Inertial Measurement Hardware for Translational Drift Robots

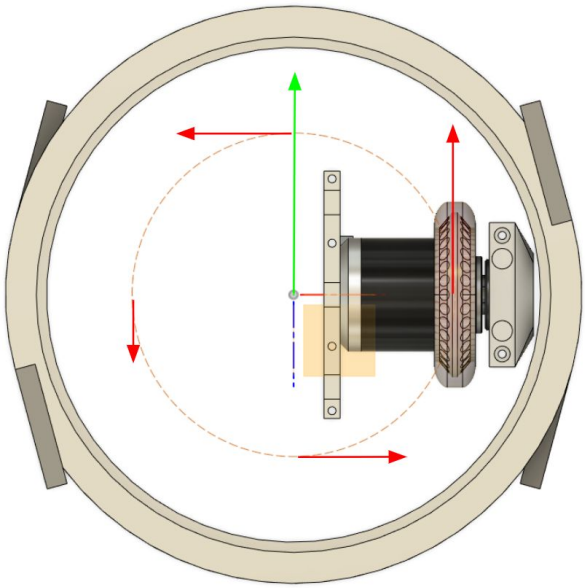
6.101 Project Report

Jackson Gray

Motivation

In the world of combat robotics, even the most optimal robots have to strike a balance between armor, agility, weapon effectiveness, and reliability. As most events feature weight limited classes, builders frequently find themselves having to decide early on in the design how much of their weight to allocate to a larger, more powerful weapon, or heavier armor and frame to be able to withstand more damage. Kinetic energy "shell/overhead spinner" type robots make use of various shapes of heavy metal bars, disks, and toothed domes spun planar to the ground and overhead of the robot. This robot type has one key optimization - their weapon functions as both an offence and a defence. When the weapon is spinning, it becomes very difficult to reach the robot below. This means that the chassis of the robot can be built very light and without armor, it is not uncommon for these types of robots whose weapon makes up over 50% of its weight. However, this still means that 50% of the robot weight is not contributing to either offense or defence. A Translational Drift horizontal spinner (colloquially known as a "meltybrain" robot) further optimizes this weight utilization problem - by using the body of the robot as a spinning weapon, 100% of the mass of the robot contributes to its overall stored kinetic energy.

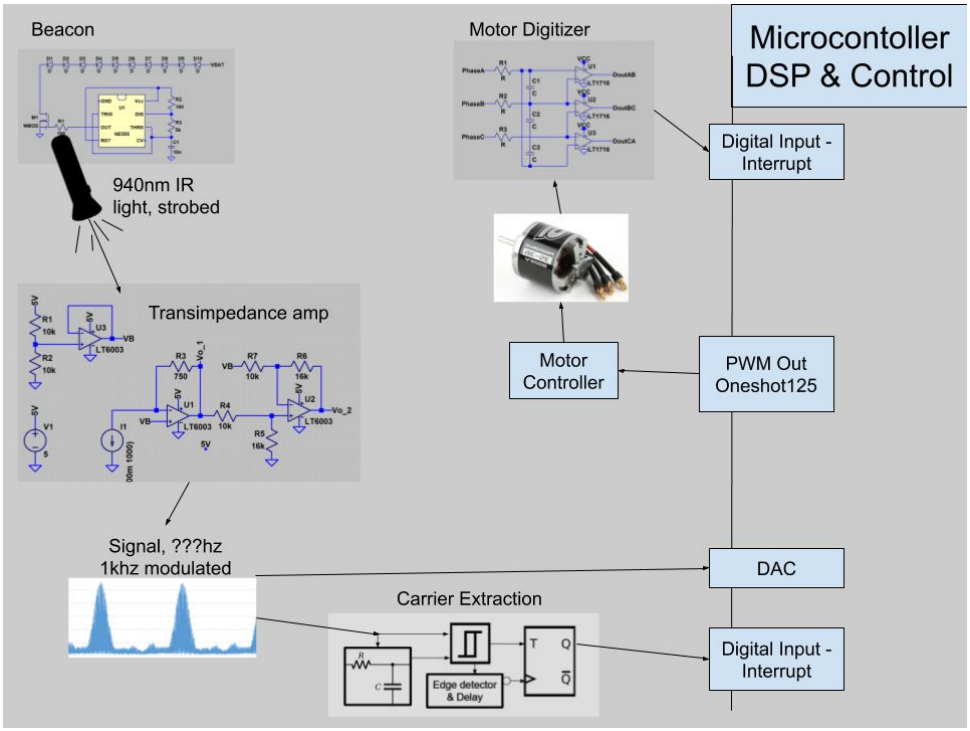
Translational Drift is a classification of robot locomotion that makes use of the rapid spin of the robot to translate across the floor in an arbitrary direction. With a single motor and wheel oriented tangent to the rotation, the robot modulates the wheel torque at the same frequency as the rotational frequency, which results in translation of the robot in a direction dependent on the phase of the applied torque. This locomotion method is particularly applicable in combat robotics, as the entire robot body is spun at a high speed while only requiring a single motor, reducing mechanical complexity and points of failure. However, while most drivers can remotely control each individual degree of freedom of their combat robot, it isn't possible for the driver of a translational drift robot to modulate the motor torque at the robot's rotational frequency. Precise computer control and reliable inertial measurements are required for an effective implementation of translational drift, and as a result meltybrain robots haven't seen much success to date.



Motor, wheel, and shell of a meltybrain robot. As the wheel travels in a circle around the center of mass, the torque is continuously modulated (red vectors), which sum to a net force applied (green vector) to the robot, causing it to drift in that direction

General Design & Goals

In this project I intend on focusing on the inertial measurement system specifically. Having tackled this type of robot in the past and having learned from previous attempts, I can attest that reliable and useful measurement of the robot's absolute (relative to the driver) rotational position in adverse conditions can make or break the performance of a meltybrain robot. Therefore, I propose a new inertial measurement solution composed of two analog circuit modules.



Block diagram of inertial measurement system

The first utilizes a IR photodiode to look for a bright, strobing IR LED array, located under the robot driver's remote controller. If the photodiode is oriented outwards, as the robot rotates, the received IR intensity from the strobing beacon will increase and decrease over one rotation (the "signal") of the robot. This allows the robot to resolve its absolute rotational heading relative to the driver. To achieve this, the photodiode current must first be amplified and transformed into a voltage signal, and then the beacon's strobing (the signal's "carrier frequency") must be identified for the microcontroller to properly sample and analyze the received signal.

The second estimates the speed of rotation of the robot's wheel via the voltage waveforms out of the motor, allowing the microcontroller to roughly know the robot's velocity. This makes the microcontroller's job of identifying the periodic signal much easier, as it would already have a reasonable estimation of the frequency of the periodic signal. Without this, the robot may face issues with falsely identifying reflections of the original signal and try to at 2x or 3x the original frequency. This estimation of the robot's rotational velocity may also be integrated to provide a fall back system in case the IR is being interfered with or is otherwise not receiving sufficient signal.

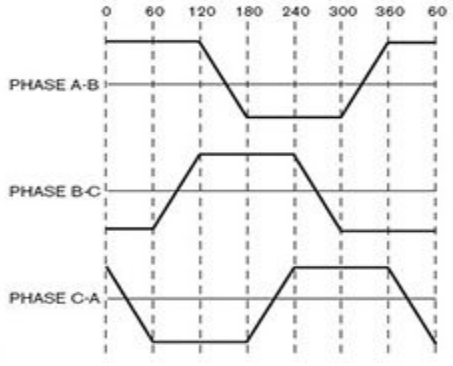
Both of these systems need to reliably operate across a range of signal strengths and rotational velocities.

Module Details

Motor velocity encoder

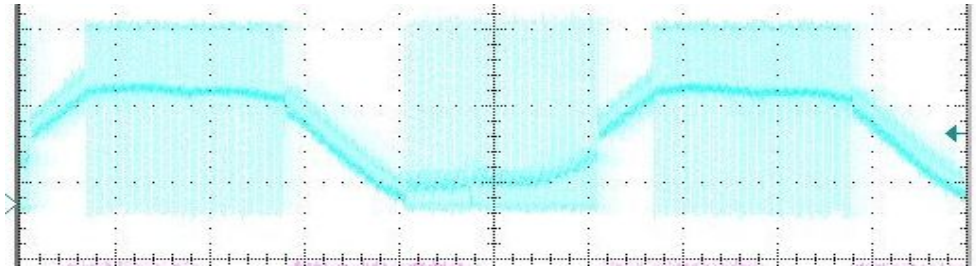
To get a moderately accurate measurement of the robot's rotational velocity, I would like to make use of the rotation of the wheel, which when it maintains traction with the ground, rotates at a rate proportional to the robot's rotational rate. I could use an off the shelf rotary encoder, and place it somewhere inline with the wheel axle. However, this approach would add complexity and mechanical points of failure, while increasing weight, volume, and cost. Luckly, I already have a electromagnetic encoder designed into the robot: it's motor. As the motor magnets move past the motor coils each coil produces a voltage associated with the change in magnetic field through the coils, called the back Electro-Motive Force, or back EMF.

