

# The Heart of the Beast Notes and Observations - 2017

## FRC 2855 - BEASTBot

By Max Narvaez, with help from Branden Asplund

*These are descriptions of our subsystems, problems we ran into and how we fixed them, observations we had and other notes about each subsystem or problem.*

### Drivetrain

The choice to use Mecanum wheels was ok, but we should have learned more about them beforehand. The wheels need to be in a specific orientation, O or X. When viewed from above, the rollers should look like one of the below diagrams.

/// \\  
←  
\\ ///

Or

\\ ///  
←  
/// \\

The 4" Mecanum wheels came preassembled. This does not mean they are ready to be used on a robot. Some of the individual rollers were harder to roll than others and all rollers should be disassembled and lubricated with synthetic grease. It is possible that bronze washers could be added to each end of each roller to act as thrust bearings for each roller. Friction lost here directly results in lost speed and power and a dramatic increase in voltage drop due to excessive current draw from the battery.

Mecanum drive benefits from equal traction on the floor from each wheel. Appropriately balancing the mechanisms to result in equal weight distribution helps. Lessening the rigidity of the frame to allow some flex also helps ensure that each wheel presses with equal force/traction and the frame does not unequally transfer weight forces.

The hubs (42 tooth AM half pulleys) used had two bumps that help make sure they are assembled correctly. When they were originally assembled, the students did not know this, so the bumps were pushing the halves apart. In the future, we need to make sure the students know this when they are assembling them.

The gearboxes, 5.95:1 Toughbox Minis, were chosen mainly because they were what AndyMark had in stock. The gearboxes worked fine, the only downside was that they took up a lot of space. In the future,

we should try to use smaller gearboxes. Another option would be to find a way to mount the gearboxes like they are in the 2014-2017 KOP drive base kit. Thoroughly greasing all gears in the gearboxes helps to reduce friction and also to reduce noise generated by gears and washers vibrating around – this takes much more grease than is provided in the KOP. Too much grease will just spin off the gears inside the box and will not cause any harm.

Another problem that we had was with attachment of the CIM motor input gears. The gear needs to be mounted with 3 washers, then the gear and machine key, then another washer, then a retaining ring. This prevents the gear from rubbing on the gearbox housing and prevents the machine key from falling out.

The gearboxes had 39 tooth AM pulleys on the output shaft that were connected to the wheels with 15mm wide, 5mm pitch belts. The 39 tooth pulleys come in two halves, like the 42 tooth pulleys, but with a hub and a spacer. The spacer must be added on the shaft before the pulley and the side the hub was inserted into must face away from the gearbox.

The gearboxes gave the robot more speed than torque. This final gear ratio should be a great starting point as long as friction reduction and other suggestions above are followed.

For whatever reason, the motors were not spinning the correct directions. This caused us to need to create our own copy of the RobotDrive class (RobotDrive2855) so that we could change positive and negative values. The original values:

```
wheelSpeeds[MotorType.kFrontLeft.value] = xIn + yIn + rotation;  
wheelSpeeds[MotorType.kFrontRight.value] = -xIn + yIn - rotation;  
wheelSpeeds[MotorType.kRearLeft.value] = -xIn + yIn + rotation;  
wheelSpeeds[MotorType.kRearRight.value] = xIn + yIn - rotation;
```

were changed to:

```
wheelSpeeds[MotorType.kFrontLeft.value] = xIn - yIn*yIn*yIn + strafe;  
wheelSpeeds[MotorType.kFrontRight.value] = xIn + yIn*yIn*yIn + strafe;  
wheelSpeeds[MotorType.kRearLeft.value] = -xIn + yIn*yIn*yIn + strafe;  
wheelSpeeds[MotorType.kRearRight.value] = -xIn - yIn*yIn*yIn + strafe;
```

