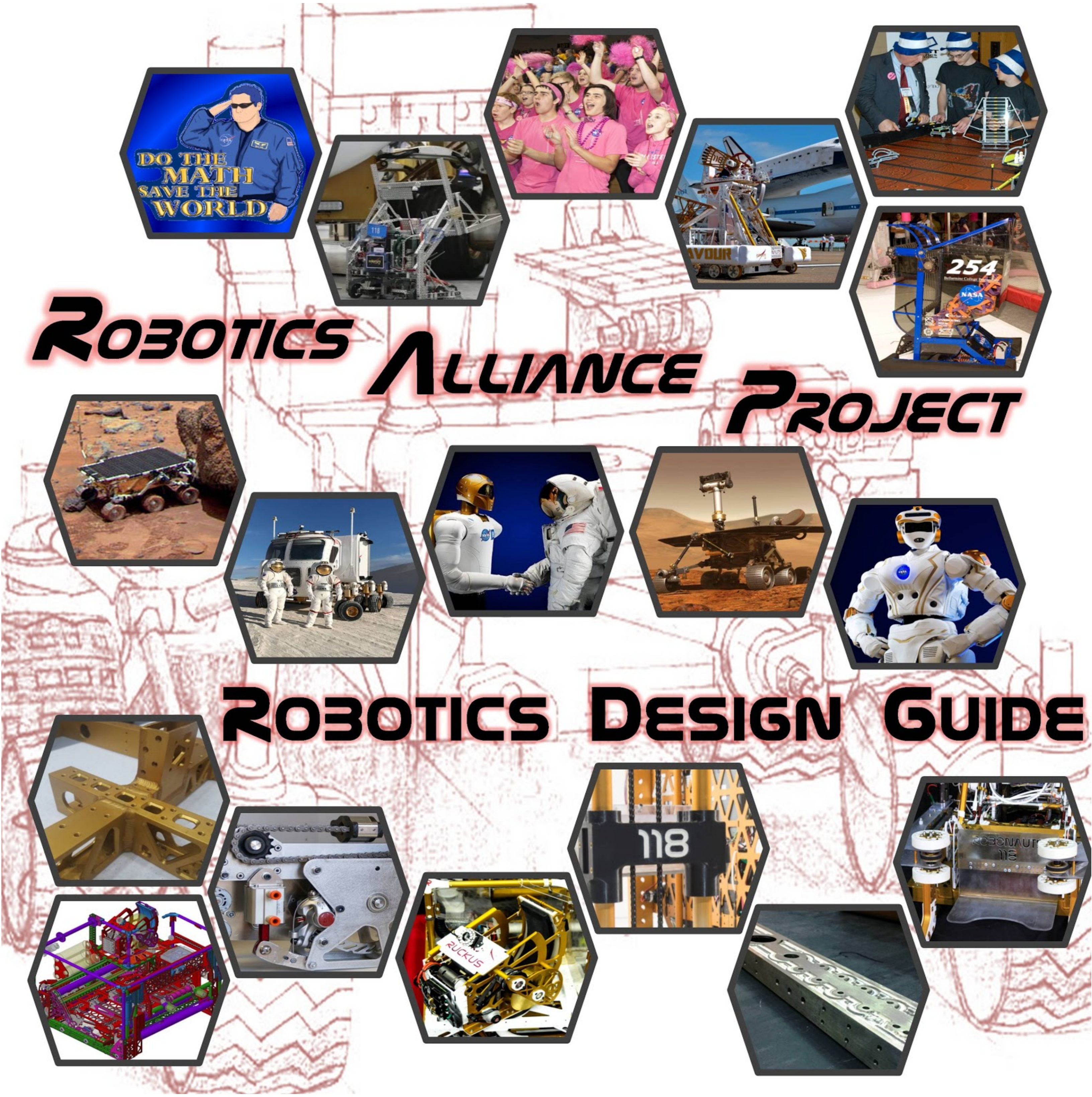
National Aeronautics and Space Administration
Robotics Alliance Project -- Robotics Design Guide

Robotics Alliance Project

Robotics Design Guide



www.robotics.nasa.gov

Version: **1.01**

The NASA Robotics Alliance Project (RAP) Robotics Design Guide was created by robotics teams and intended for use for teams that compete in robotics competitions, such as FIRST, VEX, BEST, and Botball. The NASA RAP team would like to thank those in the robotics community who have contributed to this design guide.

The NASA RAP team also encourages others to contribute to this living document. Please send any content you would like to be added to the design guide to robotics@nasa.gov, where they will be reviewed and potentially integrated into the next release of this design guide.

The latest version of this guide can always be found on the Robotics Alliance Project website at https://robotics.nasa.gov.

Image Credit: Kristine Atiyeh

The RAP Design Guide is compliant with NASA Export Control Policy Directive, NPD 2190.1B.

Table of Contents

1. Introduction **8**

1.1 Things to Consider 8

2. Manufacturing and Assembly **10**

2.1 Manufacturing Methods 10

2.2 Tolerancing 18

2.3 Fasteners 19

2.3.1 Bolts 19

2.3.2 Bolt Sizes and Threads 20

2.3.3 Nuts 22

2.3.4 Tapping and Threading 23

2.3.5 Bolt Retention 24

2.3.6 Rivets 26

2.3.7 Welding 28

2.3.8 Miscellaneous Fasteners 30

3. Design Styles **31**

3.1 Box Tube Construction 32

3.1.1 Designing for Gussets 32

3.1.2 Designing for Welding 34

3.1.3 Designing for 3D Blocks 35

3.2 Round Tube Construction 37

3.2.1 Clamping 38

3.2.2 Welding 40

3.2.3 Tube Inserts 41

3.2.4 Riveting 42

3.2.5 Rod Ends 43

3.3 Plate and Standoff Construction 44

3.3.1 Standoffs vs. Spacers 45

3.4 Sheet Metal 46

3.4.1 Recommendations for Sheet Metal 46

3.4.2 What Happens When a Part is Bent 48

3.5 Slots and Tabs 48

3.6 3D-Printing 50

3.7 Weight Savings 53

4. Power Transmission **57**

4.1 Motors 57
4.1.1 Simplified Motor Selection 60
4.1.2 Ratio Selection 61
4.1.3 Planetary Gearboxes 63
4.2 Servos 64
4.3 Bearings 66
4.3.1 Radial Ball Bearings 67
4.3.2 Thrust Bearings 67
4.3.3 Linear Bearings 69
4.3.4 One-Way Bearings 70
4.4 Bushings 71
4.5 Shafts 72
4.5.1 Live Axle vs. Dead Axle 72
4.5.2 Shaft Profiles 74
4.6 Methods for Torque Transfer 76
4.6.1 Gears 76
4.6.2 Chain 83
4.6.3 Belts 87
4.6.4 Polycord 90
4.6.5 Flat Urethane Belting 91
4.7 Motor Driven Linear Actuators 91
4.7.1 Lead Screws 93
4.7.2 Ball Screw 95
4.7.3 Rack and Pinion 96
4.7.4 Linear Elevator 97

5. Mechanism Design **98**
5.1 Drivetrains 98
5.1.1 Drivetrain Terms 98
5.1.2 Wheels 100
5.1.3 Types of Drivetrains 101
5.1.4 Drivetrain Structure 114
5.1.5 Designing a WCD Style Drivetrain 117
5.2 Elevators 122
5.2.1 Elevator Stages 123
5.2.2 Elevator Rigging 124
5.2.3 Elevator Rigging Material 126
5.2.4 Elevator Structure 127

5.2.5 Elevator Counterbalance 129
5.3 Arms 130
5.3.1 Types of Arms 130
5.3.2 Arm Design Recommendations 132
5.3.3 Counterbalancing Arms 134
5.4 Linkages 136
5.4.1 Types of Linkages 136
5.5 Intakes 140
5.5.1 Intake Architectures 140
Pincher intake on Team 5414’s 2018 robot. 145
5.5.2 Types of Intake Wheels 146
5.5.3 Intake Ratio Selection 148
5.6 Shooters 149
5.6.1 Flywheel Shooters 149
5.6.2 Catapult Shooters 154
5.6.3 Linear Punchers 157
5.7 Turrets 158
5.7.1 Turret Bearing Types 159
5.7.2 Turret Drive Types 162
5.8 Bumpers 164
5.8.1 Bumper Shape 164
5.8.2 Bumper Height 166
5.8.3 Pool Noodles 167
5.8.4 Fabric 169

6. Electronics **170**
6.1 FRC Electronics System 170
6.2 Sensors 172
6.3 Cameras 175
6.4 Communication Method 176
6.5 Motor Controllers 177
6.6 Pneumatics 179
6.6.1 Pneumatic Components and Terms 179
6.6.2 Pneumatic System Tips 181
6.6.3 Pneumatic Force Calculations 182
6.7 Electrical Connectors 182
6.7.1 High Power Connections 182
6.7.2 Medium Power Connections 183

6.7.3 Low Power Connections 184
6.8 Insulation 185
6.9 Cable Management 185

7. Prototyping **187**
7.1 Prototyping Resources 187
7.2 Prototyping Examples 191

8. CAD **199**
8.1 CAD Software Packages 199
8.2 Independent File Management Programs 200
8.3 File Naming Scheme 201
8.4 Creating DXF's (PTC Creo/Pro Engineer) 202
8.5 Designing Sheet Metal Parts (PTC Creo/Pro Engineer) 206
8.6 Setting the K-Factor of a Sheet Metal Part (PTC Creo/Pro Engineer 209
8.7 The Progression of CAD 210

9. Software **216**
9.1 Getting Started 216
9.1.1 Selecting a Language 216
9.1.2 Learn About Your Language and Related Resources 218
9.1.3 Develop a Style Guideline 220
9.1.3.1 File Management 220
9.1.3.2 Code Formatting 221
9.1.3.3 Commenting the Code 222
9.1.3.4 Managing Variables 224
9.1.3.5 Class Management 225
9.1.4 Select a Robot Framework 225
9.1.5 Plan Your Code 227
9.2 Configuring Hardware 227
9.2.1 Robot Radio 227
9.2.2 RoboRIO 230
9.2.3 Other Hardware 231
9.3 Development Tools 234
9.3.1 Git and Version Management 234
9.3.1.1 Setting up a Repository 234
9.3.1.2 Branches for Version Control 235
9.3.1.3 Planning with Issues and Milestones 235
9.3.2 Putty 236

9.3.3 Winscp 236
9.3.4 Integrated Development Environment 236
9.4 Safety 238
9.5 Driving and Troubleshooting Tools 240
9.5.1 Drivers' Station 240
9.5.2 Telemetry and Driver Feedback 249
9.5.3 Dashboard Guide 251
9.5.4 Controller Mapping 252
9.6 Robot Commanding Schemes 253
9.6.1 Tank Drive vs Arcade Drive 253
9.6.2 Swerve Drive 254
9.6.3 Presets vs Manual Control 255
9.7 Autonomous and Preset Pathing 255
9.7.1 PID Control System 255
9.7.2 Motion Profiling 260
9.8 Common Subsystems 263
9.8.1 Drivetrain 263
9.8.2 Elevator or Lift 264
9.8.3 Collector/Intake 264
9.8.4 Shooter 264
9.8.5 Climber 265
9.9 Networking 265
9.9.1 NetworkTables 265
9.9.2 IP Configuration 266
9.9.3 Port Forwarding 271
9.9.4 SSH 272
9.10 Common Problems and Troubleshooting 273

Appendix A - Acronyms and Abbreviations 276

Appendix B - Materials 279
B1 - Material Profiles 279
B2 - Metals 282
B3 - Plastics 284
B4 - 3D Printer Filaments 285
B5 - Composites 286
B6 - Rubbers 288

Appendix C - Hole Size Charts 292

C1 - Bolt Tap and Drill Chart 292
C2 - Rivet Size Chart 293

Appendix D - Additional Design Resources 294

Appendix E - Additional General FRC Resources 296

1. Introduction

1.1 Things to Consider

KISS (Keep It Simple & Straightforward): this is something you should keep in mind when designing robot systems. Simplicity is huge when it comes to manufacturing, assembling, and maintaining all of a robot's systems. Simplicity also makes it easier to create software to operate your robot. It is important to note that simplicity is relative and something that may be simple for your team would be complex for other teams. When referencing this guide during your design, it is likely best to pick the mechanism and construction method that is simplest and quickest for your team to implement despite the fact that it may be heavier or not entirely optimized.

Low Center-of-Gravity (CG): building a robot with a low CG makes your robot easier to drive and less susceptible to tipping over in matches. Creating a low CG can be achieved by locating heavy components, such as your battery, as close to the bottom of the robot as feasible and making components that are lifted high, such as a "grabber" on the end of an arm, as light as feasible.

