a derivative term could really make our speed control better. It would likely make the control

system more robust against new challenges, like variable terrain, more steep hills, and turns (which slow the car down a lot).

The Hall-Effect sensor rig we have measuring tire rotation rate works, but is a little fragile as a system. On the oscilloscope, we see gaps in the output signal every couple of seconds where the Hall-Effect misses a magnet. This error is small enough that it did not really affect our speed control, but it could be entirely eliminated by replacing the Hall-Effect sensor with a proper motor encoder (which functions according to the same principles but would be a more robust component).

Lastly, swapping out the PSoC for a Teensy Arduino would simplify the code greatly. The PSoC does have its advantages; mainly its analog integration that can offload high-speed tasks from the main CPU to dedicated hardware logic. However, the Teensy annihilates the PSoC in terms of processing power, and is much simpler to code. Given the chance, we could compare whether the loss in the deterministic consistency of the PSoC would be worth the improved performance and faster clock times of the Teensy. In addition, the Teensy is simpler to code, which would allow for faster iteration time in software development. Simplified PID logic, direct output to log files over serial using python scripts, and access to the full Arduino catalogue of libraries are all benefits that could make our software efficient enough to outweigh the analog hardware benefits of the PSoC (although, of course, we understand how educationally valuable it is to force us to work with the PSoC).

Adequate comments in your code (5 pts.) ✅