With our original setup, each gearbox was driven by a MiniCIM. This was not a good choice. MiniCIMs have the same output RPM as a regular CIM, but with 2/3 of the power. At the end of the Lake Superior regional, we switched two of the MiniCIMs to full CIMs. The other gearboxes were limited to only MiniCIMs because of their proximity to the battery box. This forced us to add a second MiniCIM to each (which were the ones from the other two gearboxes). This caused the back wheels to try to spin around to the front, like if you were trying to drive a rear-wheel drive car in snow or ice. Front location for the double mini CIMs could possibly be desirable. This was later compensated for in programming by multiplying the output of the motor controllers by 0.9.

```
m_frontLeftMotor.set(wheelSpeeds[MotorType.kFrontLeft.value] * maxOutput);  
m_frontRightMotor.set(wheelSpeeds[MotorType.kFrontRight.value] * m_maxOutput);  
m_rearLeftMotor.set(wheelSpeeds[MotorType.kRearLeft.value] * m_maxOutput * 0.9);  
m_rearRightMotor.set(wheelSpeeds[MotorType.kRearRight.value] * m_maxOutput * 0.9);
```

When the motor controllers for the two “new” MiniCIM motors were added into the programming, their outputs were originally set in our copy of the RobotDrive library.

```
m_rearLeftMotor2.set(wheelSpeeds[MotorType.kRearLeft2.value] * m_maxOutput);
m_rearRightMotor2.set(wheelSpeeds[MotorType.kRearRight2.value] * m_maxOutput);
```

One of the Talon SRXs was not getting the signal to run the motor, but was getting the enable and disable signals, and was showing that it was outputting a signal of 0 or within the Talon SRX deadband (4%). This caused one side of the robot to have less power than the other. The code for the new Talon SRXs was changed to set them to follower mode, making them follow the Talon SRX running the other motor on the same gearbox.

In RobotMap:

```
driveleftreartalon2.setControlMode(CANTalon.TalonControlMode.Follower.getValue());
driverightreartalon2.setControlMode(CANTalon.TalonControlMode.Follower.getValue());
```

In RobotDrive2855:

```
m_rearLeftMotor2.set(1);
m_rearRightMotor2.set(6);
```

This did not solve the problem. After consulting the Chief Delphi forum, it was found that the rearLeftMotor2 (CANTalon 1) was trying to follow itself. This was changed to follow the correct Talon SRX.

```
m_rearLeftMotor2.set(4);
m_rearRightMotor2.set(6);
```

This also did not solve the problem. After consultation with three CSAs, we finally found that when the RobotDrive2855 object was initiated in RobotMap, the second left rear Talon SRX was never passed to the object.

```
driveRobotDrive41 = new RobotDrive2855(driveleftfronttalon,
driveleftreartalon, driverightfronttalon, driverightreartalon,
driverightreartalon2, driverightreartalon2);
```

When this was changed to pass the second left rear Talon SRX to the object, it finally worked correctly.

```
driveRobotDrive41 = new RobotDrive2855(driveleftfronttalon,
driveleftreartalon, driverightfronttalon, driverightreartalon,
driveleftreartalon2, driverightreartalon2);
```

Because the class was copied in its entirety to our own version, there were many chances for errors to occur. After the 10,000 Lakes Regional qualification matches were completed, we decided to change the way the library was set up by subclassing the original instead of copying it.

```
public class SixMotorDrive extends RobotDrive{
```

The class then had the main method pull in the four SpeedController variables from the superclass.

```
public SixMotorDrive(SpeedController frontLeftMotor, SpeedController
rearLeftMotor, SpeedController frontRightMotor, SpeedController
rearRightMotor) {
```

```
super(frontLeftMotor, rearLeftMotor, frontRightMotor, rearRightMotor);
```

The CANTalon object is a subclass of the SpeedController class, allowing us to use CANTalon in place of SpeedController.

```
public SixMotorDrive(CANTalon frontLeftMotor, CANTalon rearLeftMotor,
CANTalon frontRightMotor, CANTalon rearRightMotor) {
```

To add the two other CANTalons, we had to add two more objects.

```
private CANTalon rearLeftMotorSlave;
private CANTalon rearRightMotorSlave;

public SixMotorDrive(CANTalon frontLeftMotor, CANTalon rearLeftMotor,
CANTalon frontRightMotor, CANTalon rearRightMotor, CANTalon
rearLeftMotorSlave, CANTalon rearRightMotorSlave) {

this.rearLeftMotorSlave = rearLeftMotorSlave;
this.rearRightMotorSlave = rearRightMotorSlave;
```

