Table of Contents

BAJA SAE BACKGROUND **5**

INITIAL VEHICLE GOALS **5**

MECHANICAL CVT FUNCTION **6**

MOTIVATIONS FOR MOVING TO THE ELECTRONICALLY CONTROLLED CVT **6**

CONS OF MOVING TO AN ELECTRONICALLY CONTROLLED CVT **7**

INITIAL DESIGN CONSIDERATIONS **8**

Initial Control Design Considerations 8
Initial Mechanical Design Considerations 9
Initial Electrical Design Considerations 12

CVT PROTOTYPE **13**

Manual CVT Prototype: Design 13
Manual CVT Prototype: Testing Results/Conclusions 15
eCVT Prototype Design: Actuator Selection 15
eCVT Prototype Design: RPM Feedback Implementation 21
eCVT Prototype Design: Implementation of Actuator Control 26
eCVT Prototype Design: Control Implementation 35
eCVT Prototype Design: PID Tuning 41
eCVT Prototype Design: Testing/Analysis 48
eCVT Prototype Redesign 51

FINAL CVT DESIGN **54**

Component Design: Overview 54
Component Design: Driving Pulley Design 55
Component Design: Driven Pulley Design 56
Component Design: Case Design 57
Mechanical Components Analysis 59
CVT Cooling 60
Electronics Enclosure Design 63
Testing of Engine 63
Initial System Selection and Design Considerations *64*
Dyno System Set-Up *65*
Engine Testing Method *66*
Final Results and Analysis *67*
Controller Software Final Design 70
PID Tuning and System Characterization 75
Final CVT Design Performance Testing 81

CONCLUSIONS AND FUTURE WORK **96**

APPENDIX **100**

Arduino Code 100
REVISION 9 ..... *100*
REVISION 24 ..... *110*
REVISION 27 ..... *123*
REVISION 28 ..... *143*

REFERENCES **174**

List of Figures

Figure 1: Example of CVT shifting; (Left) Underdrive, (Right) Overdrive 6
Figure 2: PID flowchart 8
Figure 3: Initial controller design 9
Figure 4: Example of hydraulic actuation and shifting hysteresis 10
Figure 5: Example of actuation using helical cam 11
Figure 6: Example of actuation with rack and pinion (left) and lever arm for shifting (right) 11
Figure 7: Example of actuation with linear actuator (left) and lever arm for shifting (right) 12
Figure 8: Solidworks model of Gaged driving pulley retrofitted; exploded view (left), full assembly on engine (right) 14
Figure 9: Prototype driving pulley actuated by push-pull cable 14
Figure 10: Comparison of Gaged mechanically actuated driving pulley (left) and manually actuated driving pulley (right) 15
Figure 11: M-Lot GPS data with actuation forces highlighted 17
Figure 12: Actuation force histogram of all time driven (shown on GPS map) with 10 catagories. 17
Figure 13: Actuation speed histogram of all time driven (shown on GPS map) with 30 catagories. 18
Figure 14: Actuation Speed versus Actuation Force from data shown in Figure 13 18
Figure 15: Stoddard Wells GPS data depicting actuation speed 19
Figure 16: Actuation Speed Histogram from Stoddard Wells data 19
Figure 17: Current and Actuation speed versus Actuation force relations given by manufacturer. 20
Figure 18: Current and Actuation speed versus Actuation force relations given by manufacturer. 21
Figure 19: Geartooth sensor application example (55505) 22
Figure 20: Implementation of Hall Effect and tone ring for the prototype CVT 22
Figure 21: MyChron4 data logger and display 24
Figure 22: Seeed Studio's CAN Bus Arduino shield 27
Figure 23: CAN Bus Shield retrofitted to fit Arduino Mega 27
Figure 24: Electrical Schematic of CAN Bus wiring 28
Figure 25: Frame format of CAN Bus message 29
Figure 26: Image of fully controlled CVT prototype 35
Figure 27: Potentiometer setup used for PID tuning 43
Figure 28: Electronics enclosure of prototype controlled CVT; Wiring runs to potentiometers used for PID tuning 43
Figure 29: Disturbance rejection with dyno loading 0-10A using controller gains Kp = 1.0, Ki = 1.1, Kd = 0.0 on the 2017 Baja car 47
Figure 30: Actuator overshoot problem graphed 49
Figure 31: Proper corner entry and exit maneuvering in racing 50
Figure 32: RC Low Pass Filter circuit 52
Figure 33: Sallen-Key Low Pass Filter circuit 52
Figure 34: Multiple Feedback Low Pass Filter 52
Figure 35: Oscilloscope reading of engine hall effect with passive RC low pass filter 52
Figure 36: Oscilloscope reading of engine hall effect 53
Figure 37: Driving pulley assembly with actuator and 55075 Hall Effect 53
Figure 38: Gaged V-Belt (left) and CV-Tech V-Belt (right) 54
Figure 39: Driving pulley exploded view 56
Figure 40: Driving pulley assembly (left), assembly of final CVT design without the case (right) 56
Figure 41: Secondary without torsion spring, exploded view (left), assembly (right) 57
Figure 42: Complete assembly of powertrain system in chassis (left), clos-up view of sliding shield on CVT case (right) 58

Figure 43: Heat treating of CVT case material before bending to prevent cracking due to cold working 58
Figure 44: FEA of final lever arm design 59
Figure 45: FEA of driving and driven sheaves 60
Figure 46: With no fan cooling; Internal ambient CVT temperature data, data logger temperature, and engine rpm 61
Figure 47: Fan cooling implemented; Internal ambient CVT temperature data, data logger temperature, and engine rpm. 61
Figure 48: Cooling fan deformed due to high temperature 62
Figure 49: Electronics enclosure layout (left), electronics enclosure installed in nose of chassis (right) 63
Figure 50: Dyno system components 64
Figure 51: Settings for engine test 65
Figure 52: Dyno readout window 66
Figure 53: Dyno cart 67
Figure 54: Manufacturer horsepower curve of engine specified by SAE rulebook 68
Figure 55: Manufacturer torque curve of engine specified by SAE rulebook 68
Figure 56: Dyno test results from Engine #1 69
Figure 57: Dyno test results from Engine #2 69
Figure 58: Depiction of actuator position overshooting problem using Rev 9 and Kp=1.0, Ki = 1.1, Kd = 0.0 71
Figure 59: Depiction of oscillations entering overdrive problem using Rev 9 and Kp=1.0, Ki = 1.1, Kd = 0.0 73
Figure 60: Cal Poly Test Track - Jump without maintaining throttle at landing, then acceleration after landing 74
Figure 61: 2017 Baja car strapped to chassis dyno 75
Figure 62: Setpoint change disturbance rejection of controller using gains Kp =1.0, Ki = 1.1, Kd = 0.0 78
Figure 63: Setpoint change disturbance rejection of controller using gains Kp =1.7, Ki = 0.7, Kd = 0.0 78
Figure 64: Comparison of disturbance rejection waveforms in Matlab 79
Figure 65: Futek torque sensor in line of axle shaft. 81
Figure 66: Cal Poly Test Track - Driving over mulch piles using Rev 9, engine RPM histogram with 10 categories 83
Figure 67: J-Lot Acceleration and cornering, 10 category histogram of engine rpm using Rev 9 83
Figure 68: J-Lot acceleration and cornering GPS data displaying engine RPM 84
Figure 69: J-Lot Acceleration and cornering, data from track displayed above, code used was Rev 9 84
Figure 70: J-Lot driving down stairs, 10 category histogram of engine rpm using Rev 9 85
Figure 71: J-Lot driving down stairs, data correlating to histogram of engine rpm, using Rev 9 85
Figure 72: Cal Poly Test Track – Traction limited situation - driving over mulch using Rev 24; 20 category histogram of engine RPM 86
Figure 73: M-Lot Hill - Hill climb data starting at base of hill comparing torque and engine RPM with 20 category histogram of eCVT using Rev 24 86
Figure 74: Hill Climb from 2017 California competition 87
Figure 75: Cal Poly Test Track - J-Turn on dirt using Rev 24 88
Figure 76: Cal Poly Test Track - Jump with landing while maintaining full throttle using Rev 24 88
Figure 77: Cal Poly Test Track – Jump 89
Figure 78: Shifting curves (ratio vs speed) of the eCVT (orange) and CV-Tech CVT (blue) 92
Figure 79: Acceleration run comparison between eCVT (orange) and CV-Tech CVT (blue) 92
Figure 80: Backshifting comparison with 5A load between eCVT (orange) and CV-Tech (blue) 93
Figure 81: Backshifting comparison with 10A load between eCVT (orange) and CV-Tech (blue) 93
Figure 82: Simulated tractor-pull with 5A load comparison between eCVT (orange) and CV-Tech (blue) 94
Figure 83: Simulated tractor-pull with 10A load comparison between eCVT (orange) and CV-Tech (blue) 94
Figure 84: Simulated woops with 0-5A load. 30 category histogram of engine rpm over 20 seconds for eCVT (orange) and CV-Tech CVT (blue) 95
Figure 85: Simulated woops with 0-10A load. 30 category histogram of engine rpm over 20 seconds for eCVT (orange) and CV-Tech CVT (blue) 95
Figure 86: Cal Poly Test Track – Whoops 96

Figure 87: Depiction of acceleration with ramping setpoint using Rev27 97

Baja SAE Background

Baja SAE is a collegiate design series sponsored by the Society of Automotive Engineers. Every year teams from across the globe design and manufacture one person off-road vehicles and compete in three competitions held across the United States. These four day competitions consist of Static and Dynamic events. The static events consist of a design presentation and sales presentation. The dynamic events are acceleration, suspension and traction, maneuverability and either sled pull or hill climb. Each competition ends with a four-hour endurance race where all student teams are pitted against each other on a track of one mile or greater. The dynamic events test each car's performance as well as its durability.

To maintain fairness and safety, all vehicles must conform to rules published yearly by the Baja SAE coordinators. The most notable rule in the competition is the requirement to use a 10 horsepower Briggs & Stratton engine of the same model called out and it cannot be altered in any way.

"B2.5 Engine Requirement and Restrictions (NEW)

To provide a uniform basis for the performance events, all vehicles must use the same engine: a stock four cycle, air cooled, Briggs & Stratton OHV Intek Model 19.

The only engine model accepted at all 2017 Baja SAE North American competitions is Briggs & Stratton model number: 19L232-0054-G1

No other engine models will be accepted. No engine models from previous competition years will be accepted."

Initial Vehicle Goals

At the beginning of the year, each subsystem design team for the 2017 year made a decision-matrix that ranked performance, weight, manufacturability, serviceability, and reliability in order of importance. since the endurance race is heavily weighted and pushes the vehicle to its limit, the Cal Poly Pomona Baja SAE team in recent years has highly valued the reliability of the vehicle. During an endurance race, losing time on the track for component repair or replacement can drastically affect the outcome of the race for a top-ten finish. This year, vehicle goals were set to increase overall performance and reduce weight while maintaining a reliable vehicle. Since it was expected to add the electrical system as a foundation, weight gins would end up contradicting overall goals but were expected to be justified with performance gains.

Mechanical CVT Function

In a traditional mechanical CVT shifting is based off a force equilibrium between the clamping forces of the driving pulley and the driven pulley. For the driving pulley, the engine's RPM by the centrifugal force acting on flyweights and cams to point the force towards clamping the sheaves. A compression spring aids in downshifting since the flyweight's energy would take too long to dissipate for quick down shifting characteristics but works against the flyweights in upshifting. Within the driving pulley the components that can be modified to affect CVT performance are the mass of the flyweights, angle of the ramps, and compression spring rate.

The driven pulley utilizes a torsional spring in conjunction with a helical ramp to apply a clamping force onto the V-Belt. To adjust the shifting force, the spring, spring preload and helix angle can all be changed for different tunes of the mechanical CVT.

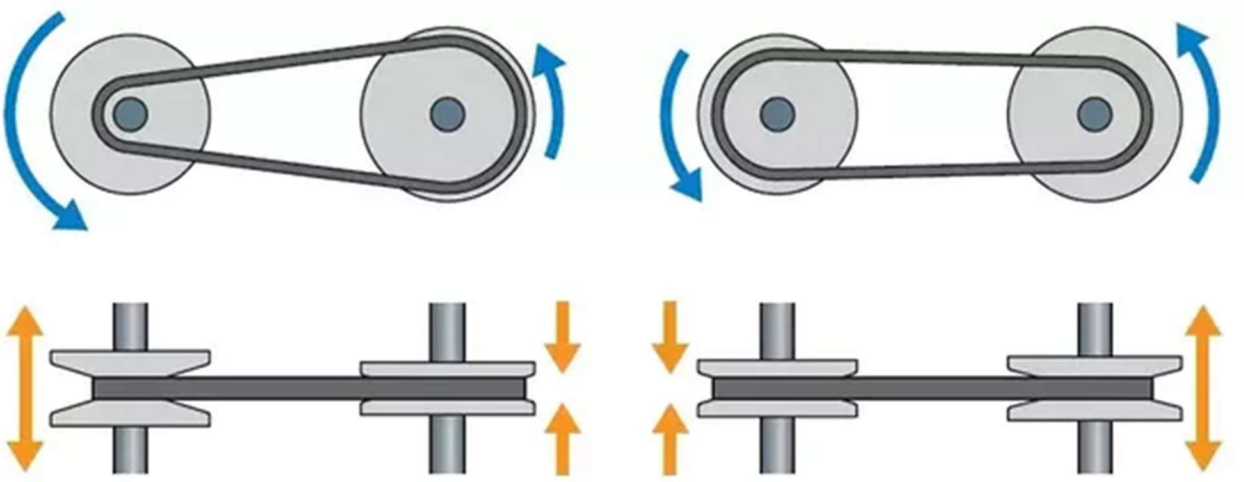

Figure 1: Example of CVT shifting; (Left) Underdrive, (Right) Overdrive

Motivations for Moving to The Electronically Controlled CVT

Mechanically actuated systems are unable to maintain a set engine RPM always, since flyweights must overcome the force of the driven pulley and the compressive spring force on the driving pulley. The driving pulley's compressive spring force is necessary for downshifting, thus creating conflicting goals for upshifting and downshifting when attempting to produce responsive shifting. With the electronically controlled system, the

driven pulley clamp force could be minimized for decreasing power loss due to over-clamping without affecting when the system would begin to shift.

Another motivation is that the mechanical system could only be optimized for a single engine while a controlled system allows for simple changes between engines of the same model. Since power output capability and characteristics of power/torque curves differ between engines, a mechanical system would have to average the best rpm to hold for all engines or design mechanical tunability to the system.

Lastly, a controlled shifting system starts the foundation of a system that can be further developed to minimize CVT inefficiencies and increase performance in scenarios of secondary importance (for example: traction control, limited belt slip control, belt failure detection, engine braking, etc.).

Cons of moving to an Electronically Controlled CVT

The greatest disadvantage with moving to the electronically controlled CVT is the added failure modes to the system. Baja SAE competitions test vehicles in various terrains and truly put cars at the limits of functionality, thus requiring robust systems. If not carefully implemented, the electronics of the system could greatly increase the system's sensitivity to the environment, increasing rates of failure to unacceptable amounts.

Another disadvantage is the added weight of the system, contradicting the overall goal of any performance vehicle, but this is compensated by a power output increase. Although a mechanical system does add rotational mass, the overall added mass is limited to the cams, weights, springs, and supporting structures.

Initial Design Considerations

Initial Control Design Considerations

To maintain maximum power output capable by the engine, closed loop feedback is implemented. Since the final powertrain system dynamics would be unknown until the system design is finalized, controller gains need to be determined using a black box tuning method (i.e. Ziegler-Nichols).

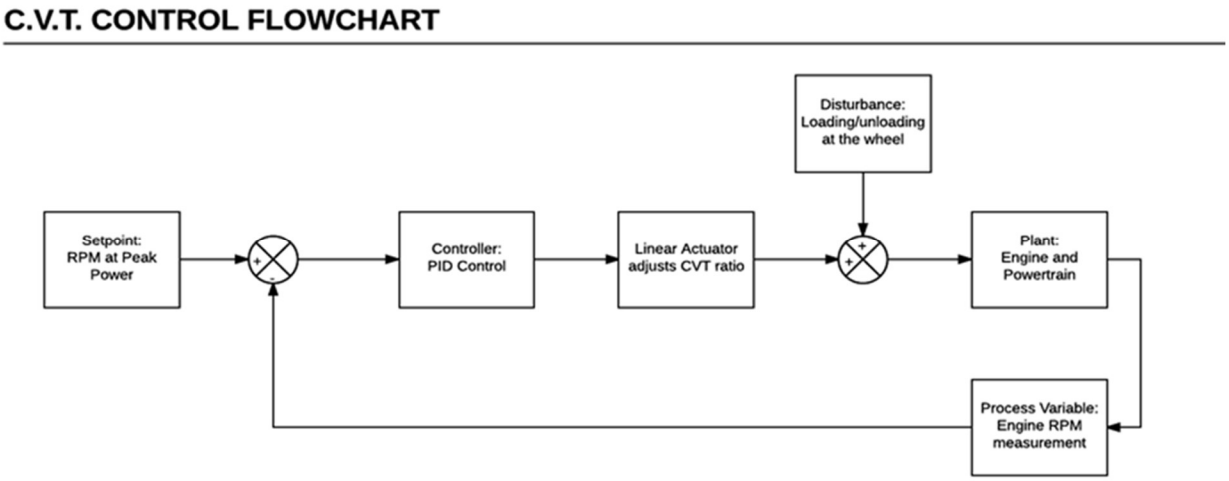

Figure 2: PID flowchart

Using the relation between rpm and horsepower, the controller is designed to use rpm as feedback with peak power rpm set as the setpoint. As the primary is actuated, the controller maintains the engine rpm but, there are a few different cases to consider while the controller is not in the main control loop:

1. Entering the control loop from a standstill, such as accelerating from a stop.
2. Entering the control loop while driving, such as when exiting a cornering scenario.
3. Exiting the control loop while driving to brake without stalling the engine.
4. Exiting the control loop and maintaining underdrive or overdrive.
    a. If actuator position accuracy poses a problem, then implementation of closed loop control for underdrive and overdrive is necessary to maintain the drive ratio. To attain CVT ratio as feedback, a second rpm sensor is needed for recording the driven pulley rpm.

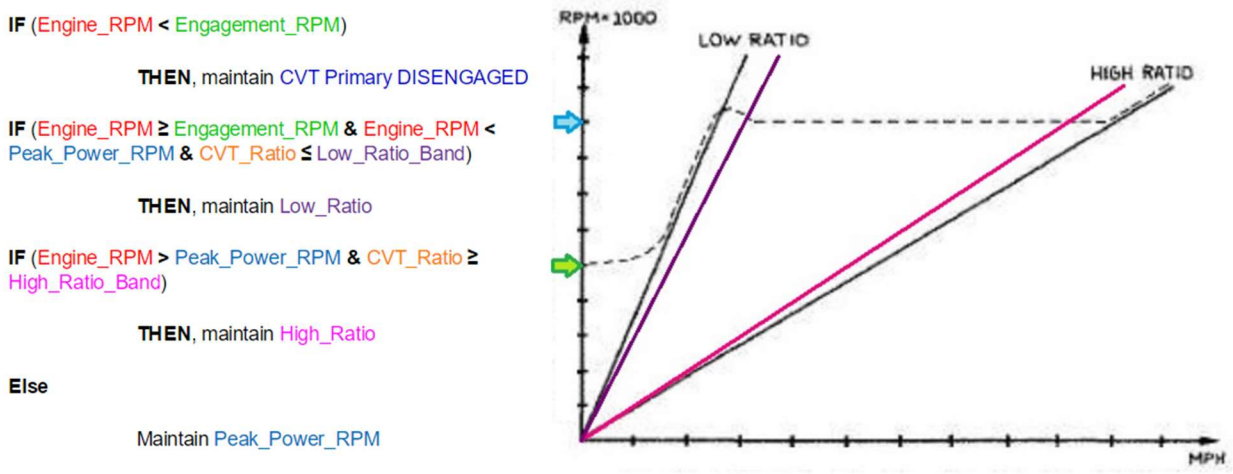

Figure 3: Initial controller design

The first attempt in resolving all the above-mentioned issues, was to use the engine rpm for not only feedback control, but also as indication of when the driver would like to initiate or stop driving. For the controller to know when to stop actuating after reaching CVT underdrive and overdrive, the CVT ratio will be used as feedback to determine actuator position. For the case of being at the overdrive position and rpm exceeding the setpoint rpm, the actuator will maintain overdrive. In a similar manner, the actuator will maintain the underdrive position once the engine rpm is below the setpoint and the actuator is near underdrive position. A buffer was given for maintain underdrive and overdrive to ensure that any fluctuations in rpm readings do not cause the system to induce oscillations when getting into the underdrive or overdrive positions.

Initial Mechanical Design Considerations

The initial development approach was to create a prototype actuation method on the driving pulley that would be simple to implement in the prototype and final design. Various approaches were considered:

- Hydraulic Actuation
    - The system would have the primary actuated by hydraulic piston while the secondary remained the same mechanical system. Hydraulic actuation was ultimately not considered due to complications of moving seals, added cost of hydraulic components when an electrically actuated piston is still required, and the difficulty of prototyping such a system.

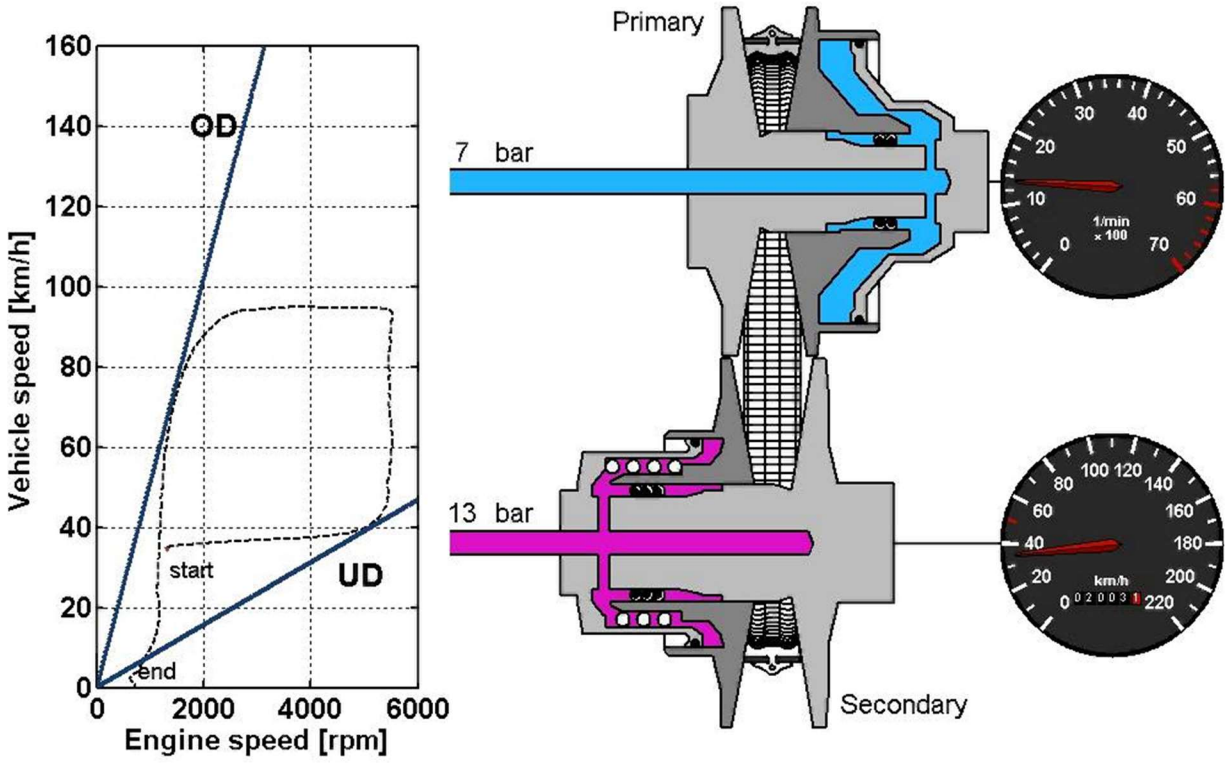

Figure 4: Example of hydraulic actuation and shifting hysteresis

- Motor with gear reduction to cam
    - In this system, the motor would drive the helical cam to actuate the driving pulley. Issues to consider would be the packaging of the motor close to the CVT, sealing the motor from the environment, the added complexity of gear manufacturing on a helical cam and a case to seal the gears from the environment.

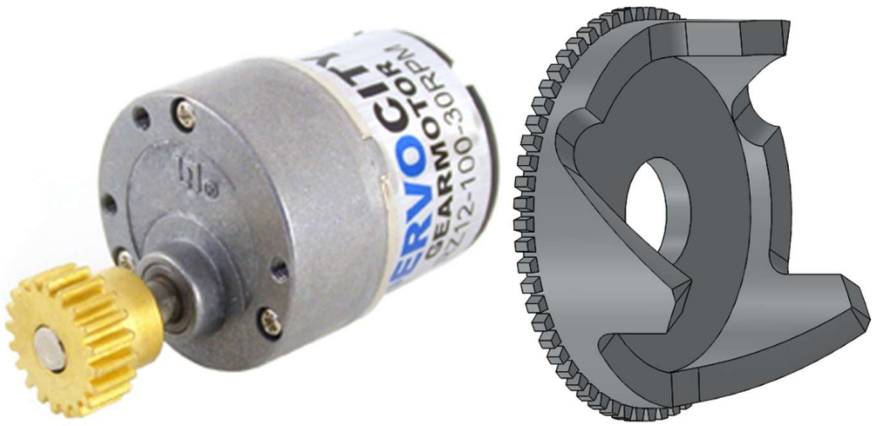

Figure 5: Example of actuation using helical cam

- Motor to single gear reduction with rack actuating clutch fork, moving driving pulley.
    - To change CVT ratio, a clutch fork was considered since the components could be quickly manufactured with the CNC Mill and lathe. As for motor linear actuation, a rack and pinion was considered due to the decreased cost in comparison to linear actuators, but sealing would still be required and the largest reduction possible was a 2:1. Motor selection available limits what can be used. The clutch fork design could be adjusted for the point of actuation, where it could be done from above the engine or in front of the engine.

Figure 6: Example of actuation with rack and pinion (left) and lever arm for shifting (right)

- Geared Linear Actuator (Acme Screw or Ball Screw) to clutch fork, moving driving pulley
    - The clutch fork was again considered with utilization of a premade linear actuator in order to reduce complexity of design. Issues to consider with a purchased actuator is the selection available and what the clutch fork design can conform to. The clutch fork design could be adjusted for the point of actuation, where it could be done from above the engine or in front of the engine.

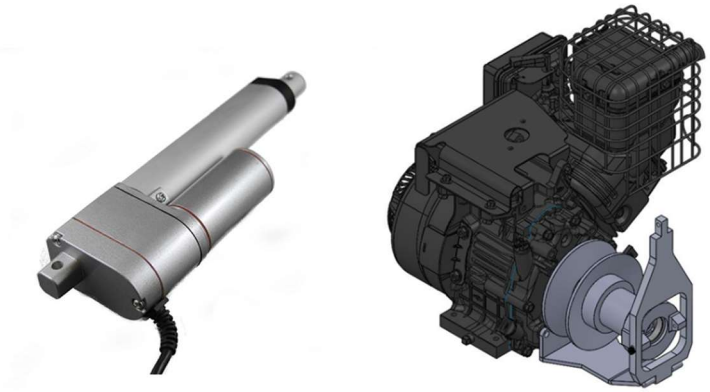

Figure 7: Example of actuation with linear actuator (left) and lever arm for shifting (right)

Considering important aspects in each design, the design chosen for prototype implementation was the clutch fork design with greater consideration for a servo linear actuator to quickly implement a control system. To consider varying cost, size, and availability, various types of linear actuators and geared motor systems were considered.

Another design consideration that had to be integrated was rpm pickup locations for the engine and driven pulley. Since the sensor to be used would be a hall effect sensor or proximity, a tone ring would be necessary for reading rpm. A tone ring was pressed onto an aluminum hub that would be pressed right against the engine output shaft when the driving pulley was attached to the engine, giving the controller the engine rpm.

Initial Electrical Design Considerations

To determine what actuating systems are available for consideration, power restrictions were considered from the 2017 Baja SAE rules.

"B2.5.13 Alternator

The engine may be fitted with an alternator to generate electrical energy. The only alternators which may be used are those which Briggs & Stratton specifies for the engine model. Available charging system includes 3, 10 and 20 AMP systems." [1]

All alternator systems provided by Briggs & Stratton are regulated to 12Vdc. With power in mind, it is necessary to determine the amount of dynamic force and speed of actuation required for CVT shifting without overdrawing current from the system. As a safety aspect, it was determined that the battery should be able to power the system in case of alternator failure during an endurance race, with a lap lasting anywhere between 3-7 minutes.

To acquire rpm readings from the engine and driven pulley shaft, sensors had to be selected based on the smallest package, reliability, ease of implementation, and ability to interface with a 5Vdc logic Arduino microcontroller. Commonly used sensors for rpm readings are hall effects and proximity sensors. For simplifying implementation of rpm sensors, it was decided that the sensor would work off a tone ring instead of a sensor based on external magnets.

CVT Prototype

Manual CVT Prototype: Design

As a proof of concept as well as a system to base the future design off, a manual CVT was built using the previous year's CVT. This was done by removing all shifting components off the driving pulley. A pressure plate assembly was made that allowed for a bearing to be used to allow the non-rotating pressure plate to actuate the rotating moving sheave of the driving pulley. A clutch fork attached to the pressure plate and a mount off the engine allowed the driver to actuate the CVT via a push-pull cable connected to a shifting lever in the cockpit of the vehicle.

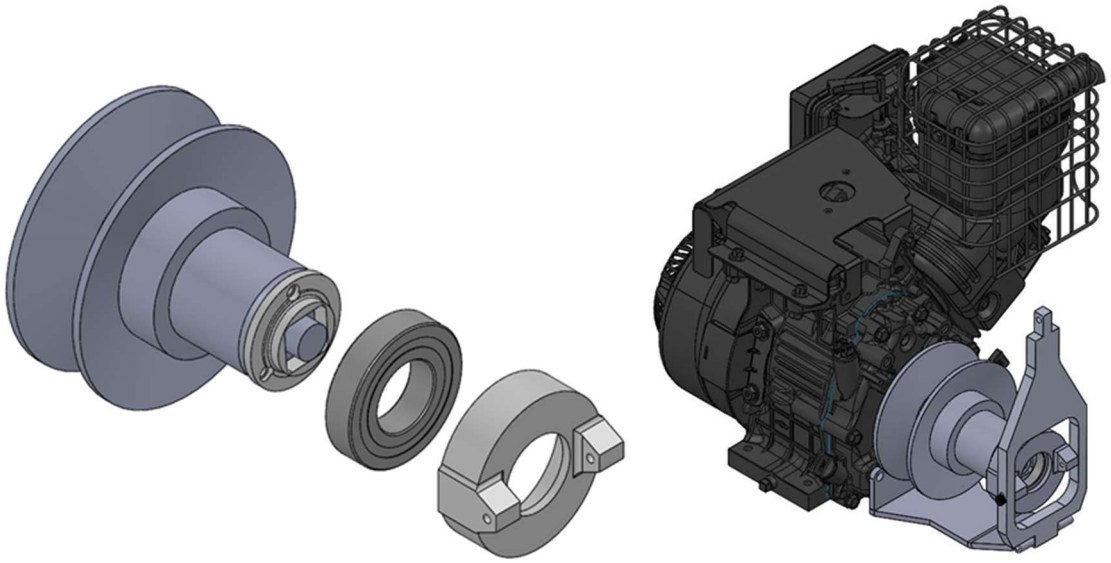

Figure 8: Solidworks model of Gaged driving pulley retrofitted; exploded view (left), full assembly on engine (right)

Figure 9: Prototype driving pulley actuated by push-pull cable

Figure 10: Comparison of Gaged mechanically actuated driving pulley (left) and manually actuated driving pulley (right)

Manual CVT Prototype: Testing Results/Conclusions

The implementation of manual shifting with the clutch fork design proved to be successful. With actuation of the moving sheave being done outboard, a large cantilever steel component was necessary, adding weight to the system. A design change was made to get rid of the cantilever beam in the final design.

eCVT Prototype Design: Actuator Selection

To implement the complete control system, a method of actuation had to be selected. To capture the dynamic forces necessary for actuation and speed of actuation, testing was conducted using the manual shifting system. Strain gages were applied to the clutch fork to attain forces and actuation speed was attained using a few methods, IR distance sensor on primary and CVT ratio change.

Since the driven pulley greatly affects the actuating force necessary, various preloads were tested using the lightest spring necessary for maintaining proper clamping force. Three scenarios tested for actuator sizing

was hill climb where the car started on a hill from a standstill, acceleration from a stop on asphalt, and cornering. All tests were conducted in M-Lot and the area behind the lot.

Expected results were that actuating forces necessary would be greatest in the overdrive position. Results from the IR distance sensor proved to be very noisy to the extent that the data was unusable qualitatively and quantitatively. Thus, CVT ratio data from the same test and desert testing was used to determine actuating speed. The desert test considered was where laps were driven with interest of inducing high loadings at the wheel to design the gearbox around. Actuating force and speed data were analyzed separately to determine maximums and time spent in each range of force and speed. Since actuators are always specified by the amount of force, speed, and current consumption at any point, speed and force were compared from the M-Lot test data.