The robot uses a brushless dc motor (BLDC) with a trapezoidal back EMF. This means that, when the motor is spinning at a given speed, we expect the voltage across each of the three motor phases to look like this:



Ideal back EMF

Unfortunately, the real world isn't as pretty, and when the motor is being driven by the motor controller and is under load, the voltage across the voltage at one of the phases will look more like this:



Back EMF when driven, from 0v to +Vbatt

The average voltage is still mostly trapezoidal, but is represented by a varied duty cycle square wave, as the controller's transistors can only pull a phase high, low, or leave the phase undriven. For the microcontroller to be able to use this as a velocity sensor, it needs a nice clean digital signal.

To achieve this I will design a three phase low pass filter, followed by three phase-to-phase schmitt triggers, which will detect when one phase rises above another. I will need to design a resistor-comparator schmitt trigger, as comparators with high enough built in hysteresis will likely not be available. I expect to have a fair amount of ripple out of the low pass filter stage as at high motor RPM's the back EMF frequency may come within an order of magnitude of the switching frequency, (insert filter roll off calculations) so to be safe I'd like to have somewhere around 500mV of hysteresis.

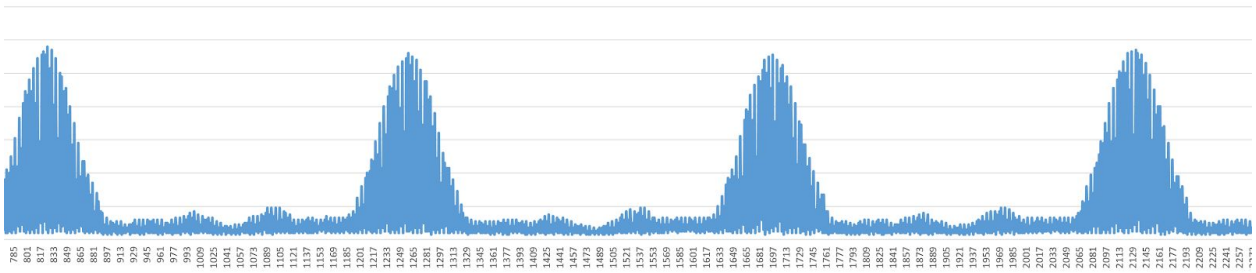
IR photodiode amplifier

For the robot to be able to detect the IR beacon, the robot needs a 940nm photodiode pointing radially outwards. As the 940nm photons impact the photodiode, they are absorbed and their energy promotes charges across the reverse biased junction, causing a proportional amount of current to flow. For the photodiode to more effectively convert photons into current, it is ideal to apply some amount of reverse bias to them. However, the currents that photodiodes generate are very small, and require amplification. By using a transimpedance amplifier and biasing it's noninverting input, we can convert small currents into useful voltages, while maintaining a constant diode forward bias. To make use of the full 0-3.3v range range of the micro's built in adc's, we follow the transimpedance amp with a differential amplifier, which removes the offset and adds a small amount of gain. The output is then read in by the adc of the microcontroller, which performs further processing to identify the robot's relationship to the beacon.

Carrier Extraction

Once the incoming signal is amplified, it will be useful to extract the 1khz carrier from the signal, so that the microcontroller can extract the strength of just the beacon signal. The microcontroller can generate its own 1khz square wave, but the beacon's 555 timer's frequency stability will not be perfect, and any difference in frequency between the 555 and the micro's 1khz will cause the two to drift in and out of phase, potentially disrupting the downstream signal processing code.

To extract the carrier, I will be using a schmitt trigger with a large amount of hysteresis to compare the current signal intensity to a low pass filtered version of itself, creating a kind of high pass filter, where only fast rising or falling edges can cause the comparator to latch in a high or low state. To avoid other signals causing the circuit to trigger, I can use an rc delay circuit and a latch to force the circuit to hold it's current state, just long enough to allow the next edge of the 1khz carrier to cause the circuit to latch.



Example of the output waveform; note that what looks like noise is actually the signal, rising and falling at 1khz

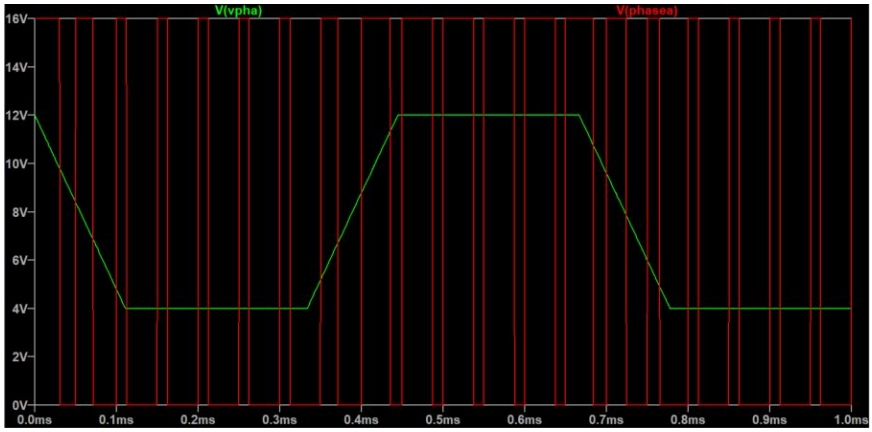
Module Design & Implementation

Motor Digitizer

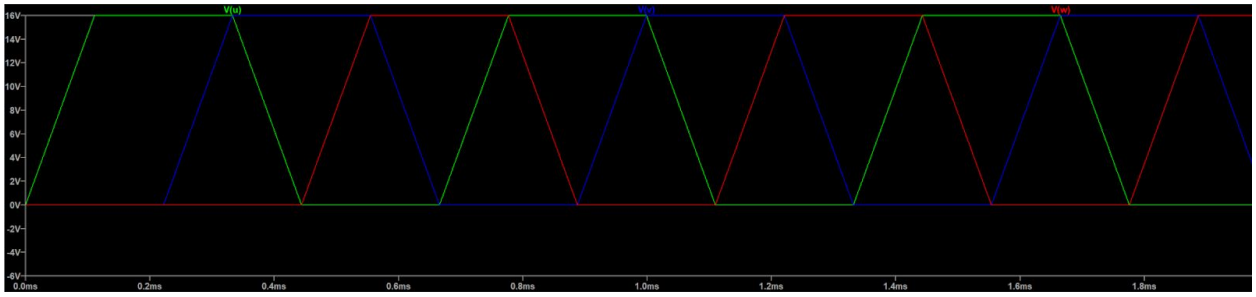
The first step in the design of the motor digitizer circuit was to determine a reasonable motor+esc model to use in my SPICE simulation. While most hobby sensorless brushless motor controllers use a fairly similar block commutated, pulse width modulated scheme for motor control, specific implementation differences exist. For example, controllers may vary in when they set a phase output to high impedance so that they can sense the back EMF of the motor.

To simplify, I made the assumption that the ESC is only applying a pwm trapezoidal voltage waveform, ignoring the periods of high impedance, as they don't contribute to the average voltage at the phase. Additionally, I made the assumption that the phase voltage waveform is centered at half of the bus voltage, which may not be the case in certain situations, but shouldn't be an issue for me as I will be measuring phase to phase.

By simulating a trapezoidal waveform and comparing it to a sawtooth for pulse width modulation, I was able to generate the following waveforms.