This creates the two extra motor controllers and sets the two local objects to the inputted CANTalon objects. The two extra motor controllers are then set to follower mode (which is redundant because it is done in RobotMap).

```
this.rearLeftMotorSlave.setControlMode(CANTalon.TalonControlMode.Follower.getValue());
this.rearRightMotorSlave.setControlMode(CANTalon.TalonControlMode.Follower.getValue());
```

The CANTalons then need to know which CANTalons they are supposed to follow.

```
this.rearLeftMotorSlave.set(rearLeftMotor.getDeviceID());
this.rearRightMotorSlave.set(rearRightMotor.getDeviceID());
```

The getDeviceID() method allows for more versatility because if an ID is changed, the class does not need to be updated, only RobotMap.

Originally, the drivetrain motors would run and randomly stop for less than a quarter second and keep running. The DriverStation was also showing errors about the drivetrain code. We did not realize these were connected until later.

```
ERROR 1 Robot Drive... Output not updated often enough.
java.lang.Thread.run(Thread.java:745)
```

This directed us to the stopMotor() method in the RobotDrive2855 class. We found that the MotorSafetyHelper class, which shuts down the robot motors if they are not updated for a certain period of time (set in RobotMap), was stopping the motors. If the motor safety thinks the motor controllers are not being updated fast enough, the message above is displayed while the motors are stopped. When we disabled motor safety, it worked perfectly.

```
driveRobotDrive41.setSafetyEnabled(false);
```

We then found that the MotorSafetyHelper class' method feed() was not being called from the mecanumDrive\_Cartesian() method. The reason we knew that it was supposed to was because the mecanumDrive\_Polar() method did call the feed() method at the end of each run through the method. This is why turning off the motor safety removed the problem.

```
if (m_safetyHelper != null) {
    m_safetyHelper.feed();
}
```

We added this code to the `mecanumDrive_Cartesian()` method and the problem was solved.

## Intake

Our intake consisted of eight BaneBots 40A T81 wheels on a ThunderHex shaft, run by a 520-5M-15 belt connected to a BaneBots RS-775-18 motor with a BaneBots 4:1 P60 gearbox. The belt was rubbing against a bolt head, but we did not ever run into problems. This was likely because we did not use the intake in competition. There was a ramp behind the wheels that was supposed to direct the balls (fuel) upwards to get over the battery box and drivetrain motors and into our 'hopper'. There was a piece of polycarbonate (a 'backboard') that protected the battery from the fuel. There were two more pieces of polycarbonate that protected the electronics board above the intake rollers from any flying fuel, and attempted to direct the fuel back towards the hopper. The intake had a piece of acrylic behind it to keep fuel from escaping between the motor and intake wheels. The overall substructure was very similar to our climber: three pieces of metal (usually aluminum) in a box shape with the moving part as the fourth side, some form of a brace across the middle parallel to the moving part, and the motor and gearbox mounted on the opposite side of the box from the moving part. This took up a lot of space.

This approach to harvest the inertia of the game piece to transfer the game piece within the robot did not work. The first one or two fuel would be shot up into or over the hopper, but once one fuel didn't make it for whatever reason, that fuel would fall back down and would be hit by other fuel, causing them to not make it, and then filling up the intake. It is desirable to have complete and positive control over the game pieces from start to finish.

At first, the fuel would fill up the intake, but then try to find other ways out that were not up. This caused some fuel to jam on one side between the intake and the backboard near where the photoelectric sensor was located (the sensor was later removed as we never used it). This prompted us to create side shields made of acrylic that would keep the fuel away from the sides and away from the electronics mounted there. This kept fuel from jamming in the sides and with enough fuel ( $\approx 15$ ), they would start to overflow into the hopper.

When two extra MiniCIMs were added to the drivetrain gearboxes on the intake side, the ramp had to be removed. The ramp could have been cut to accommodate for this, but we decided that we did not have the time to do that. The acrylic pieces on the sides had to be cut to accommodate the new MiniCIMs. After the season we then cut into the ramp to accommodate the new MiniCIMs. The MiniCIMs did push on the ramp slightly. In hindsight, ramps such as that are hard to get perfect so the wheels contact the ball throughout its travel. The shooter worked, but that was because there was 1) a rigid mounting system between the wheels and the 'ramp', and 2) it was created in CAD to ensure accuracy.