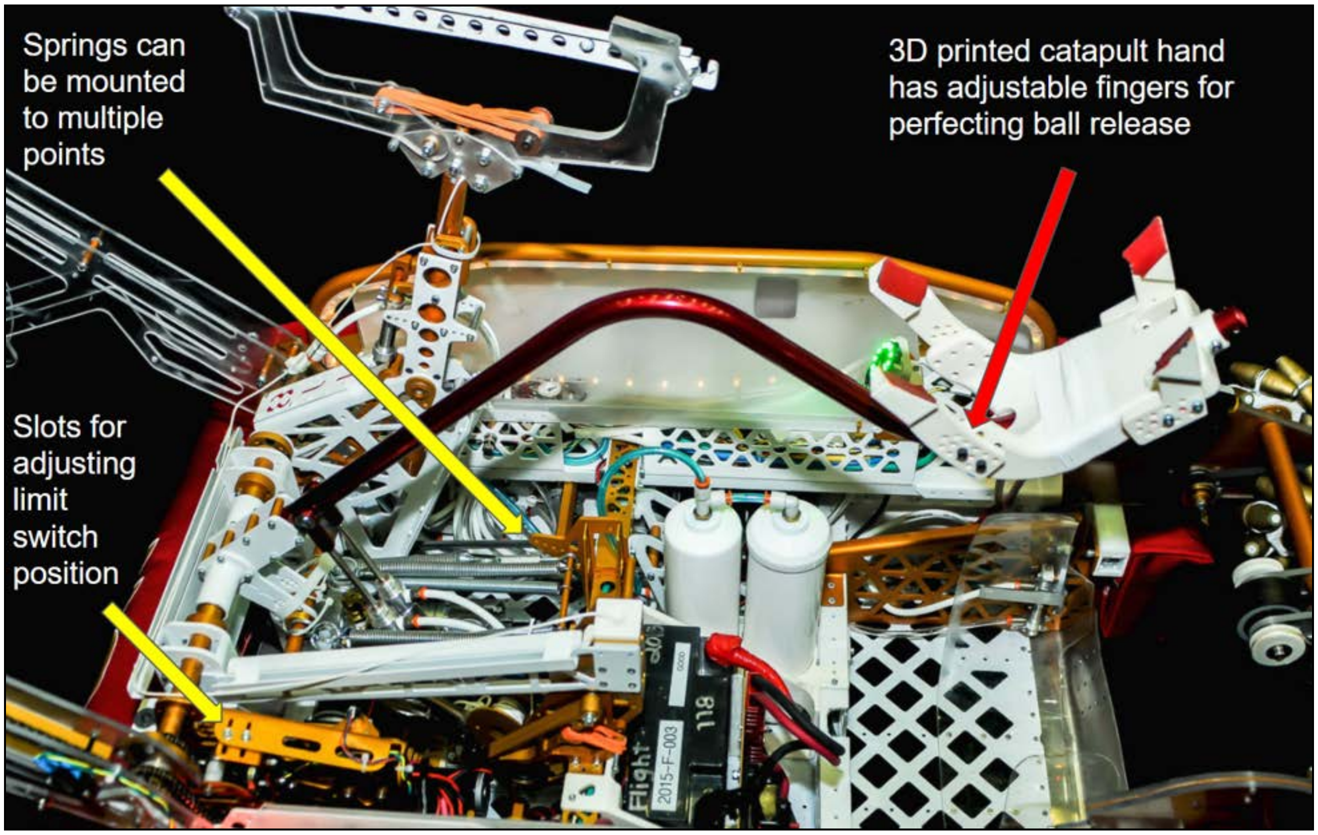
Touch It, Own It: an intake philosophy in which you strive to own and control the game piece as soon as you touch it with your intake mechanism.

Rollers are (Almost) Always the Answer: almost every game piece FIRST has utilized can be very effectively acquired with a roller intake. You can read more about intakes in section 5.5.

Drivetrain is King: without a reliable, efficient drivetrain you cannot effectively play the rest of the game. The best intake in the world is worthless if you do not have a drivetrain that can reliably get you to the game piece. You can read more about drivetrains in section 5.1.

Autonomous (Auton) Domination: in most modern FIRST games if you win auton, you are much more likely to win the match. This is something to think about when designing your robot. Some teams design their entire robot around their auton routine.

Design for Adjustability: designing mechanisms that can easily be adjusted can save you a lot of headache in the long run. This is especially true on mechanisms where accuracy is critical, such as on a shooter. The image below shows examples of design for adjustability on Team 118's 2016 robot.


Springs can be mounted to multiple points
3D printed catapult hand has adjustable fingers for perfecting ball release
Slots for adjusting limit switch position

2. Manufacturing and Assembly

2.1 Manufacturing Methods

3D Printer: a 3D printer creates parts printing a part in many layers. There are many different types of 3D printers which all use different methods for printing their layers, the most common examples are listed below. Since the printer adds material to create a part, it is the classic example of "additive manufacturing". 3D printing can be used to create parts that have complex geometry that would be impossible to create on a mill or other machine. Typical parts that are 3D printed are custom brackets for cameras and electronics, linear motion trucks, and pulleys for polycord and timing belts.

Types of 3D Printers:

- **FDM** (Fused Deposition Modeling): a 3D printing method in which a thermoplastic filament is melted and extruded through a nozzle to print each layer. This is the most common type of 3D printer.
- **SLA** (Stereolithography): a 3D printing method that uses a laser to fuse layers of a resin together. This is often used to create cosmetic parts that have very fine layer thicknesses.
- **SLS** (Selective Laser Sintering): a 3D printing method that uses a laser to sinter powdered plastic, usually nylon to create parts.
- **DMLS** (Direct Metal Laser Sintering): a 3D printing method in which metallic powder is sintered together to form 3D parts. This is the most common method used to create metal 3D printed parts.

Markforged Mark Two.

CNC Router: A CNC router is one of the most versatile and useful machines that an FRC team could have. Most routers have 3 axes which allow them to cut in 2D like a water jet, laser cutter, or turret punch, but can also cut partially through a part. Routers use drills and end mills to cut parts and require tool changes when creating parts with different sized holes. Since the router cuts with a round tool (end mill), interior corners of a part must have a radius and cannot be sharp, 90 degree corners. For example, to cut a part with a ¼" end mill, its interior corners must have a radius of at least an ⅛". However making the radius slightly larger than the absolute minimum for the tool will allow for the part to be machined faster due to the machine not having to stop at the corner and make a sharp 90 degree turn. This also decreases vibrations leading to a cleaner looking part. Due to the speeds at which a router spindle runs, a single flute endmill is required to get clean cuts. More information on CNC routers for FRC can be found here.

Velox CNC Router.

Laser Cutter: a CNC laser cutter uses a high powered laser to cut a variety of materials. Most hobby grade, low power laser cutters can be used to cut thin wood, cardboard, delrin sheet, and acrylic. Proper ventilation is important for cutting materials with a laser cutter. This can be done by venting fumes outside. If that's not an option, there are internal HEPA filters available, but they are expensive and need to be monitored for replacement. Some recommendations regarding materials on the laser cutter can be found here. Laser cutters are very useful for rapid prototyping but can also be used to manufacture competition robot parts. It is important to understand the tolerance on your laser cutter. Unlike water jets, most laser cutters cut on the line instead of inside/outside of it. This can make interfaces between parts sloppy if not designed with this tolerance in mind. Some high end laser cutters are able to cut thin aluminum sheet material as well, however these machines are very expensive compared to smaller machines.

Thunder Laser NOVA.

Water Jet: a water jet is typically a 2 axis machine however there are machines with up to 5 axes. A water jet can cut basically any material you can fit in it, usually up to several inches thick. Most water jets can cut internal radii of 0.015", but creates a slight taper which shows up more prominently on thicker parts. This taper leads to holes coming out slightly smaller than they were modeled. To get precise holes, such as for bearings, use a reamer to remove the taper. A drill bit can be used for smaller holes, such as for bolts or rivets. Water jets can also be used to cut tubing by inserting an absorbing material into the center of the tube.

Omax 2652 Water jet.

Mill: a mill is similar to a router, but is much more rigid and precise. They can also machine parts in 3 dimensions and some mills incorporate more axes to machine more complex parts. Mills can use all of the same tools as the router as well as more specialized tools. Mills are typically used to machine boxtube and complex 3D parts as its setup time is significantly longer than a waterjet or router. A 3 axis mill can machine parts on 3 orthogonal planes in one setup. To machine parts with features on more than 3 axes, another setup is required.

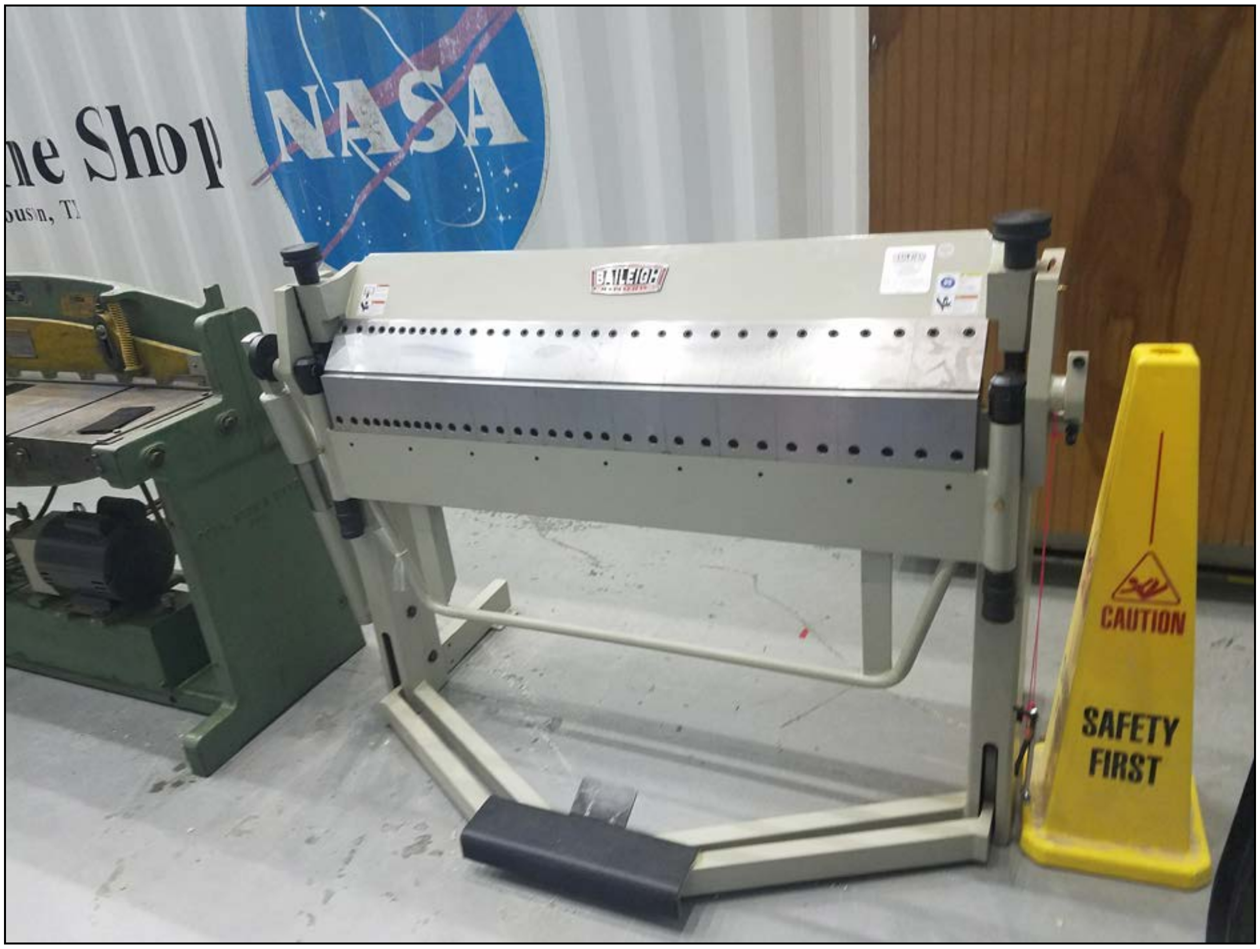
JET knee mill that is a Bridgeport style mill.

Haas VF2 CNC Mill.

Lathe: a lathe cuts round, hex, and sometimes rectangular parts by spinning the part and keeping the cutting tool stationary, unlike most other forms of machining. Lathes are typically used to create shafts, rollers, spacers, and plugs for tubes. You can find more information on lathes for FRC here.

South Bend 18” Lathe with Digital Readout.

Manual Sheet Metal Brake: a manual sheet metal brake can be used to put simple bends in sheet metal and polycarbonate parts that only have critical dimensions on one face of the part such as simple 90 degree bends for mounting parts or strengthening bends on the part. With an experienced operator a manual sheet metal brake can be used to make sheet metal parts with a high level of precision. The brake shown below does not come with an adjustable backstop but one can be made to allow for greater repeatability.



Baileigh Manual Sheet Metal Brake.

CNC Sheet Metal Brake: a CNC brake can be used for creating very complex sheet metal parts with reasonably tight tolerances. Typically flat parts are cut on a router, waterjet, or laser cutter and then bent using a sheet metal brake.

Strippit LVD PPEB 120 CNC Sheet Metal Brake.

2.2 Tolerancing

Proper **tolerancing** is critical to your parts fitting together correctly. If you are designing a part to fit a 1/4” dowel pin and dimension the hole to exactly 1/4” and cut it on a waterjet or laser cutter, the pin likely won’t fit properly. Most waterjets cut with a taper, making the hole too small; most laser cutters cut on the line, making the hole too big. Parts cut on a traditional CNC machine, such as a mill or a lathe, will hold tolerances much better, depending on the settings with which the part is machined.

For bolt or rivet holes, the proper size drill bit can be used to enlarge the hole to fit the desired bolt or rivet. For bearing and pin holes, reamers should be used to create a perfect sized hole.

For non-standard features such as slots and tabs, a tolerance can be designed into the part. For example, increasing the size of the slots by 0.005” and/or decreasing the size of the tab by 0.005.”

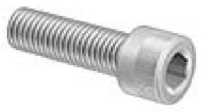
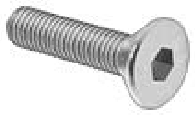
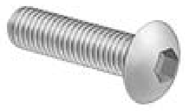
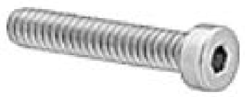
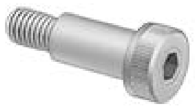
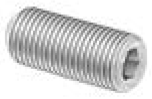
It is also important to measure the thickness of the material out of which you are cutting a part. For example 1/8” polycarbonate sheet is almost always manufactured to be 0.118” thick. So if you design a part in which an 1/8” nominal part is inserted into a slot that is 0.125” wide, the fit may be sloppy.