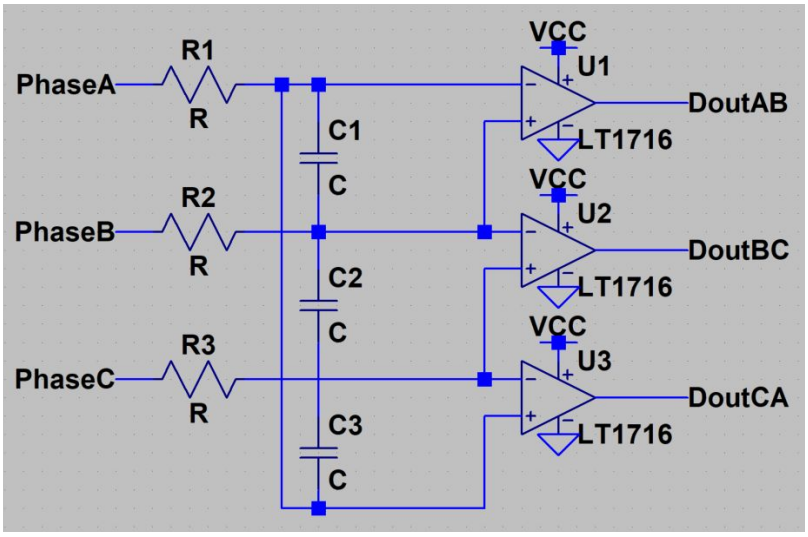


Switched trapezoidal phase waveform, green is the ideal phase voltage, red is the PWM'd phase voltage, as the motor controller would drive it. Switching frequency turned down to 20khz for illustration purposes.



All three ideal phase voltages, 120 degrees apart. (f_commutation = 1500hz)

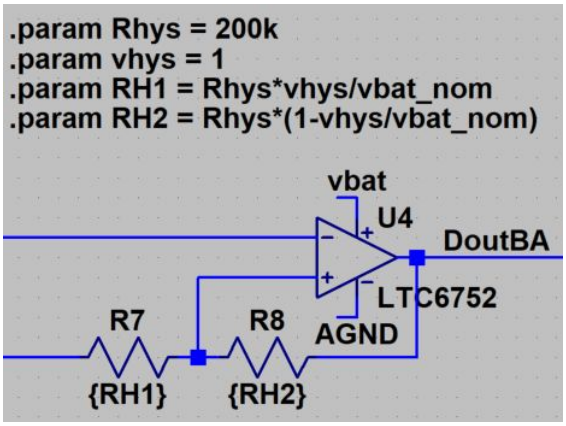
Once I knew my input waveform, I could start designing the digitizer circuit. I knew that first I had to filter out the switching frequency to get back to the trapezoidal shape that the back emf of the motor would be producing. Once I had three reasonably trapezoidal waveforms, I could simply compare pairs of phases to see which was at a higher voltage, and get out three nice square waves at the commutation frequency.



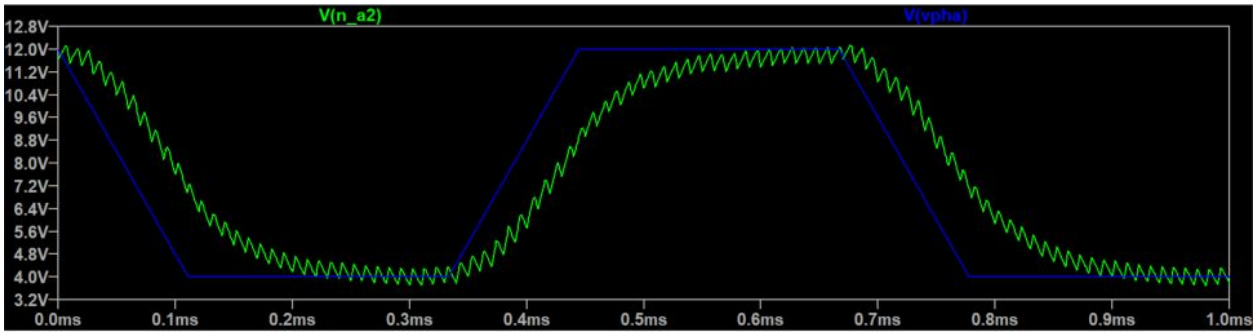
Simplified circuit module

Even with the low pass filter, some amount of switching frequency will still get through. Even though in my model the switching frequency of each phase is aligned and would contribute no ripple between two phases, I expect that the real controller will exhibit some amount of noise or difference in switching such that the ripple may not be perfectly aligned. To prevent possible issues, I decided it would be a good idea to implement hysteresis for each of the phase to phase

comparators. Making good use of parameterization directives, I calculated resistor values using these simple equations.

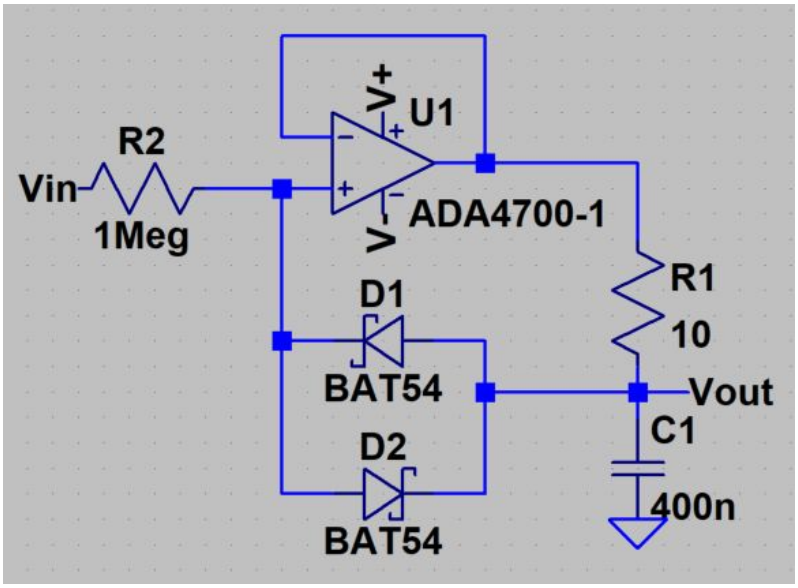


For the low pass filters, I initially decided to go with the classic RC. While this was absolutely adequate, I wondered if I could achieve better performance using a different type of filter. One “issue” I noticed was that as the phase voltage linearly increased at the start of a trapezoid, the low pass filter would follow it up, but as it plateaued, it would slowly decay exponentially. I figured since the fastest the signal would ever change was at a linear rate up and down at the sides of the trapezoids, a nonlinear filter with a fixed output slew rate could be useful for behaving as a hard limit to the speed of signals which are allowed through.



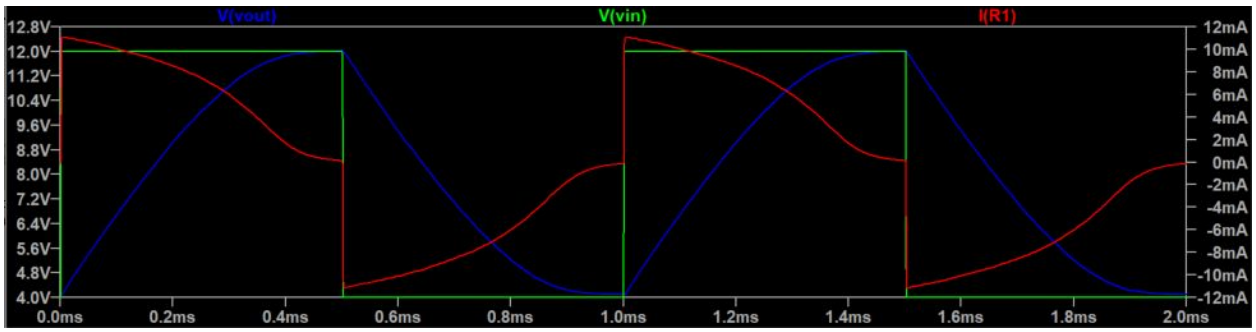
Performance of first order low pass filter

I scribbled down a few different configurations of opamp, resistor, capacitor, and reverse-parallel diodes, but I eventually came up with this design.



The idea was simple, make a constant current source that works in both directions by using reverse parallel diodes as a reversible voltage reference, and then maintain that reference across a current sense resistor. As the opamp pushes more current across R1, a voltage larger than the diode forward voltage begins to develop, and the diode begins to conduct a small amount of current, which holds the non-inverting input of the opamp to Vout, offset by the diode drop, maintaining a fixed voltage across the resistor, fixing the output current. Given a certain size of capacitor, this fixed current manifests itself as a linear rise or fall in voltage.