Analyzing the M-Lot test data, actuating forces were set up in a histogram of 10 partitions with each pertaining a percentage of time spent in a range. Forces were greatest in the startup of an acceleration run, reaching 325 lb clamping force and for 89.3% of the time forces only reached ~120 lb. Actuating speed data was laid out in a histogram of 30 partitions to determine the time spent in each actuating speed subset. Desert test results displayed a consistent actuating speed of 2.5in/sec for 98.7% of the time for 10 min of driving. M-Lot tests resulted in a gaussian spread of actuating speed with 95.7% of the time considering speeds of up to 2.7 in/sec and at 90% spread, speeds of up to 1.3 in/sec were seen. To determine the actuating speed with the corresponding actuating force an XY-plot was used to analyze the relation. The distribution of actuating speed in each direction was asymmetric due to the aspect of shifting where little resistance was imposed when downshifting (negative direction) as opposed to upshifting (positive direction). Developing a trend line to encompass most of the data points, a trend could be developed where at no load linear actuation is 3 in/sec, and at a load of 250 lbf there is no motion.

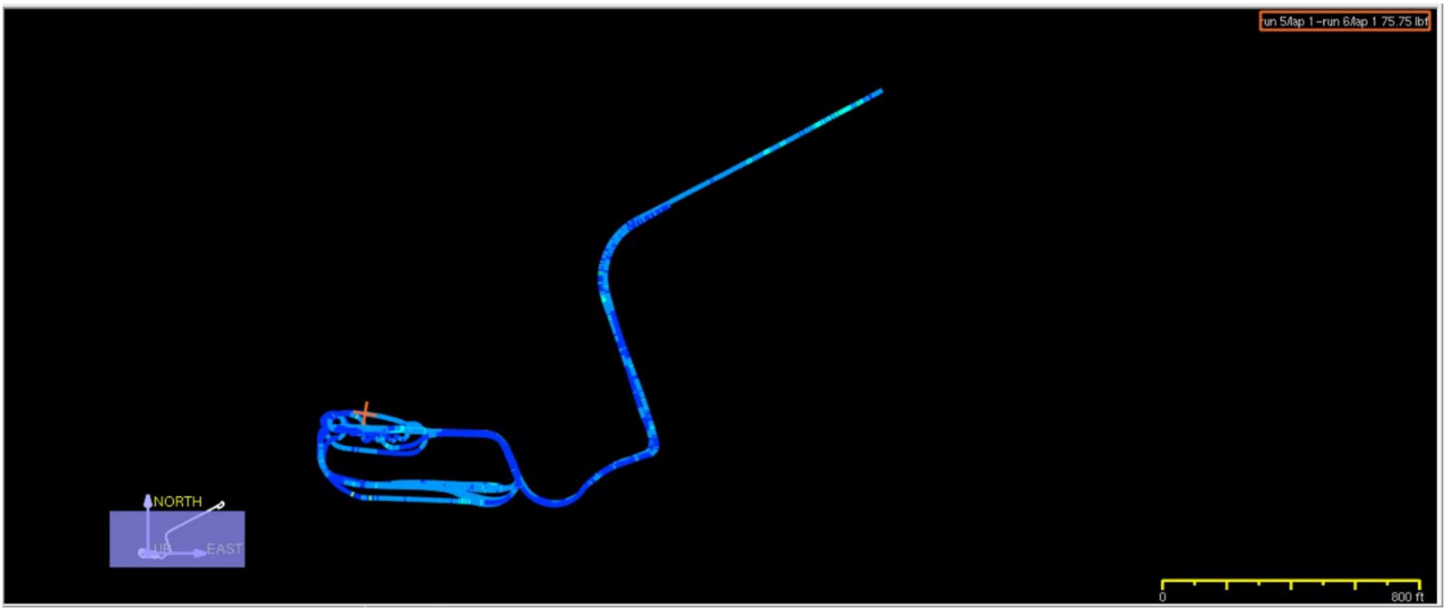

Figure 11: M-Lot GPS data with actuation forces highlighted

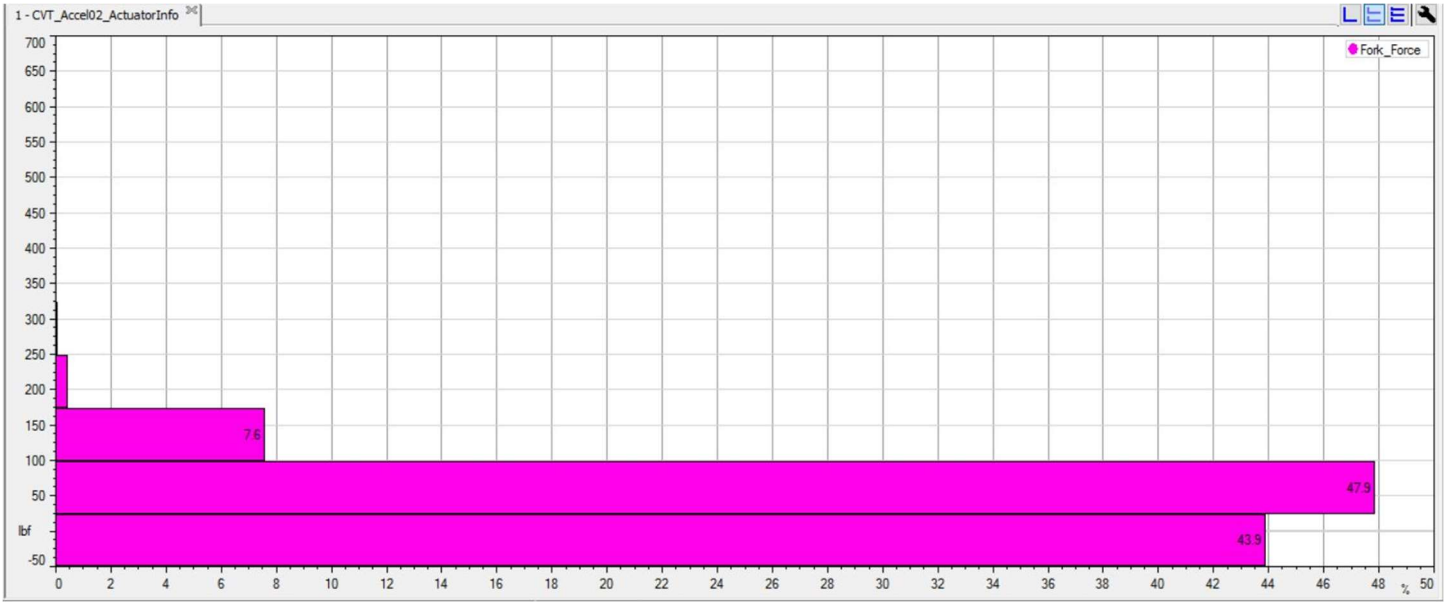

Figure 12: Actuation force histogram of all time driven (shown on GPS map) with 10 catagories.

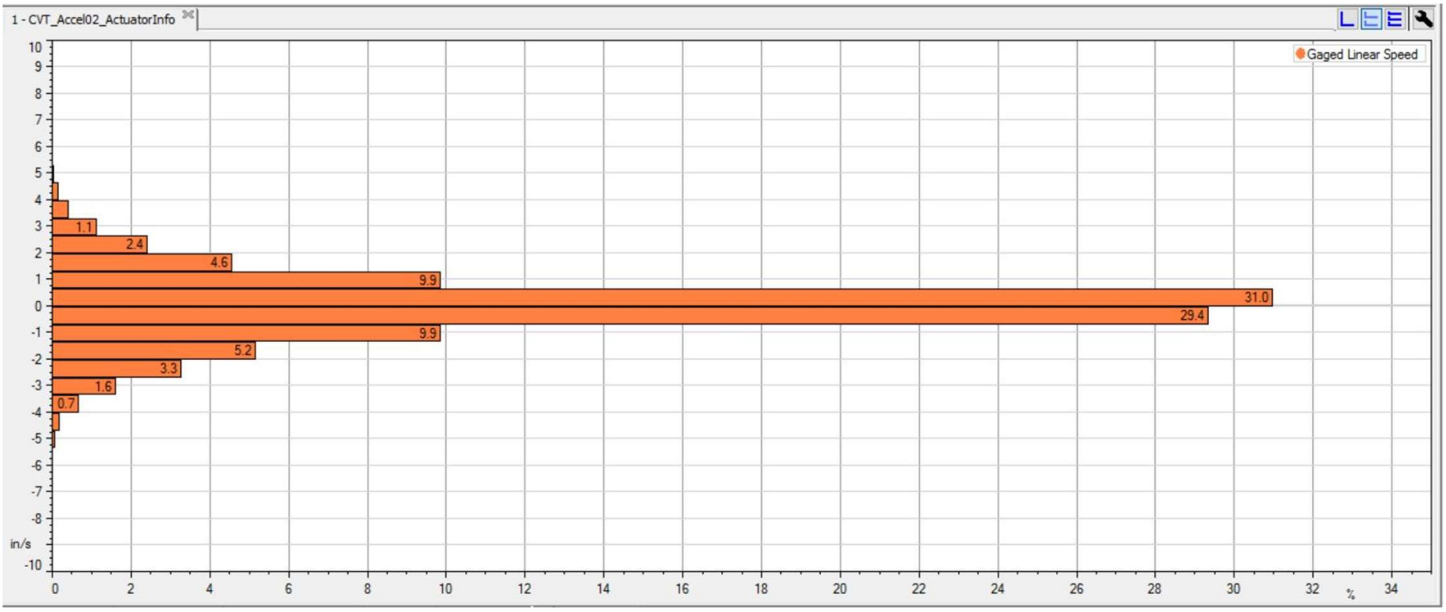

Figure 13: Actuation speed histogram of all time driven (shown on GPS map) with 30 catagories.

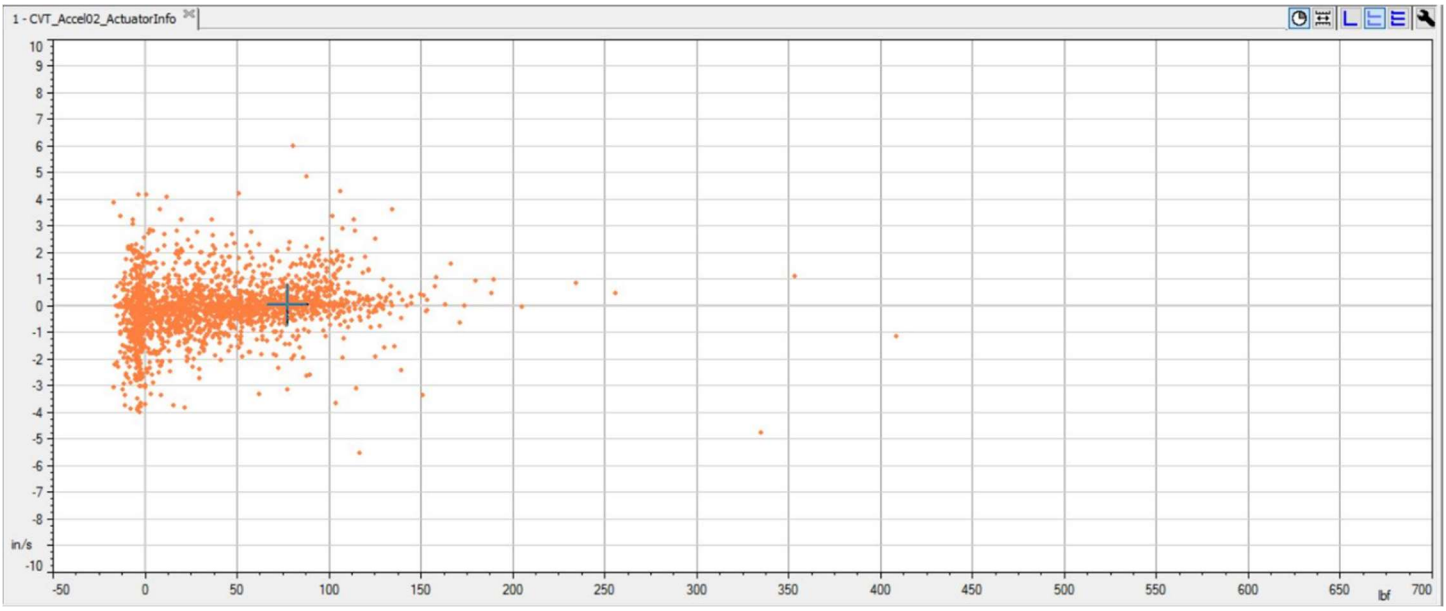

Figure 14: Actuation Speed versus Actuation Force from data shown in Figure 13

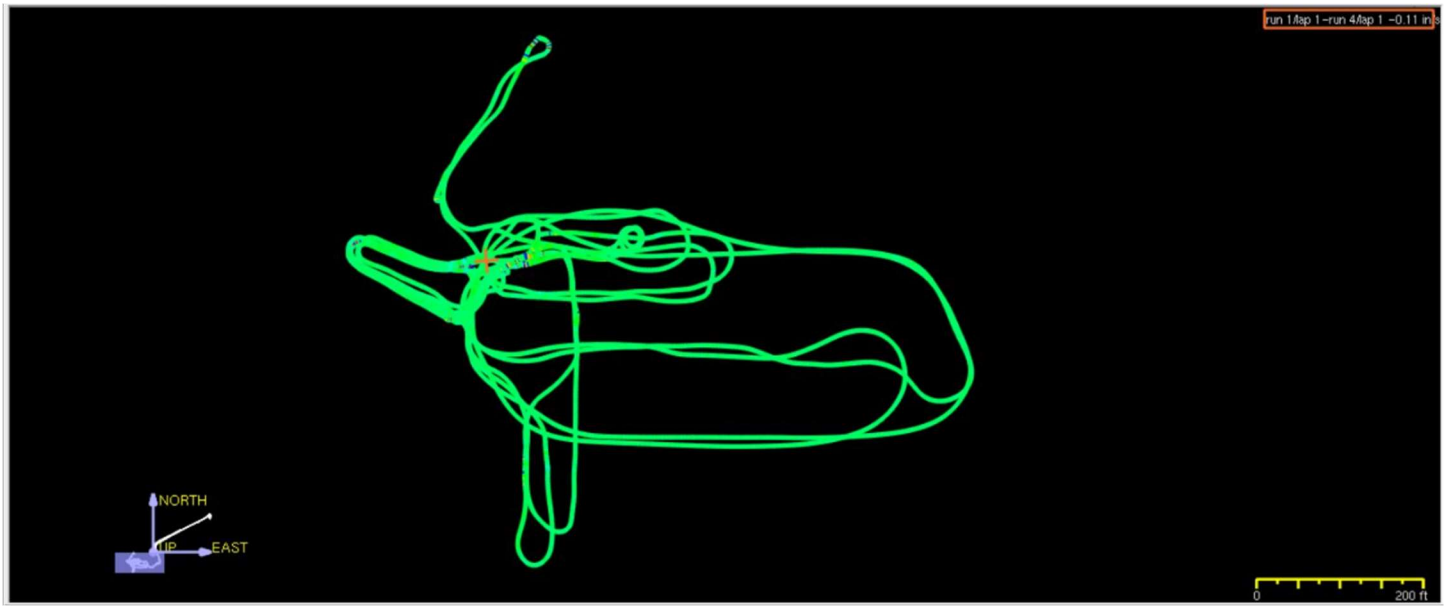

Figure 15: Stoddard Wells GPS data depicting actuation speed

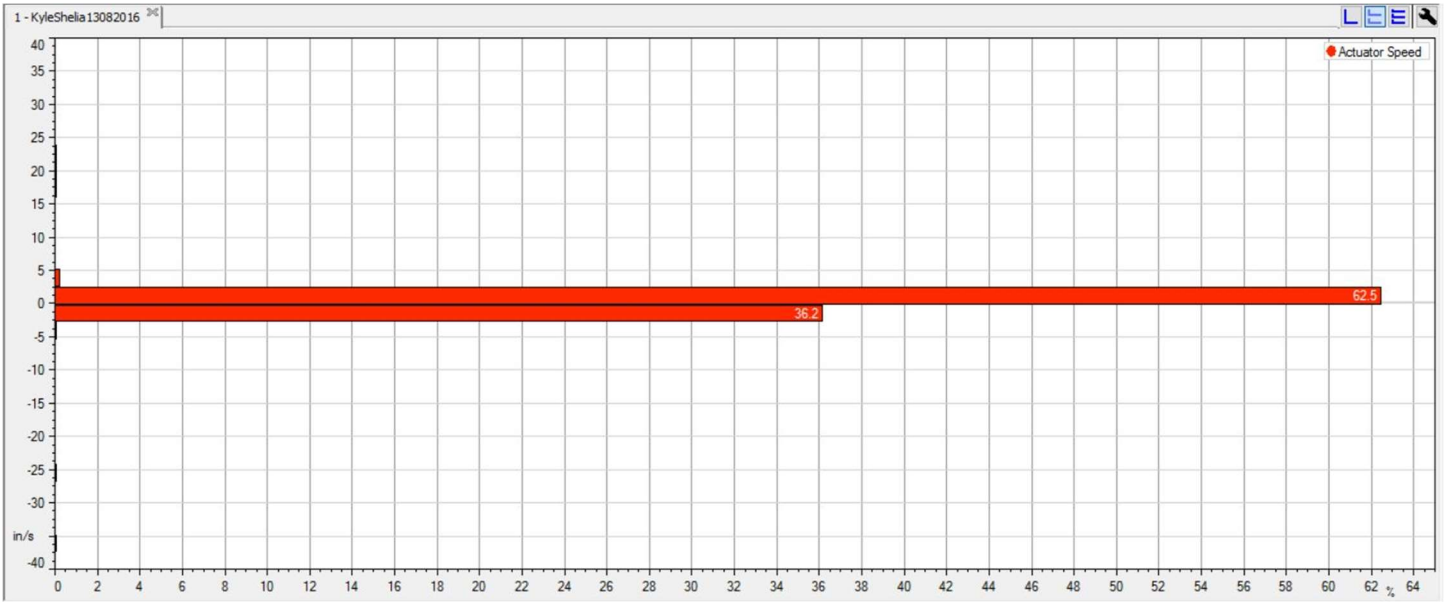

Figure 16: Actuation Speed Histogram from Stoddard Wells data

Actuators for consideration had to be water/dust proof to at least IP66 and a 12Vdc system. Once those requirements were met cost, actuation speed, force, current consumption, method of control, and feedback were considered.

Actuator Model	Control Method	Cost	Rated Duty Cycle (Full Load/Half Load)
Thomson Linear: HD Electrak	Servo - Encoder position feedback	~$700	25%/(N/A)
	Servo - Can Bus info feedback	~$730	25%/(N/A)
Warner Linear: B-Track K2x	N/A	~$600	10%/50%
	N/A - Position feedback	~$700	10%/50%

*Cost comparisons are made for stroke lengths of 4 inches (100mm)

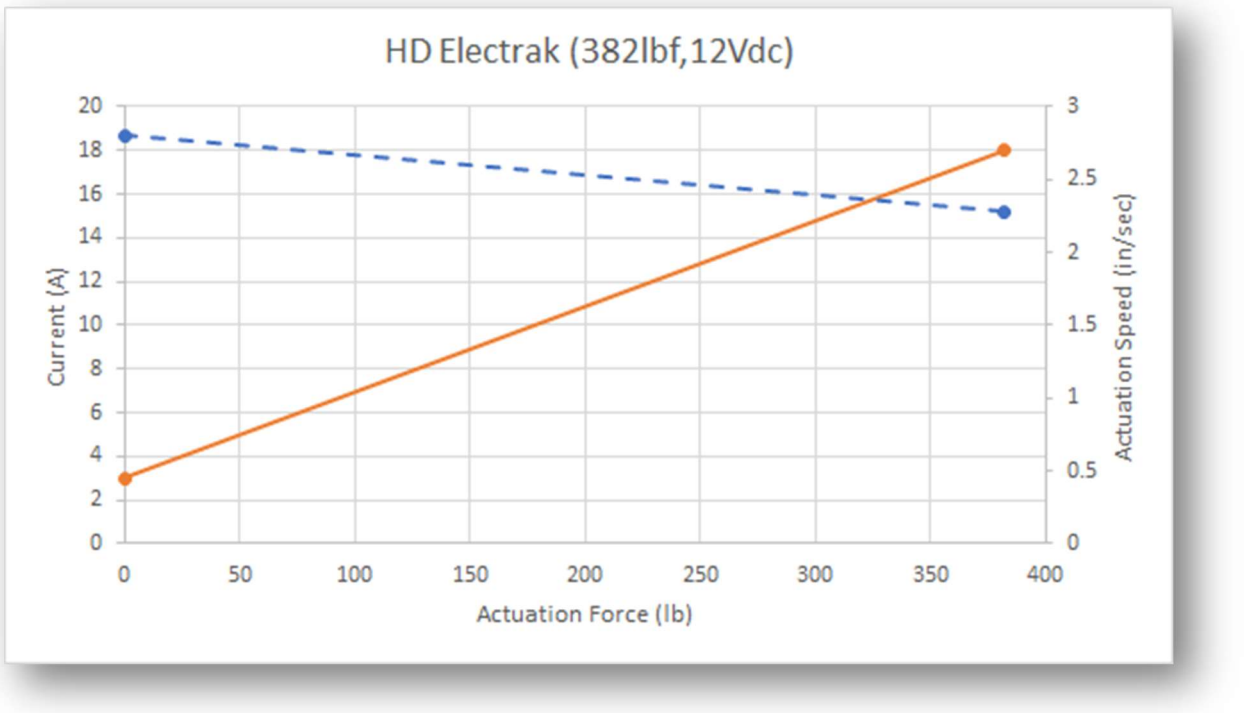

Figure 17: Current and Actuation speed versus Actuation force relations given by manufacturer.

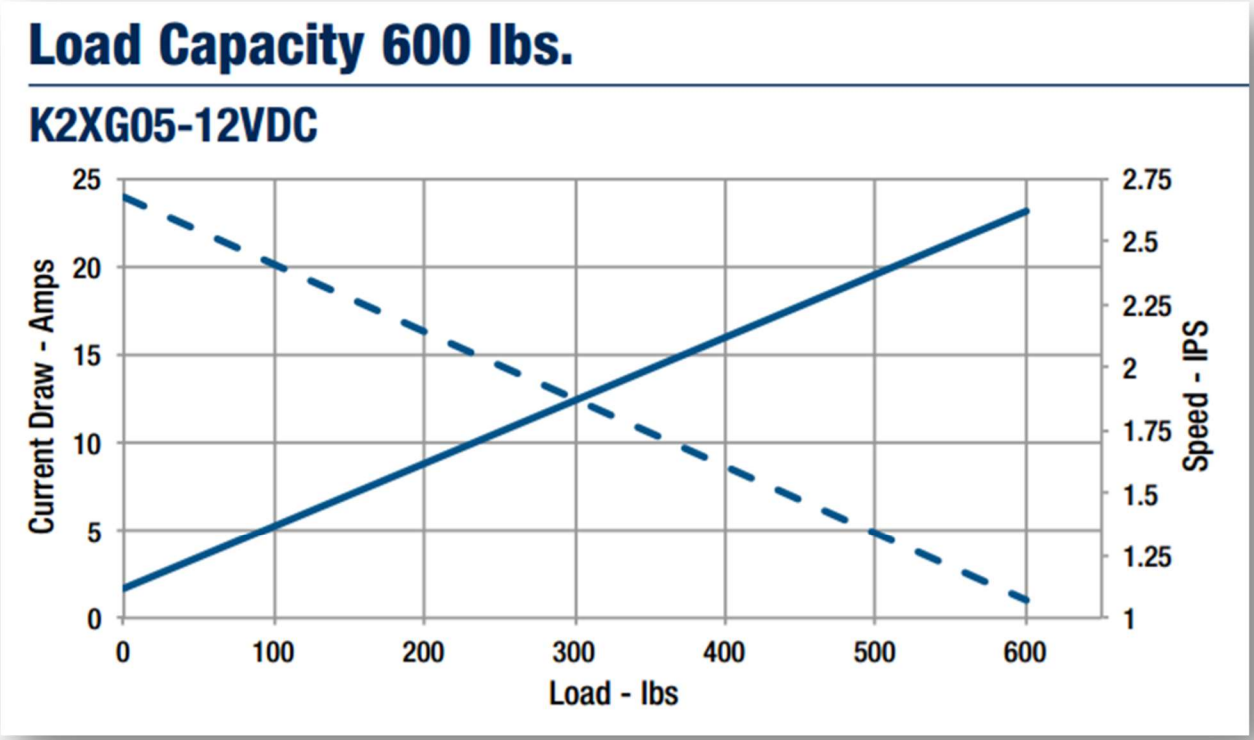

Figure 18: Current and Actuation speed versus Actuation force relations given by manufacturer.

Ultimately the HD Electrak linear actuator with CAN Bus communication was selected due to the high duty cycle, maximum speed at full load, and the added ability to attain various feedback information and control aspects besides position for a minimal addition in cost. Feedback information is provided on temperature, current, voltage, overload, and position. Controllable features are allowable current use, allowable actuation speed, position command, and enabling/disabling of actuator motion.

eCVT Prototype Design: RPM Feedback Implementation

Considerations for the RPM sensor was cost, size, method of mounting, setup and machining for sensor actuation, signal waveform, input/output voltage, and input/output current. Hall effect sensors are commonly used for rpm measurements, thus selection was a matter of choosing between having external magnets inducing a signal or a tone ring inducing a signal to a geartooth hall effect sensor. Since manufacturing a collar to install

various magnets to increase the number of readings per rotation proved less practical, a larger sensor size was accepted with the use of a tone ring.

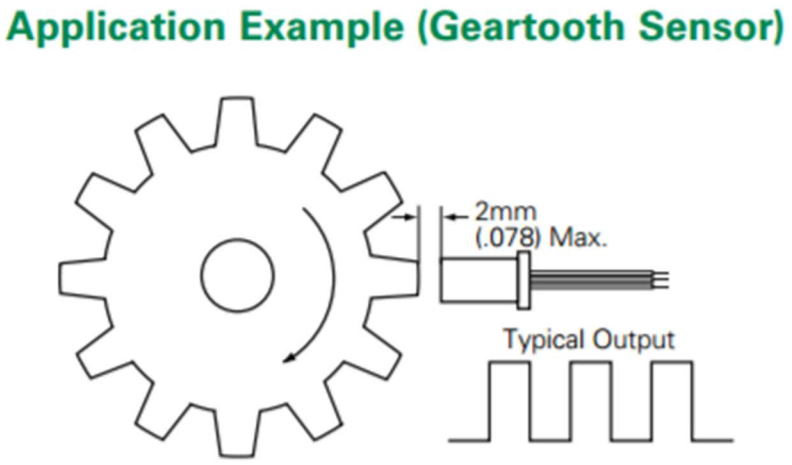

Figure 19: Geartooth sensor application example (55505)

Figure 20: Implementation of Hall Effect and tone ring for the prototype CVT

The sensor used was the Littelfuse 55505 Hall Effect since it met the criteria needed to interface with the Arduino Mega 2560 and was the most cost effective on the market.

After a tone ring and sensor mount was manufactured, the rpm feedback to the Arduino was tested. With the 5v output to the Arduino and ground going to the hall effect sensor, the hall effect output went to the Arduino interrupt pin. The interrupt pin was used to have rpm reading errors minimized while running through various other commands. As each tooth on the tone ring passes the hall effect, the *void loop* function is interrupted and the function correlated with the interrupt pin is performed, then once it completes a single cycle in the interrupt function, the program will continue where it left off in the *void loop* function. To calculate the rpm, each passing of a tooth on the tone ring is counted with the *void RPMp* function, and the rpm is calculated once every cycle within the *void loop* function.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
double Pdt; //differential time for primary pulley hall effect
unsigned long PcurrentMillis; // Timestamp after engine rpm calculation was last completed
unsigned long PpreviousMillis = 0; //Timestamp just before engine rpm calculation is completed

volatile int ErpmCounter = 0; //Primary rpm counter
double Erpm = 0; //Engine rpm read from hall effect
uint8_t  Nt = 15; // # of teeth on toner
const byte EngineInterruptPin = 3; //Pin set for interrupt (INT 1)to primary hall effect

void setup (){
attachInterrupt(digitalPinToInterrupt(EngineInterruptPin), RPMp, RISING); //Setup interrupt pin for hall effect
to read Engine RPM
}
void RPMp() {
 ErpmCounter += 1;
}

void loop (){
  if (ErpmCounter >= 1){
        int i = ErpmCounter; //Value is passed to local variable to ensure no calculations are done with a
volatile     variable that consistently updates
        ErpmCounter = 0;
        PcurrentMillis = millis();
        Pdt = PcurrentMillis - PpreviousMillis;
        Erpm = (i * 60000) / (Nt * (double)Pdt);
        PpreviousMillis = PcurrentMillis;
    }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Testing of the system was conducted by comparing to the engine RPM read on the Mychron data logging system, which reads the RPM by the engine spark. The engine RPM was compared at idle and at full throttle. Arduino RPM readings at a baudrate of 115200 resulted in high amounts of noise which necessitated a method of filtering to reduce the high fluctuations in the RPM reading.

Figure 21: MyChron4 data logger and display

A moving average filter was introduced to limit fluctuations, with each RPM reading taken as a data point. To minimize effort by the Arduino microcontroller, the moving average was implemented without the use of a shifting array with a finite number of data points.