2.3 Fasteners

Fasteners: are the hardware used to attach parts of the robot together. There is a large variety of fasteners with many different applications. It is recommended that teams standardize on fasteners to stock as few different sizes as possible. The weight saved with smaller bolts such as #4 or #6 is usually not worth the added complexity of stocking these smaller sizes and tools for them. The VEXpro VersaFrame is sized so that its holes can be used with 5/32” rivets and fit #8 fasteners, or be drilled out or tapped for larger fasteners. Many teams standardize around this ecosystem.

2.3.1 Bolts

Types of Bolts

Screw Type	Description	Photo
Socket/Cap Head Bolt	Standard go-to bolt, the strongest of all of the socket head, the hex is the hardest socket head to strip out.	
Flat Head/ Countersunk Bolt	Is meant to be flush with the material, requires you to countersink the hole. Most standard bolts use an 82° countersink. Most metric bolts use a 90° countersink.	
Button Head Bolt	A bolt with a rounded head for areas where a cap head bolt is too large or you don’t want the bolt head to catch on anything.	
Low Profile Bolt	A bolt with a head that isn’t as tall for use in areas where a cap head bolt is too large.	
Shoulder Bolt	Bolt with a precision smooth section meant to fit into a bearing or bushing and act as a shaft.	
Set Screw/ Grub Screw	Screw used to keep something positioned on a shaft or to transmit torque. It is best to avoid these. If they are used, they should only be used to hold axial position on a shaft, not to transmit torque.	

2.3.2 Bolt Sizes and Threads

Standard Bolts

We recommend using some sort of color-coded system for organizing your hardware and tools. A color code for all of your hardware and related tools makes finding the correct tools when working on your robot in the pit much easier.

Team 118 uses the color code below, but feel free to create your own. The chart below does not include the allen wrench sizes for set screws.

Bolt Size	Color	Allen	Allen (flat/but)	Torx (flat/but)	Wrench/Socket	Clear Drill	Tap Drill
2-56	Brown	5/64"	.05"	T6	1/4"	41	50
4-40	Orange	3/32"	1/16 "	T8 T10 (flt blk ox)	1/4"	30	43
6-32	Blue	7/64"	5/64"	T10 T15 (flt blk ox)	5/16"	25	36
8-32	White	9/64"	3/32"	T15 T20 (flt blk ox)	11/32"	16	29
10-32	Green	5/32"	1/8"	T25 T20 (flt zinc)	3/8"	7	21
1/4-20	Yellow	3/16"	5/32"	T27 T30 (flt blk ox)	7/16"	H	7

Metric Bolts

Most teams in the U.S. do not use metric bolts for their robots; however, many FRC motors are mounted using metric bolts.

Bolt Size	Usage	Tool
M3	Mounting 550 motors	2.5 mm Allen Wrench
M4	Used for attaching 775pros and BAG motors.	3 mm Allen Wrench
M5	Used for attaching snowblower motors.	4 mm Allen Wrench
M6	Power Distribution Panel Battery lug mount (10 mm length)	5 mm Allen Wrench

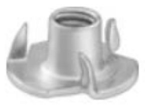
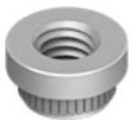
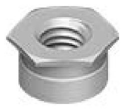
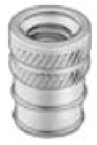
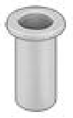
We recommend purchasing metric bolts in a different color than your standard bolts so they can easily be identified as metric. McMaster-Carr sells metric bolts with a blue-tinted finish.

Team 118 uses the color code below, but feel free to create your own.

Bolt Size (mm)	Color	Allen (mm)	Allen (flat/but) (mm)	Torx (flat/but)	Wrench/ Socket (mm)	Clear Drill (mm)	Tap Drill (mm)
M1.6	Blue	1.5	----	T5	----	1.8	1.5
M2			----	T6	----	2.4	1.6
M2.5	Pink	2	1.5	T8	----	2.9	2
M2.6			----	----	----	----	----
M3	Purple	2.5	2	T10	5.5	3.4	2.5
M3.5			----	----	----	3.9	2.9
M4	Green	3	2.5	T20	----	4.5	3.3
----	Yellow	3.5	----	----	----	----	----
M5	Orange	4	3	T25	8	5.5	4.2
----	Pink	4.5	----	----	----	----	----
M6	Blue	5	4	T30	10	6.6	5
----	Purple	5.5	----	----	----	----	----
M7	Green	6	----	----	----	----	----
M8			5	----	13	9	6.8
----	Pink	7	----	----	----	----	----
M10	Yellow	8	6	----	17	11	8.5
----	Orange	9	----	----	----	----	----
M12	Blue	10	8	----	19	13.5	10.2

2.3.3 Nuts

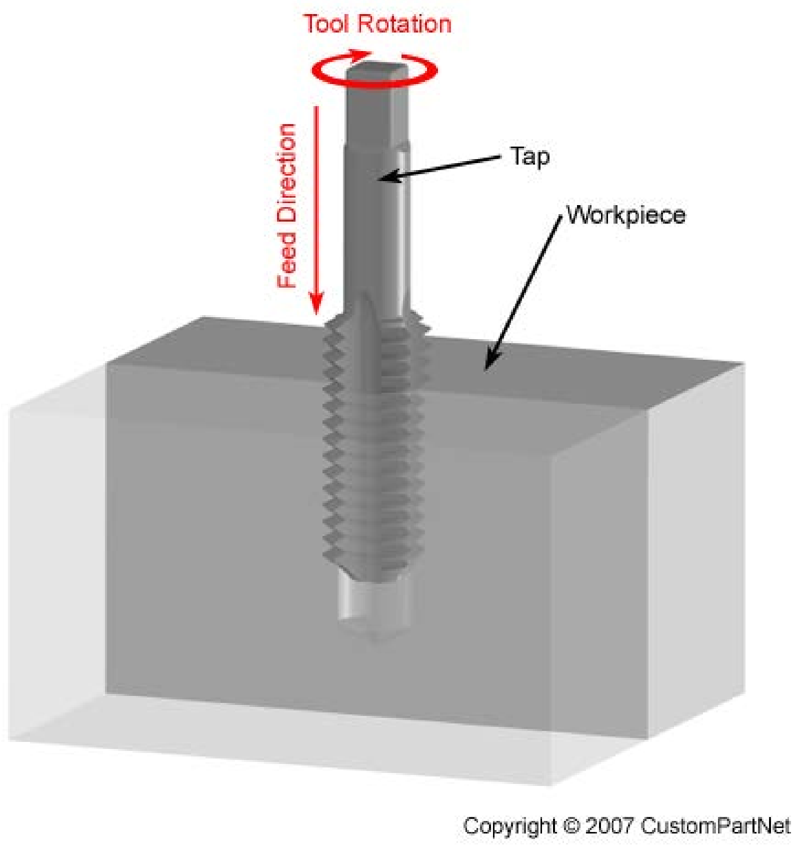
Nuts: nuts thread onto bolts and are used to attach parts. They are an alternative to threading bolts into tapped parts.

Nut	Description	Photo
Lock Nut	Standard go-to nut. Has a nylon insert that prevents the bolt from loosening.	
Thin Lock Nut	Used when a standard lock nut is too large.	
Regular Nut	A normal nut, typically only used when creating a "jam nut" where two nuts are tightened against each other on a threaded rod.	
Tee Nut	A type of nut pressed into wood. Sometimes used in bumpers.	
Wing Nut	A type of nut meant to be tightened by hand. Convenient for removing without the use of tools. Can be useful for attaching bumpers.	
Pem Nut	A nut that is pressed into a part, usually in a part that is too thin to tap well. It's important to check McMaster for what size the holes the part needs to be for the desired pem nut.	
Flush Pem Nut	A pem nut that is used in sheet metal and mounts flush with the material.	
Heat Set Inserts	Used in 3D-printed or other plastic parts. A hole is designed for a specific size listed on McMaster, and then the threaded brass insert is pressed in using a soldering iron.	
Rivnuts	Similar to a pem nut, except it is installed with a specialty tool, similar to a rivet gun. Can be used as an alternative to tapping a hole. Rivnuts are useful for hard-to-reach locations (i.e. box tubing or thin sheet). Rivnuts can be used to provide a semi-permanent nut in a location.	

2.3.4 Tapping and Threading

Taps and **dies** are used to cut or form screw threads, which is known as **threading**.

Taps: taps allow you to put threads into a part so that you don't need a nut (forming the female part of a mating pair). When tapping metals a cutting fluid such as TAP Magic should be used to lubricate the tool and prevent the tap from breaking. Usually it is best to not tap parts less than 1/16" thick. Strive to get at least 4 threads into the material. For 32 pitch screws, that means the minimum thickness is 1/8" (4/32). Typically .090" (3/32) parts can be tapped to 4-40, and for very light loads, 1/16" can be tapped to 4-40. The more load the bolts need to support, the thicker the part that you are tapping needs to be. If you have a 1/16" part that needs to support a large load, you are better off using a bolt and nut or a pem nut.



Using a tap to thread a bolt hole into a part.
(Photo from https://www.custompartnet.com/glossary/t)

Dies: dies allow you to put threads on the outside of a part and essentially make your own bolt (forming the male part of a mating pair). This isn't done very often in FRC. Threads can also be created on a lathe.

Using a die (held by a die wrench) to cut threads into a rod.

(Photo from https://www.sharpen-up.com/use-tap-die-set-right-way/)

2.3.5 Bolt Retention

Bolt Retention: Retaining the bolts on your robot is critical in FRC. You do not want your robot falling apart in the middle of a match so it is best to use some form of bolt retention on all of your bolts. The most common methods by far in FRC are using Loctite or using a nylon lock nut. However other methods that you may find useful are also listed below.

Loctite: loctite is a thread locker that helps prevent bolts from coming loose from vibrations. Blue 242 Loctite is useful for bolts sized #8 and smaller; Red 262 Loctite is useful for bolts #10 and larger without needing to apply heat to remove the bolt. It is possible to use Red 262 Loctite on smaller hardware, but care must be taken not to strip out or break the bolt. Bolts with stronger Loctite compounds can be removed by heating them up with a heat gun. There is also Green Loctite 648 which is a retaining compound that can be used to retain bearings in holes and can also be used to retain components on shafts. It is important to be careful what materials with which you use Loctite. Specifically polycarbonate will crack if the commonly available Blue 242 or Red 262 Loctite touches it. Keep in mind the hubs of the wheels on the Kit of Parts drivetrain are polycarbonate. Loctite will also ruin the nylon of a nylon lock nut, so do not use them together. Loctite 425 is a plastic safe Loctite product that can be used with polycarbonate or other plastics.

Nylon Lock Nuts: lock nuts that have a nylon section the bolt threads into that prevents the bolt from backing out from vibrations. Opinions vary on whether you can reuse a nylon lock nut, but for FRC it is typically fine to reuse them. Do not use nylon lock nuts in conjunction with Loctite.

Locking Heli-coils (Screw Thread Insert): locking Heli-coils can be used to retain bolts and to repair damaged threads. A hole is tapped with a special Heli-coil tap and then using a Heli-coil Insert Tool the insert is threaded into the tapped hole. Most Heli-coil inserts are locking by default due to the shape of the threads. They are great for creating robust threads in soft material such as aluminum or plastic.

Distorted Thread Nuts: these nuts work like a nylon lock nut but are all metal, allowing them to be used in high temperature areas. These usually are not necessary in FRC, and most are one-time use.

Interfering Thread Nut and Tapered Thread Nut: these nuts have an undersized root diameter that plastically deforms the threads on the bolt. This form of nut is usually only used on permanent fixtures, and is usually unnecessary for FRC.

Safety Wire/Lock Wire: a form of a positive locking device that prevents bolts from backing out due to vibration. Safety wire is typically used in the aviation industry in which wire is used to connect two or more bolts together making it impossible for them to back out. This is usually over complicated and unnecessary for FRC. A video on how to use lockwire can be found here.

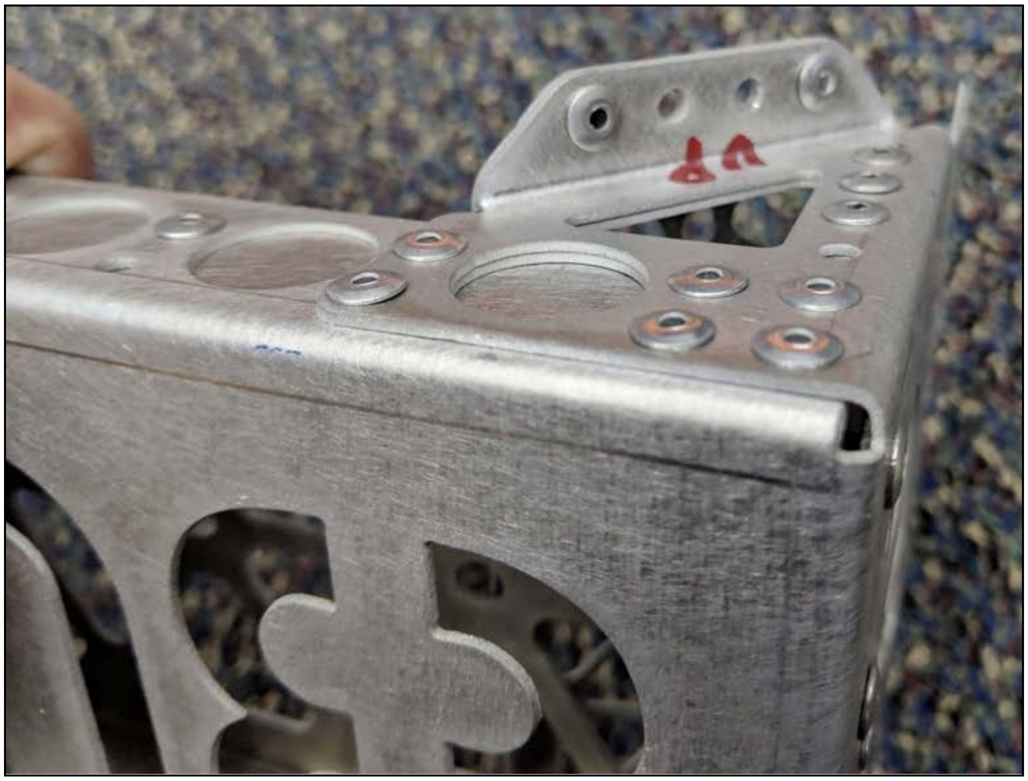
Pins and Castle Nuts: a nut with slots that is tightened like a regular nut, in which a pin inserted through a hole in the bolt fits through the slots, preventing the bolt from rotating and backing out. Types of pins used include a cotter pin, hairpin cotter pins, split rings, or safety wire. Again, like several of the methods above, this is usually unnecessary for FRC.