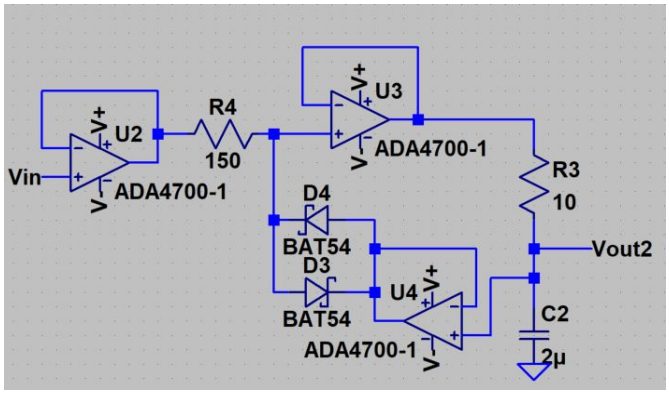
Ideally the output voltage slew rate should not be affected by the difference between Vin and Vout, but in this application, the diodes can't just be modeled as fixed voltage drops. Because of the desire for a high input impedance, the currents which end up across the diodes are very small, and they operate right around the transition to conduction. As a result, the current waveform through the resistor isn't as flat as one would hope, and begins to decay immediately following an input step.



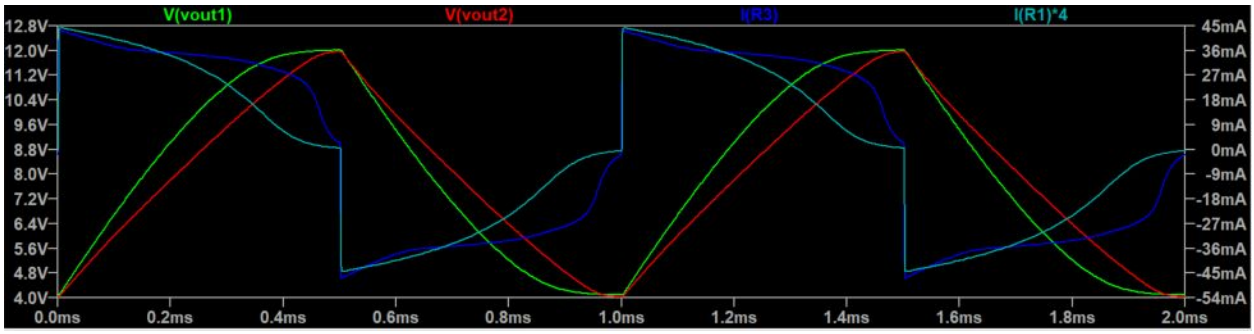
The step response in blue appears linear, but on careful inspection it has some curve to it.

The significant curvature at the end of the step response is the transition to a simple RC response.

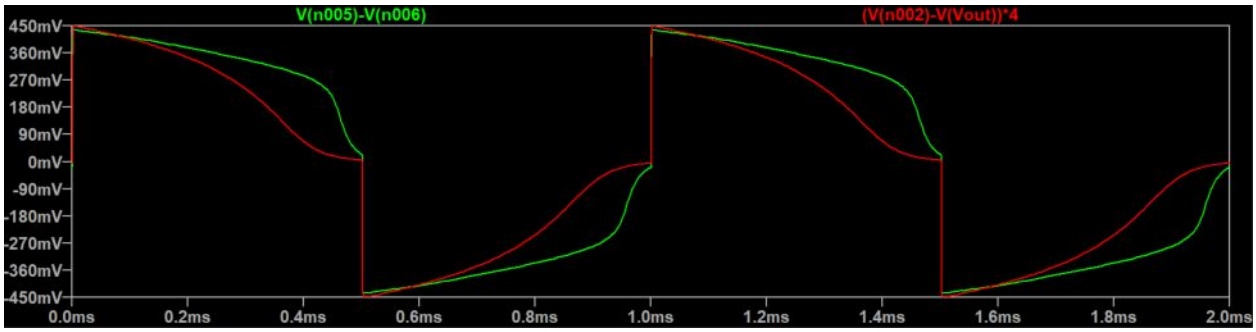
This issue isn't unfixable however. As I mentioned, the limiting component is the large input resistance, which allows the diodes to push and pull the non-inverting input around without large currents flowing back to the input source. By implementing a simple op amp buffer on the input, the diode current can be greatly increased while maintaining a large input impedance, putting the diode in a more constant voltage region. However, larger diode currents will start to affect the output current, as they draw from R1. Again, this can be fixed with buffering.



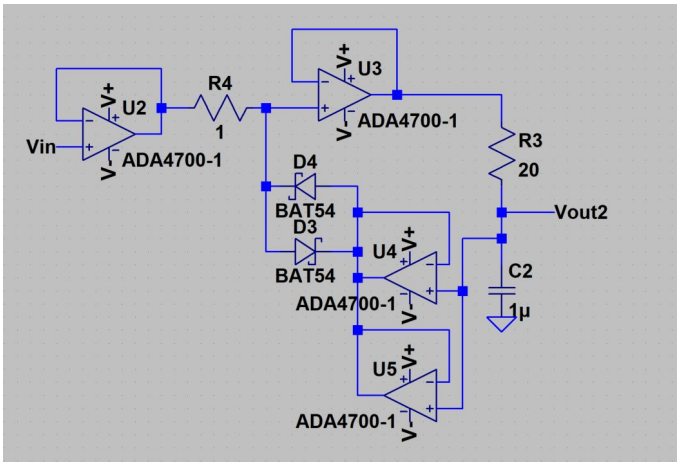
While these changes improve behavior of the circuit, the diode voltages will always be exponential, regardless of how much bias current is applied.

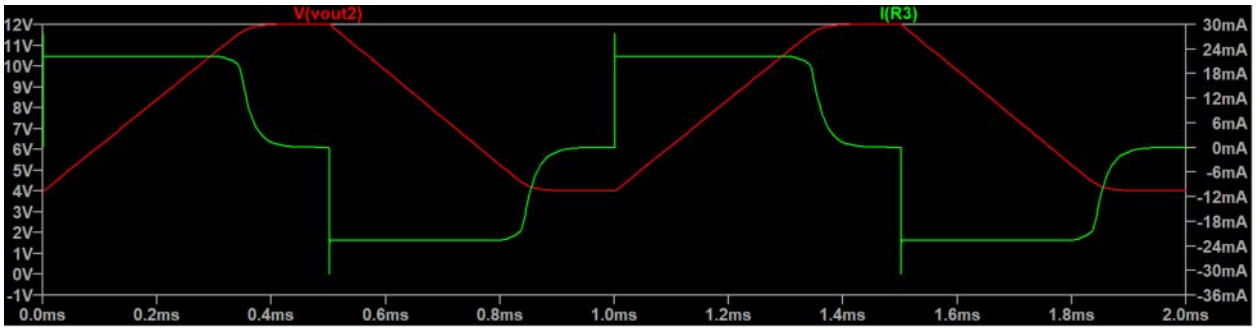


Above: I&V step response of original circuit (green and teal) vs buffered circuit (red and blue)
Below: voltage across diodes of original (red) vs buffered (green)
Notice the shallower green slow which more sharply drops off as current falls



These improvements are still imperfect, the circuit requires an R4 which is large enough that U4 can overcome U2 without either of the devices saturating, leaving some performance on the table. Optimal performance requires opamp abuse. If your opamps are tolerant of output shorts, and you add a parallel opamp to U4, then....