A better way to design the intake would have been to create one similar to successful intakes used in 2006, 2009 and 2012. Those intakes usually consisted of two sets of rollers with belts, rubber latex tubing or other materials running between them designed to act as 'belts' that directed the balls up to their destination. We thought about doing this between the Lake Superior and 10,00 Lakes regionals, but decided to work on our gear mechanism instead. One of our goals is to remake the intake before the Minnesota Robotics Invitational in October so it is actually useful.

## Climber

Our climber originally was made of three pieces of 1" square tubing spinning around a ThunderHex shaft. The tubing was attached to the shaft with each end having the short part of three AM14U3 L-brackets attached a 500EX hub and the long parts attached to the tubing. The shaft was driven by a 775pro motor with a BaneBots 256:1 P60 gearbox via a chain. The output shaft of the gearbox was connected with a ½" keyed shaft coupler to an output shaft from a Toughbox Mini or CIMple gearbox (not sure which), which then had a hex hub attached to a sprocket that drove the chain. There was a reversible ratcheting wrench on one end to prevent the climber from backdriving. The substructure was similar to our intake: three pieces of metal (usually aluminum) in a box shape with the moving part as the fourth side, some form of a brace across the middle parallel to the moving part, and the motor and gearbox mounted on the opposite side of the box from the moving part. There was a guard on each side of the top of the climber with a vertical piece of aluminum specially designed to fit in the climber, that were supposed to keep the rope from jamming in the sides of the climber and to direct it towards the part that would grab on to the rope. The motor and gearbox were mounted with part of a AM14U3 end plate that was attached to the bottom of the climber. This was done to keep the sprocket from hitting the bottom of the climber, but made replacing the gearbox or motor much harder. (Insert Climber picture here)

The climber was originally supposed to use hooks screwed into rivet nuts to grab a loop in our rope. We eventually decided that the hook-grabbing-loop approach was not fast enough to be competitive in a match. We then removed the hooks and attached Velcro to the tubing with VHB tape. They were added by running the climber and just letting it wind the tape or Velcro onto itself. A new rope was then made out of a tie-down strap with Velcro sewn onto the end of it and a slip-knot in the middle (thank you Bumper Boys with a Z)<sup>1</sup>. The hook part of the Velcro was on the robot and the loop part was on the rope. This made attaching to the rope much easier. During the 10,000 Lakes regional, a large 'donut' of VHB tape was added to the shaft on each side of the Velcro to keep the rope from getting under the protective guard and jamming the climber. Electrical tape and grease was added to keep the VHB tape from sticking to the guards and jamming the climber.

The climber's gearbox broke twice throughout the two regionals. It first broke when we forgot to flip the ratcheting wrench during testing on the Lake Superior practice field. We then replaced it with another. That new one broke partway through the 10,000 Lakes regional because the rope jammed the climber. That one was then replaced. When the gearboxes broke, the gears inside lost some teeth and those were getting in between the other gears. The connection between the two is that if the motor is pushing on the gearbox and the gearbox's shaft is unable to move, the gears inside will break. Later in the 10,000 Lakes regional, our climber stopped working again. But, when we rotated it a little bit, it started working again. We then replaced the PDP breaker and the motor. When the motor was taken apart for analysis, there was no sign of breakage. We believe that the breaker may have been the cause of the problem because when the automatic breakers are tripped multiple times, they start to degrade. When the gearbox had broken twice before, the breaker was tripping multiple times because the motor was trying to use a large amount of amperage to get the gearbox moving.

---

<sup>1</sup> As far as we know, remaking the rope during the evening after the first day of competition was legal. The manual prevents working on the robot during non-pit hours, but as far as we know, does not prevent work on the rope.

The climber was a bit underpowered by using the one 775 motor. An alternative to consider next time would be to siphon power from the drive motors to run a climbing mechanism, as some teams did successfully.