2.3.6 Rivets

Rivets: Rivets are typically used for attaching sheet metal parts that are relatively thin. Thicker parts are usually attached with tapped holes and bolts. As with bolts, adding more rivets to connect parts will make the connection stronger.

Blind rivets are semi-permanent fasteners that only require you to have access to one side of an assembly. Many teams use aluminum rivets with aluminum mandrels for their robots, but aluminum rivets with steel mandrels are available, and are stronger. The disadvantage to using rivets with steel mandrels is that they are heavier and harder to drill out. To disassemble an assembly held together by rivets, you have to drill out the rivets and replace them, which is fine for the occasional repair. However, if you need to remove something every match, you should probably use a bolt. Eventually the repeated drilling out of rivets will cause the holes to wallow out, or get larger. To solve this, you can drill out the holes for the next largest rivet size.

It is important to identify the total thickness of the parts that are being joined and select the appropriate rivet length. If a rivet that is too short is used, the connection between parts will be weaker, and if a rivet that is too long is used, the rivet may not deform enough to hold the parts together.



Example of a riveted joint on Team 118's 2019 robot.

Gespia NTX Rivet Gun

Rivet Sizes

1/8” Diameter: good for low to medium load connections.

5/32” Diameter: good go-to rivet size for medium to high load connections. Used in the VEXpro Versaframe system.

3/16” Diameter: good for high load applications. Used by some teams to assemble their drivetrain.

Steel rivets are available for increased connection strength, but add a considerable amount of weight and are very difficult to drill out.

Rivet Chart

Rivet Size	Drill Bit to Use
1/8”	30
5/32”	20
3/16”	11

2.3.7 Welding

Welding: Welding is a lightweight and stiff way of joining components. It is permanent, so it is typically only used on robot drivetrains and superstructure. When designing assemblies to be welded, it is best to create fixtures to hold the parts for welding. Team 118 has bolted or riveted the frame together with gussets that have pockets for the weld bead, as shown in section 3.1.2. Team 118 has also bolted the frame together using the tube connecting blocks, shown in section 3.1.3, before the frame is welded.

Since it is difficult to weld assemblies without them warping, especially with thin material, your design should take slight warping into consideration. Usually when welding enclosed material, such as box tubing, small holes are drilled near the welds to allow gasses to escape. When welds fail at competition, “scab” plates are usually added to brace the components, so having a “scab kit” containing thin sheet aluminum, shears, and rivets can be very helpful.

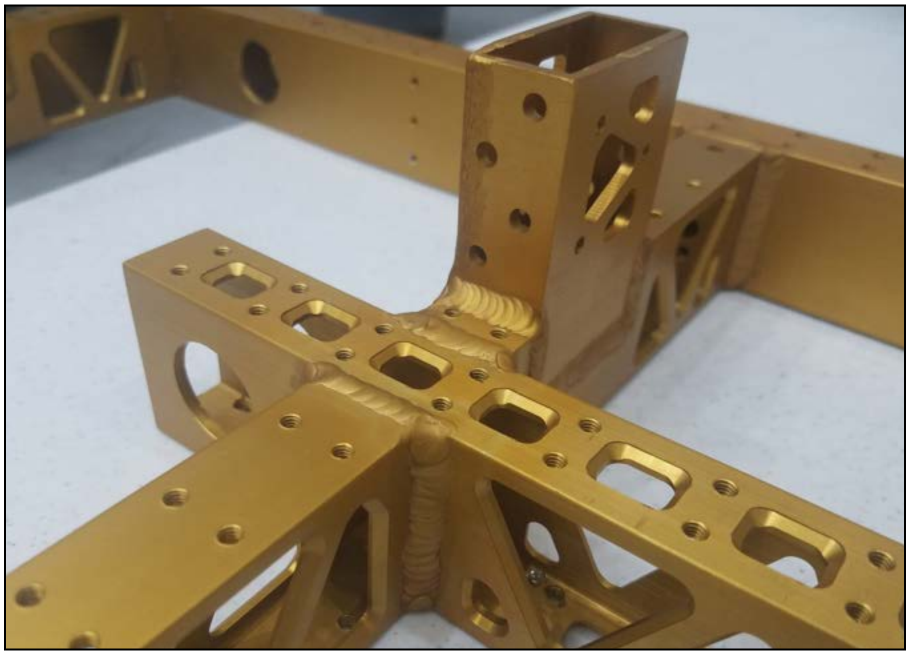
Welding for FRC is usually a large resource investment, not from cost but from complexity. It takes a highly skilled welder to weld thin aluminum well. The overall quality of the welds will be determined more by the skill of the person welding than by the equipment being used. Most teams that weld their robots have sponsors weld their frames, pay a welding shop to do it, or have an experienced mentor do it. If a team plans on welding themselves it is highly recommended to invest in lots of practice welding thin aluminum as it is difficult to not burn through the material and even more difficult to weld the frame with minimal warping.

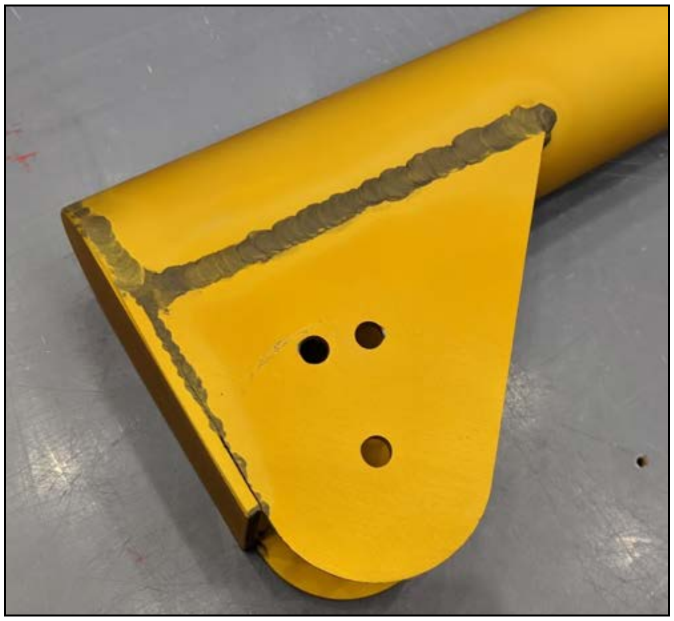
The two welding methods that can be used to weld aluminum are TIG and MIG welding. However, a TIG welder that only has DC capabilities cannot be used to weld aluminum. MIG is generally easier and is most often used for steel; TIG welding is much harder and is used on materials such as aluminum and stainless steel.

MIG welding uses a wire fed by the welder that acts as a filler and adds material to the weld. TIG welding can be used with or without the filler material. If used with a filler, the filler is fed by hand, making it a more difficult process. The arc from MIG welding is spread out over a larger area, while the TIG arc is much narrower and therefore heats up the material more. This makes it much more difficult to TIG weld thin aluminum without the frame warping. MIG welding often has small imperfections in the weld and small air holes in the welds. TIG welding is harder but does produce the highest quality welds. TIG welding is also a slower process and costs more than MIG.

If your team is passionate about building robots with welded frames it is recommended to find someone, or a potential sponsor, who is good at TIG welding. Welding is a very complex topic and people go to school just to be trained to weld.

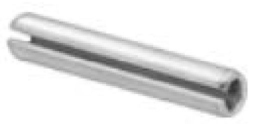
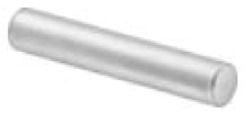
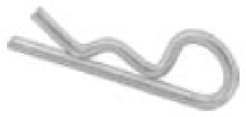
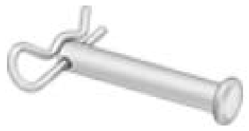
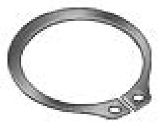
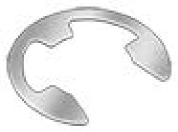
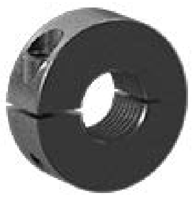
If planning to anodize welded aluminum, 5356 filler should be used; 4043 will not anodize well and will turn a dark brown/black color.


Complex TIG welded joint with 5356 filler on the frame of Team 118’s 2018 robot.

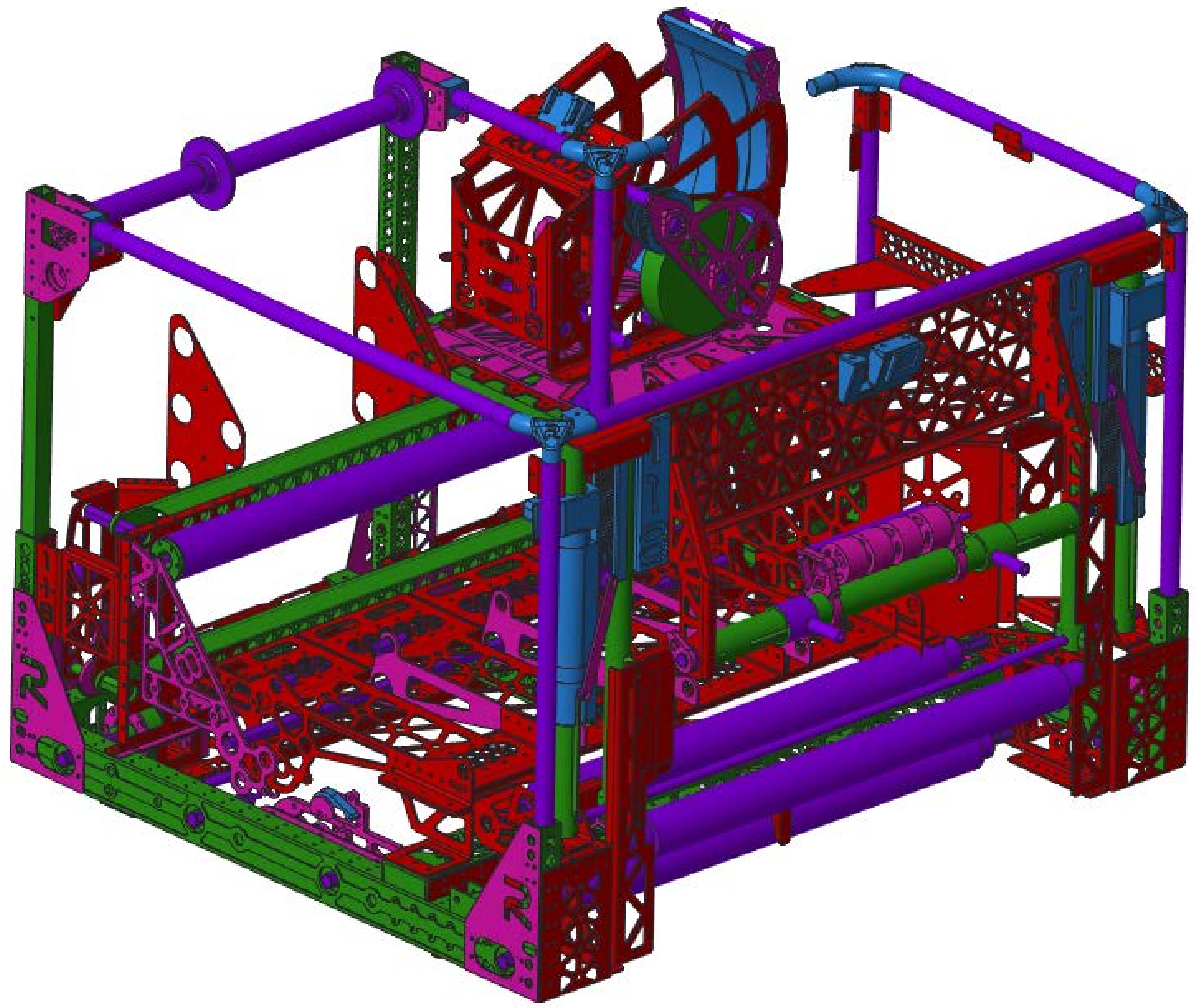

Anodized Assembly with discolored welds with 4043 weld filler.

2.3.8 Miscellaneous Fasteners

Miscellaneous Fasteners

Fastener	Description	Photo
Spring Pins/ Roll Pins	Pins that are pressed into undersized holes in two or more parts to semi-permanently connect them. Check McMaster for the correct hole size when designing for spring pins.	
Dowel Pins	Solid pins that are used for alignment and for transferring sheer forces between parts.	
Cotter Pins	Pins that go through a hole at the end of a clevis pin for quick removal and installation.	
Clevis Pins	A pin with a hole in the end in which a cotter pin is put through. Useful for quick releases or removable mechanisms.	
Retaining Rings/ Snap Rings	Fits in a groove on the end of a shaft to retain gears, wheels, etc.	
E-Clips	Fits into a groove at any point on the shaft for retaining gears, wheels etc.	
Shaft Collars	Clamp around a shaft to retain gears, wheels, etc.	

3. Design Styles



The above picture shows Team 118’s 2017 robot, Ruckus, with all the custom parts highlighted, showing their method of manufacture using the following key:

Color	Method of Manufacture
Pink	Waterjetted sheet metal/plate
Red	Waterjetted sheet metal bent on CNC brake
Purple	Turned on lathe
Green	Milled
Blue	3D-Printed

3.1 Box Tube Construction

Box tube construction: in this method, box tubes are assembled to form a strong backbone to mount other mechanisms. The box tubes can be attached using gussets, welding, or connecting blocks.