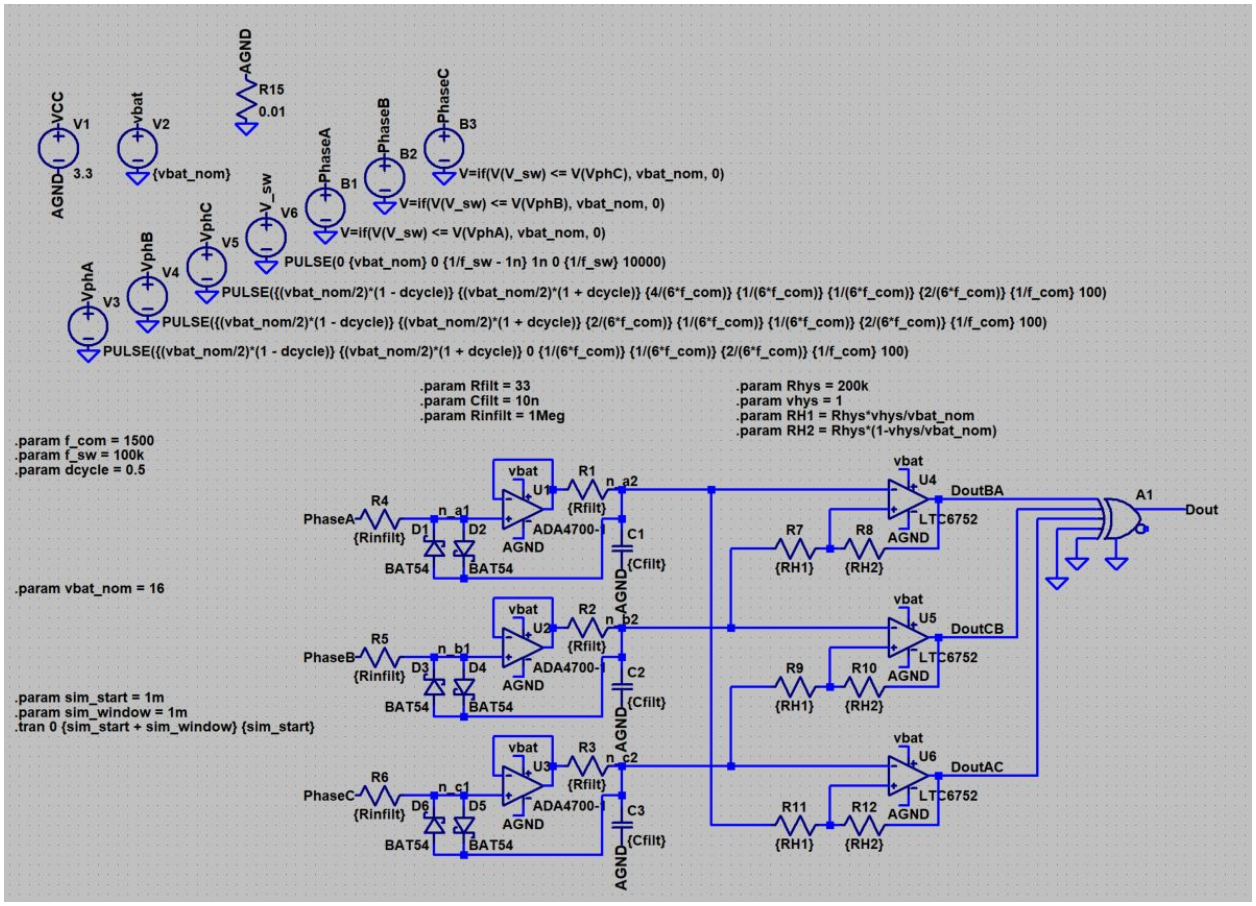




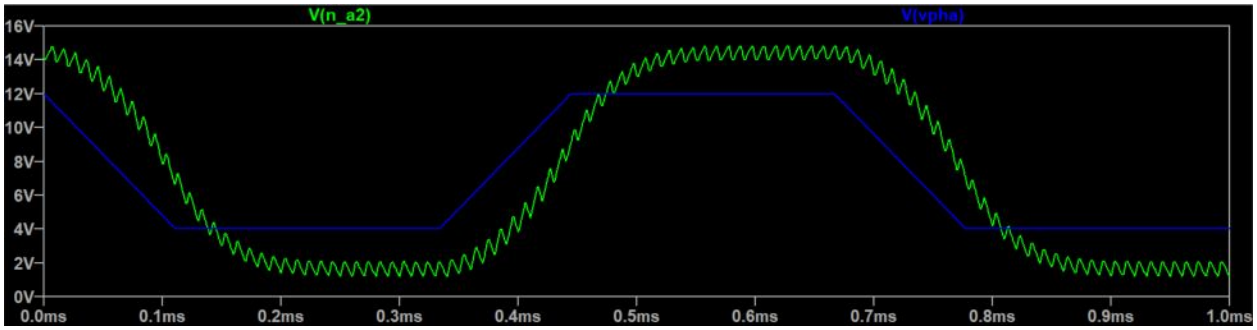
If you saturate the output of U2, you can push a very large amount of current from U4 and U5 across the diode, resulting in incredibly constant output current. Ignore the current spikes on the tops and bottoms of the current waveform, the op amps are likely becoming very unhappy.

But what does all this mean for the motor digitizer? Truthfully, not much, but it was a fun aside. In reality, the simple slew rate limiter is absolutely fine for our purpose, and the less

complex the circuit is, the easier implementing it will be. Let's see what happens if we drop in the slew rate limiter to the original motor digitizer circuit.



final motor digitizer spice model



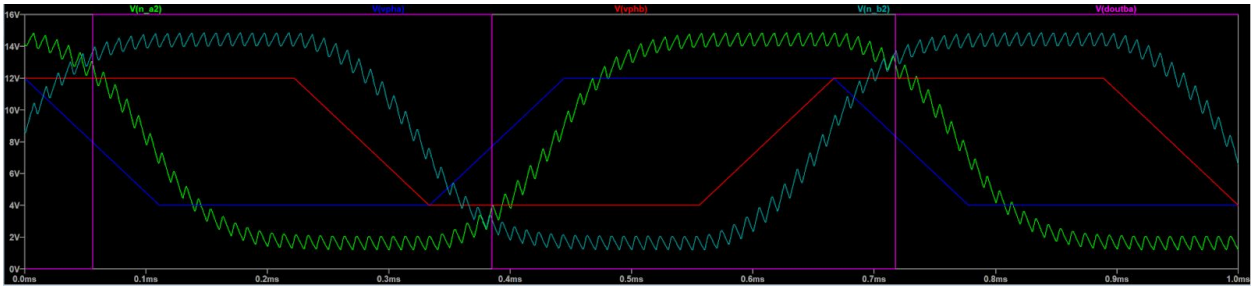
Ideal phase voltage (blue) and filtered phase voltage (green)

At first glance, it doesn't look particularly interesting or different, but there are a couple interesting things to note. First and least importantly, if you look closely at ripple in the extreme high and low points in the filtered waveform, you may notice that the rising and falling slopes

aren't equal. This is a consequence of the aforementioned issues with the slew limiter having a not insignificant relationship between the input voltage step and the output current.

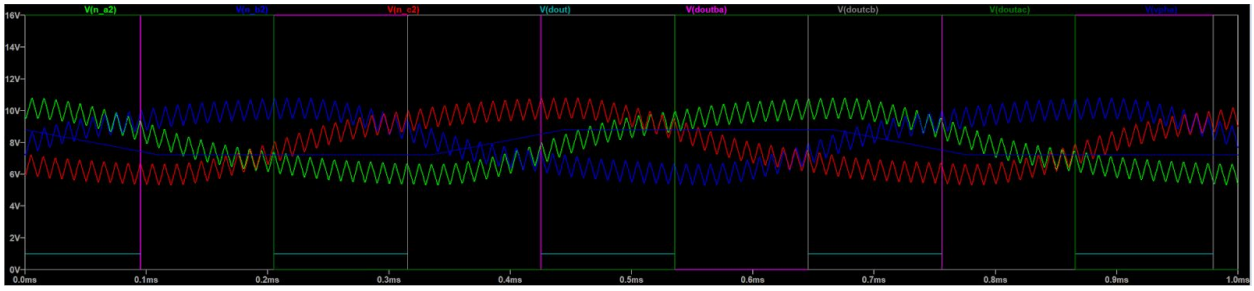
Secondly, and more importantly, notice that the output voltage waveform has a larger magnitude than the input trapezoid? Bet ur RC low pass can't do that. In all seriousness, this is actually a really interesting result of the slew rate limiter circuit's tendency not to move towards the average voltage of the input, but actually towards the point where the voltage spends most of its time. Because the input waveform is only set to 50% voltage (the trapezoid goes from 4v to 12v, while the bus voltage is 16v), the voltage that the transistors apply is higher than the average/ideal voltage at the top of the waveform. At around t=0.6ms the input is pulled to 16v for a longer period of time than it is at 0v, and as a result the slew limiter low pass filter drifts towards the 16v rail, giving this type of low pass filter better signal to noise ratio for the same amount of ripple.

But does it all work? Of course it works. Have some good plots.