## Gear Mechanism (Original)

Our original gear mechanism was intended to capture gears from the gear delivery station. Also, it was determined early on that it would be nice if the gears could be picked up from the ground. Thus, our original gear mechanism was designed as a combination bucket and dustpan gear manipulator. The original design was basically a rectangular box with various braces and brackets aimed at decreasing the chances that the gears would fall to the floor when falling from the loading station. The intention was that the bucket would go down to the bottom, then the front would swing out. This did not end up working, partly because the bumpers would interfere with the panel that swung out when the robot was on the blue alliance. The ramp was intended to use gravity but didn't really move deliberately enough with gravity to be useful. A small length of latex tubing was added to act as a 'spring', but that did not help much. Also, the ramp did not slide under the gear in practice and tended to jam up. The lesson here is that the best FIRST Robots must be able to pick up all game pieces from the floor. We then added a slit into the mechanism to allow pilots to pull the gears out when we were at the peg.

At the end of the Lake Superior regional, we worked to get some of the angles better so the gears would not fall out as easily. This mechanism was then removed at the 10,000 Lakes regional so we could add our second gear mechanism.

The bucket was made of bent steel. The supports were c-channel. The bucket was moved up and down with polyester cord wound and unwound by a RS-550 motor running a BaneBots 256:1 P60 gearbox controlled by a SPARK. The SPARK was used because we intended to use a hall effect sensor to prevent the motor from pulling the bucket too high and damaging the mechanism. This did not pan out as we did not add a magnet to trigger the hall effect sensor, and then we ended up never moving the bucket.

## Gear Mechanism (Prototype)

The first pneumatic arm prototype had an arm that grabbed the gear with a cylinder that extended towards and away from the robot. This was later changed to one that had a cylinder that extended parallel to the robot's frame.

## Gear Mechanism (New)

The new gear mechanism was intended on improving our ability to pick up gears from the floor and deliver them to the peg in a more deliberate way. We created a drawing of the original frame pieces of the robot (the bag was clear) so that the new design could be easily implemented at our next regional. It was designed to fit into the space of the original bucket gear mechanism. The new design was to be an arm of some sort driven by pneumatics that would come down and grab the gear and come back up with the gear in the position ready to be delivered to the peg. The design utilized one pneumatic cylinder to move the arm up and down around a pivot shaft and another cylinder placed within a 1" by 1" square tube with a sliding feature that fit the profile of the gear (cut out of wood).

Our new gear mechanism was added at the beginning of the 10,000 Lakes regional. The arm was driven by pneumatics which pinched and released the gear and extended and retracted the arm. It was

designed to pick gears up exclusively from the ground. This would allow us to pick up gears dropped by alliance partners at the pegs and potentially steal gears from the opposing alliance's loading station. The latter was finally used in qualification 75 during the last 40 seconds of the match because the other alliance had already achieved four rotors and thus was nowhere near the loading station.

This mechanism was a major improvement over our old mechanism. The moving piece of wood that would grab the gear between teeth gave us a better grip on the gear (more surface area contacting the gear). The other, static, end of the mechanism was made of a plate of aluminum with grip material from the KoP added to give the robot a better grip on the gear on that side. The edge of the metal furthest from the robot was angled out to help guide the gear in.

There was some trouble with the bumpers at the beginning. Similar to what happened with the original gear mechanism, one of the braces would hit the bumper. This was remedied by the Bumper Boys with a Z, who decreased the length of the bumper by 1-2" after confirming with an inspector that this was legal.

## Shooter

The shooter was constructed from plywood, sheet metal, an aluminum plate, an 8mm keyed shaft, a VersaPlanetary gearbox with a 1:1 ratio, encoder and 8mm output shaft, a 8mm keyed shaft coupler, an 8mm ID flanged bearing, a BaneBots 30A T81 wheel with an 8mm hub and two BaneBots 40A T81 wheels with 8mm hubs.

The shooter was designed in CAD. The side plates (made of plywood) were printed 1:1 on paper, then cut out on the line with a band saw. The original prototype was made too large due to a calculation error in the CAD design. This was then remedied in the second prototype. The final product added a place to brace the two sides before the sheet metal was added (removing a significant amount of the human error in the second prototype). The final product was also cut by a mentor to be sure it was cut correctly.

The wheels were arranged in a 40A-30A-40A pattern. The hubs were not able to be placed touching each other because the wheels extend slightly past the hubs. The VersaPlanetary was attached directly to one of the sides of the shooter. The flanged bearing was used on the opposite side of the shooter.

The shooter was never used in a match, mainly because we did not have the time to fine-tune it. At full speed, it ran at just under 3,000 RPM (measured by the SRX Mag Encoder and retrieved via the webdashboard).