Example of implemented moving average without array storing data points:

Erpm = (IntervalErpm / Pavg) - (Erpm / Pavg) + Erpm

Erpm = (4000/30)-(3210/30)+3210

Erpm = 133.333 - 107 + 3210

Erpm = 3236.333

Example of normal moving average:

Erpm = ((3210*29)+4000)/30

Erpm = 3236.333

If the rpm readings obtained were 4000 rpm for four more data points while comparing to a normal moving average, the results would be as follows:

Without Array Storage	With Array Storage
3236.333	3236.333
3261.7889	3262.666
3286.3959	3289
3310.1827	3315.333
3333.1766	3341.666

It's apparent that the results are close, but the method used is slightly less responsive than a regular moving average. Looking at actual readings, the method of not utilizing an array to store values is more than sufficient. The primary concern with using a filter was the phase lag between unfiltered readings and filtered readings. Initially, the number of readings was set to where noise was visually minimized on the Arduino serial plotter for a compromise between filtering noise and a phase lag no greater than ~0.25 seconds. Conducting the test for RPM reading accuracy again with the newly implemented filter, readings resulted in a difference of +/- 10 RPM from the Mychron RPM output.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
double Pdt; //differential time for primary pulley hall effect
unsigned long PcurrentMillis; // Timestamp after engine rpm calculation was last completed
unsigned long PpreviousMillis = 0; //Timestamp just before engine rpm calculation is completed

volatile int ErpmCounter = 0; //Primary rpm counter
int ErpmMAcounter = 0; //Primary rpm moving average data point counter
double Erpm = 0; //Engine rpm read from hall effect
double IntervalErpm = 0; //Intermediate engine rpm reading used as data point for moving average

uint8_t  Nt = 15; // # of teeth on tone ring
double Pavg = 5; //Window size (# of data points) for moving average
const byte EngineInterruptPin = 3; //Pin set for interrupt (INT 1)to primary hall effect

void setup (){
```

```
attachInterrupt(digitalPinToInterrupt(EngineInterruptPin), RPMp, RISING); //Setup interrupt pin for hall effect to read Engine RPM
}
void RPMp() {
 ErpmCounter += 1;
}
void loop (){
  if (ErpmCounter >= 1){
        int i = ErpmCounter; //Value is passed to local variable to ensure no calculations are done with a volatile     variable that consistently updates
        ErpmCounter = 0;
        PcurrentMillis = millis();
        Pdt = PcurrentMillis - PpreviousMillis;
        Erpm = (i * 60000) / (Nt * (double)Pdt);

        if (ErpmMAcounter < Pavg) //At arduino startup, setup is necessary to grow moving average to the size specified by variable Pavg {
                ErpmMAcounter += 1;
                SumErpm += IntervalErpm;
                Erpm = SumErpm / ErpmMAcounter;
                if (ErpmMAcounter == Pavg)
                {
                      Erpm = SumErpm / Pavg; // While growing moving average for arduino startup, output average of rpm readings collected thus far
                }
         }
        else  {
                      Erpm = (IntervalErpm / Pavg) - (Erpm / Pavg) + Erpm ; //Moving average calculation after growing the moving average to the specified window size
          }
          PpreviousMillis = PcurrentMillis;
    }
}
```

//

eCVT Prototype Design: Implementation of Actuator Control

With the selection of the actuator requiring control through CAN Bus communication, it was then necessary to set up the Arduino microcontroller as such. For ease of implementation, an Arduino shield was used, specifically the CAN-BUS Shield V1.2 from Seeed Studio. An accompanying library was used to implement the use of the shield as well.

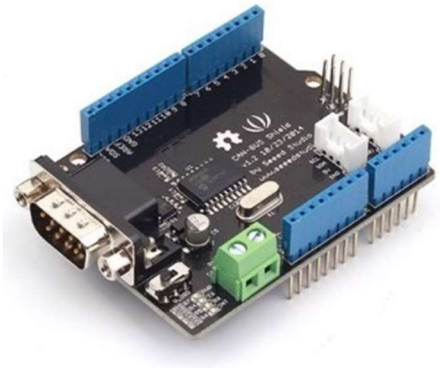

Figure 22: Seeed Studio's CAN Bus Arduino shield

Since the shield was developed for the Arduino Uno, the SPI pins going to the shield had to be rerouted for use with an Arduino Mega.

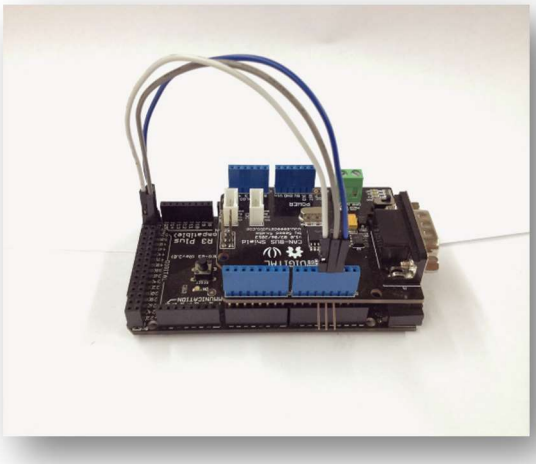

Figure 23: CAN Bus Shield retrofitted to fit Arduino Mega

Arduino Mega Pin	CAN Bus Shield Pin	SPI Pin Description
50	12	MISO
51	11	MOSI
52	13	SCK

Jumper Wire connections

Another change to the shield that was necessary was a replacement of the termination resistor. For proper communication, it is necessary to have similar termination resistors at each end of the CAN Bus lines. The reason being that for the signal to quickly switch to a low state, any potential held by the long wires would render the signals unreadable as it deviates from a square wave.

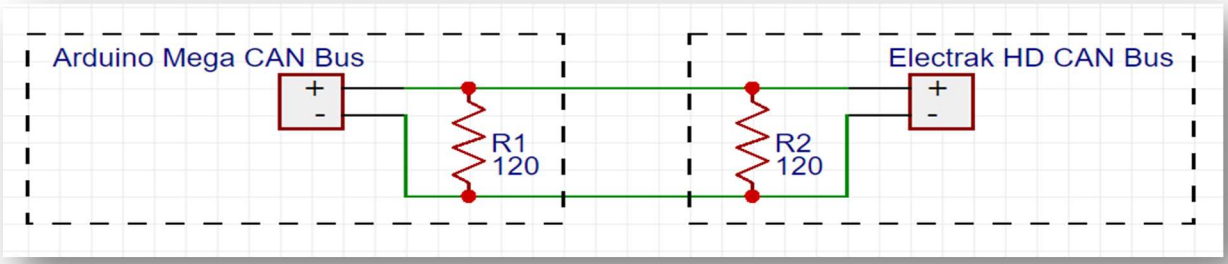

Figure 24: Electrical Schematic of CAN Bus wiring

With the hardware properly set up, the software had to be set up according to instructions given by manufacturer of the actuator.

In general, there are a few important considerations when dealing with CAN Bus communication. The method of communication is a series of square waves representing a high and a low state that can be interpreted by manipulation of the bits of a byte. Each series of signals is sent as one long message using standardized formatting, as shown in the slide image below. For the protocol in J1939, the extended format for addressing is used (29-bit extended form). All addressing information is explained in the SAE J1939 standard, but for this project, it was not directly referenced. To attain the actuator address, it was retrieved using a microcontroller with the same setup as the Arduino but a PIC was used to communicate with the microcontroller. The reasoning behind this was an issue found that the library used with the Arduino shield was setup to display an address that was not an extended format, thus always cutting off the full address. So, the address found was 0x00EF13FF where the address could be changed easily by hardware means, as mentioned in the actuator manual "*5.2.2 Address.*"

Another crucial aspect in CAN communication is the baudrate used, since all nodes in the network must interpret the same information, the timing for each node must be the same; J1939 requires a baudrate of 250k (250 kbit/sec) for communication between each node.

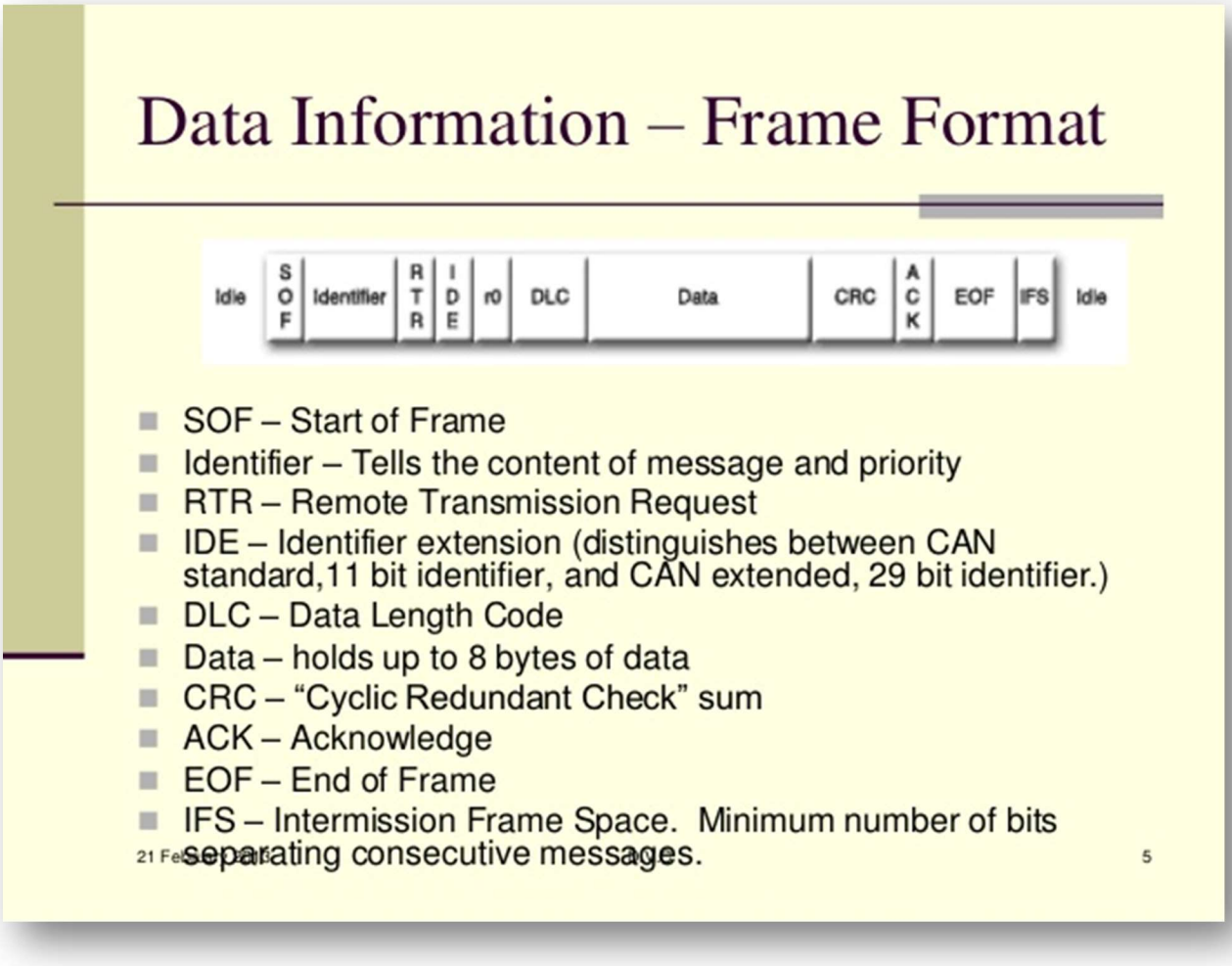

Figure 25: Frame format of CAN Bus message

After setting up the baudrate and address to the actuator, the data for the CAN messages needed to be formatted. As previously mentioned, this system allows control over various aspects of the actuator and the exact format required for commands is specified in the actuator user manual in section "*5.2.4 J1939 actuator control message (ACM).*" To fit the commanded position into a CAN message, the value must be changed to a bit value as described in the manual.

For example, if a position command of 90mm out of the 100mm travel of the actuator was to be sent:

Position = 90/0.1 = 900

Position = 0b 0000 0011 1000 0100 (binary)

To fit the command value into the message, bit masking operations are done to format the message required for the actuator to interpret the message. Taking the right number as the most significant digit, the first 8 bits are packaged into a byte as part of an array. The next byte of the array shares the last 2 bits with the current limit command, therefore an "AND" operation is performed to ensure only the first 6 bits with a high value transfer information, if there is any.

Bit masking example:

Data[0] = (uint8_t)Position = 0b 1000 0100

Data[0] = 132 = 0x84

Data[1] = (uint8_t)(Position >> 8) & 0x3F = 0b 0000 0011 & 0b 0011 1111

Data[1] = 0b 0000 0011 = 3 = 0x3

To interpret information from the actuator, the method is just conducted in reverse as shown in the code below.

Actuator Control Message Signal Information		
Start position	Length	Parameter name
1.1	14 bits	Position command
2.7	9 bits	Current limit
3.8	5 bits	Speed command
4.5	1 bit	Motion enable
4.6	35 bits	Factory use

The least significant bit of each message is indicated by the start position column

Actuator Feedback Message Signal Information		
Start position	Length	Parameter name
1.1	14 bits	Measured position
2.7	9 bits	Measured current
3.8	5 bits	Running speed
4.5	2 bits	Voltage error
4.7	2 bits	Temperature error
5.1	1 bit	Motion flag
5.2	1 bit	Overload flag
5.3	1 bit	Backdrive flag
5.4	1 bit	Parameter flag
5.5	1 bit	Saturation flag
5.6	1 bit	Fatal error flag
5.7	18 bits	Factory use

The least significant bit of each message is indicated by the start position column

To test the system, the full code was written with position feedback and position command to disengage the CVT, as if the actuator was mounted. Going through the code, to unpackage the information and package the information in an array, a function was made to call after determining all command values for messages to be sent and another after messages were received. For all messages sent to the actuator, a delay of 100 milliseconds had to be incorporated as specified in the manual of the same section for formatting the control message. A position was declared as the disengaged position in the section of variable declarations to know the expected position the actuator should return to. Actuation speed was also declared to see the speed at which the actuator would move.

Changing the actuator position was done manually with a hex cap on the end of the actuator, once powered on the actuator would return to the set home position. The change in speed commands had no visible change until 25% speed was commanded. For large changes in motion, the actuator would have an increased speed of motion while for small increments, the speed was not as quick. Actuator position accuracy was also compared with caliper measurements, and all measurements were accurate to the millimeter.

```
unsigned long MsgDelay = 0;//Variable for differential time between last message time and current time
unsigned long PreviousMsgDelay = 0;//Timestamp for when last CAN message was made
double error = 0;
uint8_t  ActuatingSpeed = 40;//Percent of  duty cycle for actuator (0% being slowest speed possible to run
forever and 100% being fastest speed possible given loading conditions)
uint8_t DisengagementSpeed = 100;
uint8_t Speed = 0;
uint8_t CurrentLimit = 20; //Amps of current accepted to be used by the actuator
double Disengage = 90; //Position at which the actuator will move to assure disengagement of the CVT (in mm)
#include <mcp_can.h> //CAN Shield Library
#include <SPI.h> //Arduino SPI Library
#define CONTROLLER_ID 0x00EF13FF //Actuator CAN address
uint8_t   Data[8]; //array to hold data for CAN message to the actuator
uint8_t   ReadData[8]; //array to hold data from CAN message received from actuator
uint8_t   DataLength; //Variable specifying data length of CAN message
uint16_t  Position; //Variable holding position command
bool MotionEnable = false; //Initiate motion parameter of actuator as false to assure no movement
//variables to hold information received from actuator
uint16_t  PositionCheck;
uint16_t  CurrentCheck;
uint8_t  SpeedCheck;
uint8_t  VoltageError;
uint8_t  TempError;
bool MotionCheck = false ;
bool CurrentOverload = false;
bool BackdriveFlag = false;
bool ParameterFlag = false;
bool SaturationFlag = false;
const int SPI_CS_PIN = 9; //SPI interrupt pin used by the CAN Shield for communication
MCP_CAN CAN(SPI_CS_PIN);// Set CS pin for CAN Bus Shield

void setup(){
    Serial.begin(115200);
    while (CAN_OK != CAN.begin(CAN_250KBPS))  // init can bus : baudrate = 250k
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println(" Init CAN BUS Shield again");
        delay(100);
    }
    Serial.println("CAN BUS Shield init ok!");
    CurrentLimit /= 0.1; //convert current limit from Amps to a bit value
    MotionEnable = true; //motion command
    Position = Disengage / 0.1; //convert position command to bit value
    Speed = ActuatingSpeed / 5; //convert actuator speed to bit value
    EncodeMotionParameters(); //Format commands into array for CAN message
    CAN.sendMsgBuf(CONTROLLER_ID, 1, 8, Data); //Send CAN message
    delay(100);
}
void EncodeMotionParameters(void) //Since the information sent must share a byte, the bits are shifted to be
read by the actuator
```

```
{
   /*  ********************************************************************
     Position                  Start Bit Position (LSb)       Length
     ********************************************************************
     Position Command          1.1                  14 bits
     Current Limit             2.7                  9 bits
     Speed Command             3.8                  5 bits
     Motion Enable             4.5                  1 bit
     Factory Use               4.6                  35 bits (all 0x00 or 0xFF)
     ********************************************************************/
   /*************************************************************************************
****
     PPPP-PPPP, CCPP-PPPP, SCCC-CCCC, FFFM-SSSS, FFFF-FFFF, FFFF-FFFF, FFFF-FFFF, FFFF-FFFF
       byte1          byte2          byte3          byte4          byte5          byte6          byte7          byte8
  ************************************************************************************
***/
    Data[0] = (uint8_t)Position;
    Data[1] = (uint8_t)(Position >> 8) & 0x3F;
    Data[1] |= ((uint8_t)(CurrentLimit << 6) & 0xC0);
    Data[2] = (uint8_t)(CurrentLimit >> 2) & 0x7F;
    Data[2] |= ((uint8_t)(Speed << 7) & 0x80);
    Data[3] = (uint8_t)(Speed >> 1) & 0x0F;
    if (MotionEnable == true){
        Data[3] |= 0x10;
    }
    else {
         Data[3] &= 0xEF;
    }
}
void DecodeMotionParameters(void){
    PositionCheck = ((uint16_t)ReadData[0]);
    PositionCheck |= ((uint16_t)ReadData[1] & 0x3F) << 8;
    PositionCheck *= 0.1; //convert from bit value to mm
    CurrentCheck = ((uint16_t)ReadData[1] & 0xC0) >> 6;
    CurrentCheck |= ((uint16_t)ReadData[2] & 0x7F) << 2;
    CurrentCheck *= 0.1; //convert from bit value to Amperes
    SpeedCheck = ((uint8_t)ReadData[2] & 0x80) >> 7;
    SpeedCheck |= ((uint8_t)ReadData[3] & 0x0F) << 1;
    SpeedCheck *= 5; //convert from bit value to motor duty cycle
    VoltageError = ((uint8_t)ReadData[3] & 0x30);
    // 0b00 Input Voltage in Range
    // 0b01 Input Voltage below operational range
    if (VoltageError == 0x1){
        Serial.println("Voltage below operational range");
    }
    // 0b10 Input Voltage above operational range
    if (VoltageError == 0x2){
        Serial.println("Voltage above operational range");
    }// 0b11 Not Used
    TempError = ((uint8_t)ReadData[3] & 0xC0);
```

```
    // 0b00 Temp within Operational Range
    // 0b01 Temp below operational range
    // 0b10 Temp above operational range
    if (TempError == 0x2){
        Serial.println("Temp above operational range");
    } // 0b11 Not Used
    MotionCheck = (bool)ReadData[4] ;
    CurrentOverload = (bool)(ReadData[4] >> 1);
    BackdriveFlag = (bool)(ReadData[4] >> 2);
    ParameterFlag = (bool)(ReadData[4] >> 3);
    SaturationFlag = (bool)(ReadData[4] >> 4);
}
void loop(){
    CAN.readMsgBuf(DataLength, ReadData);    // read data from actuator (data length, data buf)
    DecodeMotionParameters(); //call function to decode CAN message
    error = Disengage - PositionCheck; //Determine if actuator is in a position where it engages the belt
    if (error > 0)//If actuator is engaging contact with the belt, change actuator position to disengage belt
    {
        MotionEnable = true;
    }
    else //Let actuator maintain current position since it is disengaged
    {
        MotionEnable = false;
    }
    Position = Disengage / 0.1;
    Speed = DisengagementSpeed / 5;
    MsgDelay = millis() - PreviousMsgDelay;
    if (MsgDelay >= 100) //Actuator commands can only be updated every 100 milliseconds or more
    {
        EncodeMotionParameters();
        CAN.sendMsgBuf(CONTROLLER_ID, 1, 8, Data);
        PreviousMsgDelay = millis();
    }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///////
```

eCVT Prototype Design: Control Implementation

With all electronics and mechanical hardware installed, PID control of the system was implemented.

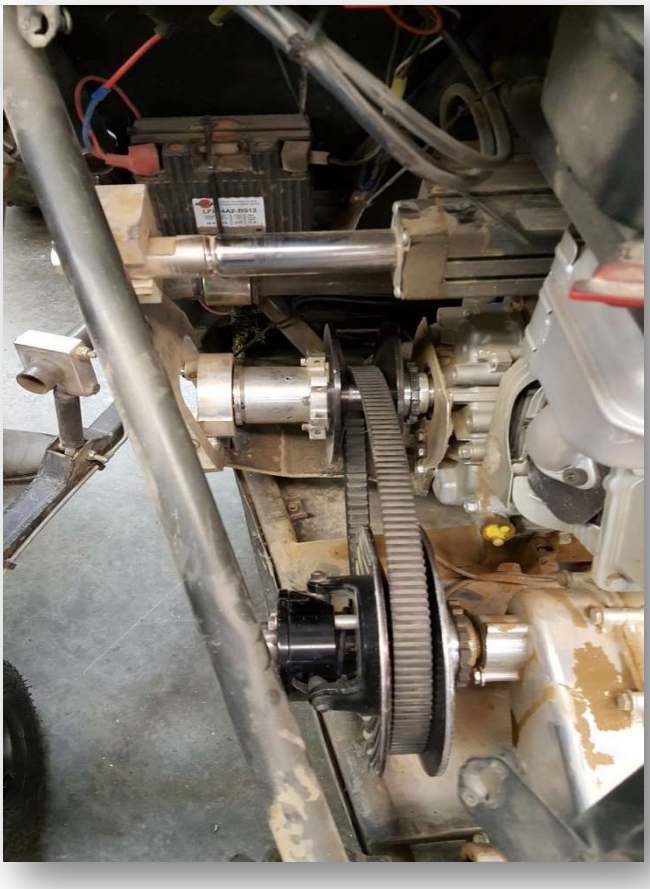

Figure 26: Image of fully controlled CVT prototype

Since actuator control turned out to be quite accurate, PID control was used for only maintaining engine RPM. Before delving into the controls software, parameters had to be set for CVT underdrive, overdrive, and actuation speed. To set the actuator overdrive position, the V-Belt was removed and using the previous code, the overdrive position was set to the point where the sheaves were in contact with each other. Underdrive position was set by placing the prototype car on the chassis dyno and inducing a large wheel load (10A on the dyno which equates to ~130ft-lb) to visually determine if there was not enough clamping force on the belt at the position the actuator maintained. Actuator position was adjusted in 1mm increments until there was visually no slipping between the V-belt and sheaves. Ideally the underdrive should have been set by comparing actuator

position to the CVT ratio to ensure that it is obtaining CVT ratios given by the manufacturer. This process would have taken time and to speed up the process for a working prototype, it was determined satisfactory and could be adjusted later.

With the actuator boundaries set, it was then necessary to determine the actuation speed necessary. Despite the actuator being selected based on data collected for actuation speed, it was necessary to validate these findings. A code was written to determine actuation speed by actuating to overdrive position if the RPM exceeded a determined setpoint, and if below the actuator would move to the underdrive position. The actuation speed parameter was adjusted in increments of 5% since that was the resolution provided by the actuator (Refer to manual section "*5.2.4 J1939 actuator control message (ACM)").* Actuation speed is limited by the belts ability to withstand the dynamic shear induced while shifting (~1800 - 3600 rpm is the full working range).

Without a dynamic model of the actual forces induced and knowledge of the belts breaking point dynamically, it would not be possible to accurately determine the optimal speed. Therefore, actuation speed was based on visual inspection of the V-belt's behavior while shifting. Upshifting applies a larger load to the belt for clamping due to the mechanics of the driven pulley. As the actuator would upshift, attention was placed to any deformation the belt may have exhibited and jerking, if there was any. The resulting actuation speed resulted in 30% of full capacity, but as noted before it is a number representing motor duty cycle so speed does change with load despite being limited.

Actuation Speed	30% of full capacity
Underdrive Position	64 mm of extension
Overdrive Position	4 mm of extension

Results from determining actuator parameters

```
double dt; //time difference of current PID loop (same variable reused)
double Pdt; //differential time for primary pulley hall effect
unsigned long PcurrentMillis; // Millis = Milliseconds
unsigned long PpreviousMillis = 0;
unsigned long MsgDelay = 0;
unsigned long PreviousMsgDelay = 0;
volatile int ErpmCounter = 0; //Primary rpm counter
int ErpmMAcounter = 0; //Secondary rpm average data point counter
double Erpm = 0; //Engine rpm read from hall effect
double SumErpm = 0;
double IntervalErpm = 0; //Represents rpm calculated between each tooth
const byte EngineInterruptPin = 3; //Pin set for interrupt (INT 1)to primary hall effect
//CVT Primary closed position
double PosUR = 64; //Position of actuator (in mm) when CVT is in underdrive (6-22-17; CV-Tech Underdrive:
82, Overdrive: 25)
//CVT Primary open position
double PosOR = 4; //Position of actuator (in mm) when CVT is in overdrive
double Disengage = 93; //Position at which the actuator will move to assure disengagement of the CVT (in mm)
(6-22-17; CV-Tech Disengaged Position: 93)
uint8_t CurrentLimit = 20; //Amps of current accepted to be used by the actuator, if exceeded will lock up the
actuator
double Pavg = 5; //Counter for # of teeth thats counted on Primary before outputing a new rpm
uint8_t  Nt = 15; // # of teeth on tone ring
double SPpp = 3000;  //setpoint of Peak torque RPM of engine
double EngagementRPM = 1500;  //engagement rpm of engine; where actuator begins to maintain low ratio of
CVT
uint8_t  ActuatingSpeed = 30;//Percent of  duty cycle for actuator (0% being slowest speed possible to run
forever and 100% being fastest speed possible given loading conditions)
uint8_t  DisengageSpeed = 5;
uint8_t Speed = 0;
#include <mcp_can.h>
#include <SPI.h>
#define CONTROLLER_ID 0x00EF13FF
uint8_t   Data[8];
uint8_t   ReadData[8];
uint8_t   DataLength;
uint16_t  Position;
uint16_t RemappedPositionCheck;
uint16_t  PositionCheck;
uint16_t  CurrentCheck;
uint8_t  SpeedCheck;
uint8_t  VoltageError;
uint8_t  TempError;
bool MotionCheck = false ;
bool CurrentOverload = false;
bool BackdriveFlag = false;
bool ParameterFlag = false;
bool SaturationFlag = false;
bool      MotionEnable = false; //Initiate motion parameter of actuator as false to assure no movement
uint32_t  LastTxTime = 0;
```

```
uint32_t  LastPosChangeTime = 0;
const int SPI_CS_PIN = 9; //SPI interrupt pin used by the CAN Shield for communication
MCP_CAN CAN(SPI_CS_PIN);// Set CS pin
void setup(){
     Serial.begin(115200);
     while (CAN_OK != CAN.begin(CAN_250KBPS))             // init can bus : baudrate = 250k
      {
         Serial.println("CAN BUS Shield init fail");
         Serial.println(" Init CAN BUS Shield again");
         delay(100);
      }
     Serial.println("CAN BUS Shield init ok!");
    CurrentLimit /= 0.1; //convert current limit from Amps to a bit value
   //assure disengagement of the CVT
    MotionEnable = true;
    Position = Disengage/0.1;
    Speed = DisengageSpeed/5;
    EncodeMotionParameters();
    CAN.sendMsgBuf(CONTROLLER_ID, 1, 8, Data);
    delay(100);
    attachInterrupt(digitalPinToInterrupt(EngineInterruptPin),RPMp, RISING); //Setup interrupt pin for hall
effect reading Engine RPM
}
void DecodeMotionParameters(void){
    PositionCheck = ((uint16_t)ReadData[0]);
    PositionCheck |= ((uint16_t)ReadData[1] & 0x3F) << 8;
    PositionCheck *= 0.1; //convert from bit value to mm
    CurrentCheck = ((uint16_t)ReadData[1] & 0xC0) >> 6;
    CurrentCheck |= ((uint16_t)ReadData[2] & 0x7F) << 2;
    CurrentCheck *= 0.1; //convert from bit value to Amperes
    SpeedCheck = ((uint8_t)ReadData[2] & 0x80) >> 7;
    SpeedCheck |= ((uint8_t)ReadData[3] & 0x0F) << 1;
    SpeedCheck *= 5; //convert from bit value to motor duty cycle
    //Serial.print("   Instantaneous Speed [%]:  ");
    //Serial.println(SpeedCheck);
    VoltageError = ((uint8_t)ReadData[3] & 0x30);
    // 0b00 Input Voltage in Range
    // 0b01 Input Voltage below operational range
    if (VoltageError == 0x1){
      Serial.println("Voltage below operational range");
    }
    // 0b10 Input Voltage above operational range
    if (VoltageError == 0x2) {
     Serial.println("Voltage above operational range");
    }
    // 0b11 Not Used
    TempError = ((uint8_t)ReadData[3] & 0xC0);
    // 0b00 Temp within Operational Range
    // 0b01 Temp below operational range
    // 0b10 Temp above operational range
```

```
    if (TempError == 0x2){
      Serial.println("Temp above operational range");
    }
    // 0b11 Not Used
    MotionCheck = (bool)ReadData[4] ;
    CurrentOverload = (bool)(ReadData[4] >> 1);
    BackdriveFlag = (bool)(ReadData[4] >> 2);
    ParameterFlag = (bool)(ReadData[4] >> 3);
    SaturationFlag = (bool)(ReadData[4] >> 4);
}

void EncodeMotionParameters(void){
      Data[0] = (uint8_t)Position;
      Data[1] = (uint8_t)(Position >> 8) & 0x3F;
      Data[1] |= ((uint8_t)(CurrentLimit << 6) & 0xC0);
      Data[2] = (uint8_t)(CurrentLimit >> 2) & 0x7F;
      Data[2] |= ((uint8_t)(Speed << 7) & 0x80);
      Data[3] = (uint8_t)(Speed >> 1) & 0x0F;
        if(MotionEnable == true){
        Data[3] |= 0x10;
        }
        else {
            Data[3] &= 0xEF;
        }
}
void RPMp() {
    ErpmCounter += 1;
}
void loop(){
      if (ErpmCounter >= 1) {
       int i = ErpmCounter;
       ErpmCounter = 0;
       PcurrentMillis = millis();
       Pdt = PcurrentMillis - PpreviousMillis;
       IntervalErpm = (i*60000)/(Nt*(double)Pdt);
                  if (ErpmMAcounter < Pavg){
                              ErpmMAcounter += 1;
                              SumErpm += IntervalErpm;
                              Erpm = SumErpm/ErpmMAcounter;

                              if (ErpmMAcounter == Pavg){
                                    Erpm = SumErpm/Pavg;
                              }
                   }

         else {
                    Erpm = (IntervalErpm/Pavg)- (Erpm/Pavg) + Erpm ;
         }
          PpreviousMillis = PcurrentMillis;
    }
```

```
  Serial.print(",");
  Serial.print("Erpm: ");
  Serial.print("\t");
  Serial.print(Erpm);
   CAN.readMsgBuf(DataLength, ReadData);    // read data,  len: data length, buf: data buf
   DecodeMotionParameters();
   RemappedPositionCheck = PositionCheck*100;
  Serial.print(",");
  Serial.print("  Instantaneous Position [mm]:  ");
  Serial.print("\t");
  Serial.println(RemappedPositionCheck);
    if ( Erpm < SPpp && Erpm > EngagementRPM) //Maintain underdrive position
    {
       MotionEnable = true;
       Position = (PosUR)/0.1;
       Speed = ActuatingSpeed/5;
       MsgDelay = millis() - PreviousMsgDelay;
       if (MsgDelay >= 100)
       {
          EncodeMotionParameters();
          CAN.sendMsgBuf(CONTROLLER_ID, 1, 8, Data);
          PreviousMsgDelay = millis();
       }

    }
  else if (Erpm > SPpp) //Maintain Overdrive position
    {     MotionEnable = true;
       Position = (PosOR)/0.1;
       Speed = ActuatingSpeed/5;
       MsgDelay = millis() - PreviousMsgDelay;
       if (MsgDelay >= 100)
       {
          EncodeMotionParameters();
          CAN.sendMsgBuf(CONTROLLER_ID, 1, 8, Data);
          PreviousMsgDelay = millis();
       }
  }
else //Disengage the CVT
     {
        MotionEnable = true;
        Position = Disengage/0.1;
        Speed = ActuatingSpeed/5;
        MsgDelay = millis() - PreviousMsgDelay;
            if (MsgDelay >= 100)
            {
                EncodeMotionParameters();
                CAN.sendMsgBuf(CONTROLLER_ID, 1, 8, Data);
                PreviousMsgDelay = millis();
            }
```