Two phases being compared, with the comparator output in pink

All phases and their digitized comparisons
Note the small teal waveform at the bottom, that is the Xor of all phase comparisons, which the microcontroller will read

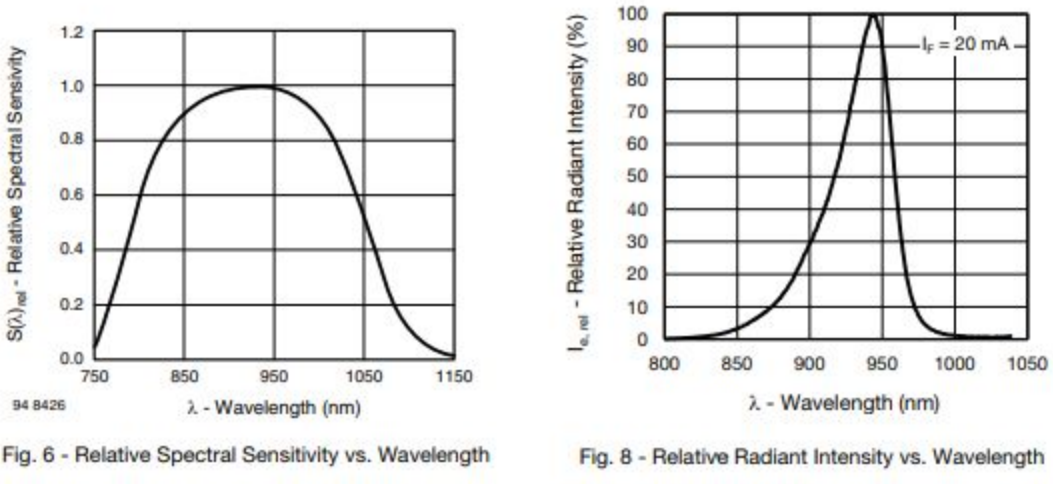


Continues working down to 10% duty cycle

As a final note, all tests were run at max commutation frequency (the frequency of the trapezoidal waveform), regardless of what duty cycle was being applied to the motor. This isn't very realistic, as lower duty cycles apply less voltage to the motor, causing it to spin proportionally slower. However, a lower spin frequency is actually easier to filter and distinguish between commutation and switching, so for the sake of clarity and convenience, I ran all tests at the same commutation frequency, as a sort of (impossible) worst case scenario.

Transimpedance Amplifier

I started off by doing a bit of calculations in matlab. Before starting the design, I wanted to know how much peak photodiode current to expect, and how much I would be seeing from other sources like sunlight. Even though the IR LEDs and the receiving photodiode were both tuned for 940nm photons, they both emit and receive many other wavelengths around 940nm.

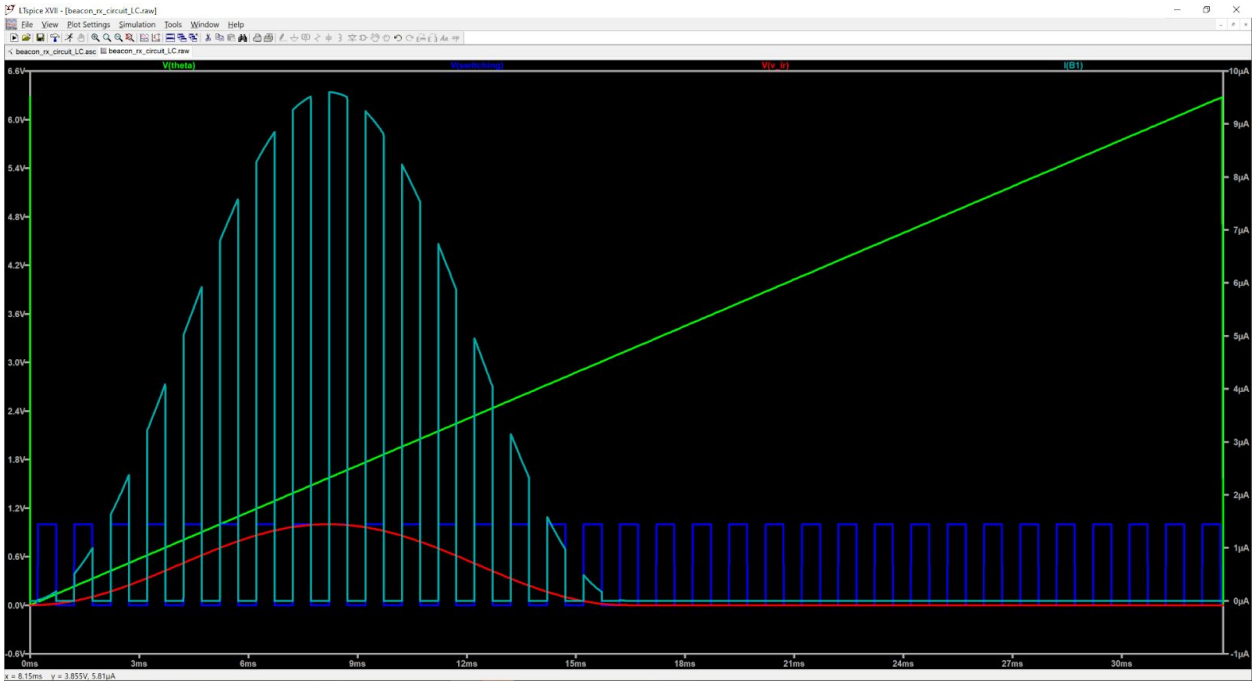


Plots from the photodiode datasheet (left) and LED datasheet (right)

To properly calculate the expected signal currents, I would first have to integrate the product of the two plots, which would give me the total power absorbed by the photodiode from a single LED. Now, I didn't actually do that, I misread the datasheet initially and just kinda ran with a number that wasn't correct. I finally noticed the number was incorrect, but at this point I don't really have time to go back and perform the calculation properly, so I went back and eyeballed a better number, and we're going with around about a 9.6uA peak signal current. I'll definitely address this in the future, but at the moment I got like 6 days of work to squeeze into 4 days.

Once I had a general signal strength, I moved onto simulating the signal I expected to be seeing. Spice behavioral sources made this quite easy. I started off by generating two signals, one 0-1v square wave at the strobing frequency, and a rising slope saw tooth at a rotational speed reasonably close to that which I calculated the robot would see, somewhere around 30hz. Though

the motor could do much more, I haven't yet been able to get the robot spinning too much faster without mechanical issues. The saw tooth waveform ranged from 0 to 2*pi, and represented the rotational position of the robot. Thinking back to the earlier plot of the experimental waveform I had earlier, the general shape of the signal for a fixed rotational speed, which looked like a kinda curvy triangle, occupying about half the signal. In theory, as the diode is pointed away from the beacon at >90 degrees, the signal should be at zero. It isn't due to reflections and ambient signal, but we can model that as dc offset later if we like. I ended up using a single cycle of a cosine function at 2x the frequency, ending up with a cute little lump every cycle, which is more than good enough representation of what I expect to be seeing for the sake of testing my circuits in spice.

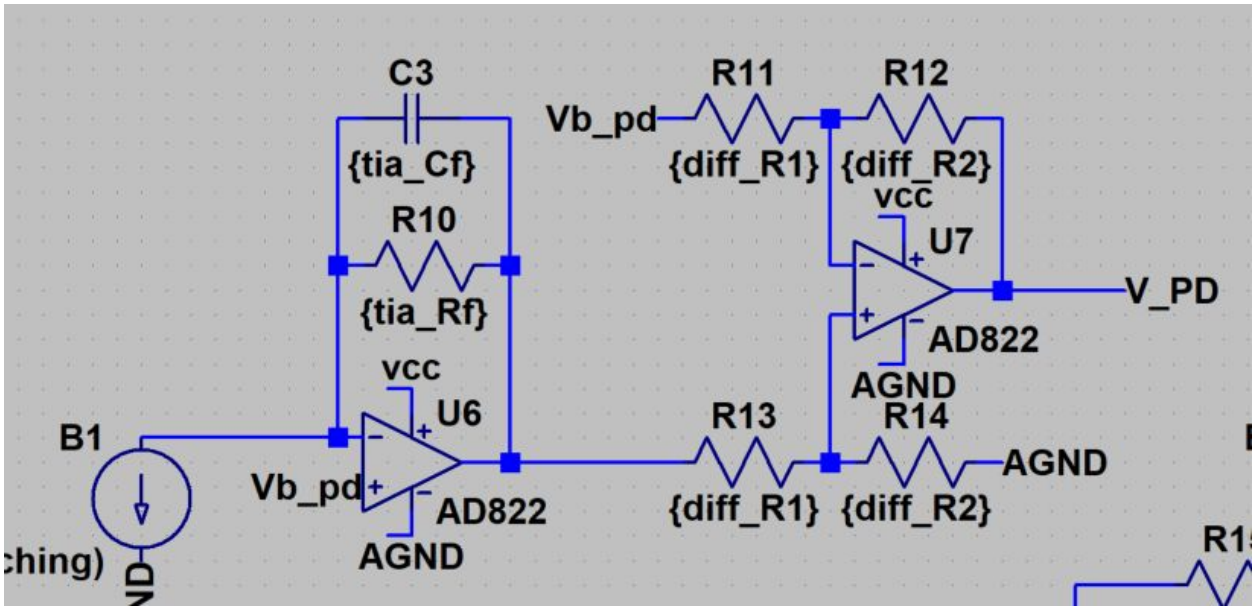


The synthetic current waveform, along with its component waveforms

The datasheet for my photodiode does provide a polar plot of angular displacement vs sensitivity, which I could use to this properly, but as I said, it isn't necessary for the sake of testing my circuits.