Box tube construction is one of the easier methods for design and implementation. Many teams build a box tube superstructure and attach sheet metal parts. The 3 main ways for attaching box tubes together are:

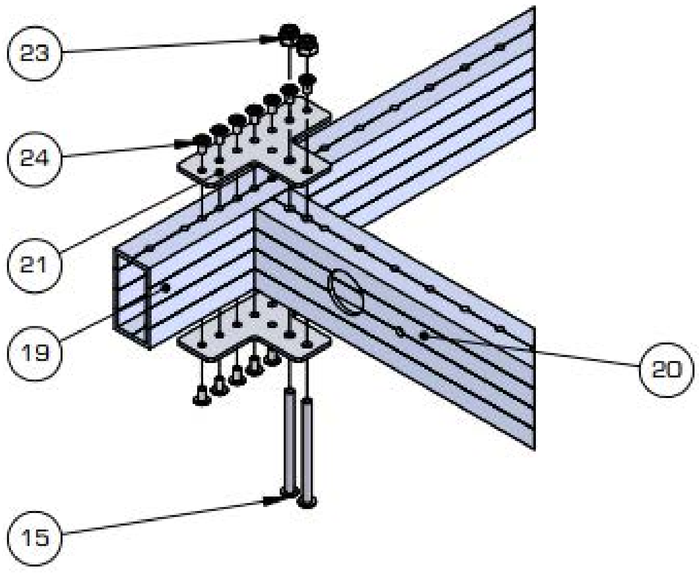
- Gussets with fasteners (typically rivets)
- Welding
- 3D milled blocks, such as the chassis connecting blocks, formerly sold by 221 Robotics.

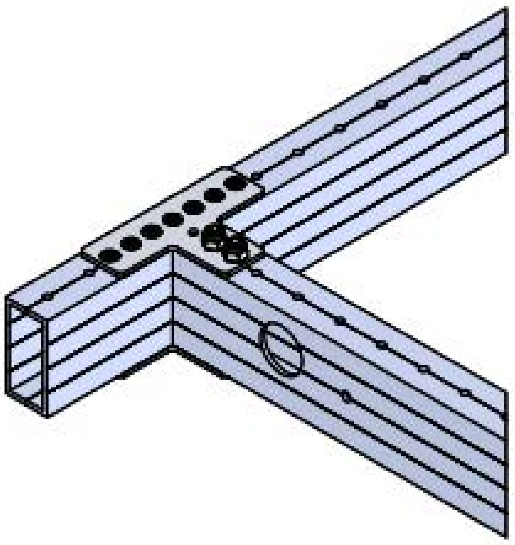
3.1.1 Designing for Gussets

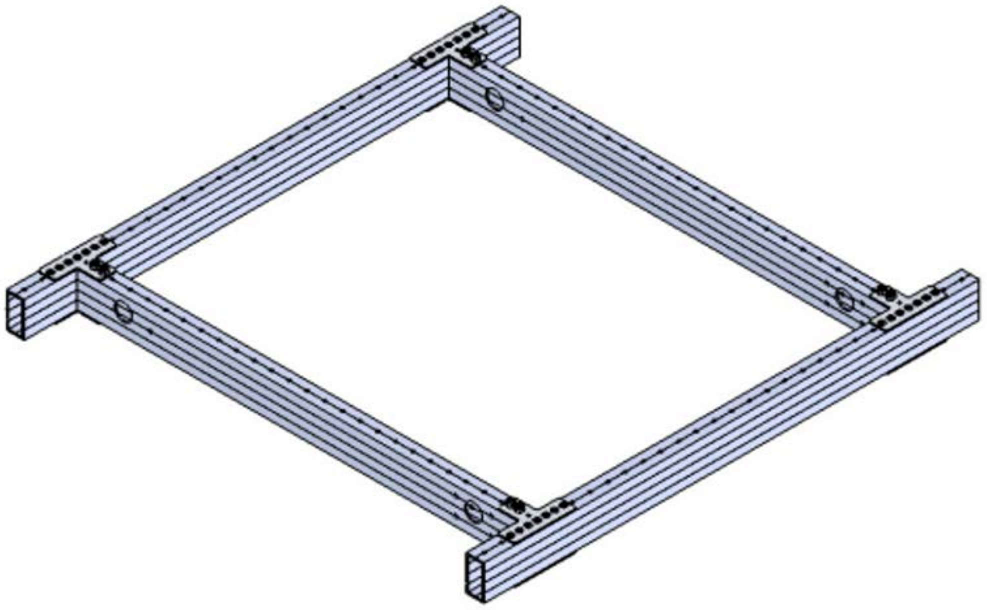
When designing for gussets, you typically design the box tube structure and then design plates or bent sheet metal parts that connect the box tubes together. A good example is the VEXpro VersaFrame system, which uses a system of box tubes with holes spaced 1” apart, as well as an array of gussets for both attaching box tubes together and for attaching motors and gearboxes to the box tube. This system will help immensely during prototyping, so you should try to become familiar with it. Here are links to robot chassis designed by VEXpro to give you an idea of how the VersaFrame system works: 2+2 drivetrain and 6 wheel drivetrain.

The following is an example of VersaFrame gussets being used to build a drivetrain frame.

Step 1
Attach VersaChassis Gussets to front and back VersaFrame with 5/32 pop rivets (match drill any absent holes). Match drill holes from gusset to the side rail, then fasten with #10-32 screws and nuts as shown.



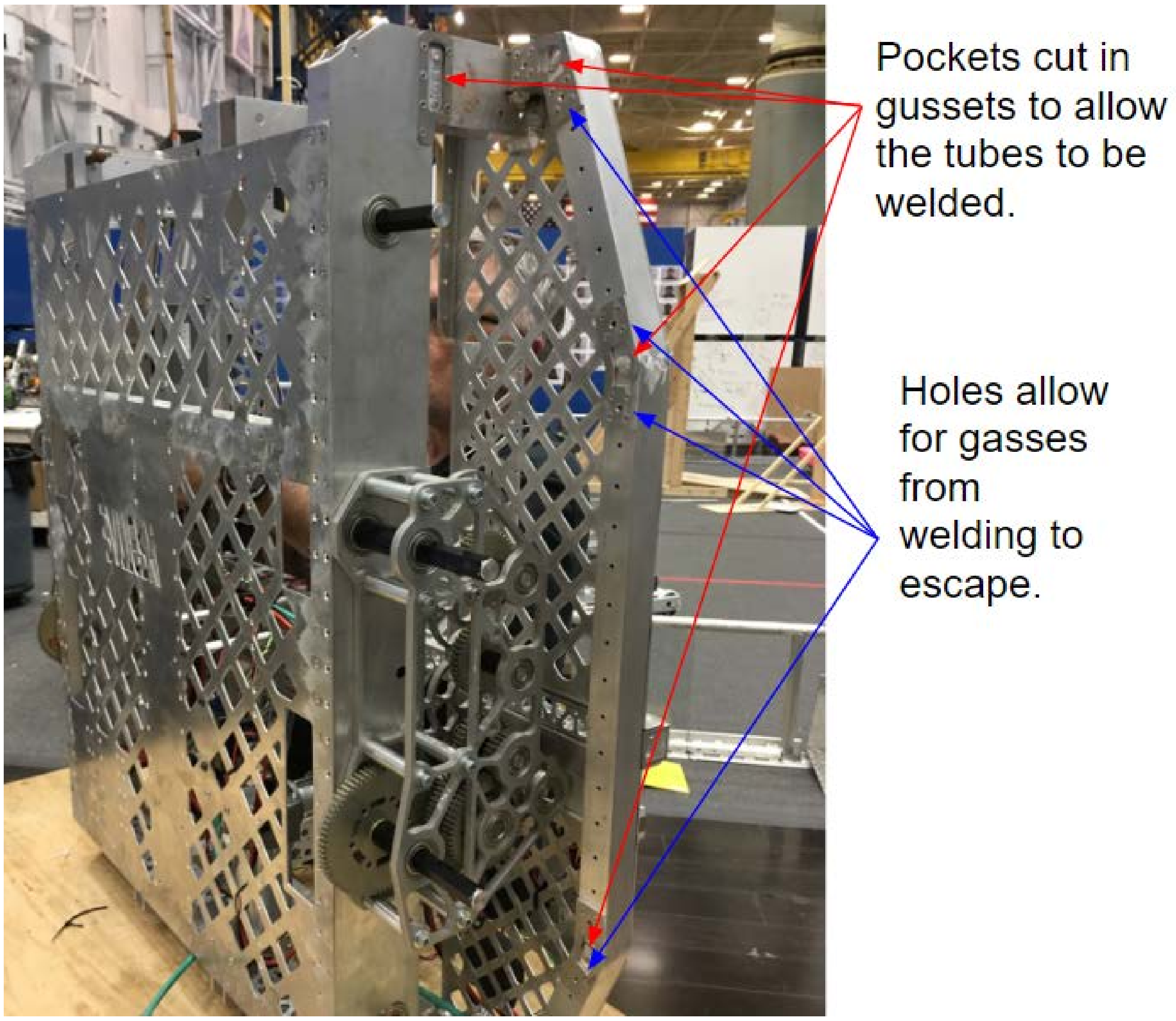




vexpro.com

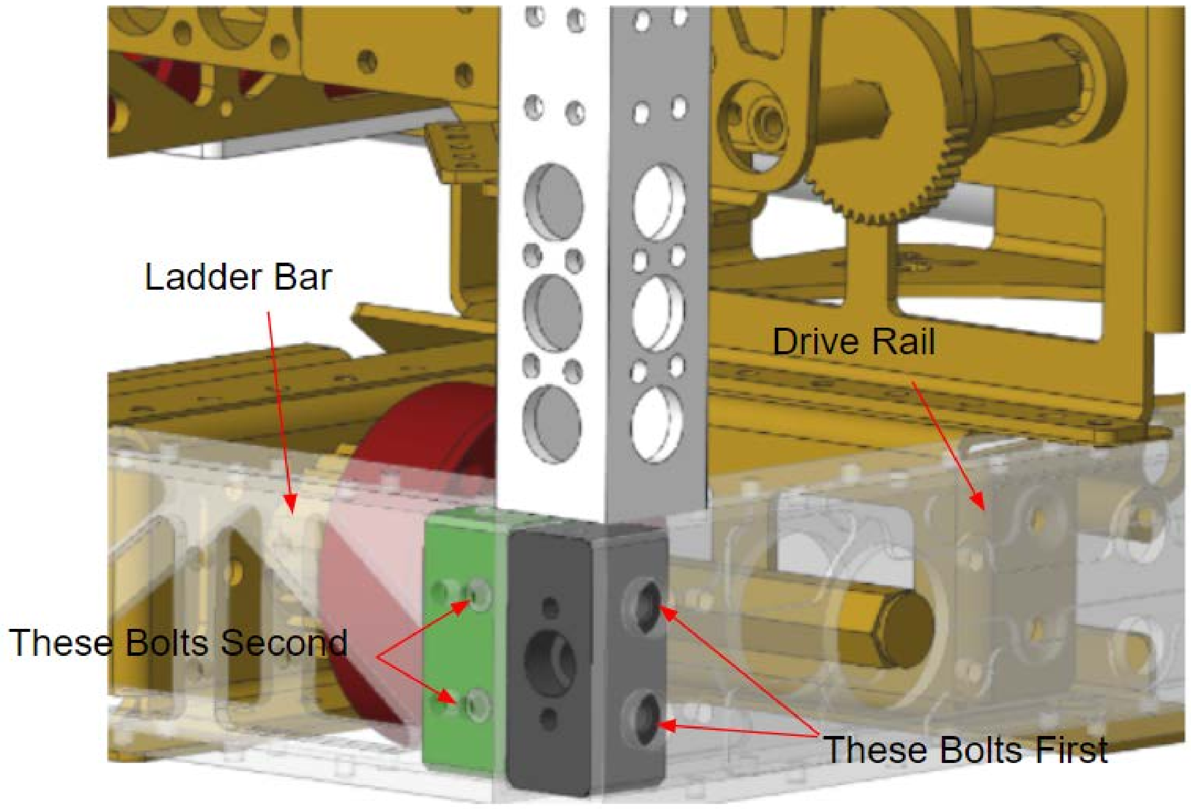
3.1.2 Designing for Welding

When designing for welding, you don't have to physically design the welds, but you need to be aware of where the welds will be in order to prevent them from interfering with robot mechanisms. Welding is a great method for fastening because it greatly increases the strength of an assembly, without adding much weight or structure. To make the welding process easier, design welding gussets that connect the parts with rivets, with a cutout for the weld to be applied. You should plan for this weld in your design. It is possible to grind the weld down flush with the material, however this greatly decreases the strength of the weld. The following picture shows Team 118's 2016 practice robot with gussets, which allow for welding.