```
        }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///////
```

Once the actuator parameters were set, PID control of the actuator was implemented. The control code is set up where engine RPM is always calculated per cycle through the main loop, then if the condition where engine RPM is seen above the threshold set, then the code will cycle through the PID control code. If engine RPM ever dips below the threshold, the actuator will disengage the CVT.

When the conditions are met and goes through the PID control code, the error is calculated and proportional, derivative, and integral terms are calculated. Once the terms are added, the value is rescaled with a conversion of the underdrive actuator position divided by the maximum RPM output from the engine then constrained to the predetermined limits of actuation. With a position value determined the function to encode the CAN message is called and sent. After going through the PID control statement, if the condition where engine RPM is below the threshold is met, all the variables keeping a timestamp or any sort of history (integral and derivative terms) are equated to zero.

The PID control code was tested by having the car on the chassis dyno without a load applied, and adjusting the proportional gain positively from zero till the actuator began to respond according to the engine RPM. It proved successful, but the system possessed an offset from its setpoint value (2500rpm).

eCVT Prototype Design: PID Tuning

With a working control code, black box tuning for PID gains was done on the chassis dyno. The prototype vehicle was strapped down to the chassis dyno and tightened down so that traction would not be lost during testing. PID gains were set to be adjusted using rotary potentiometers, and if they were to be set at zero, the potentiometer would be deactivated to prevent drifting and inaccuracies inherent to the potentiometers. The method of black box tuning was Ziegler-Nichols, where the proportional gain was increased till the system became marginally unstable. At that point, the time was kept for 10 oscillations to determine suitable gains for

the integral and derivative terms. Calculation of gains were tested with each control type in the chart shown below. The goal was to see how the system would react for different methods of control.

Initial tuning goals for normal loads were as follows:

- 0% Overshoot
- Rise Times of <0.25sec
- Settling Time of <0.25sec
- Steady State Tracking of +/- 50 rpm

These tuning goals were possibly unrealistic and subject to change based on testing results and possible tuning parameters. It was still unknown how stable the system was for control and how large of a disturbance the system could reject without any overshoot and large settling times.

Ziegler–Nichols method[1]			
Control Type	K_p	T_i	T_d
P	$0.5K_u$	-	-
PI	$0.45K_u$	$T_u/1.2$	-
PD	$0.8K_u$	-	$T_u/8$
classic PID[2]	$0.6K_u$	$T_u/2$	$T_u/8$
Pessen Integral Rule[2]	$0.7K_u$	$T_u/2.5$	$3T_u/20$
some overshoot[2]	$0.33K_u$	$T_u/2$	$T_u/3$
no overshoot[2]	$0.2K_u$	$T_u/2$	$T_u/3$

$$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau)d\tau + T_d \frac{de(t)}{dt} \right)$$

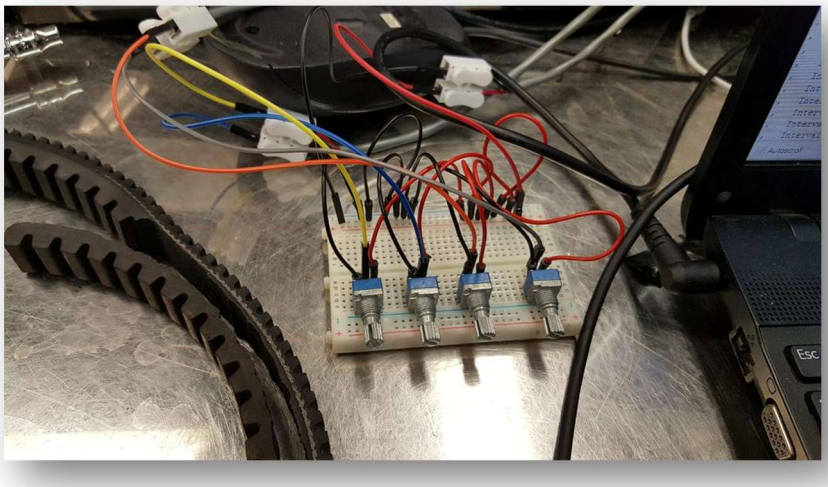

Figure 27: Potentiometer setup used for PID tuning

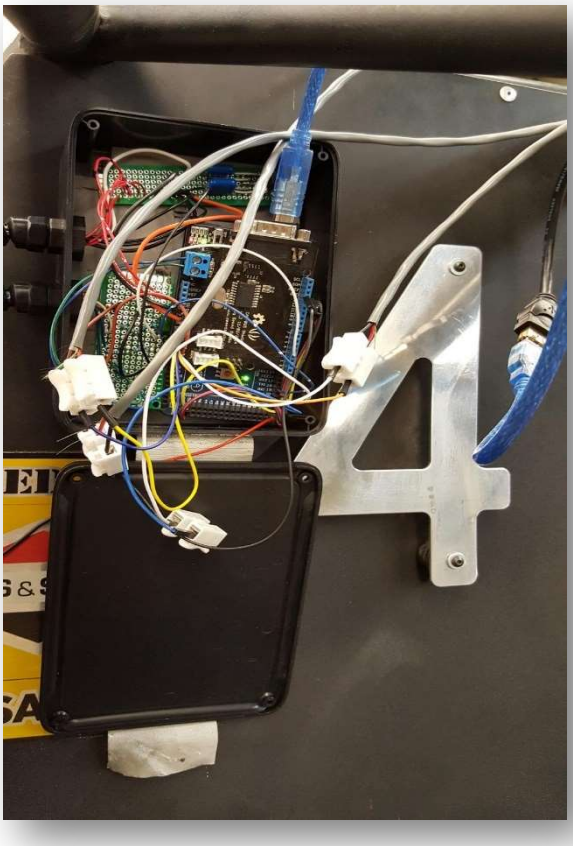

Figure 28: Electronics enclosure of prototype controlled CVT; Wiring runs to potentiometers used for PID tuning

Upon implementing various other control types, the system behaved in an unstable manner. The actuator would stick to the underdrive or overdrive position and then overshoot, either repeating from the opposite end

or stabilize. This was due to the PID code not compensating for integral windup. Another noticeable issue was the RPM value used as a condition to engage the CVT would allow the CVT to engage while the engine idled. The engine's idle RPM would deviate each time after throttling the engine, causing an accidental start.

To stop the accidental engagement, the RPM necessary for engagement was increased, but it then affected the step rejection capability of the system. If the RPM dropped enough, the actuator would disengage instead of adjusting the ratio. It was determined a switch for driver input was necessary as an indicator for driving instead of the engine RPM.

To compensate for integral windup, the PID control code was adjusted to have the integral term equate to zero at the underdrive position or overdrive position; the same was applied to the derivative term.
With those two issues solved, the black box PID tuning process was restarted. The process was as follows:

1. Apply 5A load from chassis dyno (eddie current dyno)
2. Zero integral and derivative gains
3. Scale potentiometer to cover values 0-10. The potentiometer will drift too much for ranges that are greater with a 0.1 resolution.
4. Throttle the car to full throttle capable at the pedal
5. Use separate potentiometer as driver input switch
6. After meeting the PID loop conditions, the actuator will engage the CVT. The proportional gain potentiometer is then increased from zero till the system becomes marginally unstable for about 15 seconds. If the limit of the potentiometer's travel is reached, then range for the potentiometer needs to be rescaled (Ex: 0-10 changed to 10-20)
7. With the system in a marginally stable state, a stopwatch is used to determine the frequency of the oscillations by timing 10 oscillations.
8. Referring to the chart for Ziegler-Nichols tuning method, gains for each control method is done.
9. The calculated PID gains were written into the code instead of using the potentiometers to prevent any drift.

All controllers besides the PI control type resulted in unstable tracking, which could be determined to be that the input signal is still too noisy for derivative control. While the PI controller did not go unstable, it still displayed terrible tracking at steady state. Gains were then manually adjusted to find a range of stability. With a potentiometer controlling each gain setting, various strategies were taken to produce steady state tracking that not display random oscillations.

After several hours, it was determined that the filter was not enough to provide a clean signal for the PI controller to use. Therefore, using the most stable controller gains found, the moving average size was increased from 5 data points to 15 data points incrementally by 2 data points. The moving average size was settled on 15 data points since at that point, the controller became very stable in steady state tracking.

To find a new set of initial controller gains, the Ziegler-Nichols tuning method was done again.

To explore other black box tuning methods, an intuitive approach was taken from a technical document of Cross Company Integrated System's website by Chris Hardy: <https://innovativecontrols.com/blog/basics-tuning-pid-loops>

The tuning process is as follows:

1. Increase the proportional gain value until marginal instability
2. Halve the proportional gain and increase the integral gain until marginal instability
3. Halve the integral gain and this should be the point where to start fine tuning
4. Test the step rejection with a disturbance and watch the input RPM versus the actuator response
5. If there is too much overshoot, either increase proportional or decrease integral gains
6. If the input RPM is 90 degrees or more out of phase from the actuator output, then the integral gain is too large
7. Once controller is tuned, check integral response by zeroing proportional gain. If the RPM takes too long to reach its setpoint, then there is not enough integral.
8. Introduce the derivative gain if a bump in the response is acceptable. Integral and proportional gains will have to be adjusted iteratively with change in derivative.

Again, the PI controller proved to be the best controller type versus the others when using the Ziegler-Nichols method. The method of fine tuning was then used after to ascertain different gains as an optimal tune. Contrary to the technical documents method of introducing a step response with a change in setpoint, the method used was a disturbance change with the chassis dyno. The process for introducing a disturbance was as follows:

1. Turn dyno control knob for a 0A load onto the vehicle
2. Start the car, apply full throttle and hold it to accelerate to top speed
3. Once the RPM starts to flatline at its maximum, the dyno knob is immediately changed to a higher load such as 5A. The highest load setting was selected to ensure an immediate loading at the wheel.
4. The high load is maintained to watch the controller stabilize
5. Once stabilized the, load is reduced to 0A
6. Based on how the controller reacted in rejecting the disturbance and upshifting, the controller gains were adjusted accordingly with regards to Hardy's tuning method.
7. Controller gain settings with notable disturbance rejection were recorded.

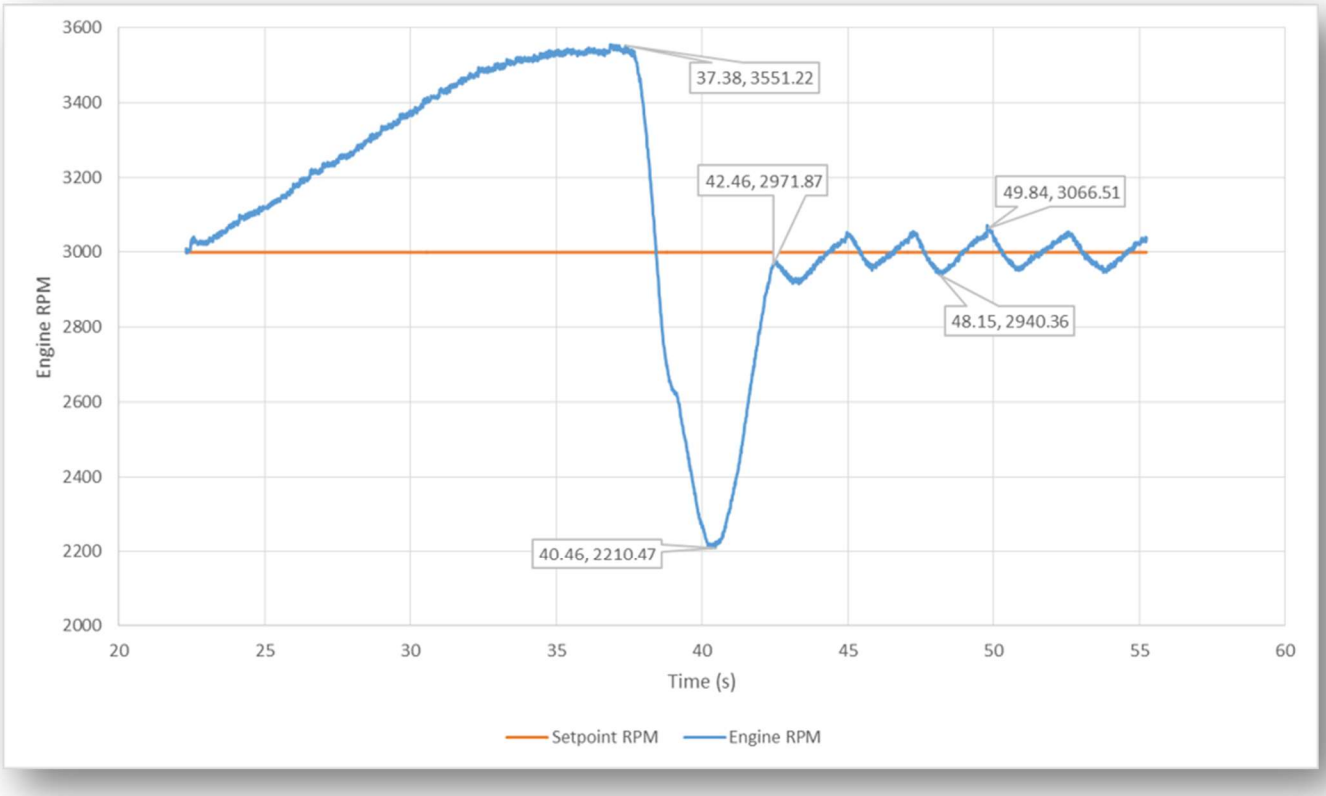

Figure 29: Disturbance rejection with dyno loading 0-10A using controller gains Kp = 1.0, Ki = 1.1, Kd = 0.0 on the 2017 Baja car

Using these methods, it was found that the controllers with the best disturbance rejection characteristics and steady state tracking were integral heavy despite references stating rules of thumb for having a proportional gain 3 times greater than the integral. Most gains found either had an integral gain almost equal or greater than the proportional gain. Through the tuning process, the controller gains found were:

Proportional Gain (Kp)	6.8
Integral Gain (Ki)	6.6
Derivative Gain (Kd)	1.3

Using this controller, testing was conducted.

eCVT Prototype Design: Testing/Analysis

Throughout the tuning process, there were issues encountered. Engine problems resulted in necessitating an engine swap that caused the PID gains to become unstable in steady state where before the engine swap the controller was stable. This was a problem because three different engines of the same model would be used and if an engine needed to be changed at a competition, PID tuning would not be possible at the competition site. It was speculated that the reason the system had to be retuned was that the governor system of the engine is the major contributor to what the actuator had to be tuned for. So, it was possible that if the engine model is changed, retuning was necessary, otherwise there would be no issue.

The second problem encountered was the rate of belt wear and failure. With the mechanically actuated system, belt failure was not as common. Since the quality of the belt was poor, usually having cracks, it was deemed acceptable that they would last one week when tested 4-6 hours a day (28-72 hours).

A third problem was a hall effect failure due to overheating. The hall effect's steel tab mounting location had two bolts going to the engine block that would transfer heat to the sensor via conduction. To remedy the issue, the sensor was replaced with a sensor having some thermal tape applied to the contacting surface with the tab, and the same was done for the bolts and tab to prevent heat transfer. This fix was successful in preventing hall effect failures due to heat.

Before vehicle testing could begin, a throttle switch was implemented in front of the throttle pedal to produce a driver input condition. The sensor selected was a proximity hall effect sensor (Littelfuse, 55100-AP-02-A), allowing for a threshold adjustability.

Initial testing was done at M-Lot of Cal Poly Pomona, where the car could be driven to highlight any possible issues in drivability. Corner entry and exit, acceleration, and hard stops were tested. The vehicle was able maintain power in and out of a corner, but issues were encountered every time the driver let their foot off the throttle. When accelerating from a stop, the vehicle would accelerate, slow down, then continue to accelerate. What would take place was the driver would throttle the car to the maximum the driver could input,

and immediately the actuator would upshift. Engine RPM would rise to the setpoint RPM in a parabolic like curve, and decrease as such since the actuator would continue to upshift.

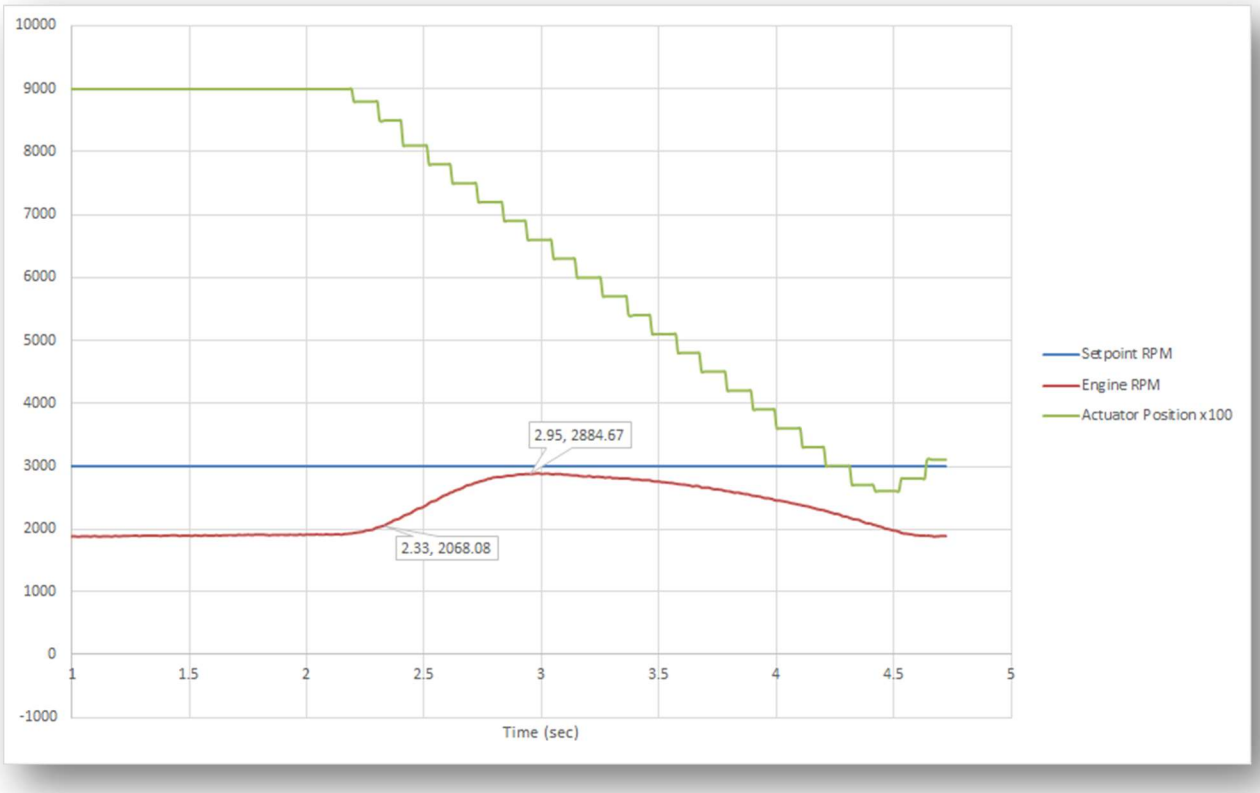

Figure 30: Actuator overshoot problem graphed

Another issue noticed was the recognition of driver input not being seen by the microcontroller when going in and out of a corner. So, the CVT would have a delay in engagement instead of immediately engaging the CVT when the throttle was pressed. This posed an issue for two notable reasons, it was somewhat unpredictable when it would occur and did not allow for proper cornering technique in a racing environment. As shown in the image below, the driver must be able to let off the throttle, brake, then engage the throttle again. Despite the issue, it was still possible to brake while maintaining the throttle.

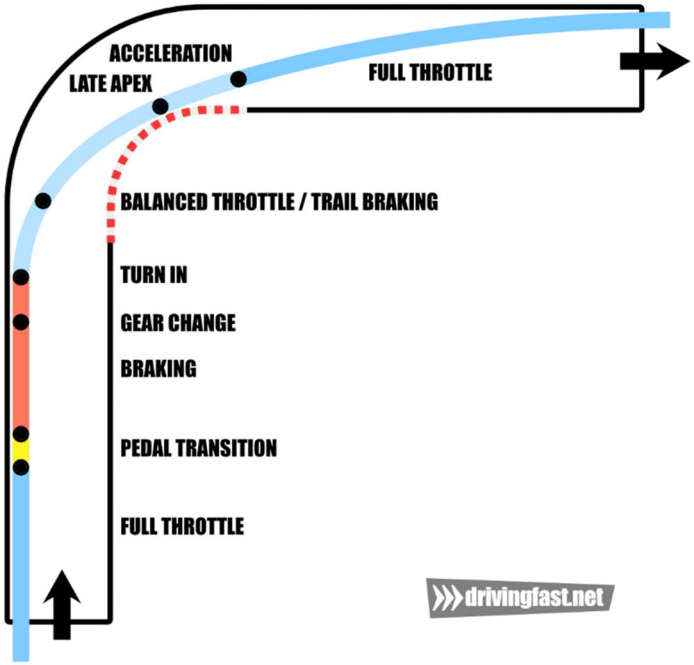

Figure 31: Proper corner entry and exit maneuvering in racing

After testing in M-Lot, the prototype car was then taken to Stoddard Valley, near Barstow to test capability in an off-road setting. Aspects tested was cornering, hill climbs of various inclines and traction, woops, J-turns, hard braking, jumps, starts on hills of various inclines, acceleration on flat ground, up inclines and down inclines.

Notable problems found in the CVT's performance was stalling of the engine when performing J-turns as it would stall the engine. There were also severe oscillations when getting into the overdrive and when applying a large load that would cause the system to get into the underdrive. Despite all these problems, the prototype vehicle could outperform the new 2017 vehicle using the CV-Tech system (mechanical CVT) in a few scenarios. In a side by side acceleration run (~200 ft), the electronic CVT could outperform the mechanical system despite the car running the mechanical system being 100lbs lighter. In a hill climb scenario, the 2017 vehicle was unable to climb hills of similar inclines (~30-45 deg from flat ground) while the prototype vehicle could climb those same hills with either dirt or gravel. The prototype vehicle could keep pace with the lighter 2017 vehicle in figure 8 tracks and various other terrain.

Upon returning from testing, the vehicle started to stop reacting properly. The result was increased noise in the RPM reading. To diagnose the root cause, the Arduino was replaced, sensor output was checked, power input to the sensor, and all electrical connectors. The sensor was tested for overheating failure but it responded properly, but had an increasingly noisy signal with increasing engine RPM. It was then decided to treat the noise with a filter going to the input pin (digital pin 3, Int 1) of the Arduino mega.

eCVT Prototype Redesign

To solve the noise problem various filter types were implemented. The first filter implemented was a passive RC filter with the cutoff frequency of ~120Hz, where the maximum RPM of the engine is 3800 rpm equating to 63.33 Hz. The Arduino was unable to attain a reading due to the voltage drop and change of the waveform. To ensure this, a first order and then a second order active filter (Multiple feedback Active Filter and Sallen-Key Active Filter) were attempted without much success. The higher order filters were used to improve noise rejection, as it was thought that the noise was of a lower frequency but the voltage drop was too great before getting to the amplifier. Another attempt at signal conditioning was passing the hall effect output signal to the amplifier without an added filter causing a voltage drop, which ended in attaining a usable signal. With a change in the signal quality, the moving average was increased in size to produce a clean signal usable by the PID controller. Using the same method as before to increase the moving average size:

1. Start moving average from 5 data points
2. Ensure the V-Belt is removed and have the Mychron data logger reading the engine spark
3. At idle, increase the moving average size in increments of 5 data points until visually smooth (deviations of +/- 10 rpm) on the serial plotter at a 115200 baudrate.
4. Compare readings with Mychron RPM reading
5. If similar, move on to compare full throttle RPM. Otherwise, increase moving average size.
6. If full throttle RPM deviates by a similar amount as idle, then increase proportional gain to test stability using the feedback data. Otherwise, increase moving average size.
7. If actuator does not maintain the same position, then the moving average needs to be increased in size.

When checking the phase lag, it was not as clear on the serial monitor due to the noisy input signal, but the filtered signal moved closely in relation to the unfiltered signal.

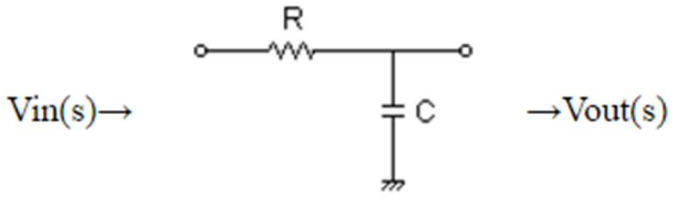

Figure 32: RC Low Pass Filter circuit

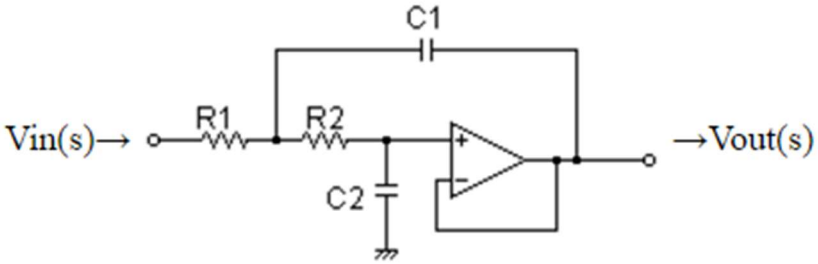

Figure 33: Sallen-Key Low Pass Filter circuit

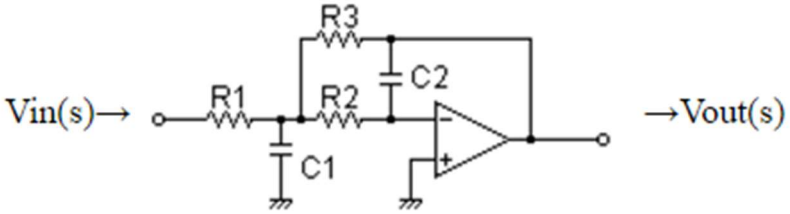

Figure 34: Multiple Feedback Low Pass Filter

Figure 35: Oscilloscope reading of engine hall effect with passive RC low pass filter

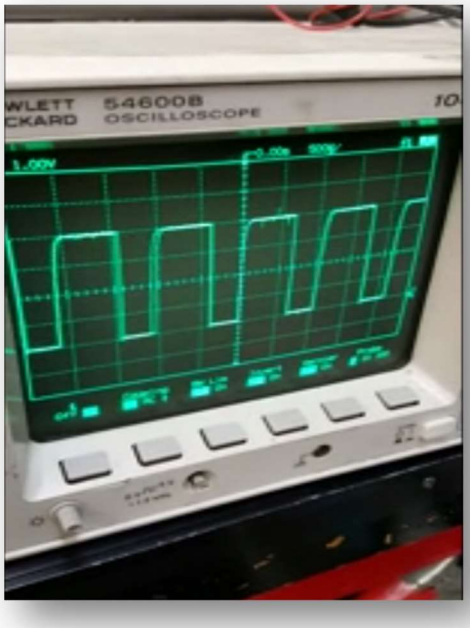

Figure 36: Oscilloscope reading of engine hall effect

To accommodate improved sensor packaging in the area between the inner sheave and case for the final CVT design, the hall effect sensor was changed from a plastic flange mount sensor to a steel threaded sensor (Littelfuse 55075). Similar precautions were taken for the new hall effect to prevent failure from overheating. Since the 2017 vehicle required to be retrofitted with all electrical components, the controller issues were solved using the 2017 vehicle upon completion of hardware installation.

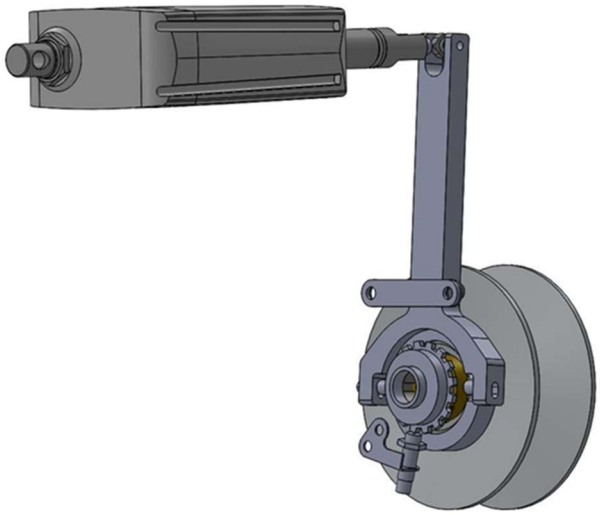

Figure 37: Driving pulley assembly with actuator and 55075 Hall Effect

Final CVT Design

Component Design: Overview

When the final iteration of the CVT was being designed, a few factors were considered for some of the overall specs of the system. With a reduction in vehicle weight, the decision was made to change the v-belt and sheave dimensions to allow for an increase in ratio range. With the weight reduction of the vehicle, it is possible to take advantage of the higher gear ratios that could not be used before due to lack of torque at the higher RPM.

	Previous CVT Specs	Current CVT Specs
Min Ratio	4.0:1	3.0:1
Max Ratio	0.77:1	0.43:1
Ratio Range	5.19	6.97
Center to Center (in.)	10 in.	8.5 in.

Figure 38: Gaged V-Belt (left) and CV-Tech V-Belt (right)

Component Design: Driving Pulley Design

Overall system integration was crucial as the initial prototype increased belt removal time to a point that was considered unacceptable. The initial prototype requiring an additional two tools and 3 pieces of hardware to be removed to allow user access to the v-belt. Through Fault Tree Analysis and Failure Mode Effects Analysis it was determined that ease of access to the belt was crucial as it was a component that was most likely to fail.