For the amplifier itself, there were a couple of features I wanted to add to the traditional transimpedance amplifier. By biasing the noninverting input to some positive voltage, the amplifier will maintain the same constant bias across the photodiode. This has small benefits, the first being better dynamic performance by lowering the diode junction capacitance, as well as providing the peace of mind that the diode will maintain the behaviors that the datasheet claims were tested for at the same forward bias.

The other feature I wanted to add was just a hint of filtering across the feedback resistor, which will attenuate higher frequency signals, and also adds a bit of stability to the opamp, which I did experience some issues with in simulation.



Photodiode amplifier, TIA stage followed by diff-amp stage, with "photodiode" in the bottom left

One of the downsides to biasing the non-inverting input is that, in addition to the inverting input, it also biases the output. Were I to add this bias and try to get my teensy to read the output signals, I'd get absolutely nothing useful, as the signal I care about is above 5v, and the teensy can only read up to 3.3. I could use a resistor divider to scale down the signal, but I'd still have a not insignificant portion of the ADC range unused. By adding a difference amplifier after the transimpedance amplifier, I can rescale the signal from its 5v-12v range to 0-3.3v for the teensy. In practice, I won't actually scale it from 5v-12v to 0v-3.3v, because I want a 0-12v signal for the carrier extractor, and I may not want the transimpedance amplifier to do all of the amplification.

Why wouldn't I want the transimpedance amplifier to do all the gain in one stage? Good question, I'm not entirely sure. Traditionally, when you have two voltage amplifiers in parallel, you want to evenly split the total desired gain across both evenly (stage_gain = sqrt(total_gain)) for maximum bandwidth. Opamps commonly have gain-bandwidth-product (GBP) specifications, which tell you at what frequency the open loop op amp's gain begins to fall off. Using feedback resistors to close the loop intentionally lowers the gain of the of the amplifier, which increases the bandwidth (GBP = gain*bandwidth). By having both voltage amplifiers have the lowest gains overall, the bandwidth of the two amplifiers in series is maximized.

However, this isn't the case for the opamp in the transimpedance amplifier configuration. With a TIA, the opamp isn't the one providing amplification, the resistor is, and I'm not sure it

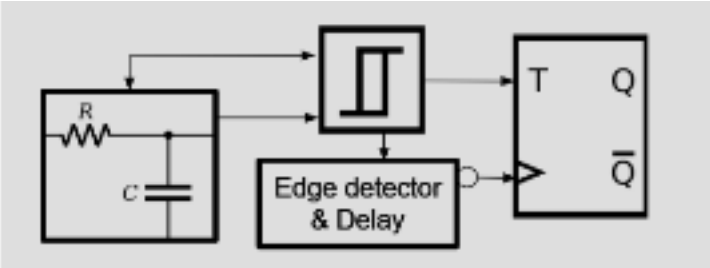
can even be called amplification, as it's more of a conversion from current to voltage with a certain relationship set by the resistor, there are no unitless gains to be had. All the opamp is doing is maintaining the voltage at the other leg of the resistor to match its non inverting input, effectively acting as a buffer. The output of the opamp simply needs to be able to slew at the rate that the input current times the resistance. While yes, adjusting the feedback resistor size for a fixed amount of signal will affect your bandwidth, bandwidth won't be affected if you proportionally increase the resistor size as you decrease the input signal size. Bandwidth will only be affected by the amplitude and frequency of the signal you want to see on the output.

Even though I believe everything I just said to be true, I seem to be missing something somewhere. While testing the behavior of the circuit, I experienced some amount of attenuation on the output of the transimpedance amplifier as it attempted to reproduce the 1khz square waves of the beacon. I had to crank down the size of the feedback resistor to reduce the distortion, but I'm not sure why. The opamp I was using has a GBP of like 1Mhz, and should have had an easy time producing a 1khz 7v signal. I ended up using a 20k volts/amps gain on the transimpedance amplifier, followed by a 62x gain on the difference amplifier, which worked nicely.

Carrier Extractor

This was the least well defined circuit when I started to take on this project. The idea was that because the signal I was looking for was a series of rising and falling edges, spaced a specific amount of time apart, I would need to first need to detect rising edges, and then wait a fixed period before looking for a falling edge. By holding the output latched on a timer, I hoped to be able to avoid general noise or other interference from being able to retrigger the edge detector.

Generally for this task you'd want to use an LC resonator with a narrow bandwidth to filter out the carrier frequency of a signal, and to be honest I really wanted to as well. However, simple calculations indicated that for the Q factor I wanted, I'd need and impractically sized inductor. While not huge relative to other inductors in the world of inductors, the constraints of my robot don't favor tall, fragile masses of ferrite hanging off the pcb. Ideally I could find an smd inductor core which is low profile enough, which would allow me to epoxy it down to the PCB around the perimeter. With this in mind, I decided to pursue an approach which only utilized resistors, capacitors, and ICs.



The initial diagram for the carrier extractor

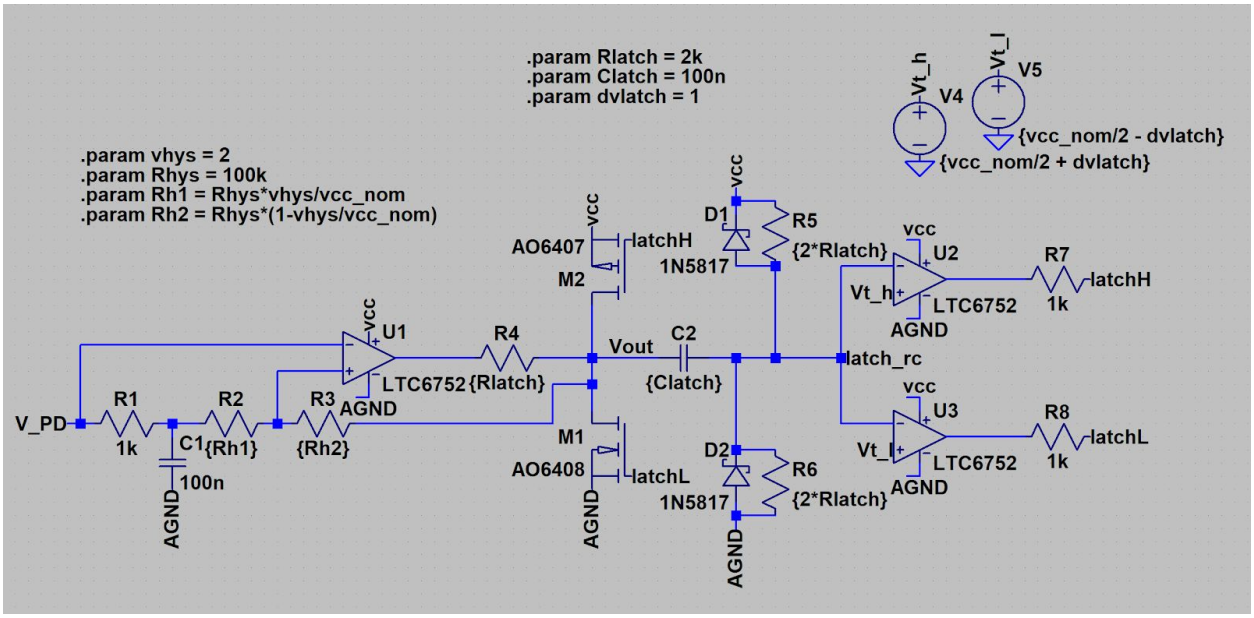
As the signal enters on the left, it is compared to a low pass filtered version of itself.

The initial rough design I was considering was a type of edge detector made from a hysteresis comparator circuit with a latching output, which was controlled by an RC delay. To detect an edge, I could compare the incoming signal to a low pass filtered version of itself. With the addition of a hysteresis circuit, I effectively set an amplitude threshold for rising edges. If I set a +-1v hysteresis threshold, a rising or falling edge would only trigger a comparator output transition if it had a > 1v step from its average value.

From there, I figured that I would need some sort of way to detect a comparator transition, as well as a way to hold that new high/low value for about half the carrier period, so that noise or other signal sources couldn't retrigger the circuit. I figured watching for some sort of RC low pass and a threshold could be used to latch the comparator output using a d-latch.