If we were to use the shooter, we likely would have used PID. The Talon SRX has a speed mode that uses an encoder input to regulate its speed.

```
ShooterTalon.changeControlMode(TalonControlMode.Speed);
```

This is explained in detail in the Talon SRX software manual (starting page 86).

## Electronics

Our electronics 'board' was spread throughout the robot. The RoboRIO, VRM, RSL, Network Switch and a Talon SR for the intake were above the intake. Two drivetrain Talon SRXs, a Spike for an LED ring and



the OM5P-AC radio were on a side panel on the shooter side of the intake. Two more drivetrain Talon SRXs and an Ethernet Microcontroller intended to run the LEDs were attached to a side panel on the climber side of the intake. The PDP, the Jetson TX1 (until the 10,000 Lakes regional), a Talon SRX for the shooter (attached to the shooter's encoder via a Talon SRX data cable) and a PCM (added at the 10,000 Lakes regional) were on a perforated polycarbonate base plate. The SPARK for the original gear mechanism was attached to an AM14U L-Mount Bracket next to the motor and gearbox for the mechanism until it was removed at the 10,000 Lakes regional. A Spike was added within one of the c-channel mounts for the original gear mechanism to run another LED ring. The Talon SR for the climber was attached to the side of the climber, a drivetrain Talon SRX was attached to the bottom of the climber (added at the end of the Lake Superior regional), a power converter (from the days of D-Links before the VRM) was attached to the top of the climber (added after the end of the season) and a Talon SR for the agitator was attached to the back of the climber. The last drivetrain Talon SRX was added to the bottom of a piece of c-channel on the shooter side of the intake. A pneumatic solenoid was attached to each side of the old gear mechanism's mounting c-channel.

The CAN Bus went from the RoboRIO to the three Talon SRXs on the climber side, then to the three Talon SRXs on the shooter side, then to the shooter Talon SRX, then the PCM and lastly to the PDP.

The robot consisted of a few sensors at different times. There was the SRX Mag Encoder in the VersaPlanetary gearbox for the shooter. At first, a photoelectric sensor was added to the climber side of the intake to sense lines on the ground. This was later removed because we never had time to utilize it. There was a FIRST Choice Ultrasonic Sensor on the shooter side of the intake, which was later removed because it was very difficult to program and we never figured that out. This was briefly replaced with a VEX ultrasonic sensor, but this was removed because we were not using it. There was a hall effect sensor on the gear mechanism that was later removed because we never added a magnet to use it.

The main breaker was originally attached to the side of the original gear mechanism, but was moved to being attached to the ThunderHex shaft and churro shaft supporting the climber. It was then turned around to face inwards because we thought we had hit the switch during a match, though it was a battery cable coming loose that caused the loss of power. It was turned around to prevent the button from being unintentionally pressed in the future.

## LEDs

Our LEDs were supposed to be run by an AndyMark Ethernet Microcontroller. At first, we ran into some problems. When we first switched the robot on, the breaker powering the microcontroller started tripping constantly. We then removed the breaker to pass inspection. We later switched it to a higher amperage breaker (NOT a good idea), and then the 18 AWG wire that split the power between the power converter and the microcontroller started to smoke. We then completely removed the power wires from the microcontroller and power converter and focused on other parts of the robot. After the Lake Superior regional, research found that the wire colors for the LED connectors were not arranged the same as the LED strips. This could have been our problem. At the end of the 10,000 Lakes regional, we started to switch some of those wires, but then decided to go practice. After the season was over, we removed all of the LED parts to diagnose the problem.

When diagnosing the problem, we decided to create a small custom program for testing (it would turn the first 29 LEDs to full green). The microcontroller was powered by a computer via the USB to Serial

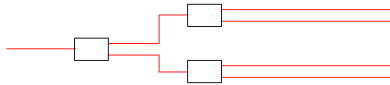
cable and one LED connector was connected to the 5V pin, a Ground pin and a DIO pin. When each LED strand was tested with the microcontroller, all but one of the LEDs worked perfectly. When the one LED was removed, the rest of the strand worked perfectly.