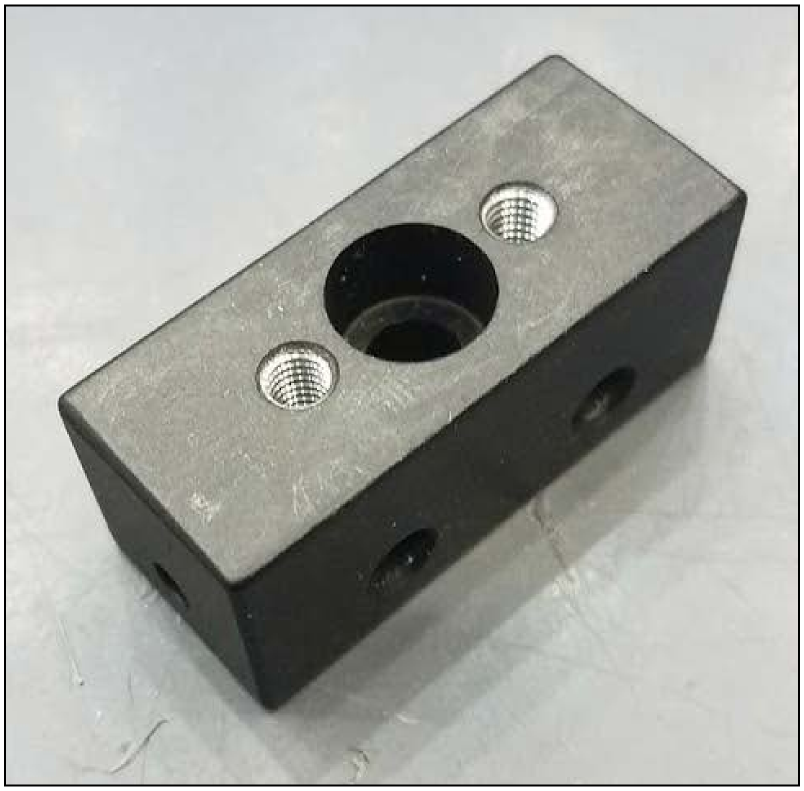

Pockets cut in gussets to allow the tubes to be welded.
Holes allow for gasses from welding to escape.

3.1.3 Designing for 3D Blocks

Designing for 3D blocks is slightly more difficult than the other two methods, and requires the most machine time. 3D blocks are bolted together, inserted into the tubes, and then bolted to the tube. Team 118 has used them on all their robots since 2013. 3D blocks are usually combined with welding and gussets to make a very rigid frame. Here is an example of 221 chassis blocks being used to attach the ladder bar to the drive rail on Team 118's 2017 robot.

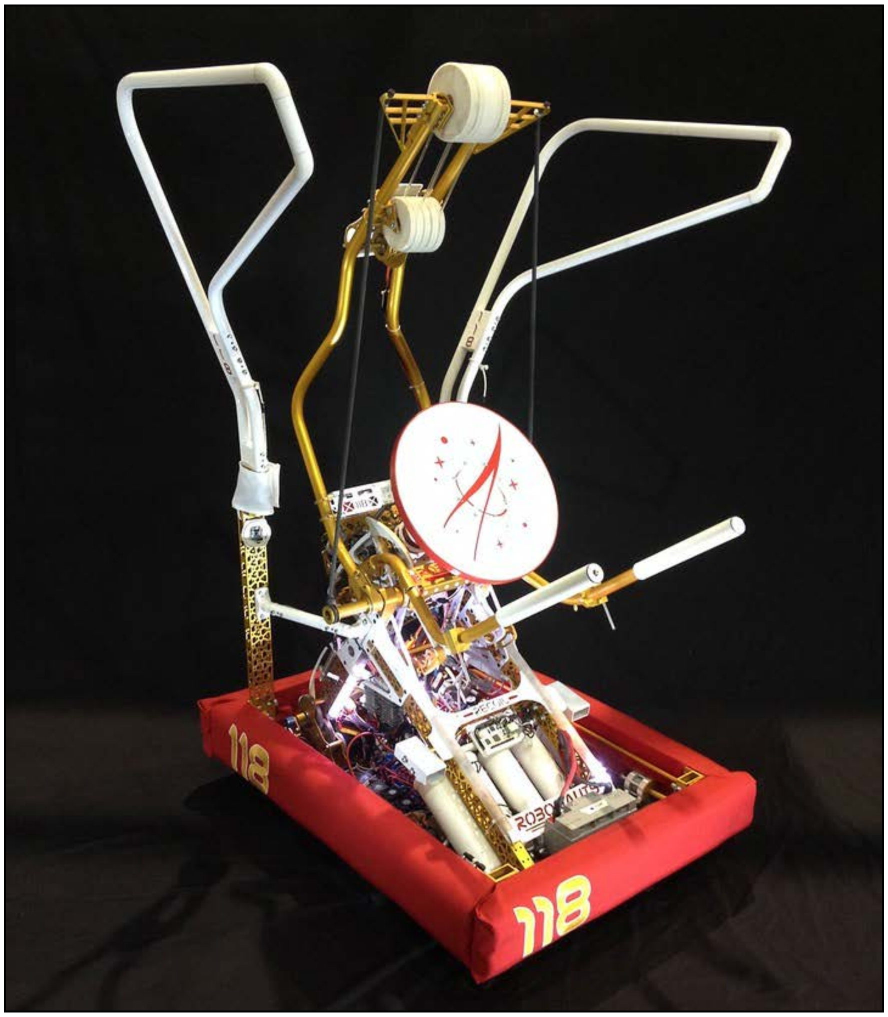


The correct order of assembly for these blocks would be to run your bolts through the drive rail, through the black chassis block, and tighten the green chassis block until it is flush against the side of the drive rail. Then you would insert the green block into the ladder bar, insert the bolts in it, and tighten, making sure the two box tubes are flush.


3D Tube Connecting Block for 2” by 1” Box-Tubing.

3.2 Round Tube Construction

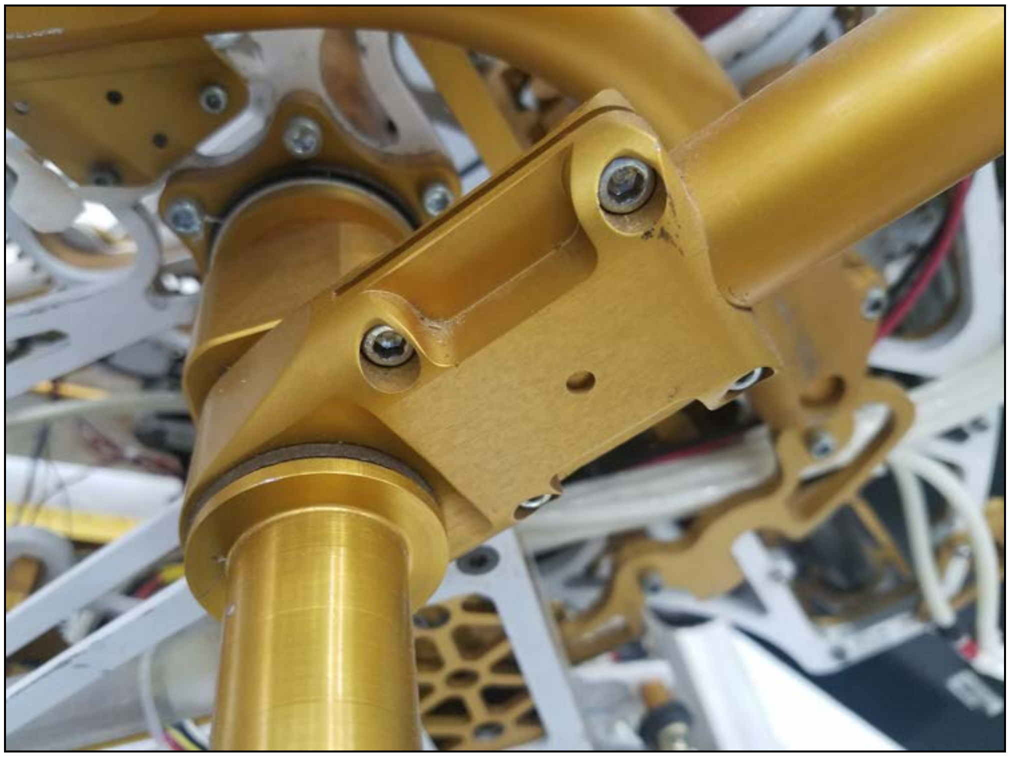
Round Tube Construction: round tube structure can be very light and strong. Round tubing is stronger in torsion than square/rectangular tubing. However square/rectangular tubing is stronger when a bending force is applied. Round tubing is also typically 20% lighter than square tubing of the same size and thickness. It can be bent into complex geometries that can be quite strong. The downside of round tubing is that it is difficult to machine and interface with. It is also one of the hardest constructions to effectively use; it is not trivial to manufacture parts with multiple bends with a high level of repeatability.



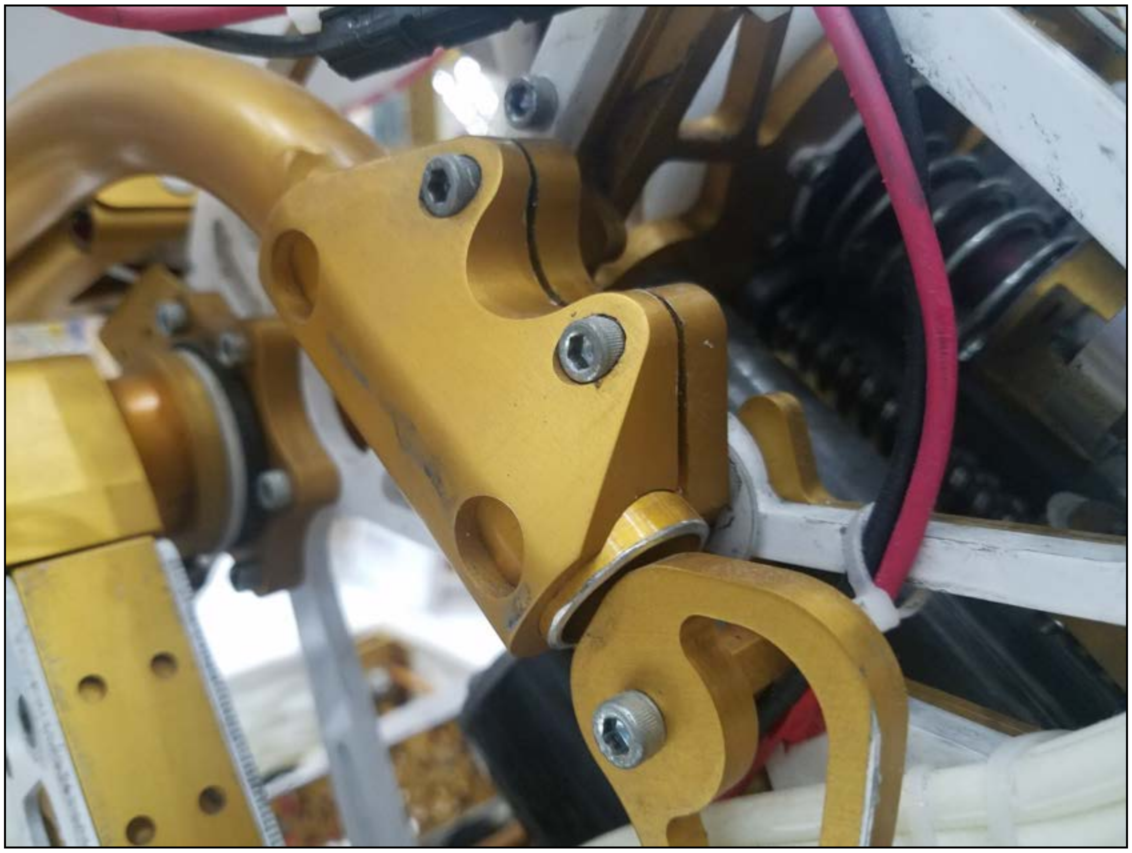
Extensive use of bent round tubing on Team 118's 2014 robot.

3.2.1 Clamping

Clamping: one method of attaching round tube to other structure is using clamping parts that are attached to another part of the robot. Below are a couple examples of round tube clamps.

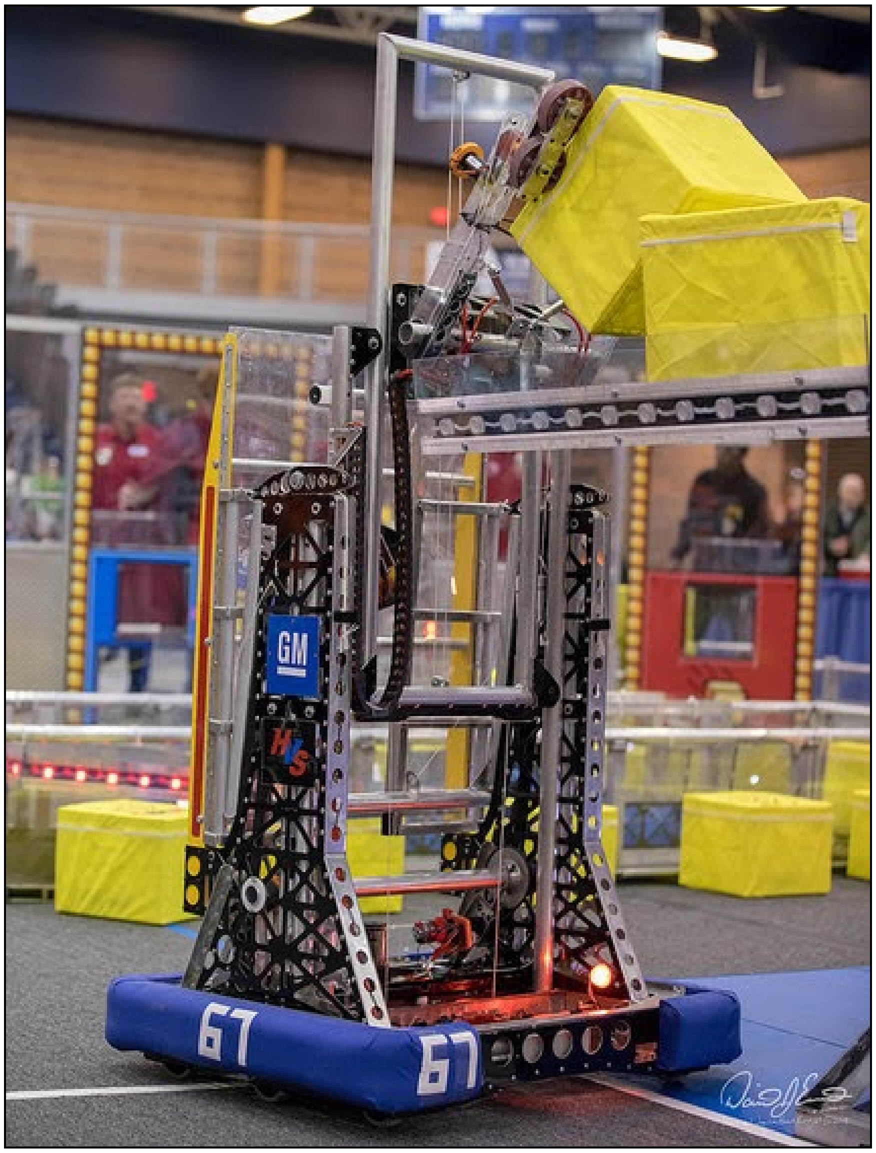


A CNC machined round tube clamp on Team 118's 2014 robot that attaches the intake mechanism tubing to the master pivot.