To allow for quicker belt removal times the driving pulley was designed as two separate sub-assemblies. The fixed sheave and shaft are separable from the rest of the driving pulley. This allows for the V-belt to be accessible by the removal of just one bolt, where as in previous vehicles that was not the case. The removal of the engine bolt allows for the fixed sheave and shaft of the driving pulley to be removed from the engine. At this point it is possible to remove the belt and driven pulley for maintenance. The moving sheave has a bearing surface that in combination with a spacer keeps the bearing locked to the sheave. To keep both the fixed and moving sheave of the primary spinning in unison, for proper power transfer through the belt, a square bronze bushing on the moving sheave mates with the square portion of the shaft. The square bronze bushing, as well as another circular bronze bushing, were used at the points of contact between the fixed and moving sheave. This allowed for the points of contact to have reduced friction to keep actuation force low. The clutch fork uses a much smaller and lighter mount in comparison to the initial prototype and places the actuator further above the engine. The actuator mount placement was changed from the initial prototype to keep the heat from the engine further away from the actuator. As the engine does move in relation to the chassis to allow for belt tension to be adjusted, the mounts for both ends of the actuator had to allow for the deflection. A rod end was used on the clutch fork side of the actuator to account for the deflection, spacers were implemented to prevent the clutch fork from coming into contact with the rod end throughout the range of motion of the actuation. On the chassis side two large rubber washers were used in between the actuators mounting location and the tabs made on the chassis, this made it possible for a bolt to be properly tightened while having the washers deform as the actuator became misaligned.

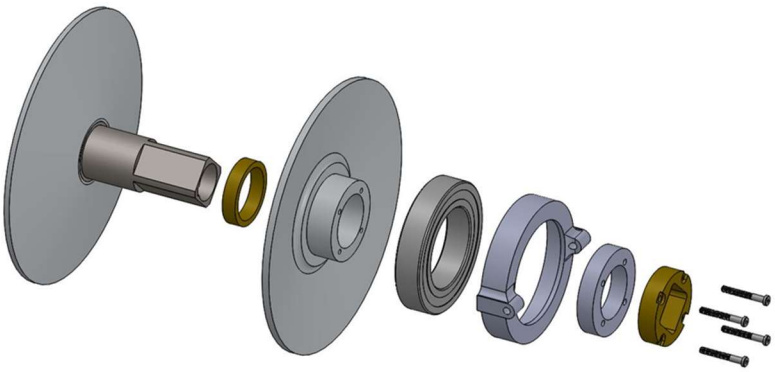

Figure 39: Driving pulley exploded view

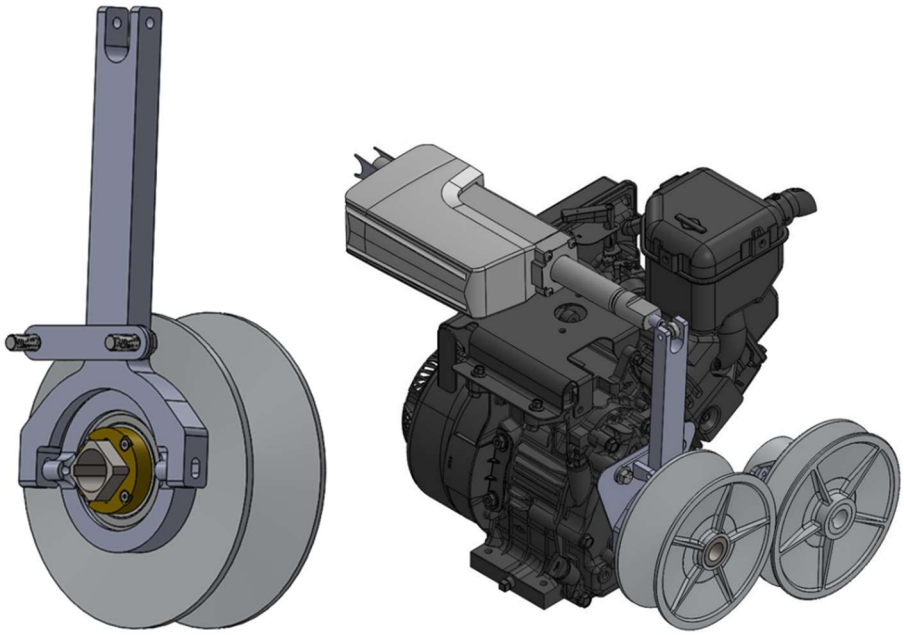

Figure 40: Driving pulley assembly (left), assembly of final CVT design without the case (right)

Component Design: Driven Pulley Design

Unlike the driving pulley, the driven pulley stayed very similar in design to a typical driven pulley in commercially available CVTs. The main goals for the driven pulley were reducing the shifting friction and weight reduction. The use of track rollers and brass bushings were utilized to offer the least amount of

resistance possible through the shifting of the secondary. An aluminum shaft with a steel sleeve was used as opposed to an all steel shaft to reduce weight out of the system. As the multiple iterations of the CVT did require many manufactured components throughout the year, manufacturability was always a priority in the design of components. As the team had access to a three axis CNC machine on campus, all components were designed with the machining capabilities of the team in mind. The ramps of the helix for the secondary were designed in a way that allowed for three axis CNC work where some helix designs of commercially available CVT's require a fourth axis machine. All components aluminum components were machined out of 6061-T6 aluminum.

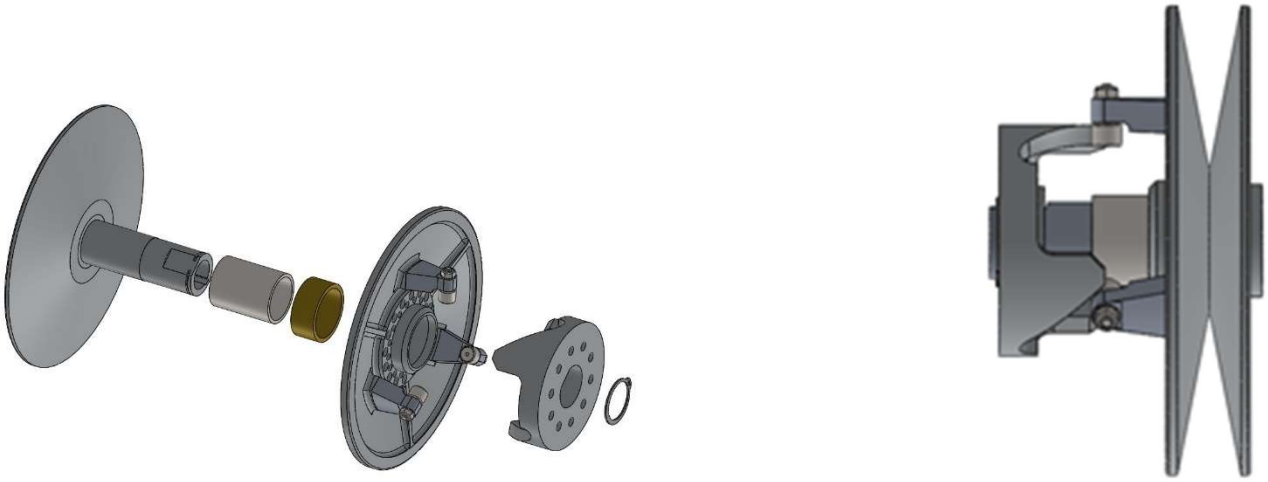

Figure 41: Secondary without torsion spring, exploded view (left), assembly (right)

Component Design: Case Design

The Baja SAE rulebook section B15.1 states that: "All rotating parts such as chains, sprockets, primary CVT pulleys, and belts that rotate at the rate of the drive axle(s) or faster, must be shielded to prevent injury to the driver or bystanders should the component fly apart due to centrifugal force." These design considerations are what led to the major change on the driving pulley of flipping the moving and fixed sheaves so that the clutch fork and pressure plate assembly could be fit between the engine and the driving pulley. Section B15.1 further explains requirements for holes into the CVT case: "Holes and/or vents in the portion of the powertrain

guard surrounding the rotating components are acceptable provided that in the event of a powertrain failure, no parts can escape. No direct path shall exist tangent to any rotating components." To accommodate for this rule a sliding guard was incorporated to ensure that the case remain sealed through the entire actuation stroke of the clutch fork.

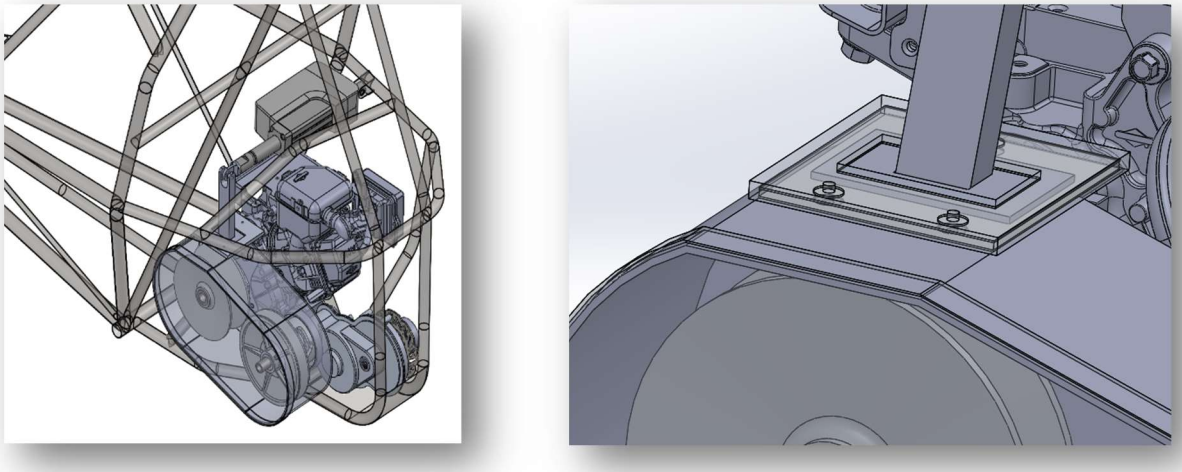

Figure 42: Complete assembly of powertrain system in chassis (left), clos-up view of sliding shield on CVT case (right)

Center to center distance was also reduced from 10 inches to 8.5 inches. As a complete overhaul was done on the chassis and suspension components a large reduction in the drivetrain bay was made. The reduction in center to center does affect the angle of wrap of the CVT in a negative manner but the improvement to overall vehicle dynamics justified the change.

Figure 43: Heat treating of CVT case material before bending to prevent cracking due to cold working

Mechanical Components Analysis

Unlike components from the Structures or Kinematics teams, analysis of impact loads was not considered for the CVT. Due to the location of the CVT it was considered unlikely that a foreign object would inflict damage to it without first coming into contact with either the rear suspension or the chassis. Also, high wheel loadings such jump landings at full throttle are not of concern due to the inherent characteristics of the CVT. The V-Belt is a flexible power transfer element that relies on the friction between the V-Belt and sheave surfaces to transfer torque. If the torque the sheave tries to transfer is too great, slipping between the V-Belt and sheaves would occur. With high torque transfer from the wheels, the gearbox would fail before the V-Belt.

A large concern in the driving pulley design is the effects of loose fitments and deflections to the relation between the CVT ratio and the actuator positioning. Another important consideration with sheave deflections is that proper V-Belt and sheave contact could be reduced. Deflection in the clutch fork became a concern after seeing the first clutch fork design of the prototype CVT begin to bow to one side and finally break due to fatigue. To analyze each individual sheave and clutch fork, a single loading scenario of the actuator applying its maximum force was considered. In the loading scenario, the V-Belt has reached half of its travel from underdrive to overdrive. Based on FEA, ribs were added to the sheaves resulting in decreased deflections.

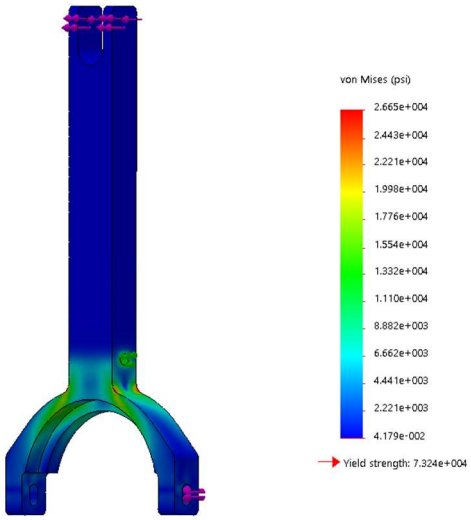

Figure 44: FEA of final lever arm design

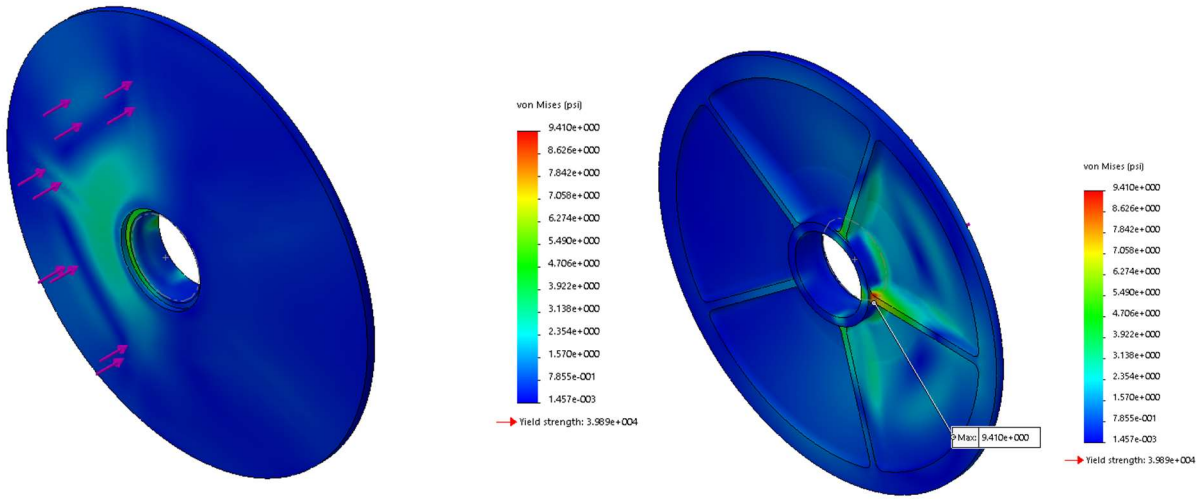

Figure 45: FEA of driving and driven sheaves

CVT Cooling

A common problem when using a sealed case for the CVT is overheating of the belt, resulting in poor power transmission. The increased temperatures start to vulcanize the contact area between the belt and sheave, hardening the belt and changing its frictional characteristics. An acceptable high temperature is 140 degrees Fahrenheit. Without any cooling, the temperature rose 100 degrees Fahrenheit above the maximum while testing various scenarios for powertrain and suspension. Severely reducing performance and reducing the usable life of the belt. To remedy the issue, a cooling fan was installed to pull ambient air away from the engine bay and push air out the CVT case, toward the side with the transaxle. In a separate test for powertrain and suspension, ambient temperature rise within the CVT was measured. The peak temperature measured was 160 degrees Fahrenheit, which is still 20 degrees above the recommended maximum but is still a large improvement.

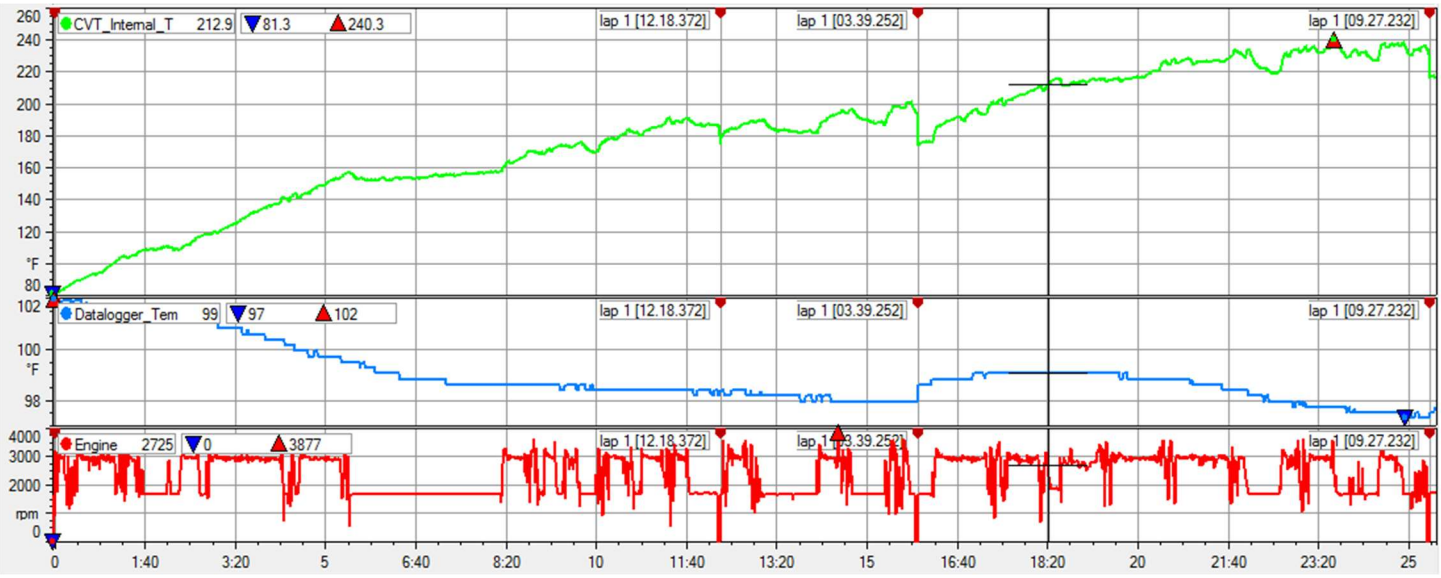

Figure 46: With no fan cooling; Internal ambient CVT temperature data, data logger temperature, and engine rpm

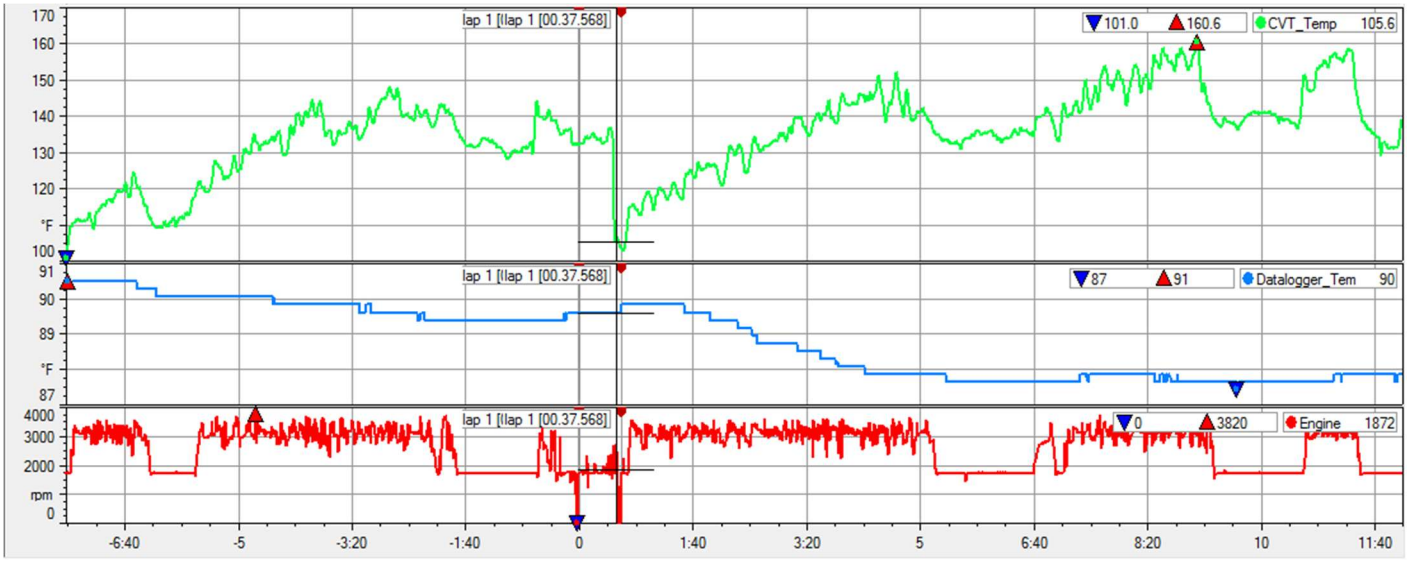

Figure 47: Fan cooling implemented; Internal ambient CVT temperature data, data logger temperature, and engine rpm

During the 2017 California competition, the battery was drained affecting power supply to the actuator. It was later found to be the fan pulling too much current at idle RPM. Another issue that appeared again was the failure of the hall effect due to the high temperatures inside the CVT case.

With the hall effect failing from insufficient cooling, the RPM pickup location was moved outside the CVT case and to the engine's flywheel.

To prevent power consumption problems by always running the fan, it was wired to turn off at idle. The Arduino was set up with a relay to activate the fan at an appropriate engine RPM suitable for providing power to the fan and actuator. Current was measured during tests after the California competition, and there was always enough for all devices.

At the next competition in Kansas, a problem with battery voltage running low was discovered. The source of the problem was the melting of the fan housing and stalling of the fan motor.

Figure 48: Cooling fan deformed due to high temperature

In the end, the problems could not be solved with the limited time until the next competition (2017 Illinois), so the fan was removed and the inlet sealed. During the endurance race temperatures rose so high, a bearing removed itself from its press fit and pushed the sheave onto the belt where the V-belt was destroyed.

Electronics Enclosure Design

All major electronics components were to be in one central location. Since accessibility to batteries, the controller, and fuses were important to do maintenance and charge batteries the enclosure was made to open. Restrictions made in the Baja SAE rule book for electronics state's "**B3.2.1** Batteries must be mounted with sound engineering practice and not come loose during a roll over." All mounts and enclosures were riveted to the aluminum case, with the front panel also riveted in place. The Shorai battery was zip tied to its mount, but later required a hose clamp to pass rules at the 2017 California competition.

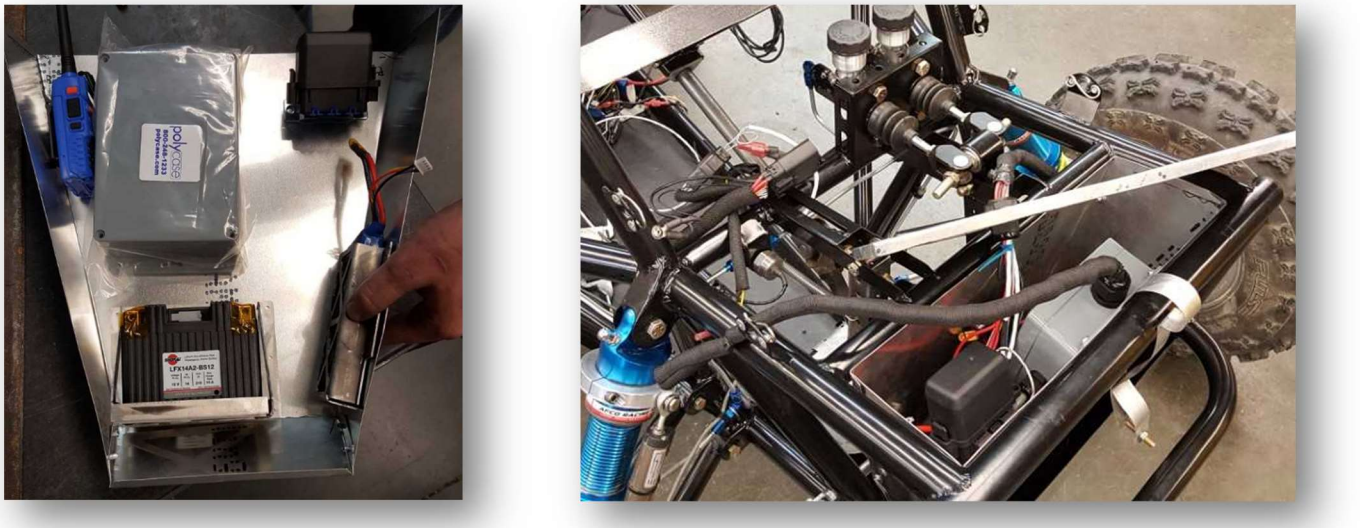

Figure 49: Electronics enclosure layout (left), electronics enclosure installed in nose of chassis (right)

Testing of Engine

An important aspect of this project is understanding the value to be tracked by the PID controller. Ideally the setpoint would be the peak horsepower of the engine, but the characteristics of the horsepower curve could change this. Since every engine is subject to manufacturing intolerances, the horsepower and torque characteristics vary from engine to engine. To determine the characteristics of each engine used in the three competitions of 2017, a dynamometer system had to be constructed as part of this project. With the data collected, an RPM could be selected to improve power output in all instances of driving.

Initial System Selection and Design Considerations

Since the fundamental idea behind a dynamometer is a simulated load applied to the engine in order to attain torque and power curve data, the first consideration would have to be how to apply that load. There are several ways to do this, some of which can get quite pricey. It was determined that the system should be within a reasonable budget yet sufficient to provide accurate data, have a user interface that's easy to work with, and be something that future teams can utilize. The choices were narrowed down to the Go Power D100 Small Engine Dyno, the Dyno-mite Small Engine Dyno, and the Hewitt Inertia Dyno. After researching company feedback and customer experience, users obtained the most accurate data with Dyno-mite products. From a financial standpoint, it provides a good balance of quality components and cost, offering self calibrating features that are not available on the DH-100. The price range also allows for the ability to spend a portion of the budget on a closed loop water cooling setup that is necessary for water brake dynamometers. The Dyno-mite system made use of a water brake for applying load to the engine's output shaft and an easy-to-use data acquisition system. Water brake size sufficient to load engine through all gear ratios of the CVT. Power is calculated directly from brake stator RPM and torque is obtained from strain gages on an arm mounted to the brake stator housing. Design considerations for the water system included pump size, reservoir size for adequate pump head, and tubing diameter. For mobility, a cart design was implemented.

Figure 50: Dyno system components

Dyno System Set-Up

The Dynomite Small Engine Dyno system is in no way limited to handling the output power of the Briggs and Stratton engines used in Baja SAE collegiate series competitions. The control panel therefore accepts input data for a wide range of testing parameters. For this project's purpose, a majority of these parameters were deemed unnecessary with the exception of engine, weather, and dyno specifics. In addition, several of the gauges provided on the Dyno-mite system console were of no use in the primary objective of retrieving the power and torque curves needed to program the system necessary to control the CVT. Air Fuel Ratio (AFR), Oil Pressure, Air Flow, Brake Specific Fuel Consumption (BSFC), Engine Temp, and Exhaust Gas Temperature all have little or no bearing on the desired data, so these gauges were turned off during testing. With the control panel and console set up correctly, it was then time to physically set up the dynamometer for testing. The engine was securely bolted to the Dyno cart and the Dyno-mite water brake was mounted to the engine output shaft via key set and retaining bolt. The reservoir was then filled with 30 gallons of water.

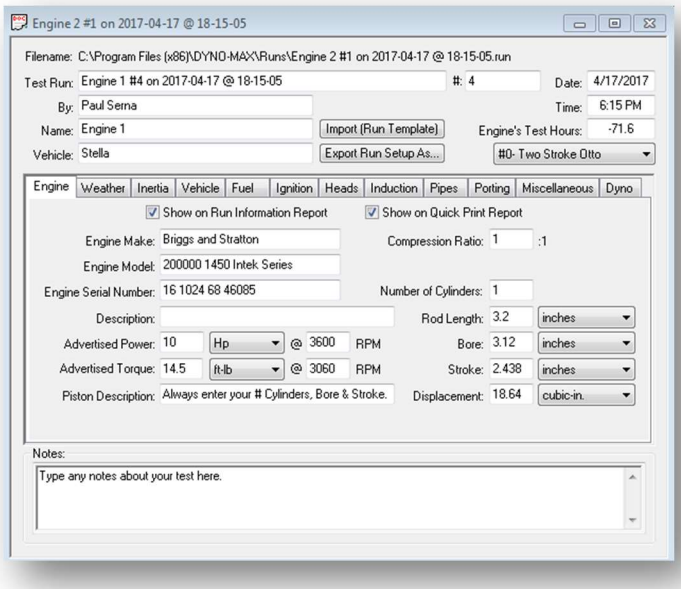

Figure 51: Settings for engine test

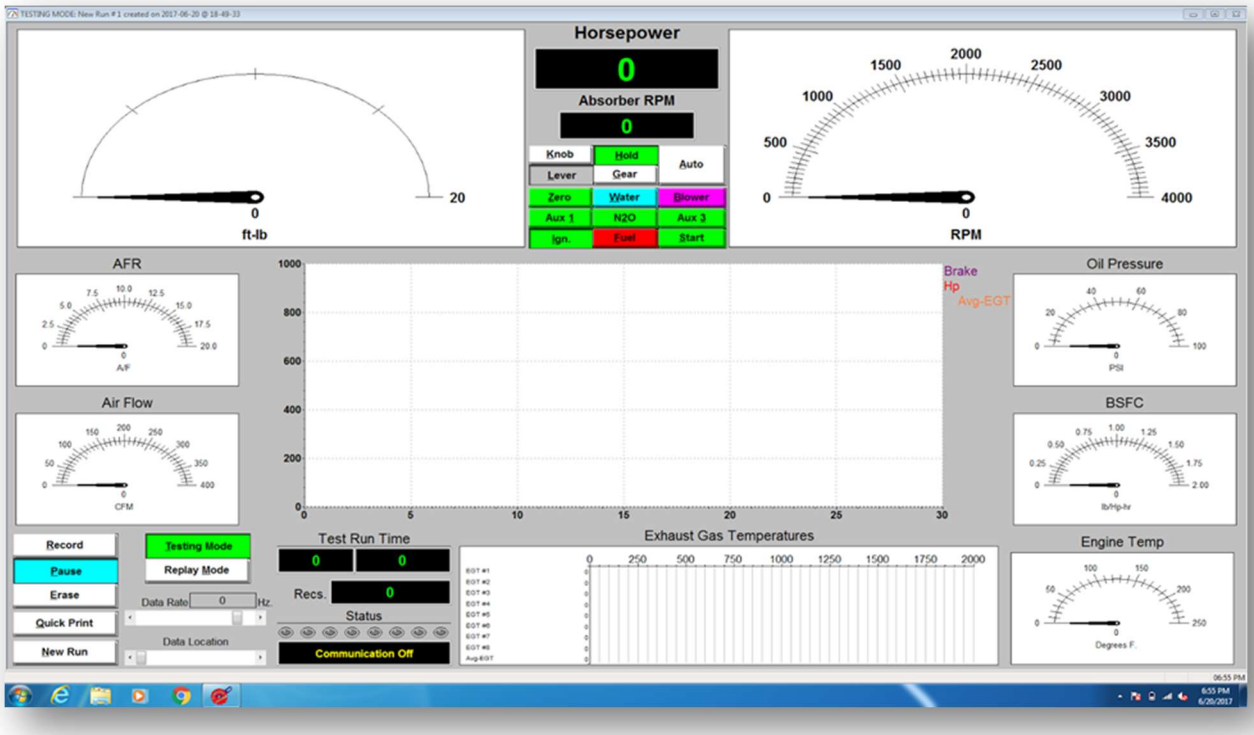

Figure 52: Dyno readout window

Finally, the data acquisition box was plugged into the computer containing the control console. Testing runs are then performed using the following principles:

1. With engine at maximum RPM, water is pushed through the system via pump
2. Load is adjusted using a load valve
3. Water is pushed in a spiralling motion through the circumference of the stator creating an opposing torque on the engine's output shaft
4. Water then exits the water brake and is dumped back into the reservoir for recirculation

Engine Testing Method

Since the program developed would have to precisely track the RPM of the engine, it was postulated that the Briggs and Stratton published values for horsepower and torque may not be accurate. To determine the validity of these values initial testing loads would be applied directly to the engine's output shaft as opposed to through the RPM-amplifying gear ratios of the CVT. In addition, testing results would have to provide the full curve with respect to power and torque. Therefore, the employed testing method, conducted at wide open

throttle, took the engine from an unloaded state through a steady progression of increasing load until a load maximum stalled it out. This method provided the data needed to create a smooth curve over the entire load capacity of the engine, including an accurate load ceiling that can then be implemented into the CVT code.

Figure 53: Dyno cart

Final Results and Analysis

After five successive runs conducted on each of the two engines tested for competition, the expected torque and horsepower of the engines per manufacturer specs were much lower than the actual values. The manufacturer's data states the engine should have a maximum of 9hp at 3600 rpm and 14 ft-lb at 2800 rpm, exhibiting a flat torque curve across the engine RPM range. Results from testing Engine #1 and Engine #2 exhibited similar torque and power characteristics but quickly dropping off after ~3600 rpm, where it is not displayed on the manufacturer curves. The maximums derived from testing each engine resulted with 10.5hp at 3210rpm for Engine #1 and 11.2hp at 3283rpm for Engine #2 after averaging peak values of each curve. Since one data set of Engine #2 was not consistent with the other data sets, the peak horsepower RPM was determined

to be 3344rpm with a peak horsepower of 11.4hp. Peak torque was found to be 18.5ft-lbs at 2382rpm for Engine #1 and 19.1 ft-lbs at 2310rpm for Engine #2.

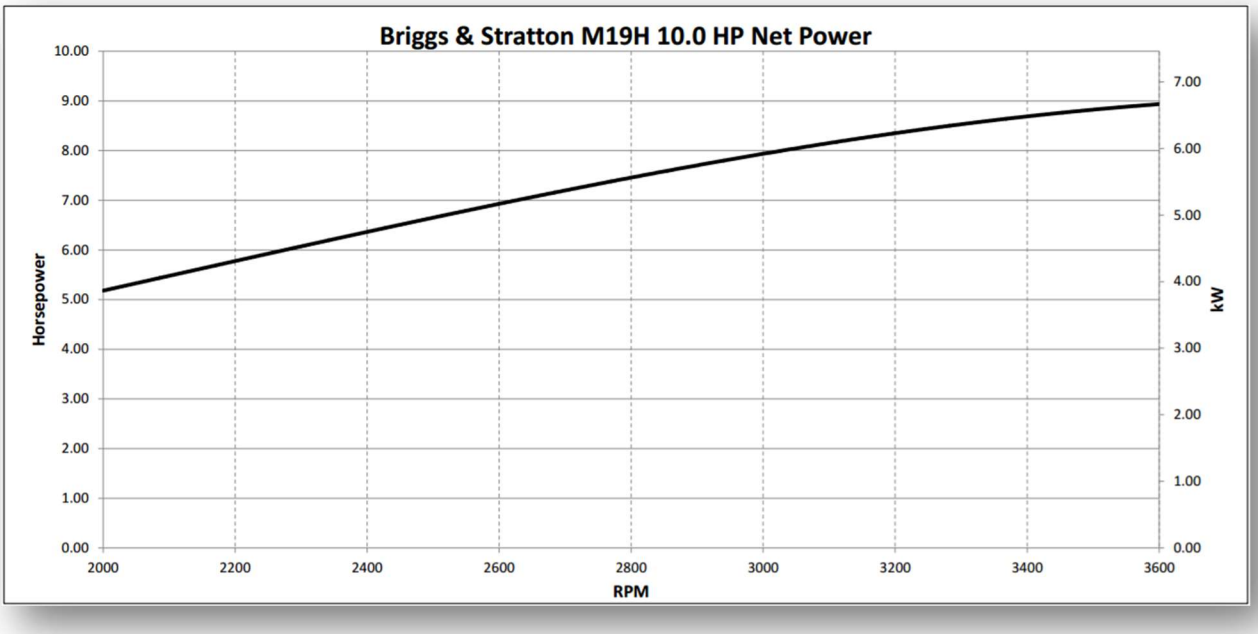

Figure 54: Manufacturer horsepower curve of engine specified by SAE rulebook

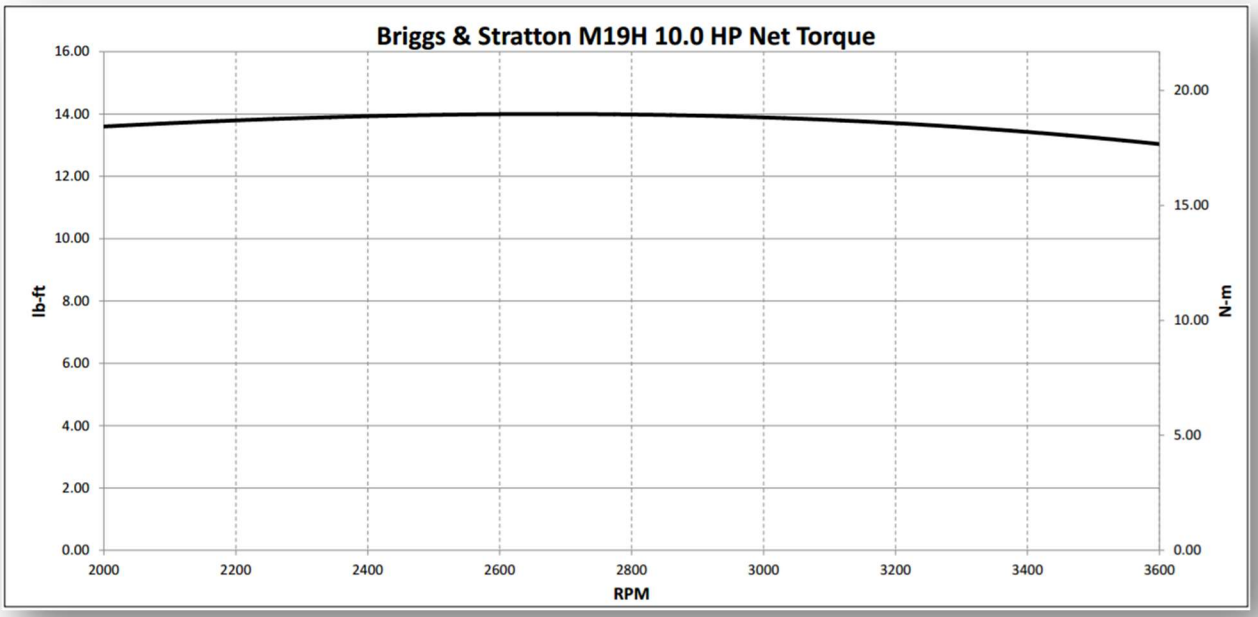

Figure 55: Manufacturer torque curve of engine specified by SAE rulebook

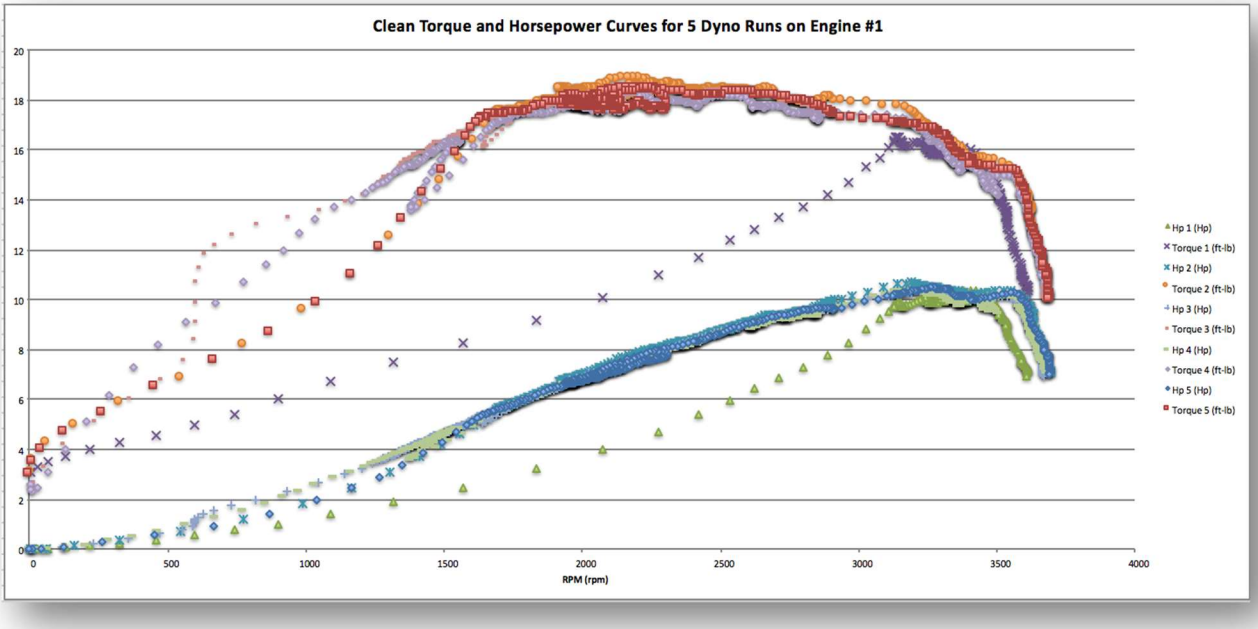

Figure 56: Dyno test results from Engine #1

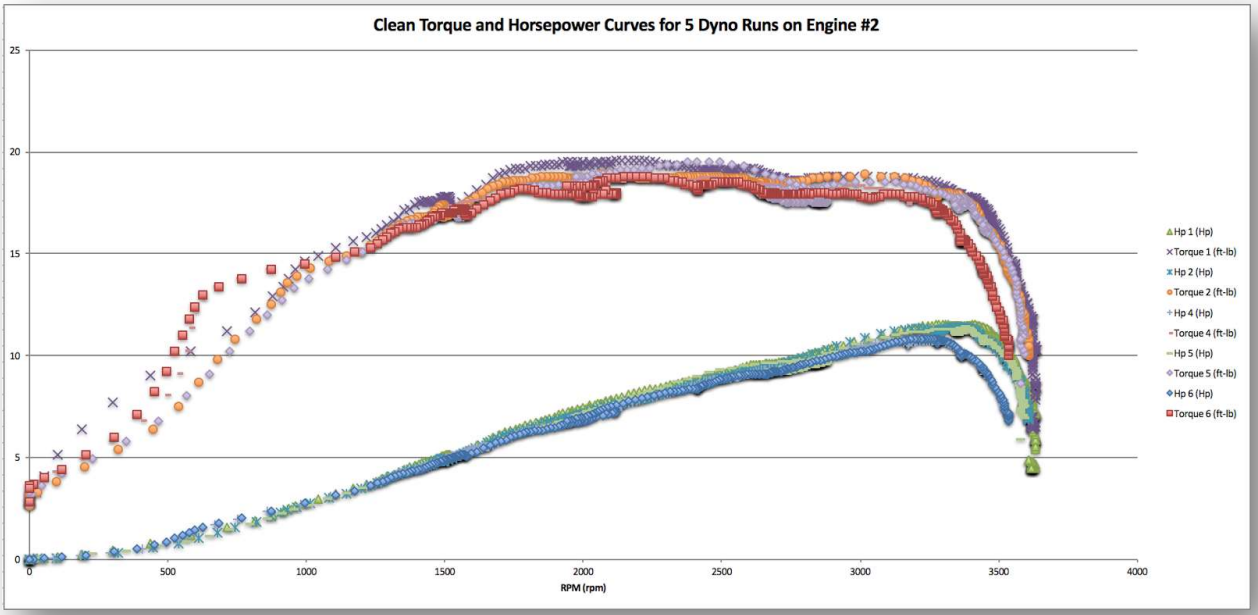

Figure 57: Dyno test results from Engine #2

Controller Software Final Design

Changes made to fix the issues found in the prototype were made, and resulted in a stable controller for driving at all times. Comparative analysis was done using the controller before properly accounting for acceleration from a stand still.

Each problem had to be accounted for separately. The inconsistent engagement of the prototype vehicle was found to be an issue with varying readings from the hall effect proximity sensor. Any change in orientation or alignment of the sensor from the magnet produced different readings. It was possible that the surrounding ferrous material in the pedal are caused some interference to the sensors readings. To solve the problem, a button indicating throttle input at the pedal was used. As a safety consideration, a spring was used to induce an opposing force for depressing the button when not applying force to the throttle pedal.

When examining the overshooting problem, it became apparent that the reason it occurred was because of the underdrive boundary condition, dictating the iTerm not to update. So, when just starting from a stop the variables would have a value of zero, only allowing the proportional term to control the actuator. To solve the actuator overshooting problem, the initialized integral term value had to be declared with a value other than 0. To ensure effects from not initializing the PID timestamp at the start of the PID calculation were not seen, the PID timestamp (PPpreviousMillis in Rev 9) had to be declared when starting the PID controller loop.

$$\text{Proportional (P) part : } u_{\mathrm{P}}(t) = k_{\mathrm{c}}(y_{\mathrm{s}}(t) - y(t))$$

$$\text{Integral (I) part : } u_{\mathrm{I}}(t) = \frac{k_{\mathrm{c}}}{\tau_{\mathrm{i}}} \int_0^t (y_{\mathrm{s}}(\tau) - y(\tau))\, \mathrm{d}\tau$$

$$\text{Derivative (D) part : } u_{\mathrm{D}}(t) = k_{\mathrm{c}} \tau_{\mathrm{d}} \frac{\mathrm{d}(y_{\mathrm{s}}(t) - y(t))}{\mathrm{d}t}$$

For example, if the boundary condition at underdrive is considered with the engine at 2000rpm: PPintegErr = 0, since the actuator is at the underdrive position, the variable has not been updated from what it was declared

iTerm = 0, since the actuator is at the underdrive position, the variable has not been updated from what it was declared

pTerm = ppKp * error = 0.8*(3210-2000) = 968

dTerm = 0, since ppKi = 0

Output = 968

Output = Output * (newMax / PPmaxOut) = 968 *(82/3800) = 20.9

Position Command = PosOR = 27mm

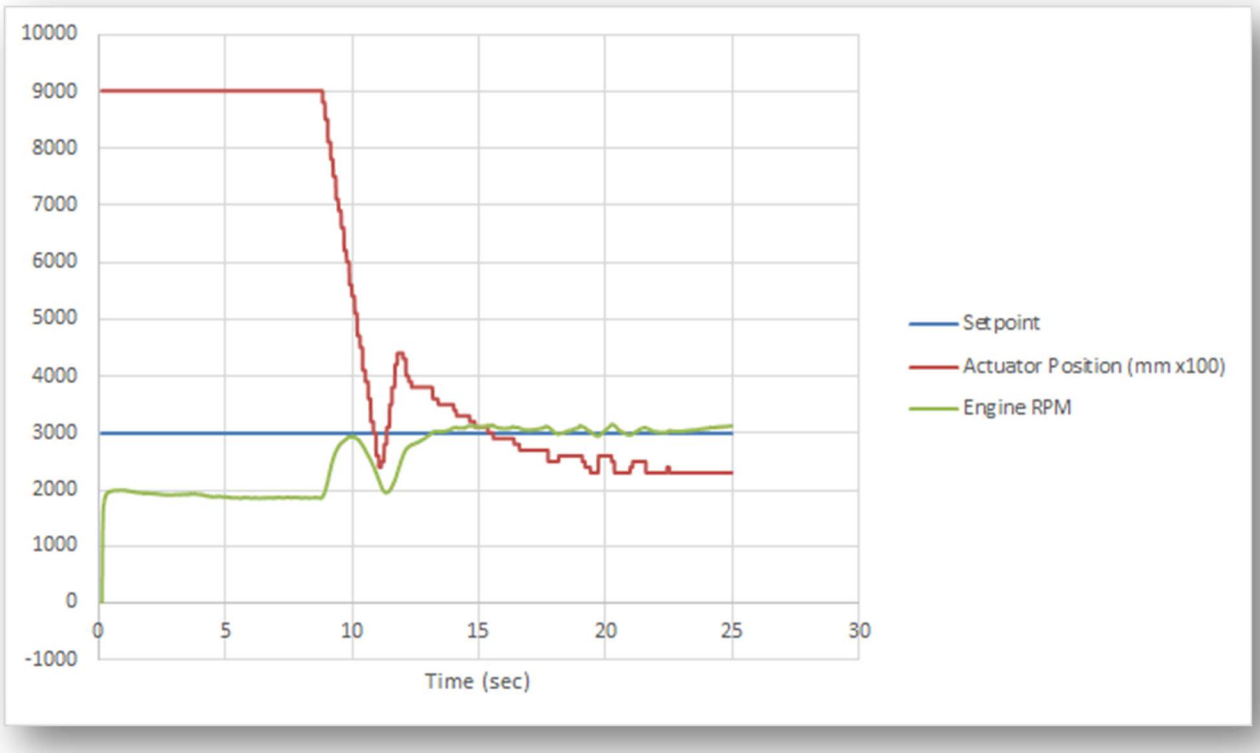

Figure 58: Depiction of actuator position overshooting problem using Rev 9 and Kp=1.0, Ki = 1.1, Kd = 0.0

For the solution to the severe oscillations getting into the overdrive or underdrive, the conditions had to be handled differently when considering integral anti-windup. The issues were the conditions used for neglecting the integral term and allowing the other terms to affect the system when the integral term is zero. Before the fix, the controller took the actuator feedback information and compared it to what the underdrive and overdrive position was set as. If the underdrive or overdrive position was reached then the integral term was not

added to the output value for the PID (neither was the derivative since they used the same conditions). The PID controller would then calculate a new output based on the proportional gain. To ensure that the controller would respond appropriately, the condition for neglecting the integral term was changed. If the RPM was above the setpoint and in the overdrive position, then the integral term would not be calculated (nor the derivative). If the rpm dipped below the setpoint and the actuator position was at the underdrive, then the integral term was not calculated (and derivative). These changes were not enough since the oscillations did decrease but were still present. The last change made was to make the output maintain the underdrive or overdrive position if the integral term was being neglected, which would ignore the input from the proportional term.

Code before the fix:

```
if (PositionCheck < (PosUR-1) && PositionCheck > (PosOR+1))
      {
        PPintegErr = (error * dt) + PPintegErr; // Take the numerical integral
        iTerm = (ppKi * PPintegErr);// Compute Integral term
      }
      else
      {
        iTerm = 0;
      }
```

Code after the fix:

```
if (Erpm > SPpp && Output <= (PosOR))
          {
           iTerm = iTerm; //Do not update integral term if the actuator is at the limits of its position
           PPintegErr = PPintegErr;
          }

          else if (Erpm < (SPpp) && Output >= (PosUR))
          {
           iTerm = iTerm; //Do not update integral term if the actuator is at the limits of its position
           PPintegErr = PPintegErr;
          }
          else
          {

           PPintegErr = (error * dt);
           iTerm = (accelKi * PPintegErr) + iTerm; //sum constant with error in order to prevent bumpless transfer
from one PID to another PID
          }
// ============================ Compute PID control command, u(t):
============================
          if (Erpm > Setpoint && Output <= (PosOR))
```

```
{
  Output = PosOR;
}

else if (Erpm < (Setpoint) && Output >= (PosUR))   // implement after acceleration
{
  Output = PosUR;
}

else
{
  Output = pTerm + iTerm + dTerm;     // Compute u(t) to drive the plant

  Output = Output * (newMax / PPmaxOut); //Scale Output

          if (Output > PosUR)
          {
            Output = PosUR;
          }

          else if (Output < PosOR)
          {
            Output = PosOR;
          }

}
```

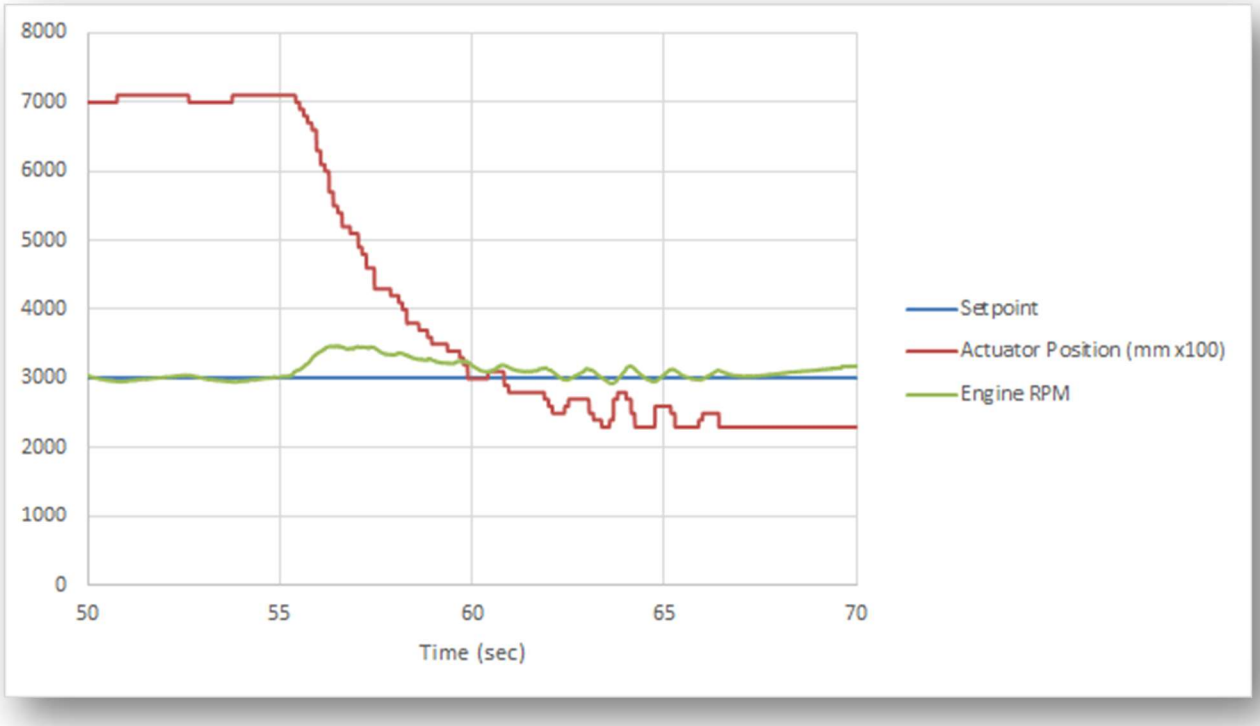

Figure 59: Depiction of oscillations entering overdrive problem using Rev 9 and Kp=1.0, Ki = 1.1, Kd = 0.0

Notable characteristics with these implemented changes was that upon starting the car from a stop, the underdrive position was maintained until the RPM increased enough to get out of the “hold underdrive” condition. The initial solution to the problem was to ensure the actuator would keep the CVT disengaged until PID started to command the actuator to move past the underdrive position. Resulting from this change was a problem that would cause the vehicle to engage at a high RPM; acceleration times suffered due to this problem (4.6 sec, which equated to slower than 75% of any number of vehicles at a competition).

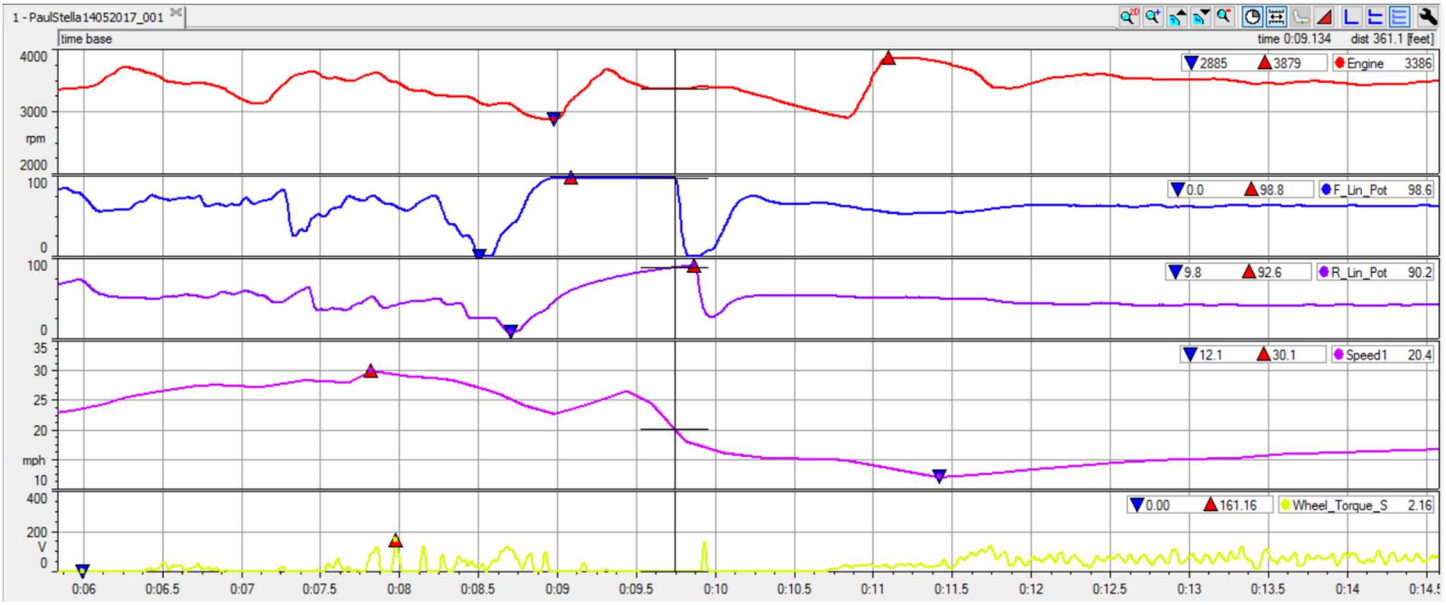

Figure 60: Cal Poly Test Track - Jump without maintaining throttle at landing, then acceleration after landing

To improve acceleration from a stop, there is an optimal engine RPM for a given torque to start power transmission. The controller software was further developed to cure this problem, but it was not thoroughly analyzed. Therefore, all tests conducted used revision 24, whereas the improved code is revision 27.

Another change made was the addition of a cooling fan to reduce temperatures of the CVT case. Power consumption problems at idle RPM during the 2017 California competition prompted the use of a controlled switch to turn the fan on once sufficient power could be generated at higher RPM.

PID Tuning and System Characterization

Initial method of PID tuning was done in the same way the tuning of the prototype was conducted. Once the range of stability was found, various gains were recorded to find trends. The test was conducted on the chassis dyno, as before, and the process was:

1. Apply throttle to the vehicle until top speed is reached with no load provided by the dyno
2. Immediately apply a 10A load and wait for system to stabilize and hold for ~10sec
3. Immediately reduce load to 0A and allow for system to reach top speed
4. Repeat the process with application of 5A load
5. Release throttle to let CVT disengage
6. Load dyno to 5A and apply maximum throttle
7. Wait for system to stabilize, if it doesn't let go of the throttle
8. Repeat process with a 3A load

Figure 61: 2017 Baja car strapped to chassis dyno

All results exhibited the actuator overshoot problem, since this tuning process was conducted before the actuator overshoot solution was applied. When going through the process of tuning, the addition of the derivative term was always detrimental, therefore it was not used. Determination of the gains used was four conditions:

1. Disturbance rejection capability
2. Rise time

3. Overshoot
4. Steady state stability

Based on all data collected, a greater integral gain produced better characteristics. Disturbance rejection capability was quantified by looking at the drop in RPM the controller allowed before adjusting the CVT ratio. When checking a tune's disturbance rejection capability, the time taken to get from the minimum engine RPM seen after load application to the setpoint. Overshoot was sought to be minimized to ensure that in cases of impact wheel loadings, the controller would not exhibit severe oscillations. Since overshoot was minimized, settling time was not considered much, or at all. Steady state stability was determined by checking the actuator's movement in a steady state condition. It was previously decided that steady state stability would be compromised to achieve greater step rejection capability. Gains used after the given test was:

Proportional Gain	1.0
Integral Gain	1.1
Derivative Gain	0.0

This PI controller was tested and used for the 2017 California competition but no analysis compared it directly to the CV-Tech's performance, which will be discussed in the testing/analysis portion of this paper. Critique received by judges in the design presentation portion of the competition stated there was a lack of information regarding the system. Specifically, a dynamic model was requested to understand the system. The goal of this system was to be implemented in a short time frame without knowledge of final designs, which could greatly impact any dynamic model, and was why black box tuning was implemented. Further research showed that some systems can have their open loop plant model determined using Matlab's PID tuner app. The app allows a user to input the ideal response of the system (input) and the actual response of the system (output) to determine a model for the system, and allow the user to find controller gain suitable for controller design. To acquire input/output data for this PID tuner app, a potentiometer was used to toggle the setpoint between two values while conducting the test on the chassis dyno. The exact process conducted:

1. Adjust PID gains. Depending on the characteristics of the step rejection curve, it may make curve fitting easier when using the PID tuner app from Matlab.
2. Ensure time, input, and output data are collected. (Ex: time in seconds, setpoint rpm, and filtered engine rpm)
3. Set setpoint values to be used with a difference of ~1000rpm (Ex: 3000 rpm & 2000 rpm). Based on various tests with high loadings, the RPM would drop about 1000rpm.
4. The chassis dyno load is increased to about 5A to ensure the system stays in between the underdrive and overdrive ratio.
5. Full throttle is applied and maintained with the potentiometer set at the lower rpm setpoint.
6. Once the system is at full speed and stabilized at the lower end setpoint (Ex: 2000rpm), change setpoint to upper bound (Ex: 3000 rpm) and allow for system to settle and maintain for ~10 sec.
7. Again, change the setpoint back to the lower bound (Ex: 2000 rpm) and allow the system to settle.
8. Repeat process as needed

Once the data was collected and placed into an excel document, the data section of interest is cut out to import to Matlab.

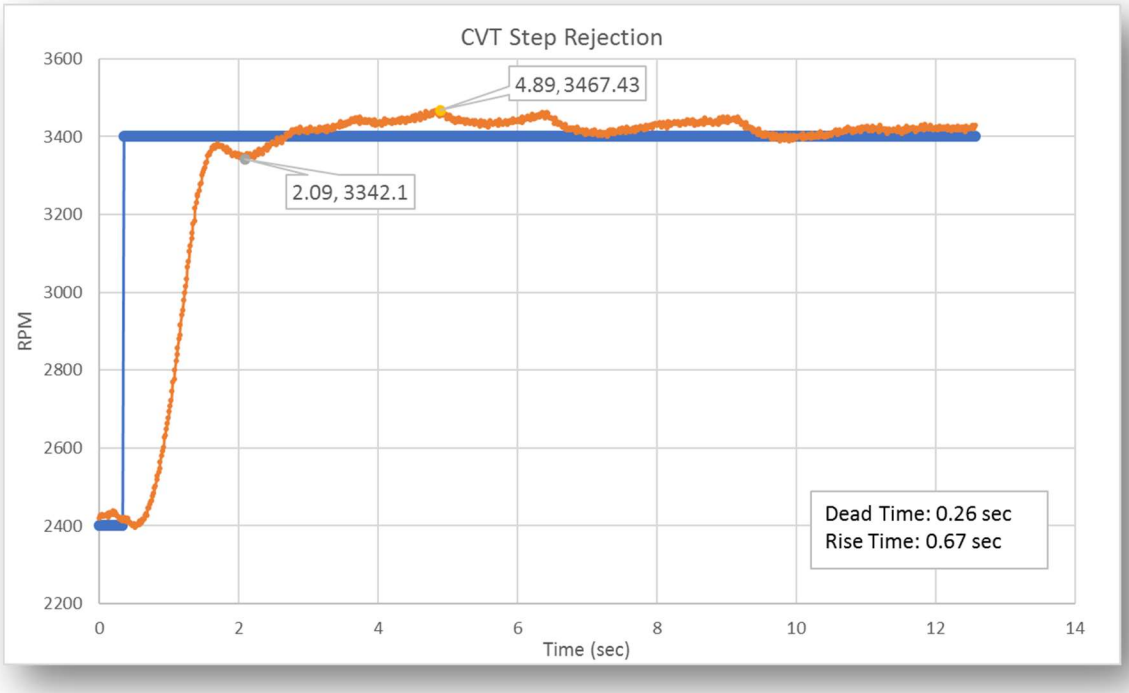

Figure 62: Setpoint change disturbance rejection of controller using gains Kp =1.0, Ki = 1.1, Kd = 0.0

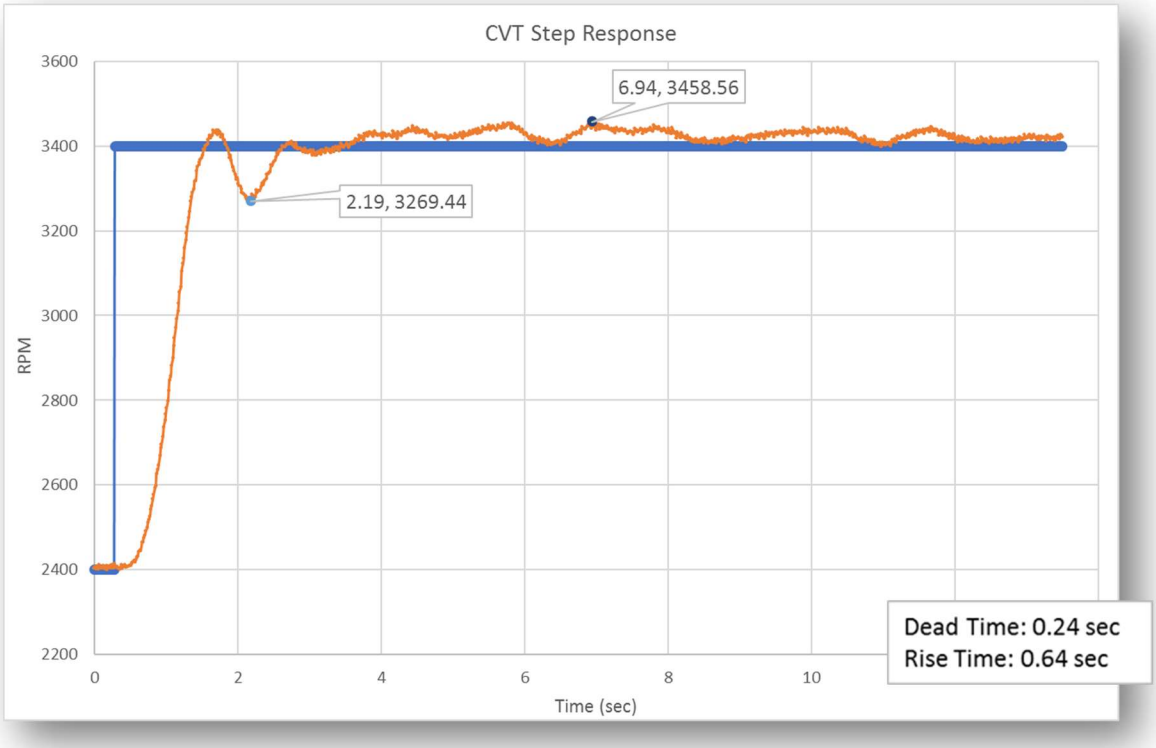

Figure 63: Setpoint change disturbance rejection of controller using gains Kp =1.7, Ki = 0.7, Kd = 0.0

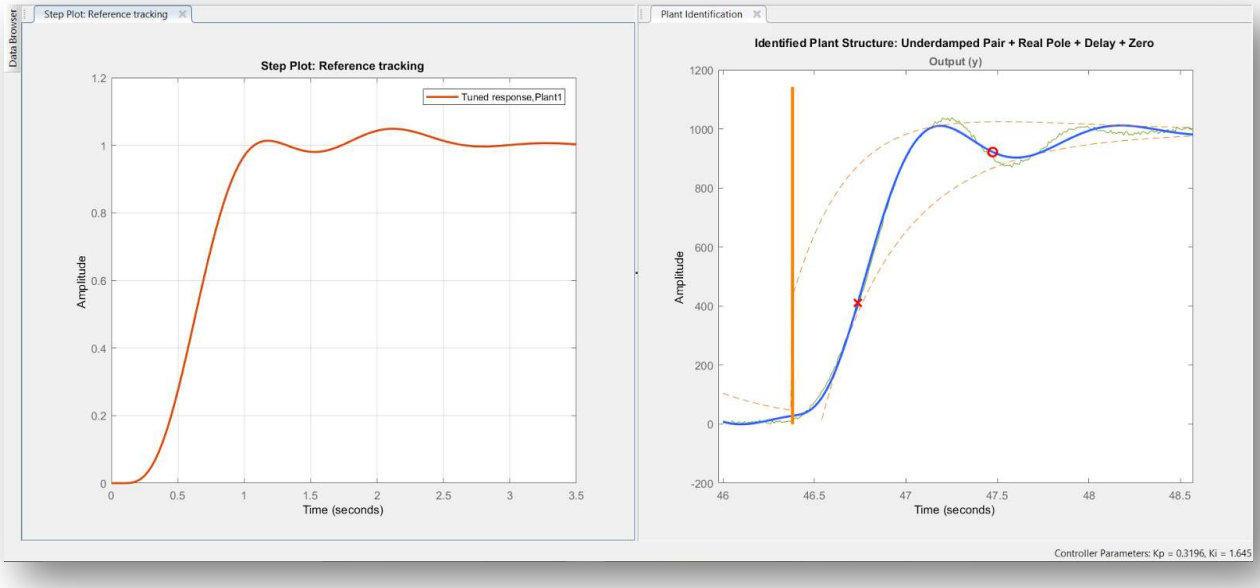

Figure 64: Comparison of disturbance rejection waveforms in Matlab

From identified plant structure:

```
Process model with transfer function:
                          1+Tz*s
  G(s) = Kp * ---------------------------------- * exp(-Td*s)
              (1+2*Zeta*Tw*s+(Tw*s)^2)(1+Tp3*s)

         Kp = 0.99831
         Tw = 0.16429
       Zeta = 0.6551
        Tp3 = 1.4961
         Td = 0.17104
         Tz = 1.7474