We then wired the microcontroller into Sparky (with a 20A breaker). The microcontroller turned on as expected. We then wired the power converter in to a different WAGO port with a separate breaker (20A), rather than pulling from the same port as before. The LED connector was connected to the power converter and to the same DIO port on the microcontroller. The 5V ground from the converter was also connected to the ground on the DIO row on the microcontroller, as recommended by the online guide. When the LED was connected, it flashed and then went out. All subsequent LEDs would not turn on.

When the power converter was tested, the input read 12V, as expected, but the output read 0V. After the power was cut and the breaker was removed, the continuity was tested, and the ground read positive, but the power gave no reading (no continuity). Obviously, something had burnt out either when we connected the LEDs incorrectly originally or when the first LED was connected during testing.

We then got one of our old power converters out of our cRIO control system bin in our storage closet. When it was wired in, we decided to not connect the 5V ground to the microcontroller. When a LED was connected, it shone bright green as it was supposed to. This then proved we had a circuit that worked.

The LEDs were connected to the Heart of the Beast again, but this time PWM cables were used to connect to the LEDs from the microcontroller and power converter because the wires would then stay together. The power and ground were connected to the four sets of LEDs with the following setup:



*□ are terminal blocks - Diagram 1*

When the LEDs were all connected and the previous code was loaded on the microcontroller, the power was switched on and the LEDs worked as expected. We then connected PWM cables to DIO ports 1-4 on the RoboRIO and connected the signal to DIO ports on the microcontroller that were set for input. We cut the power and ground cables on the microcontroller side of the wire to prevent the microcontroller port from accidentally getting 5V instead of a signal. (Note: Do Not use these as PWM cables in the future!) These were used to signal the microcontroller of different states such as if the shooter or intake were running, and then triggering those animations and stopping when the signal is low again. We found that the ground from the DIO ports on the RoboRIO do not need to be connected to the microcontroller's ground for them to work properly.

Also, for whatever reason, when the robot is starting up, some of the LED sections will show one of the animations randomly, and then stop after ≈30 sec. We don't know if this is because of the microcontroller, or if the RoboRIO is sometimes setting one or two of the DIO ports to high during startup.

## Pneumatics

The pneumatic system was added to the robot at the beginning of the 10,000 Lakes regional. A large plastic tank was added next to the climber on the gear side, one was attached inside the climber, and a small metal tank was added to the shooter side of the intake. The pressure switch was added next to the PCM on the bottom electronics board. The compressor was attached to c-channel beneath the climber. The emergency relief valve was attached to one side of a t-fitting on the compressor, with the other end containing a tube connector. This tube then split into two tubes, one that went to the pressure switch and ended there, and one that went to the rest of the system. The tube then split again to one that went to the storage-side pressure gauge and the manual pressure relief valves. One of the manual valves had a silencer on it to lower the sound of the air escaping the system. The other manual release was added to confirm that all of the air made it out of the system in a timely manner. The other branch of the tube went to the tank next to the climber, then the tank in the climber, then all the way around to the small metal tank. This then went to the working-side pressure gauge and main regulator. This then went to another t-fitting that split the tube into two, one to each solenoid. Those solenoids then had two outputs, one going to each side of the cylinder they controlled.

The pneumatic system worked as it was supposed to. There was only one problem when one of the wires for the solenoid controlling the arm movement cylinder was pulled out of the PCM. This was luckily caught during a full-system-check when we were in queue. When the main breaker was turned around, the battery cables put the solenoid cables under extra strain. This strain was removed after our last match. Even though the Festo solenoid was a double solenoid and used two PCM ports, the solenoid needs all four wires plugged in to operate correctly.

Meter-out flow control valves were used on the arm cylinder to control the speed it extended and retracted to prevent it from breaking itself by smashing into the floor.

## Final Notes

- Make sure that the drivetrain has as little friction as possible
- MiniCIMs are not a good choice for a drivetrain when used at a 1:1 ratio with each wheel
- Check your code carefully with multiple people
- Don't rely on gravity or momentum to get game elements or mechanisms where they need to be
- Be ready to replace gearboxes that will be experiencing high-torque loads
- Ensure your battery connection is completely secure (and preferably soldered)
- Check that wire colors correspond with what they are connecting to when using connectors and components from different suppliers and manufacturers
- Full system checks can save a match even before it starts