A CNC machined round tube clamp on Team 118’s 2014 robot that attaches the intake tusks to the side of the shooter gearbox.

3.2.2 Welding

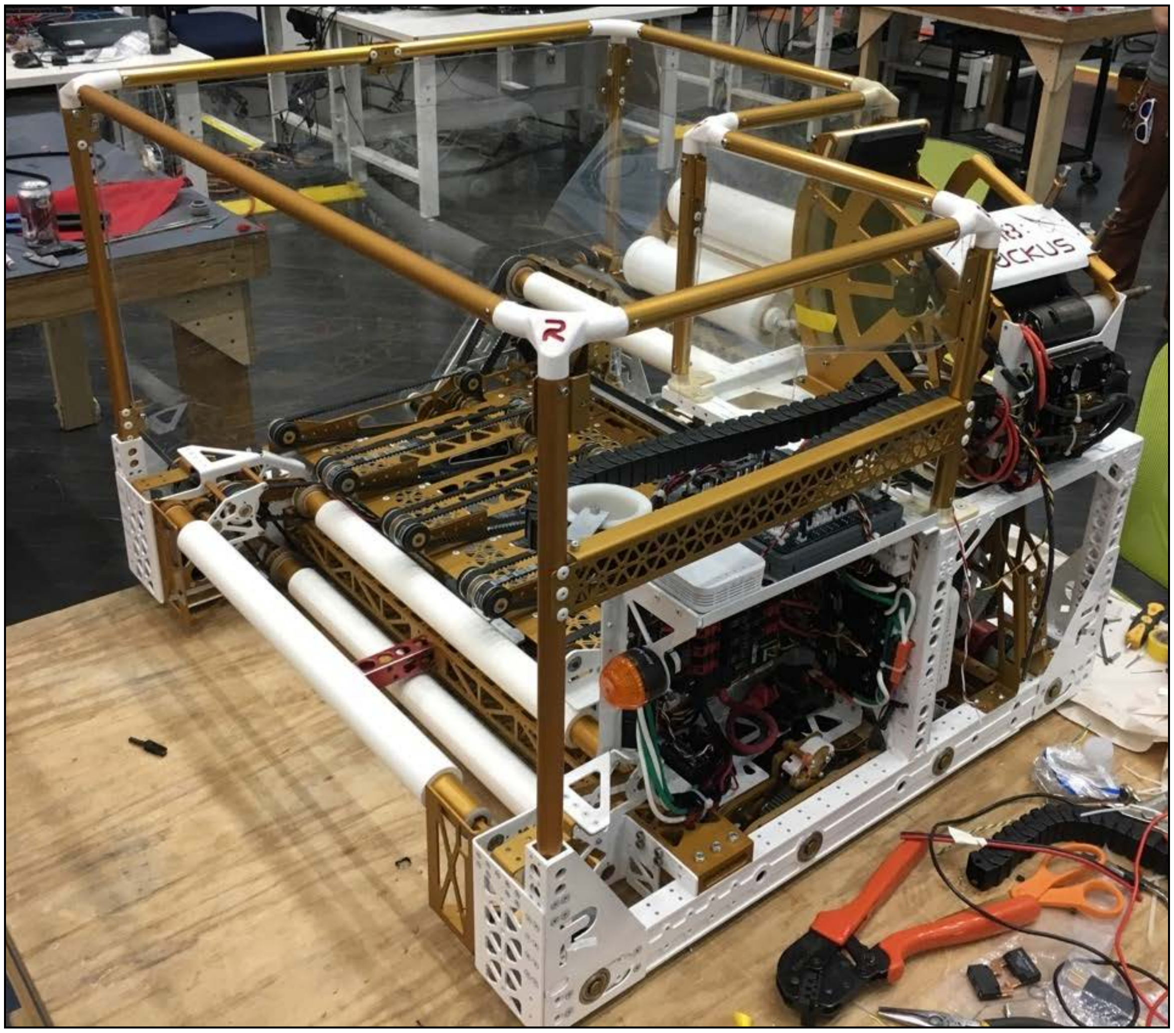
Welding: welding can be used to attach round tube to other structure such as box tube. One thing to note is that it is difficult to weld thin wall aluminum tubing. Usually a small hole is drilled into the tubing near the weld joint to allow the hot gasses created during the welding process to escape.



Team 67 is well known for consistently using welded round tube on their robots, as seen at the top of their elevator in this photo.

3.2.3 Tube Inserts

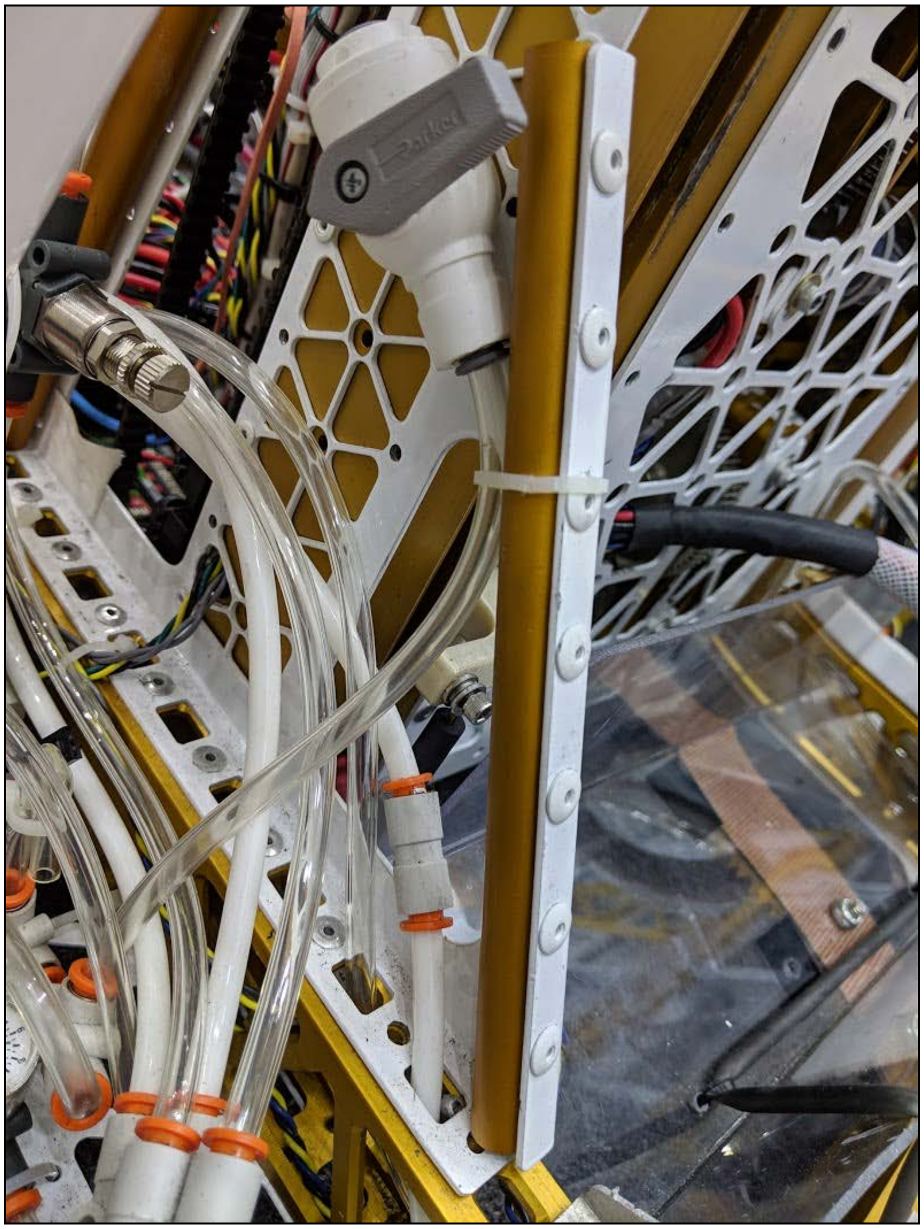
Tube Inserts: aluminum blocks can be machined with a feature that inserts into a tube, allowing the tube to be attached to some other structure. 3D printed parts can also be used to attach round tubing when it is not exposed to high loads and violent impacts.



3D printed parts at the corners of the hopper structure on Team 118's 2017 robot insert into the round tubing and have a 4-40 bolt going through them.

3.2.4 Riveting

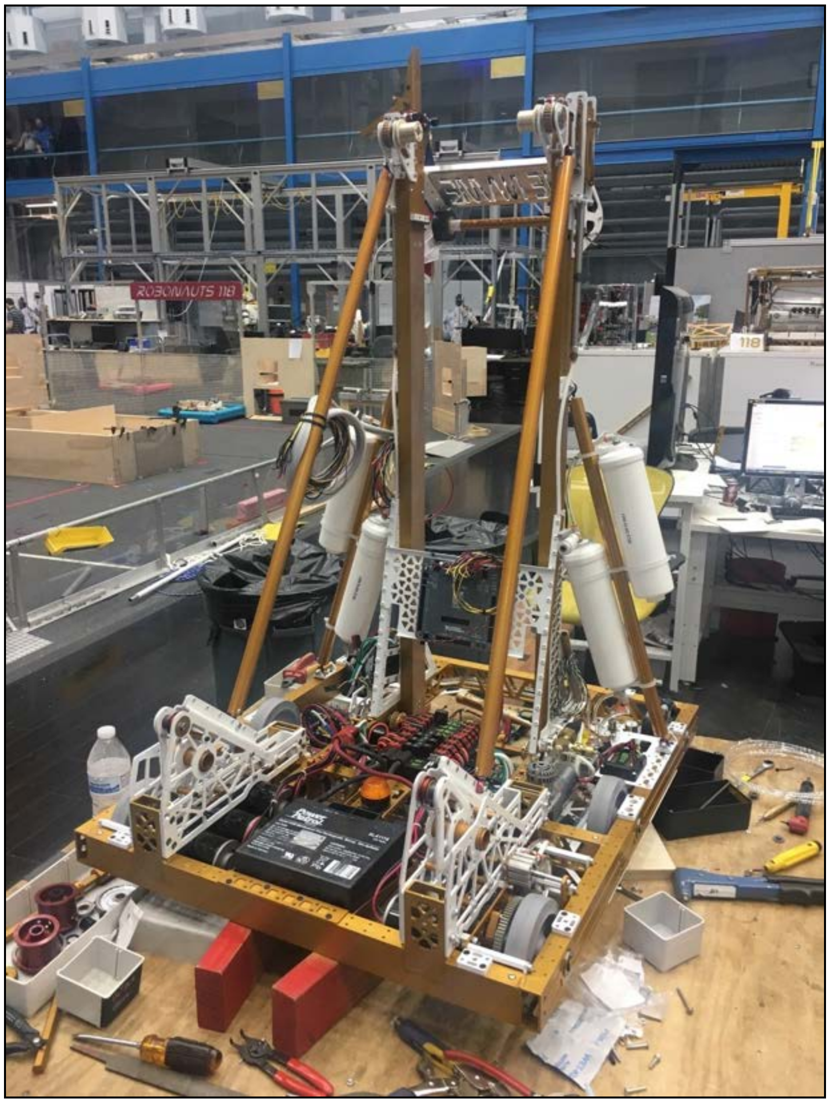
Riveting: one interesting use of thin wall round tubing is to use it to strengthen a large sheet metal part. A round tube can be riveted to a sheet metal part to strengthen it along that axis.



1/2” OD tubing riveted to a sheet metal part on Team 118’s 2018 robot.

3.2.5 Rod Ends

Rod Ends: by inserting a plug into each end of the tube, tapping one for a right handed thread, tapping the other for a left-handed thread, and threading in ball joint rod ends such as these, you are able to create a turnbuckle that acts as a compression and tension member. This lets you add very light triangular supports to structure, such as elevators.



7/8" OD 0.035" wall tubing with tapped end plugs used as a structural member to support the elevator on Team 118's 2018 robot.

3.3 Plate and Standoff Construction

Plate and Standoff Construction: plate and standoff construction is one of the simpler construction methods. It's common in VEX EDR, and is also how the kitbot chassis is assembled.

Plate and standoff construction is most commonly used on gearboxes, usually to support the end of the shafts. When designing a mechanism, try to use COTS standoffs to minimize machining time. McMaster has a very large variety of standoffs in standard lengths in all the common thread sizes. Note that you can tap the ends of VEXPro thunderhex shaft for a 1/4-20 bolt to easily make custom standoffs. Below is a photo of Team 118's shooter from 2017, showing the shooter shafts supported by an ⅛" plate, supported by three, 8-32 standoffs.