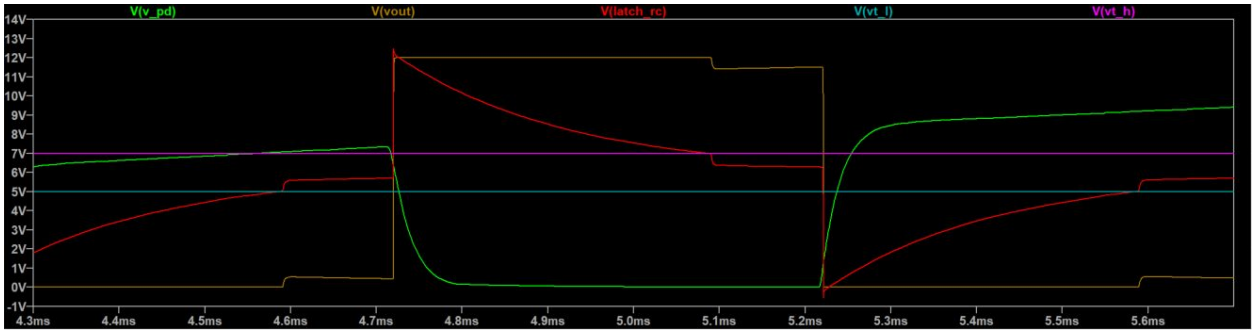
As I began to play around with the design, I realized that using a d-latch would be a mistake, as my circuit needed to have symmetric behavior, treating the incoming rising edges the same as incoming falling edges, and output the corresponding signal. The circuit needed three states, latched & delaying high, latched & delaying low, and unlatched. Using a digital logic block in my analog circuit would only complicate things.

I sat around and thought about it for a bit, and eventually realized that an ac coupling capacitor could be used to indicate a high or low transition of the comparator, as I could detect if it was above or below its resistive biasing point, and use the RC decay back to its biasing point to time the latch system. After some quick sketches, I had the general shape of my latching and delay circuit.



A bit confusing, but I can explain. Starting with the left most portion of the circuit, we have the signal input which goes straight to the inverting input of the comparator, and to the low

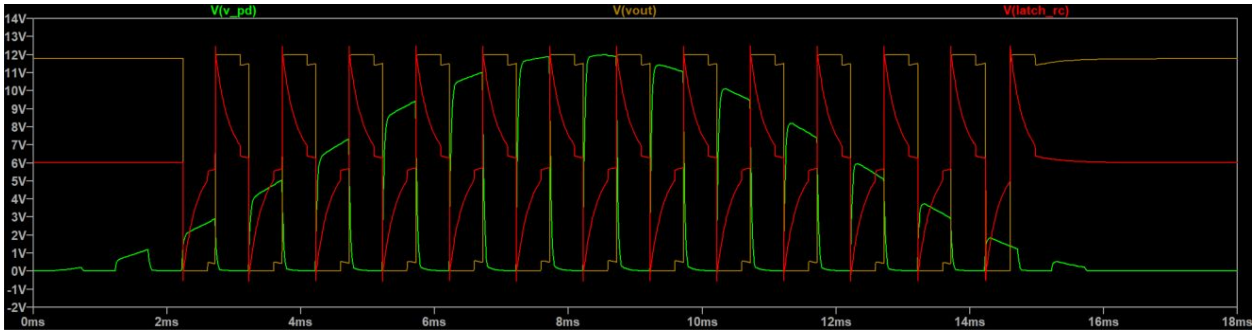
pass filter (R1 and C1). This filter is closely followed by R2 and R3, which form the hysteresis positive feedback loop. As the [.param Rhys=100k] implies, this leg of the feedback is designed to have a 100k impedance, much higher than the impedance of the low pass filter, so to avoid issues with affecting the bias point of C1. However, unlike a traditional schmitt trigger, the feedback leg is not connected directly to the output of the comparator. Instead, it is connected through a 2k resistor. This intentionally increases the output impedance of the comparator, which will allow us to override the comparator output with the high and low latching transistors, M2 and M1. At rest, these transistors are not conducting, but we will get back to them.



Behavioral plot of the latching and delay system, the magenta and teal traces are the high and low thresholds, and the red trace is the latch_rc node, which is holding the system latched while above Vt_h

The next node is critical to how the latch and delay system works. The capacitor C2 AC couples the Vout node with the latch_rc node. At rest, the latch_rc node is biased at half vcc by R5 and R6. However, when an edge transition comes in and triggers U1 to flip its output, for example from low to high, the latch_rc node is driven upwards. Because the capacitor was resting with 6 volts across it, and it was driven high with a 12v rising edge, we included two diode clamps D1 and D2 to prevent latch RC from pushing dangerous voltages into the inputs of the two comparators U2 and U3. These two comparators are used for detecting if C2 has pushed latch_rc above or below its quiescent point by a certain threshold, Vt_h or Vt_l. While the node latch_rc is above Vt_h, U2 applies voltage to the gate of M2 which causes it to switch on. While on, Vout is clamped high, locking the hysteresis state of the schmitt trigger high, so that even if the comparator sees a rising or falling edge at its input, nothing will happen.

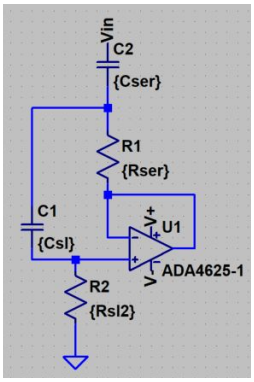
While Vout is clamped high, latch_rc decays back towards its quiescent point by the parallel resistance of R5 and R6. Once latch_rc has decayed back below Vt_h, the comparator U2 drives the gate of M2 back to the off state, and then the circuit is once again ready for another rising or falling edge. This system is able to implement a delay and latching system with only one capacitor storing the state of the system.



While the system works well in simulation, it has certain downsides that I fear may cause issues if implemented on the robot. The first is that, while the latching system will reject periodic interference from sources which are higher frequency than the carrier and lower frequency than the carrier, it may be susceptible much higher frequency sources, which are able to generate multiple rising and falling edges in the span of time between the output being unlatched and when the next edge comes in from the beacon. Additionally, the method of detecting edges at the input, by comparison to a filtered signal, has a significant tradeoff between rejecting interference and allowable signal threshold. As a result, you may notice that the circuit is only generating output pulses when the IR signal is peaking, but not outside of that period. This would create more problems that have to be solved in software, resulting in projects not getting finished.

While I maintain that the inductor packaging can be problematic, the narrowband resonator approach would solve a lot of problems. Firstly, it is by nature extremely good at rejecting other frequencies, and is difficult to convince to start resonating without consistent periodic impulses. Additionally, with a large enough Q factor, a resonator would be able to pick up and amplify the small amounts of signal that gets reflected off the walls, maintaining resonance which the microcontroller would be able to detect at any point in its rotation.

Though an inductor for a resonant LC would be clunky, I may investigate using synthetic inductor circuits to create fake LC resonators without the need for a large core, simply by faking it by using an op amp and capacitor in a transimpedance configuration.



LC resonator using a synthetic inductor

Software Blocks

The microcontroller has the tough job of turning all of this information into robot motion. As a safety measure, if the micro doesn't see an IR signal (signal strength is too low), the robot only operates using the motor encoder information. The user's throttle input will command an average motor output, and as the bot spins, the robot will assume the wheel is a perfect measurement of it's own rotational position, and try to modulate a sine wave from the driver's translation inputs.

In normal operation, when the micro sees a state transition in the extracted carrier via a software interrupt, it will sample the current analog signal value using one of it's built in DACs. It will compare this value to the previous value before the carrier state transition, to determine the strength of the signal from the beacon. Remember, as the beacon strobes, the actual strength of the signal as it is received can be viewed as the height of the rising and falling edges in the signal, not the overall signal amplitude (as it may include other stray IR sources).

Once the micro has taken the sample to sample difference of the signal, it has a representation of the strength of the signal vs time. Using the rotational velocity given by the motor encoder circuit as a base line, the micro will generate an estimate of the robot's rotational phase and frequency. By using a synthetic version of the signal strength vs rotational position plot, the robot both mixes and phase-compares the incoming signal strength with what the signal strength should look like according to the estimated phase and frequency. The robot then adjusts the phase and frequency estimates to achieve the best alignment between the two. This Phase Locked Loop (PLL) gives the robot a robust way to know it's heading at any given point in time, relative to the beacon.

References

IR 940nm LED datasheet
PIN IR photodiode datasheet
http://www.ti.com/lit/ug/tidu535/tidu535.pdf
http://www.physics.unlv.edu/~bill/PHYS483/LED_PIN.pdf