Parameterization:
    'P3DUZ'
   Number of free coefficients: 6
   Use "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using PROCEST on time domain data.
Fit to estimation data: 93.7%
```

Once the plant identification process was done, a tuned response was adjusted to find a similar response using the model Matlab has determined. If the PID gains used were similar enough, the model could be used to describe our system and attain gains with characteristics that improved its performance. The results produced different controller gain values for a similar response using collected data. Therefore, the exact model produced could not be used for quantitative information.

Results were then used for a qualitative understanding of the system. The PID tuner app showed that the proportional gain needed to be decreased and the proportional gain increased to improve disturbance rejection. Adjustments made were again tested using the setpoint adjustment method for a disturbance source. The adjustments produced results that exhibited similar characteristics to the model in the PID tuner for a positive step, but when the setpoint was changed to the lower bound there was more overshoot and an increased settling time. It was attempted to characterize that response separately from a positive step rejection, but it was not possible to create a curve fit from the data of 85% accuracy or greater. A curve fit for a pulse input was attempted and the same issue resulted.

With the gains found from the comparative analysis to the Matlab plant model, the controller gains were tested using the initial test method of using the dyno load as a disturbance with the setpoint maintained at the peak power RPM of the engine. The controller gains tested had improved disturbance rejection with a few exhibiting increased overshoot and in the settling time. Despite these characteristics, a new set of controller gains was implemented and tested for drivability.

Proportional Gain	0.80
Integral Gain	2.00
Derivative Gain	0.00

Due to improved performance in step rejection and no issues found with the increased amount of overshoot, the controller was used for comparative analysis with the CV-Tech system.

Final CVT Design Performance Testing

To determine vehicle performance, the 2017 baja car was outfitted with a torque sensor in line of the axle, spark sensor for engine rpm, hall effect reading the rear wheel, and brake pressure sensor.

Figure 65: Futek torque sensor in line of axle shaft.

Testing conducted before the 2017 California competition (Competition dates: April 27-30, 2017) was not comparative to the CV-Tech system. The goal of these tests was analyzing system tracking capability and controller stability. For these tests, Revision 9 was used with gains of Kp = 1.0, Ki = 1.1, and Kd = 0.0. Due to time constraints, testing had to be restricted to locations on campus and tests. The tests conducted needed ensure a reduced risk of inducing component failure since the California competition was 1 week away.

To analyze the data, histograms were taken for driving scenarios of interest. The three scenarios available were acceleration and cornering on asphalt, driving across areas of limited traction (mulch and dirt), and high wheel loading from jumping off stair in J-Lot. To consider the amount of deviation setpoint in each scenario, a majority was looked for in each histogram. A majority of the time was determined to be 70% of the time or greater. The minimum threshold for %time was determined by the distribution of engine RPM in each category. So, some resulted in higher thresholds to meet the cutoff with the number of categories set.

Results from each scenario:

Scenario	% of time	Tolerance about setpoint
Acceleration and Cornering	70%	+/- 200 rpm
Limited Traction	70%	+/- 200 rpm
High Wheel Load	77%	+/- 400 rpm

*Engine #1 was used with the setpoint set to 3210 rpm

Since the tests conducted only used Engine #1, the deviation in RPM was used for tracking performance effects on average power output. Results indicated a larger power drop when RPM swung positively from the setpoint with Engine #2 but Engine #1 remained similar in each direction of RPM fluctuation. Since the horsepower curve of Engine #1 does appear flat and peaks early, the peak was set lower. With Engine #2 having more of a peak in the horsepower curve, the drop was not equal when dropping off the peak in each direction. So, to prevent large horsepower drops in Engine #2, the setpoint needs to be readjusted to a lower RPM which will reduce steady state power, but increase average power output. Data displaying engine RPM out of a corner or any other loading situation shows that the controller starts from the outer bounds of its tolerance, then getting to the ideal RPM value.

	± 200 rpm	± 400 rpm
Engine #1	0.5 hp loss at each boundary	1.0 hp loss at each boundary
Engine #2	1.0 hp loss in positive boundary 0.5 hp loss in negative boundary	6.6 hp loss in positive boundary 1.2 hp loss in negative boundary

*Horsepower losses excludes data sets deviating from most of data sets (i.e. HP6 of Engine #2 & HP1 of Engine #1).

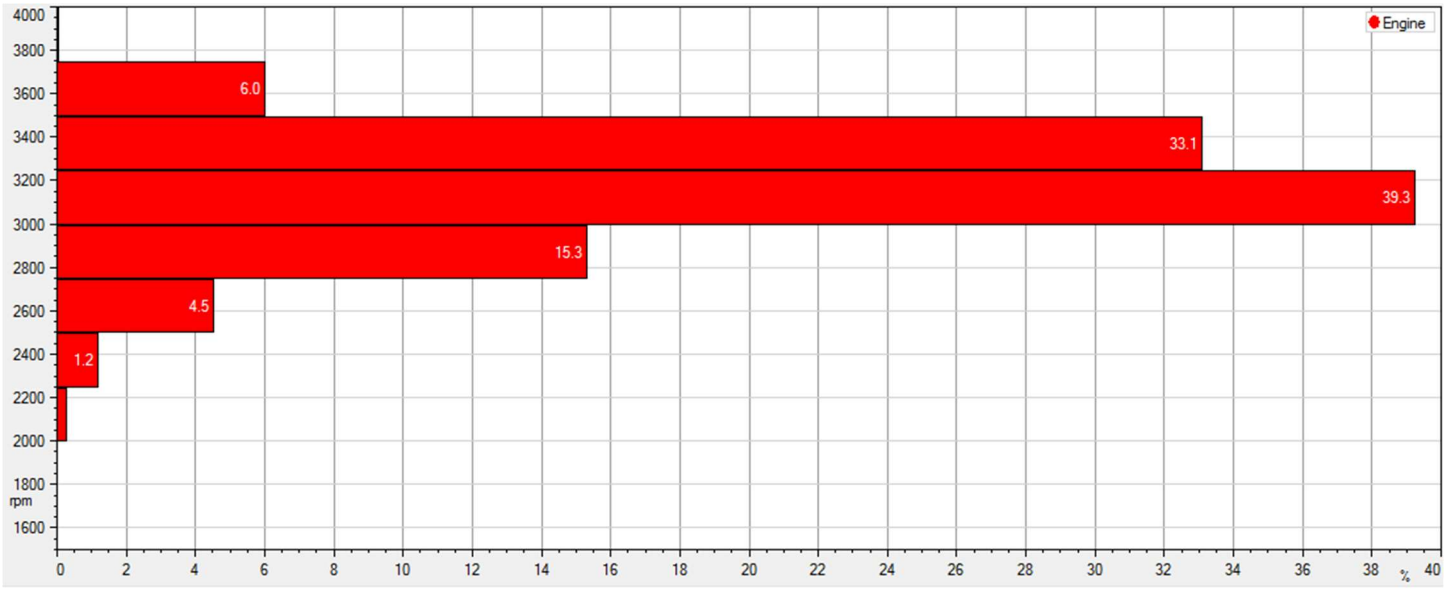

Figure 66: Cal Poly Test Track - Driving over mulch piles using Rev 9, engine RPM histogram with 10 categories

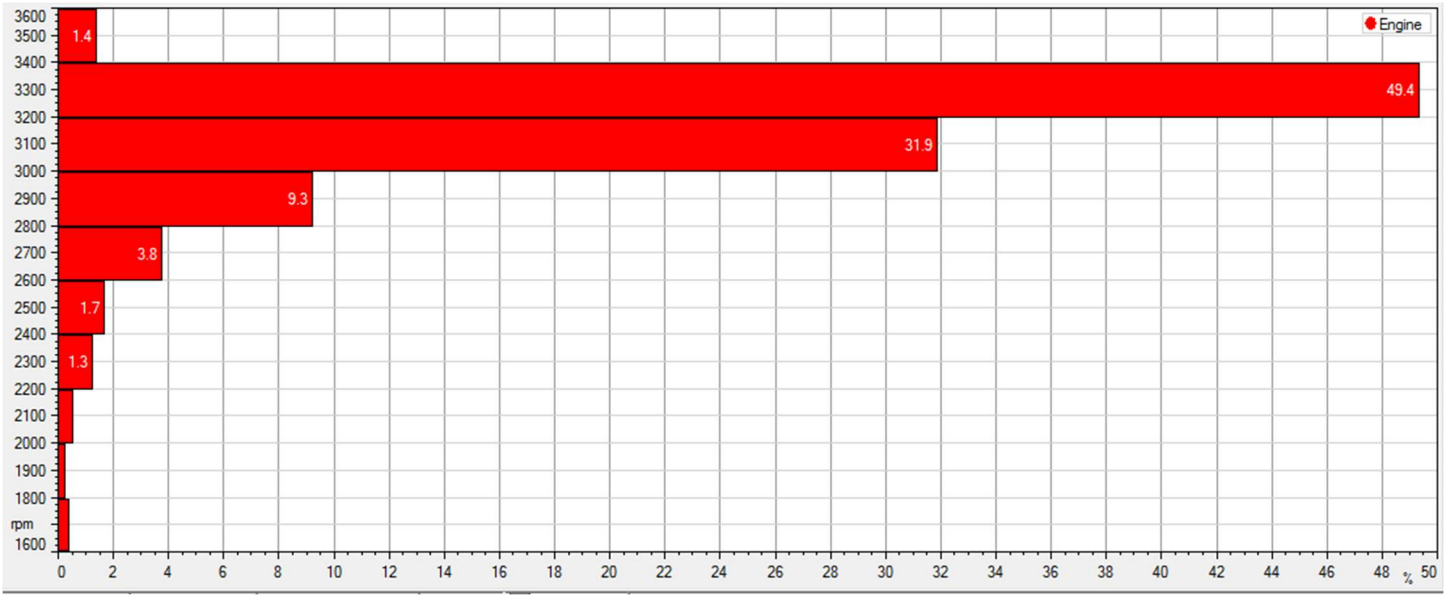

Figure 67: J-Lot Acceleration and cornering, 10 category histogram of engine rpm using Rev 9

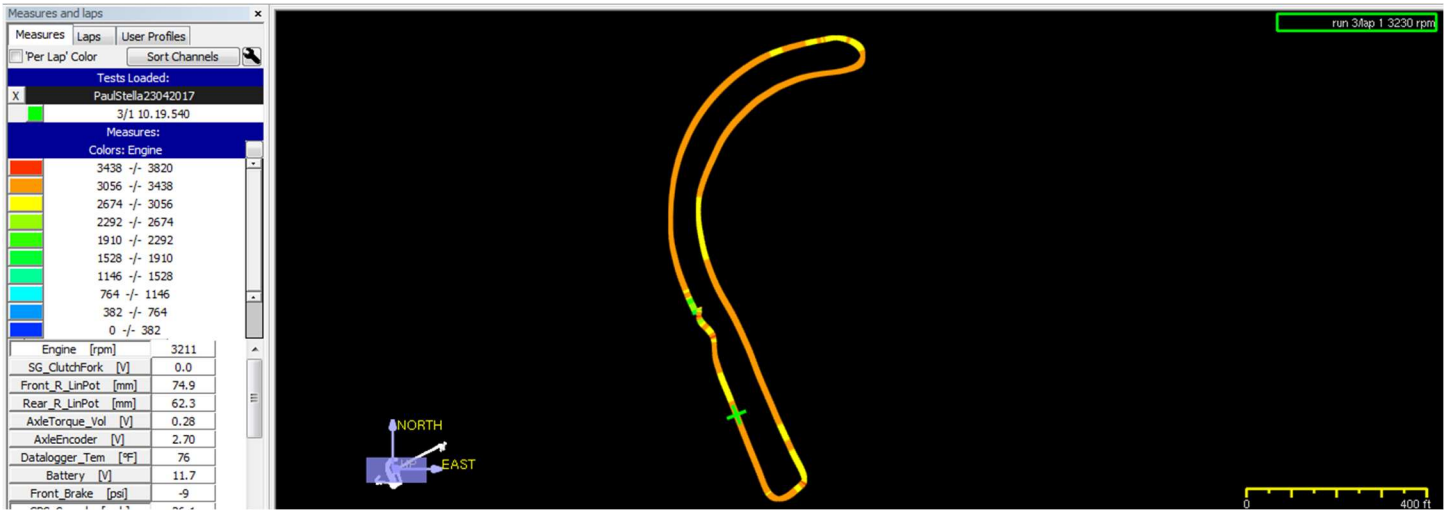

Figure 68: J-Lot acceleration and cornering GPS data displaying engine RPM

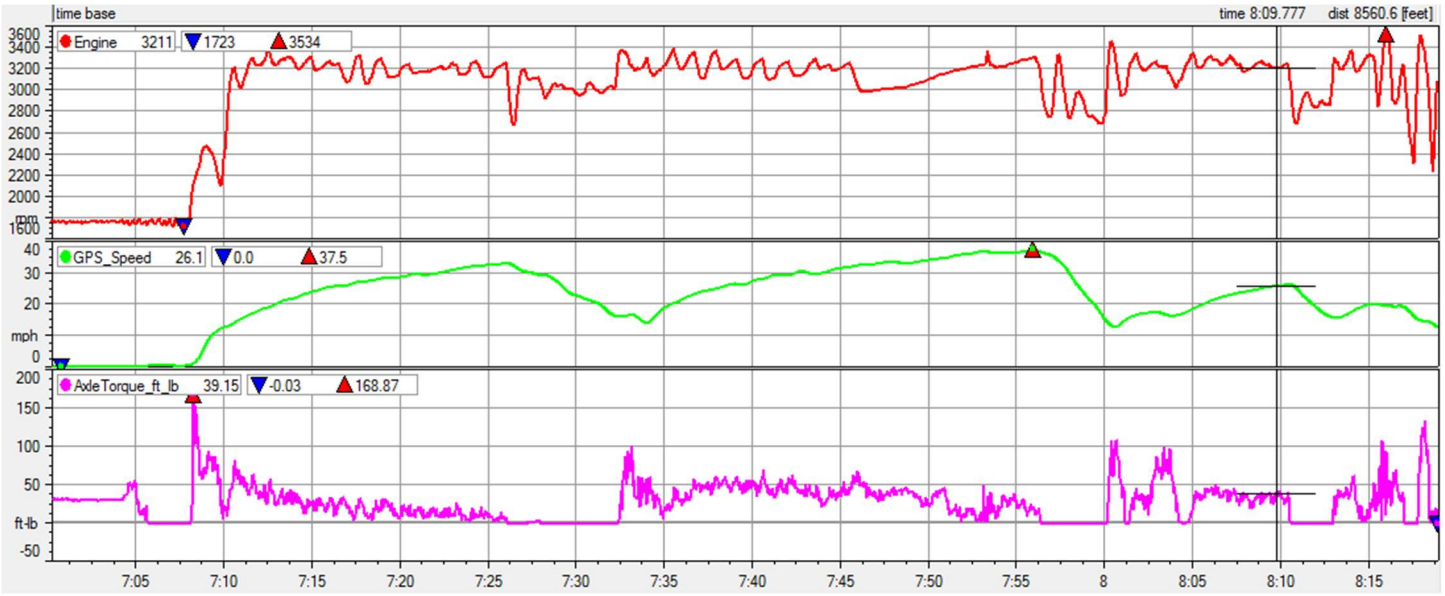

Figure 69: J-Lot Acceleration and cornering, data from track displayed above, code used was Rev 9

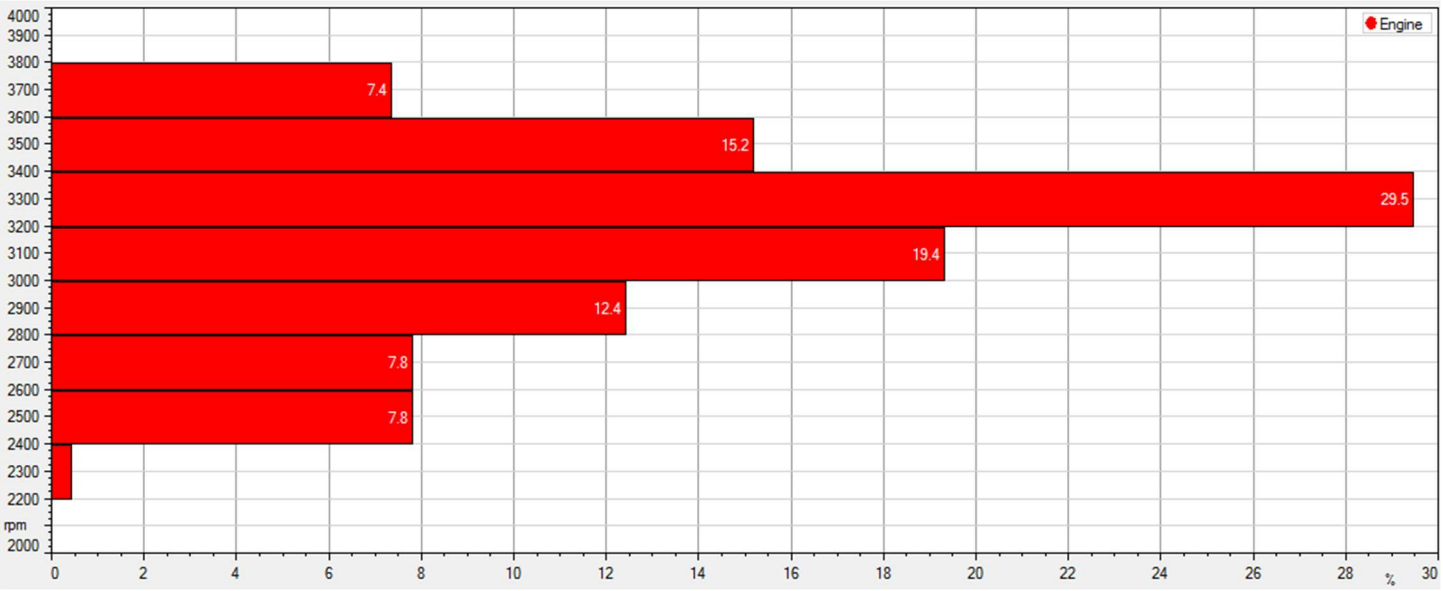

Figure 70: J-Lot driving down stairs, 10 category histogram of engine rpm using Rev 9

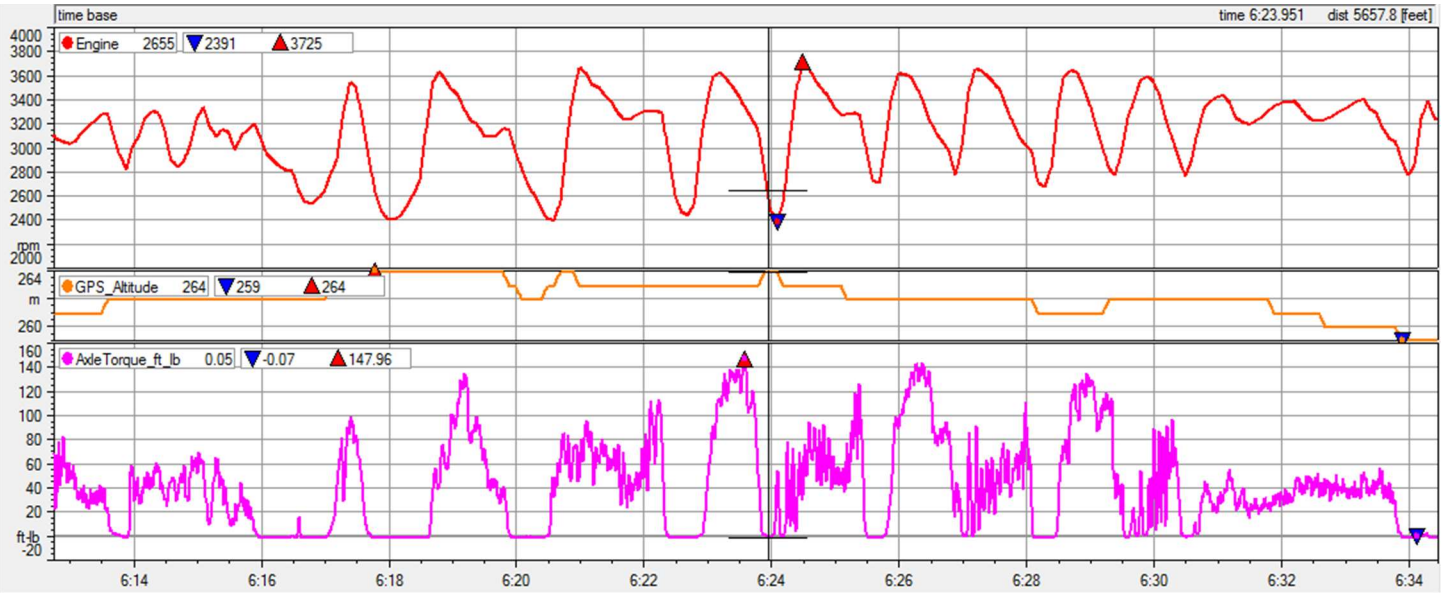

Figure 71: J-Lot driving down stairs, data correlating to histogram of engine rpm, using Rev 9

Comparative data collected was all done on the chassis dyno due to loss of field testing data on the CV-Tech system. Data collected allowed for analysis on effects of the updated controller structure to the performance of the PID tracking and the deviations in RPM for other scenarios. In the following test data, Engine #2 was used and the setpoint used was at peak power. Deviations from the setpoint were similar that found in the initial tests conducted with Engine #1, therefore the setpoint was later adjusted.

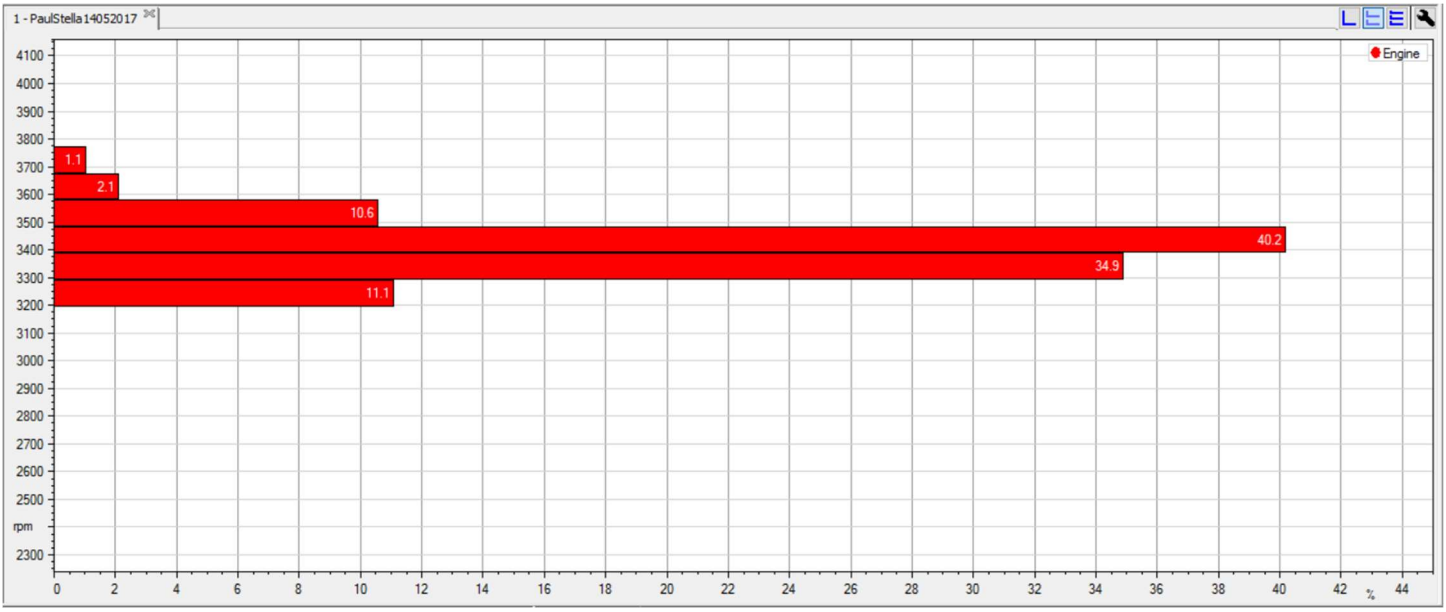

Figure 72: Cal Poly Test Track – Traction limited situation - driving over mulch using Rev 24; 20 category histogram of engine RPM

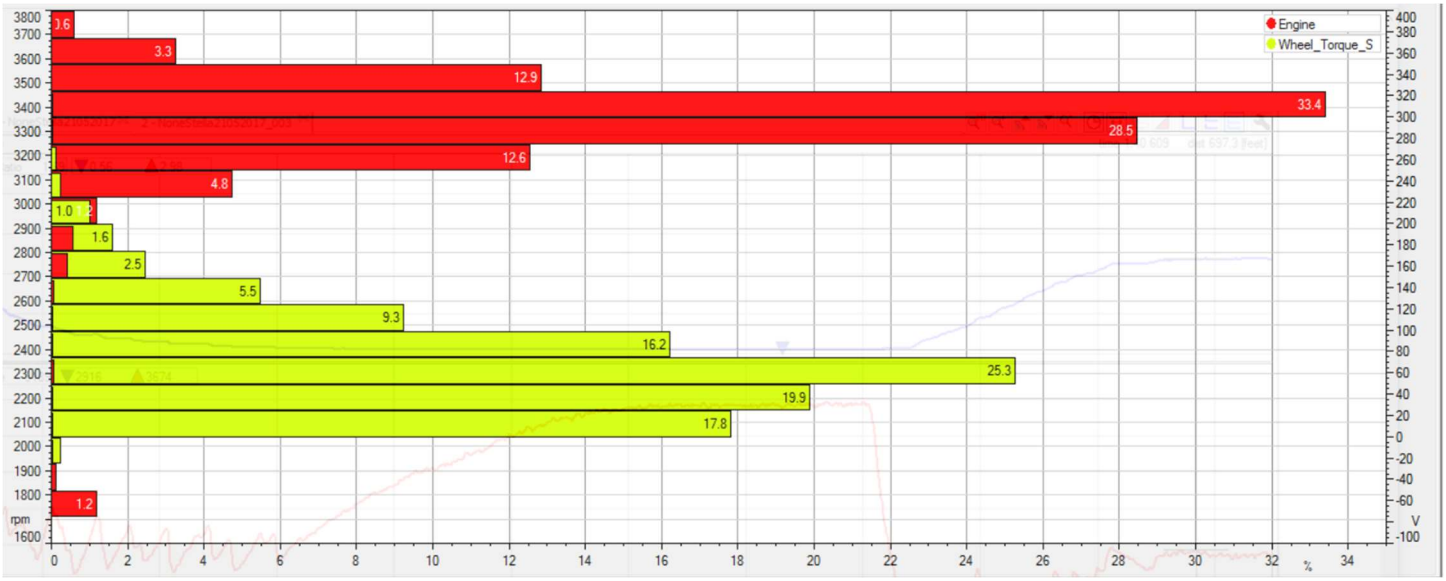

Figure 73: M-Lot Hill - Hill climb data starting at base of hill comparing torque and engine RPM with 20 category histogram of eCVT using Rev 24

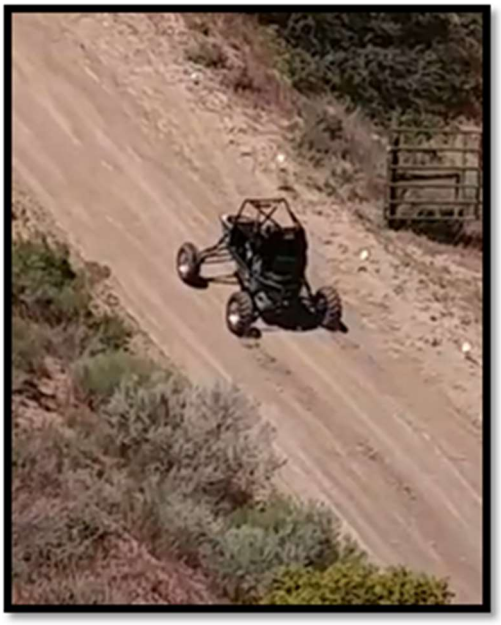

Figure 74: Hill Climb from 2017 California competition

Other information extrapolated from the tests is the ability of the vehicle to reject a greater disturbance compared to previous observations. In *figure 72*, a J-turn is performed but the engine does not stall, and the engine RPM drops to 2600 RPM which correlates to 9 hp. Examining the critical scenario of a jump while maintaining throttle upon landing, referring to *figure 73*, the car is able to keep the engine from stalling. Just after landing, the engine drops to 2379 rpm and overshoots by + 400rpm. This proves the robustness of the controller meeting the requirements of the system to maintain engine RPM even in the worst-case scenario. Due to the high instantaneous loading of this scenario, 317 ft-lbs of torque at the wheel, it cannot be recreated on the chassis dyno to test the controller's stability and other disturbance rejection characteristics during the worst-case scenario. The only testing method that could recreate this disturbance would have to be a setpoint change disturbance, like what was conducted to characterize the open loop plant model.

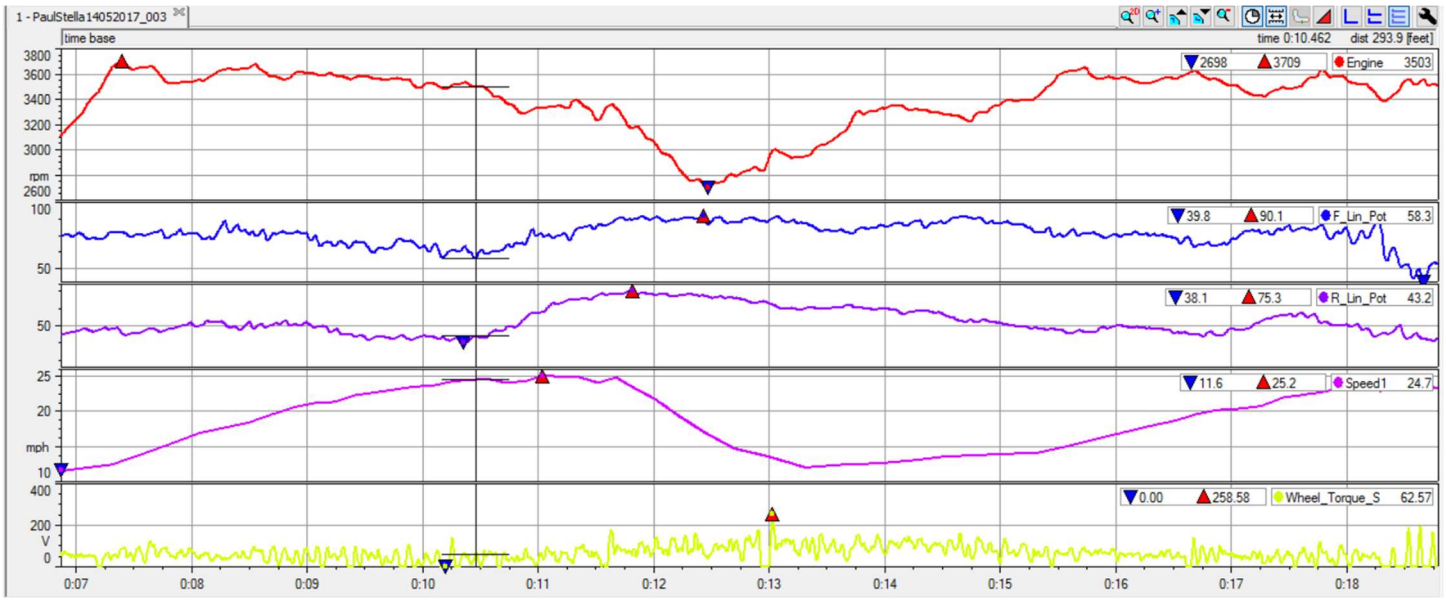

Figure 75: Cal Poly Test Track - J-Turn on dirt using Rev 24

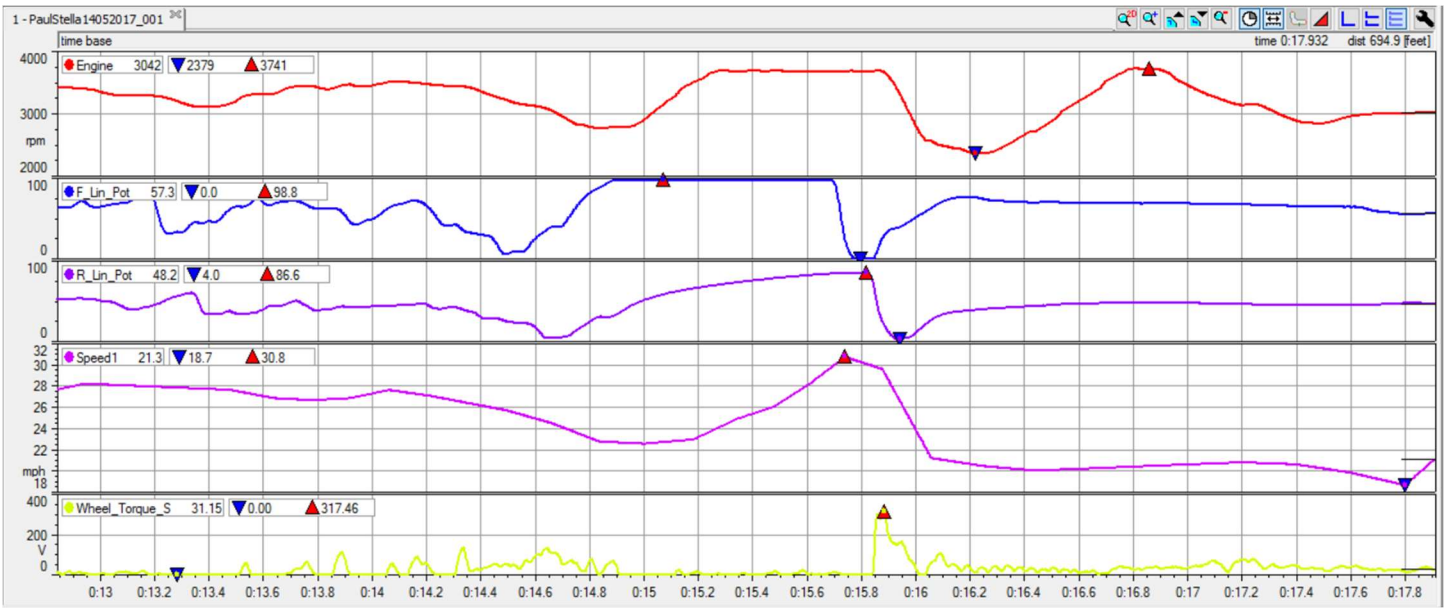

Figure 76: Cal Poly Test Track - Jump with landing while maintaining full throttle using Rev 24

Figure 77: Cal Poly Test Track – Jump

The controller gains used for the comparative tests were the most recently implemented (Kp =0.8, Ki = 2.0, Kd = 0) and the controller code used was Revision 24. The CV-tech system was used the way it came from the manufacturer, but when looking at acceleration data, it is set properly near peak power.

The testing process conducted was:

1. Acceleration
    a. The dyno was set to 0A
    b. Full throttle was applied until maximum speed was reached
    c. Then the throttle was completely released.
2. Back shifting with load disturbance (5A and 10A) [the same method used in the initial tuning process]
    . Set dyno to 0A load
    a. Apply full throttle and maintain. Do nothing until top speed is reached
    b. Immediately set dyno load to 5A.

c. Allow for system to settle

d. Release throttle

e. Repeat process but with high load applied being 10A instead of 5A

3. High load at vehicle start (5A, and 10A), exemplary of a tractor pull with high traction

. Set dyno to desired load, 5A

a. Apply full throttle and maintain until system settles

b. Release throttle

c. Repeat with 10A load

4. Oscillating load, exemplary of woops

. Set dyno to 0A

a. Apply full throttle and maintain

b. Oscillate dyno load between 0A and 5A, maintaining each load for 1 second for a total of ~15 second

c. Release throttle

d. Repeat process but oscillating between 0-10A

Highlights from the data collected in the tests, was a 15% speed increase in the simulated tractor-pull and 15% improved tracking in the simulated whoops. The acceleration runs on the dyno looked almost the same with the difference being the RPM the controlled CVT and CV-Tech start. The controlled CVT does not reach its setpoint until 3 seconds after launching from a stand still. Whereas the CV-Tech CVT reaches the same rpm in 1 sec, providing more power for 2 seconds due to the steep drop in power above 3500 rpm. When comparing 100ft acceleration times, there was a 5% decrease in acceleration times compared to the CV-Tech system. This result is contrary to what was seen on the dyno, but may be attributed to the traction capability. With the car strapped to the chassis dyno, there is 100% traction whereas on loose dirt, traction is a bit limited.

Comparing the back-shifting ability of both systems (figure 77 and figure 78), it's clear that the CV-Tech cannot maintain tracking a single RPM value in all scenarios. The difference in RPM relates to a difference in about 2.5hp with the CV-tech outputting 8.5 hp after reaching its steady state with a 5A load. With a 10A load, the difference is 3hp. An interesting aspect when comparing the disturbance rejection ability of both systems, the time taken to settle at its steady state RPM is similar. This may be due to the overall powertrain system dynamics, the dynamic characteristics of the engine, or effects from the driven pulley system.

Similar differences in power output are seen in the simulated tractor-pull data (figure 79 and figure 80), where even a speed difference of 1mph is seen. The problem with this controller is the increased settling time and overshoot. The system oscillates so much that wheel speed is greatly affected which is detrimental and bothersome to the driver. Again, analyzing the simulated whoops data for both loads, the RPM fluctuates greatly, losing power with greater time spent in the higher RPM ranges.

PI Controlled CVT	CV-Tech CVT
4.608 sec	4.37 sec
4.144 sec	4.46 sec
3.989 sec	4.52 sec
4.298 sec	4.62 sec
	4.51 sec
Avg: 4.26 sec	Avg: 4.50

*100ft acceleration times in Cal Poly Test Track area

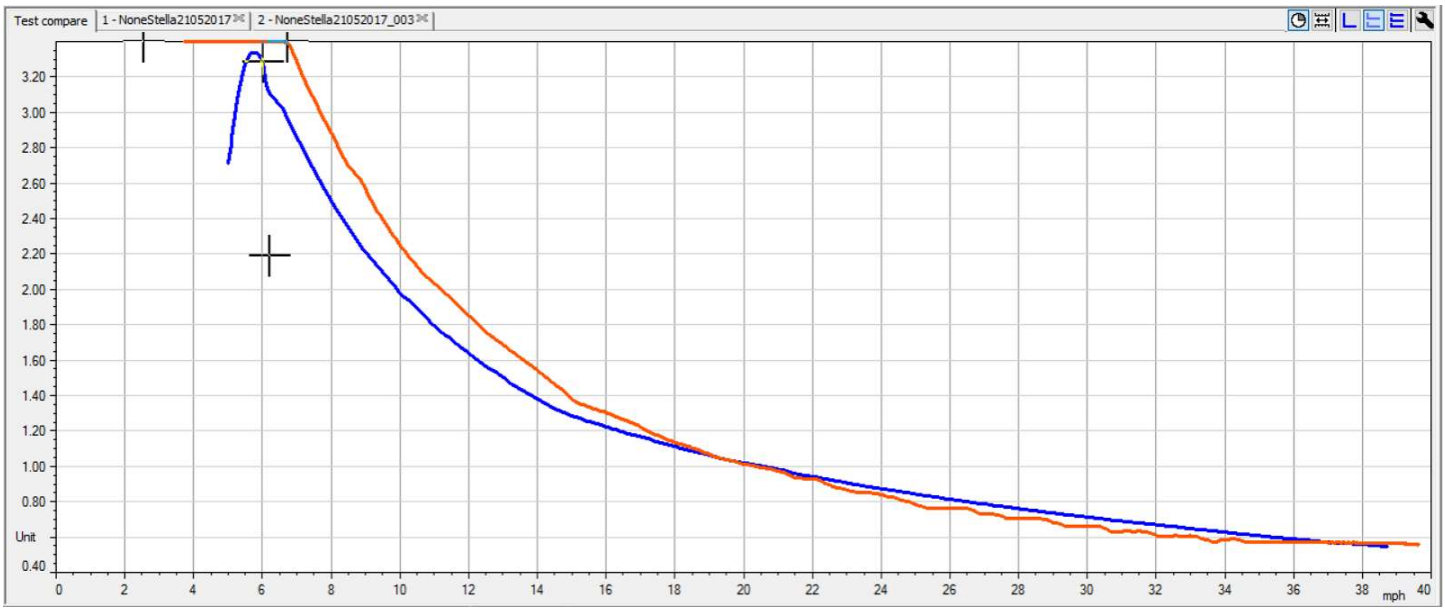

Figure 78: Shifting curves (ratio vs speed) of the eCVT (orange) and CV-Tech CVT (blue)

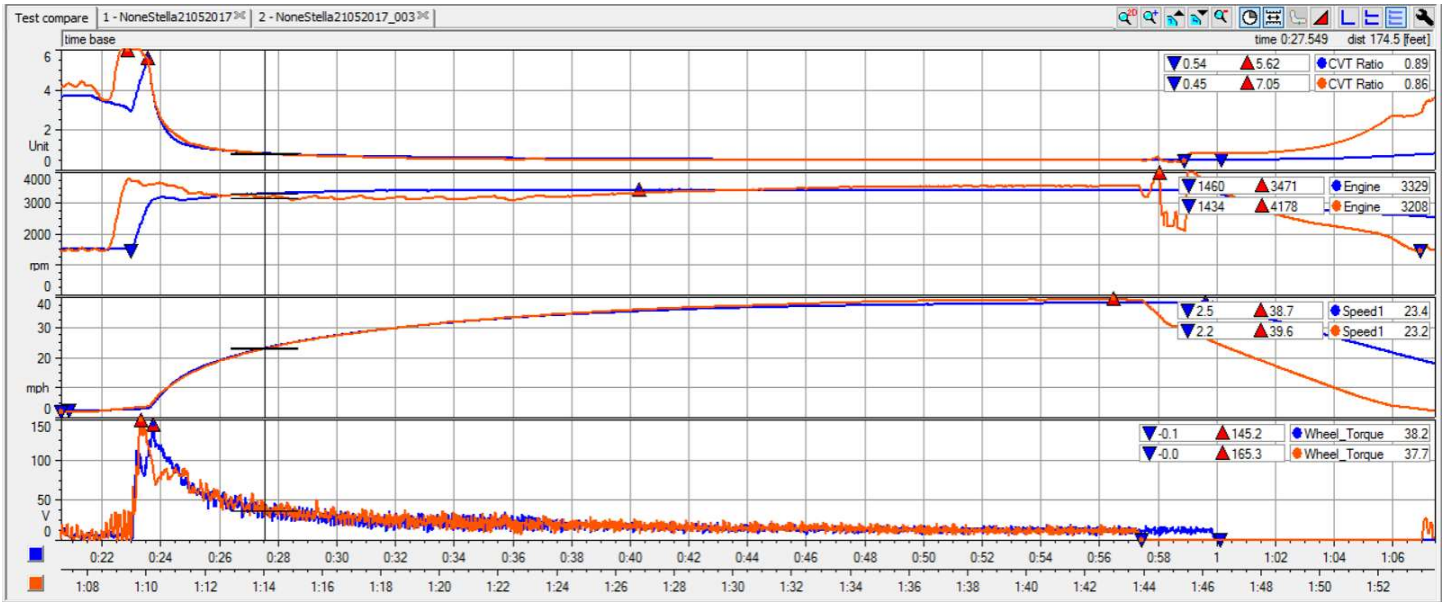

Figure 79: Acceleration run comparison between eCVT (orange) and CV-Tech CVT (blue)

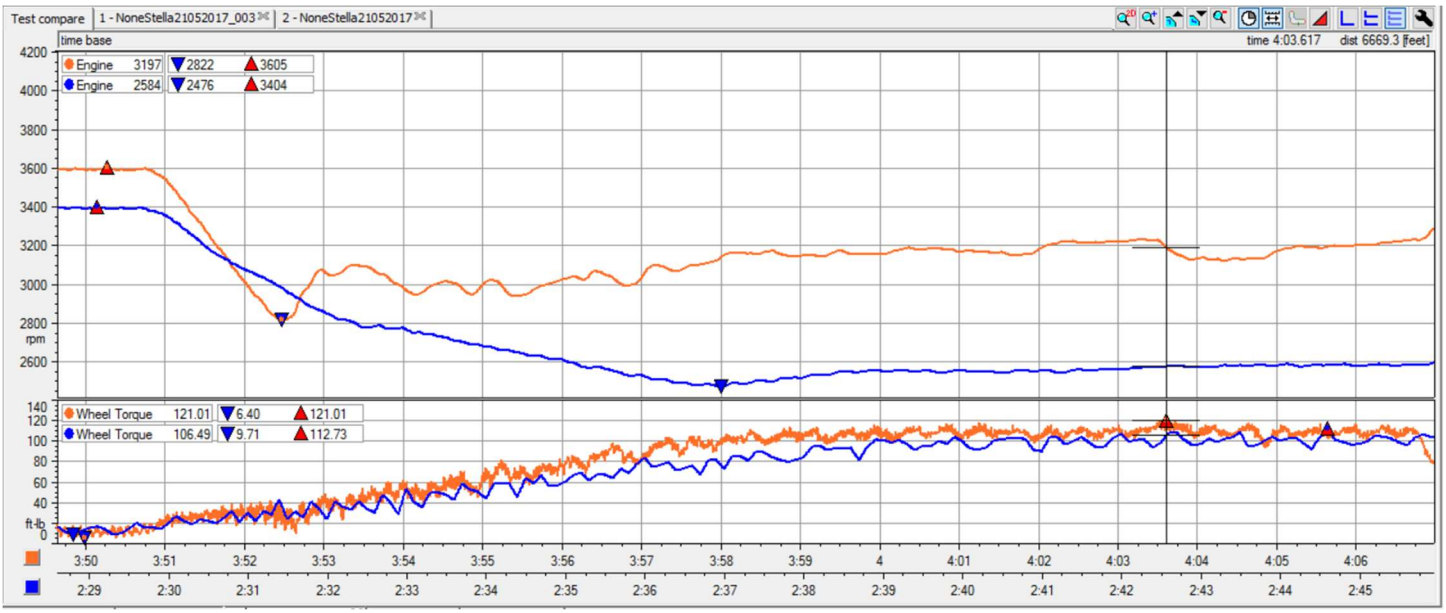

Figure 80: Backshifting comparison with 5A load between eCVT (orange) and CV-Tech (blue)

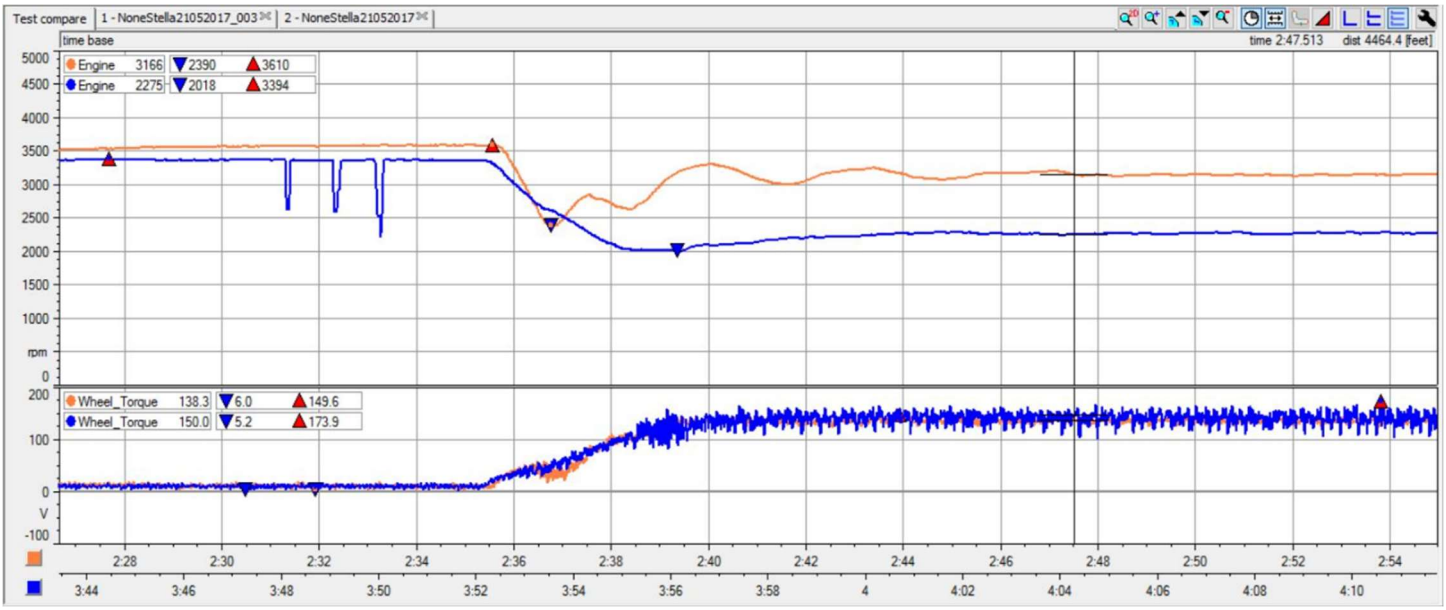

Figure 81: Backshifting comparison with 10A load between eCVT (orange) and CV-Tech (blue)

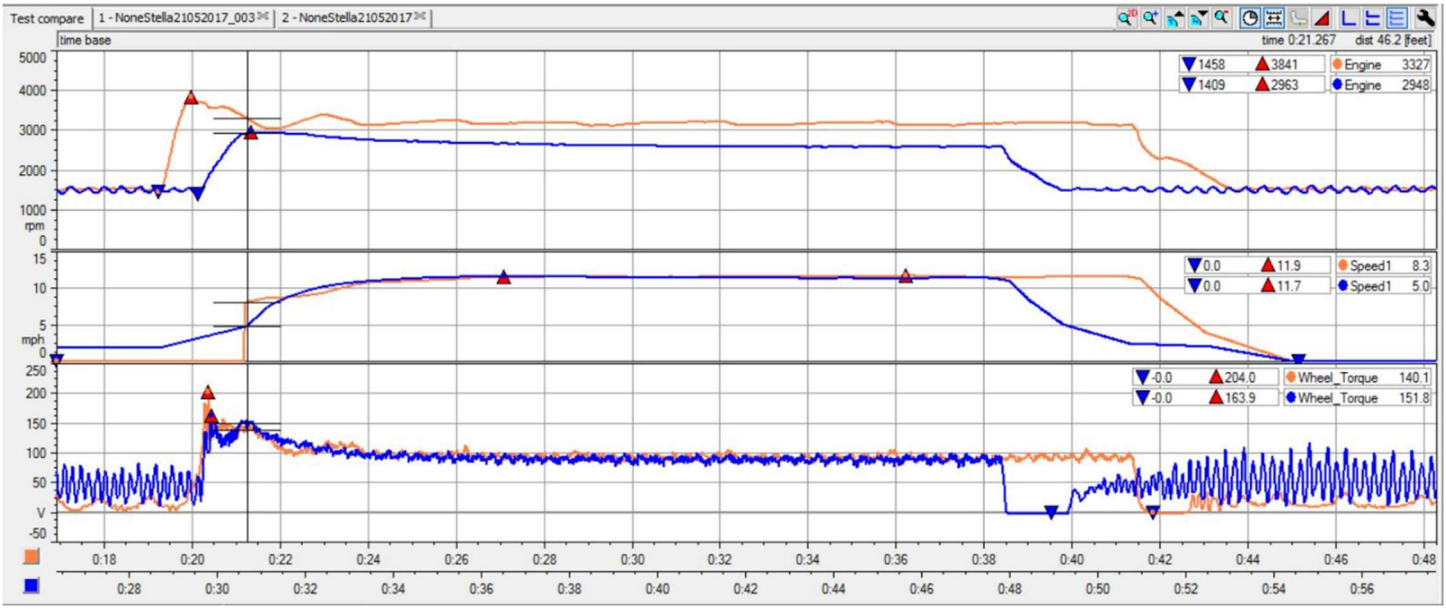

Figure 82: Simulated tractor-pull with 5A load comparison between eCVT (orange) and CV-Tech (blue)

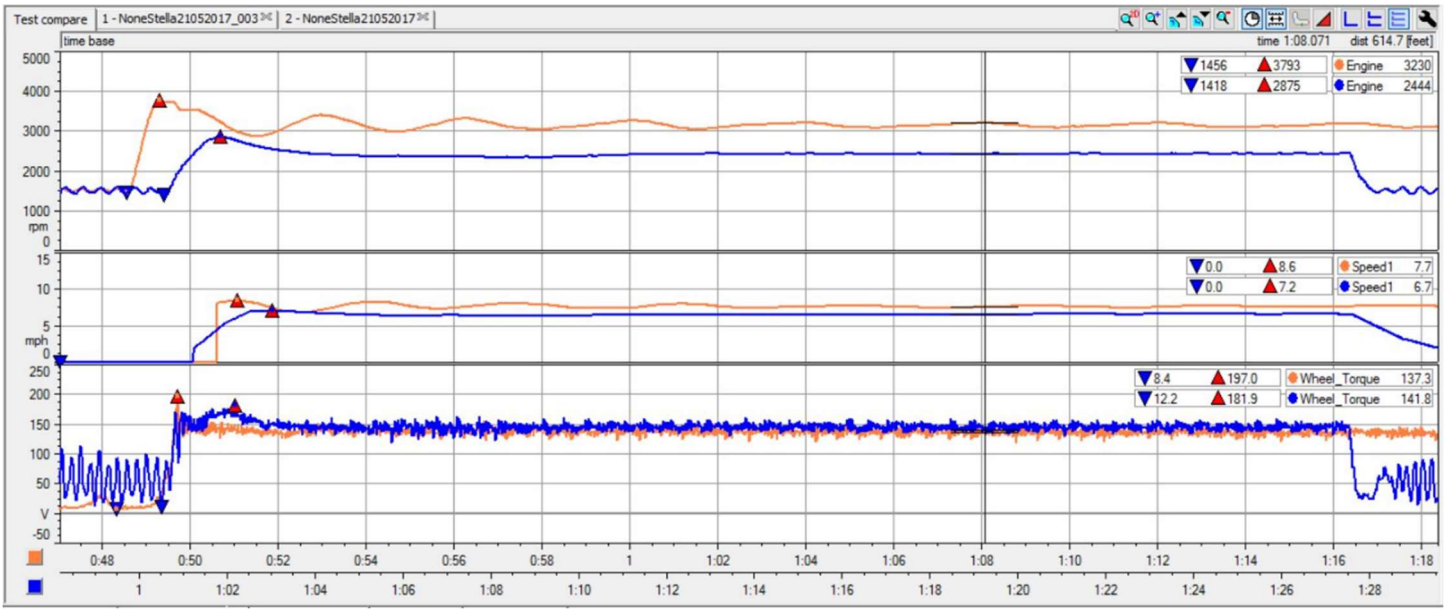

Figure 83: Simulated tractor-pull with 10A load comparison between eCVT (orange) and CV-Tech (blue)

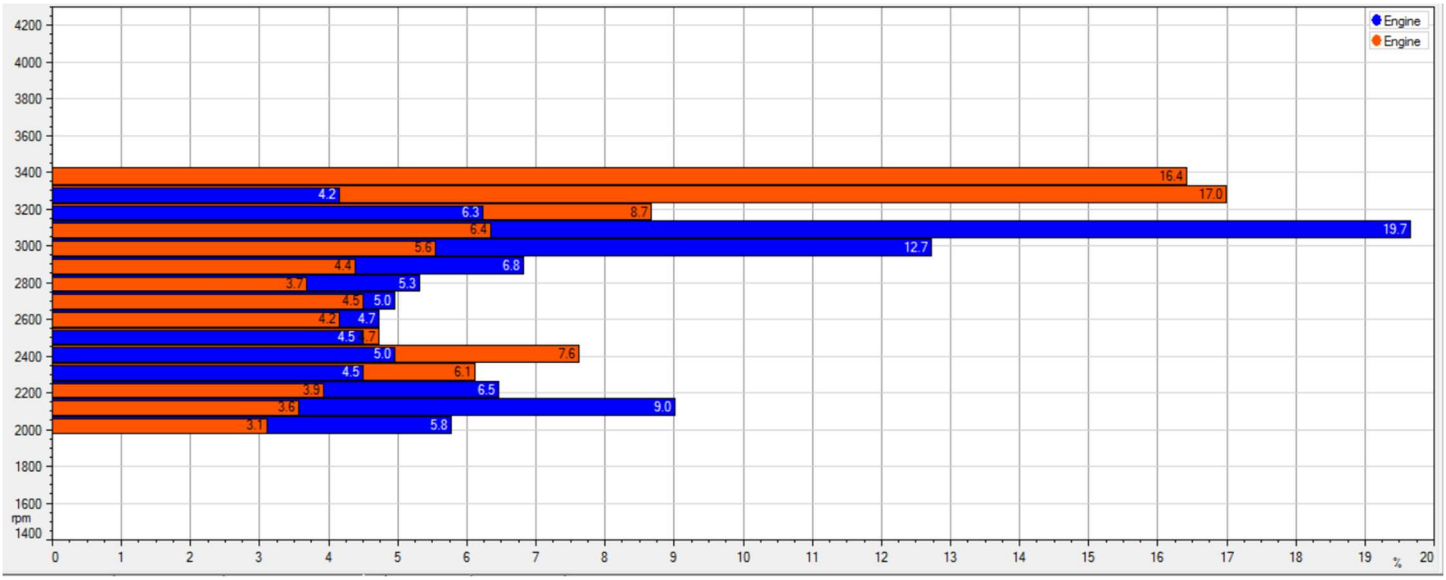

Figure 84: Simulated woops with 0-5A load. 30 category histogram of engine rpm over 20 seconds for eCVT (orange) and CV-Tech CVT (blue)

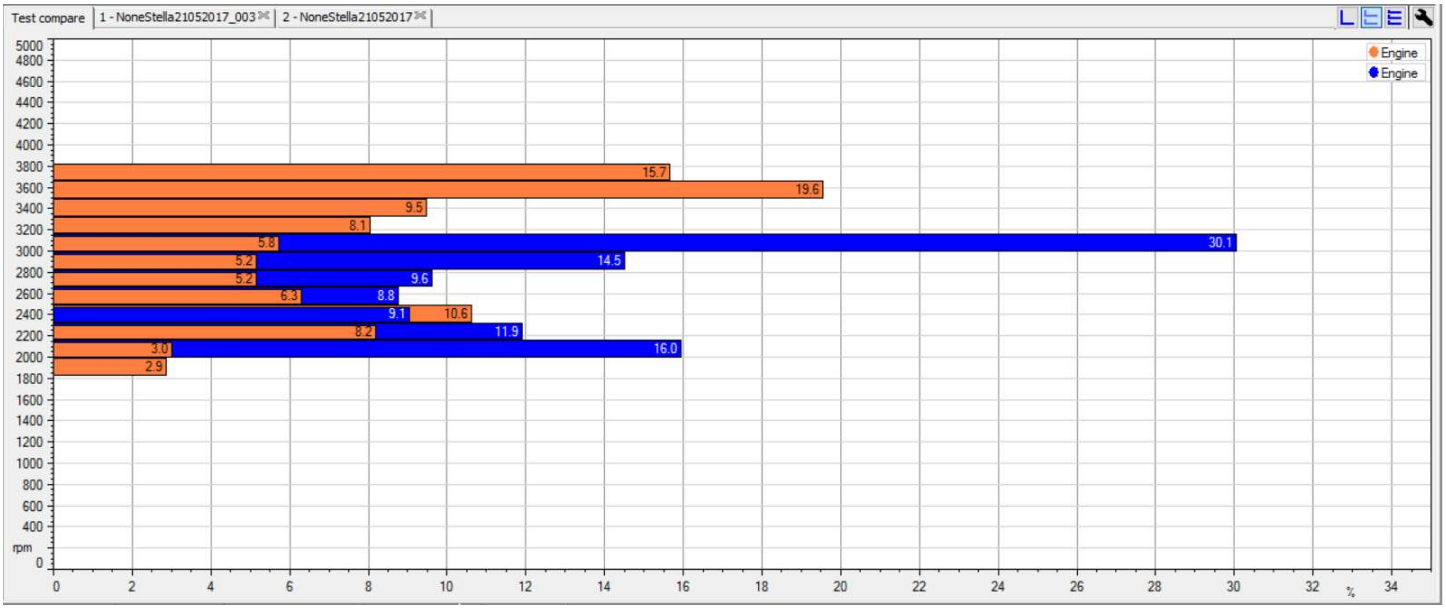

Figure 85: Simulated woops with 0-10A load. 30 category histogram of engine rpm over 20 seconds for eCVT (orange) and CV-Tech CVT (blue)

Figure 86: Cal Poly Test Track – Whoops

Conclusions and Future Work

The overall performance gains in a comparative analysis were not as substantial as expected, but the control system is still considered unfinished. Despite a 5% decrease in 100ft acceleration times, acceleration times at the Kansas and Illinois competitions both were 4.6 sec. Startup acceleration played a large role in the results of the sled pull and hill climb events. In the hill climb event, the car would only start to pick up momentum once the car started going up the hill (fully inclined) instead of picking up momentum in the less sloped section of the hill climb. In the sled pull event, the engine RPM would reach its peak RPM and only then would the CVT start shifting, jerking the car forward from the engine speed but lacking in power due to the torque drop. If the system started shifting at a lower engine RPM, the torque necessary to start the vehicle would be more than sufficient and the horsepower drop is only due to the RPM between 2000 – 3200 rpm. Revision 27 was developed to implement shifting at a lower RPM and was minorly successful, but it was not consistent in starting to shift from the same RPM. It was an improvement from Revision 24, but was not consistent. A notable situation that the driver could induce was by revving the engine and not letting the controller actuate, then letting off the throttle, and finally applying full throttle, the car would launch forward as ideally wanted. Since Revision 27 has a ramping setpoint from 1800 rpm (idle), if the the engine RPM started higher by a few hundred RPM immediately entering the PID, the PID would respond aggressively. All of this was tested by observation in the Illinois competition, but there was not enough time to test other changes to the code. Another solution was coded to create consistency in the launch characteristics, where an if statement will

look for the ideal RPM to start the PID calculation with the ramping setpoint. This solution remains to be tested and will be given to the 2017-2018 design team to implement. The ramping setpoint may be the ideal solution for an acceleration run, but it may be possible that the slope of the ramp would have to be different for different situations such as a sled pull or hill climb. Looking at graphs displaying the actuator overshoot problem, the characteristic of the RPM curve is almost parabolic, unlike a straight ramp. It is possible that the setpoint should be ramped on a parabolic curve. For the future team, it would be ideal to determine the right starting wheel torque to find the right engine RPM to begin actuation. From there, the straight ramping setpoint should be changed to determine the ideal starting setpoint and the ideal slope to get the best acceleration.

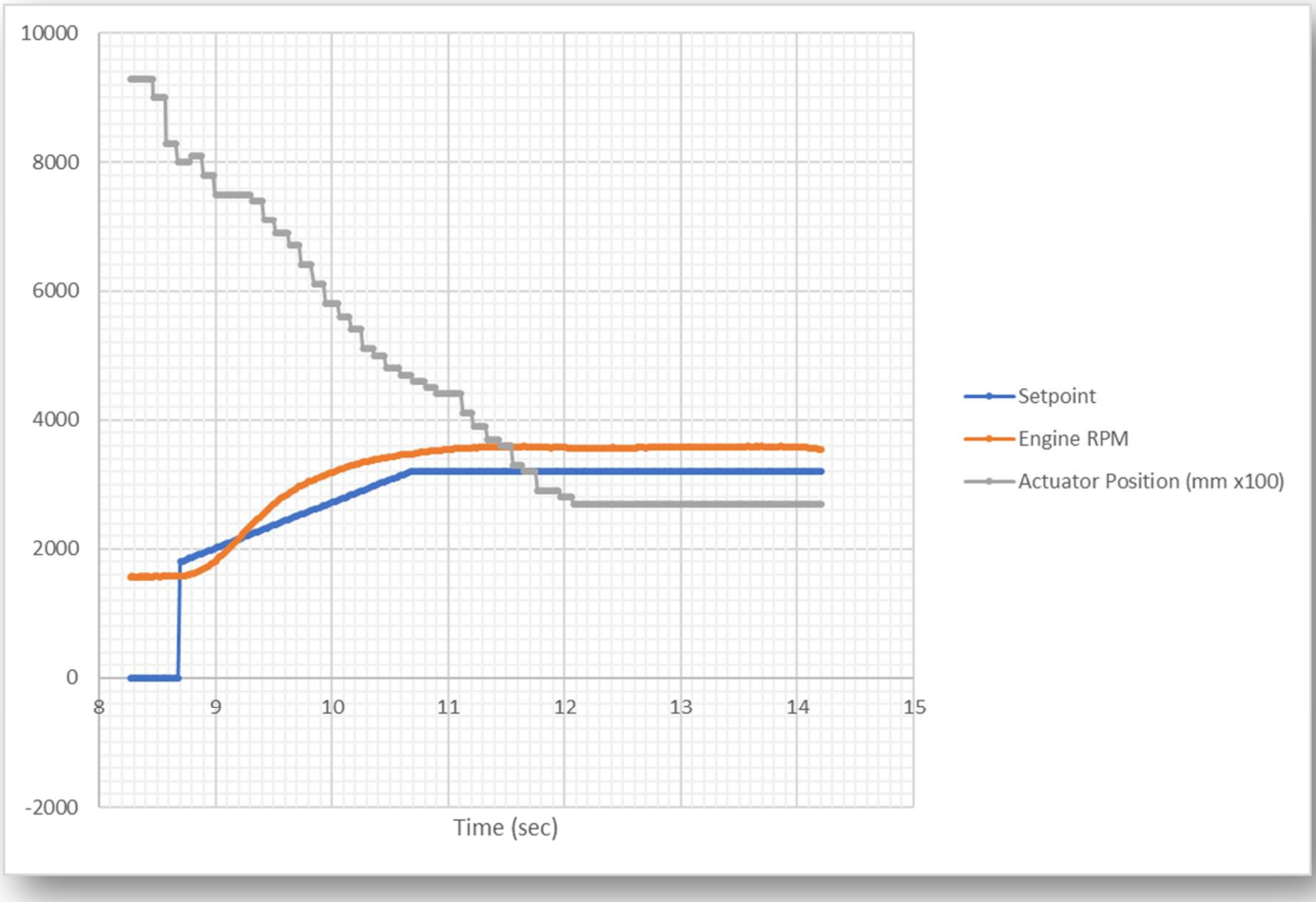

Figure 87: Depiction of acceleration with ramping setpoint using Rev27

Other possible future controller improvements, is the introduction of different PID controllers for upshifting (towards overdrive) and downshifting (towards underdrive). Based on histograms and tuning by a

setpoint change, having a single PID for upshifting and downshifting creates a need for compromise between the ideals of each. When changing the setpoint in a positive direction the controller was very responsive and had no overshoot, but when the setpoint was changed in the negative direction, the controller was. By looking at the direction of the error, a different set of gains could be implemented.

Despite the PI controller dictating the disturbance rejection, the mechanical limitations play a greater role to the system's ability to reject a disturbance. To further improve shifting characteristics, changes to the driven pulley's design could be made. Also, if the actuator can increase the speed of shifting then response could be improved, but is again dependent on physical shifting limitations.

During all competitions this season, the cooling system was the main issue that led to power consumption problems and component failures. The high case temperatures created problems with hall effect failures in the California competition. In the Kansas competition, the cooling fan housing was deformed from heat and stalled the motor resulting in overdrawing of power. In the Illinois competition, the pressure plate assembly fell apart as the press fit of the bearing was lost since aluminum housing around the bearing expanded causing a catastrophic failure of the V-Belt. For the same competition, the actuator failed due to overheating problems and stopped actuation altogether and maintaining disengagement. The future CVT team at CPP Baja SAE will have to take large precautions in considering temperature effects to component design and if cooling is implemented, a reliable power system that will not affect the control system in case of failure. Power loss had the worst failure mode seen during the California competition, endangering the driver, with the CVT stuck in the engaged position driving into a wall and tree. The only method of combating this failure was the engine kill switch, removing power. For these failures, troubleshooting issues took too long in time critical events. Simple indicators to the driver or individual servicing the system could have aided in troubleshooting.

An important aspect of the control system is ensuring the driver feels under control of the vehicle at all times. With the issues encountered by each controller design, the driver had to be well informed as to understand what issues are known and how they can avoid them or if unknown, how can they expose problems during tests. Considering the attempt to get a more responsive controller, the result was a controller that would

overshoot and had increased settling times. Oscillations cause the driver to be concerned and uncomfortable while driving if they have not become accustomed. It is of great importance to ensure that any controller revision does not induce a feeling of loss of control to the driver.

As a summary of goals that the future Baja SAE CVT team should greatly consider are a far more robust/reliable system, greater performance gains, greater integration within the Powertrain subsystem, and increase in serviceability in likely failure points.

12) Ziegler, J.G & Nichols, N. B. (1942). "Optimum settings for automatic controllers" (PDF). Transactions of the ASME. **64**: 759–768

13) "Filter Design Analysis." Okawa Electric Design. Okawa Electric Design, 2004. Web. 29 Jun 2017. <http://sim.okawa-denshi.jp/en/Fkeisan.htm>.

14) "Ziegler-Nichols Tuning Rules for PID." *Microstar Laboratories.* Microstar Laboratories, Web. 29 Jun 2017. <http://www.mstarlabs.com/control/znrule.html>.

a. "Getting in tune with Ziegler-Nichols," Thomas R. Kurfess, PhD, in the Academic Viewpoint column, *Control Engineering* magazine, Feb 2007 issue, p. 28, http://www.controleng.com/.
b. The classic original paper: "Optimum settings for automatic controllers," J. B. Ziegler and N. B. Nichols, *ASME Transactions*, v64 (1942), pp. 759-768.
c. A more recent survey that covers the Ziegler-Nichols and Kappa-Tau tuning rules: "Automatic Tuning of PID Controllers," Karl J. Åström & Tore Hägglund, Chapter 52, *The Control Handbook*, IEEE/CRC Press, 1995, William S. Levine ed.
d. Selected from the tuning rules listed in the paper "Rule-Based Autotuning Based on Frequency Domain Identification," Anthony S. McCormack and Keith R. Godfrey, *IEEE Transactions on Control Systems Technology*, vol 6 no 1, January 1998.
e. For more information about obtaining frequency response curves automatically, see the article A Self-Testing PID Loop on this site.