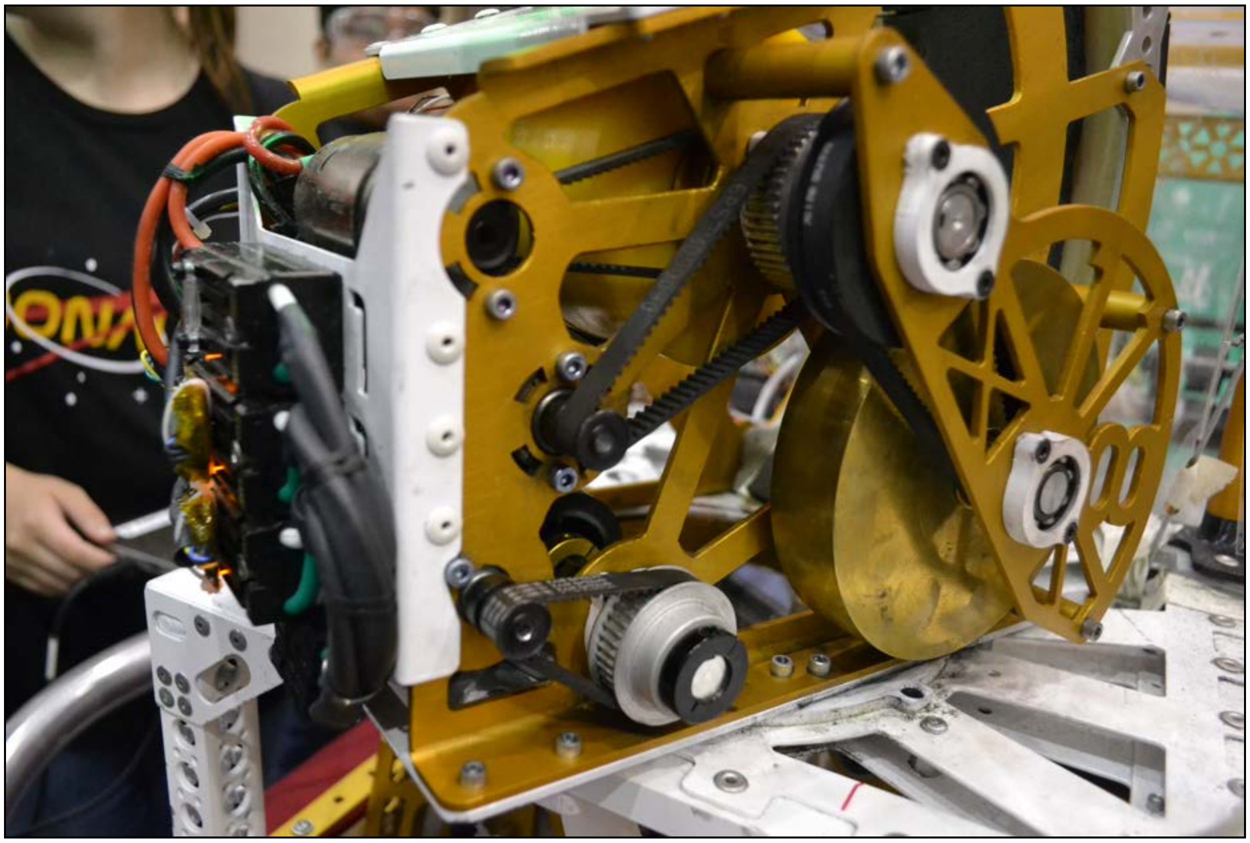


Plate and standoff construction on Team 118's 2017 shooter.

3.3.1 Standoffs vs. Spacers

Standoffs and spacers are often mixed up, and the terms are sometimes used interchangeably, however they are different.

Standoffs: standoffs are threaded and usually have bolts going into them from either end.

Tube Connecting Nuts: threaded inserts that can be pressed into tubing to create custom, large diameter standoffs. They are sold with a variety of thread sizes and are sized to press into specific tube inner diameters.

Spacers: are not threaded and are meant for a bolt to go all the way through them, this is technically better than standoffs because you can preload the assembly and you get the added strength of the steel bolt. You can read more about why preload is important here.

3.4 Sheet Metal

Bent sheet metal: in this method, metal or plastic (usually polycarbonate) sheet is bent using a sheet metal brake to add strength to a part, to combine multiple functions into one part, and to reduce part count. When designing for bent sheet metal, it is important to keep in mind how the part will be bent to ensure manufacturability. Bent sheet metal allows you to create parts that are very strong but relatively light compared to other design styles. Sheet metal design is the most exotic of the design styles mentioned here; it is also the hardest to master. If you are designing parts to be bent on a manual brake, it is important to understand which dimensions of the part are critical.

3.4.1 Recommendations for Sheet Metal

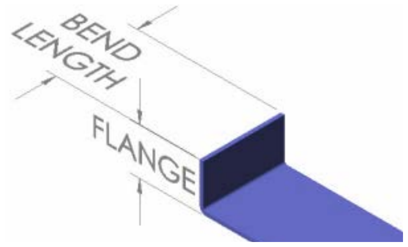
Bend Radius: the bend radius of a part is the inside radius of a bend. Usually having a bend radius equal to or greater than the part thickness is safe for 5052 aluminum. There are bend tables that tell you the optimal bend radius for a given thickness. It is important to know however, these tables frequently use non-standard radii, and sheet metal benders won't have dies for them, so you will have to make your radius the closest standard size. The following is a recommended bend radius chart for different aluminum alloys. Typically 5052 alloy aluminum is used for all bent parts. 6061 alloy aluminum can be bent with larger radii, but you typically don't need 6061 for sheet metal parts.

Aluminum Minimum Bend Radii for 90 Degree Cold Forming of Sheet and Plate

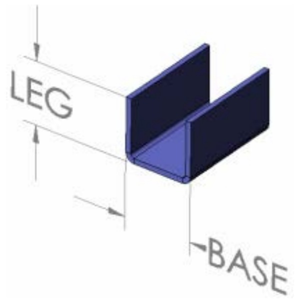
per The Aluminum Association, Inc.

Alloy	Temper	RADII FOR VARIOUS THICKNESSES EXPRESSED IN TERMS OF THICKNESS "t"							
		1/64 in.	1/32 in.	1/16 in.	1/8 in.	3/16 in.	1/4 in.	3/8 in.	1/2 in.
3003	H12	0	0	0	1/2t	1t	1t	1.5t	2t
	H14	0	0	0	1t	1t	1.5t	2t	2.5t
	H16	1/2t	1t	1t	1.5t	2.5t	3t	3.5t	4t
3105	H12	0	0	0	1/2t	1t	1t	1.5t	2t
	H14	0	0	0	1t	1.5t	1.5t	2t	2.5t
	H16	1/2t	1t	1t	1.5t	2.5t	3t	3.5t	4t
5052	H32	0	0	1t	1.5t	1.5t	1.5t	1.5t	2t
	H34	0	1t	1.5t	2t	2t	2.5t	2.5t	3t
	H36	1t	1t	1.5t	2.5t	3t	3.5t	4t	4.5t
	H38	1t	1.5t	2.5t	3t	4t	5t	5.5t	6.5t
5086	H32	0	1/2t	1t	1.5t	1.5t	2t	2.5t	3t
5454	O	0	1/2t	1t	1t	1t	1.5t	1.5t	2t
	H32	1/2t	1/2t	1t	2t	2t	2.5t	3t	4t
	H34	1/2t	1t	1.5t	2t	2.5t	3t	3.5t	4t
6061	T6	1t	1t	1.5t	2.5t	3t	3.5t	4.5t	5t

Flange: the flange is the feature you bend. This is the feature you typically stick inside the brake to bend it. The flange length should be no shorter than 4 times the material thickness. While it is possible to bend parts with shorter flanges, it requires special tools, or will turn out poorly, so try to avoid it.



C-Channels: C-channels are a type of bent sheet metal part you may be familiar with from VEX. C-channels are usually pretty simple to bend, however the leg of the C-channel should be equal to or less than the length of the base of the C-channel, otherwise it might crash into the die when attempting to bend.

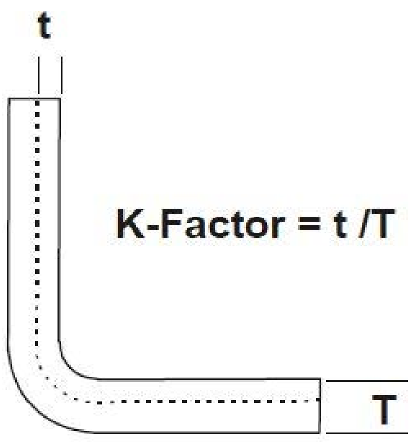


Holes Near Bends: it is not ideal to place holes close to a bend. You should try to leave a distance of at least 2 times the part thickness between the edge of a hole and the start of a bend.

3.4.2 What Happens When a Part is Bent

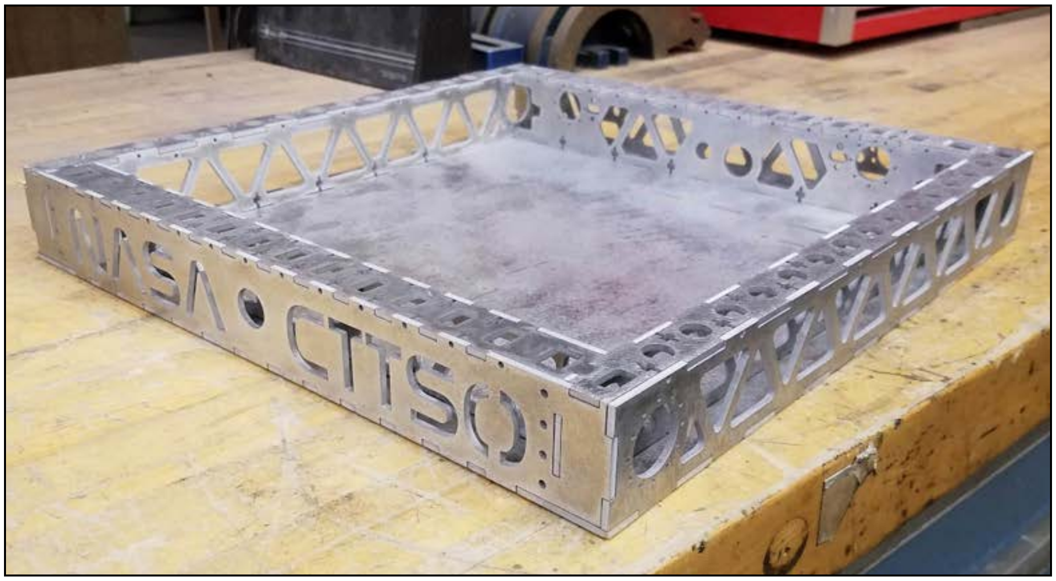
Shrinking and Stretching: when a part is bent in a sheet metal brake the inside of the part shrinks, while the outside of the part stretches. This will lead to small cracks on the outside of a bend if the radius of the bend is too small or the part is not bent with the grain of the material. These cracks are usually negligible when it comes to strength. The line between where the material stretches and shrinks is called the **neutral axis.**

K-Factor: when a part is bent, the material stretches. This means when you go cut a part on a water jet, router, or laser cutter, and it's 4" long, if you put a bend in the middle of it and measure the length of the outside of the part, it will be slightly greater than 4". To compensate for this material stretch you must set the K-factor for the part. The K-Factor varies a small amount, based on material thickness, but the difference is negligible. For 5052 aluminum Team 118 uses a **K-factor** of **.44** to compensate for material stretch. If you use the wrong K-factor, then your parts may not fit together. The included diagram shows how K-factor is calculated. The dotted line in the diagram is the **neutral axis**.



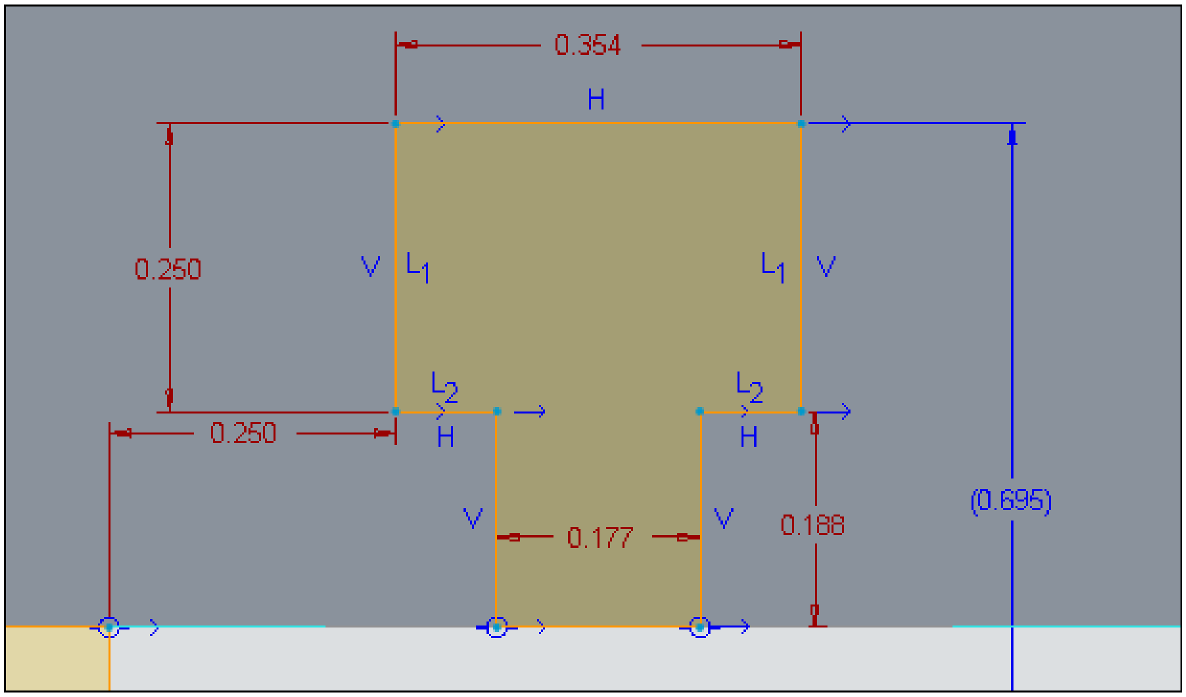
3.5 Slots and Tabs

Slots and Tabs: one way of attaching perpendicular parts. Tabs on one part are inserted into slots cut into other parts. This allows for easy alignment and the transfer of forces. To bolt the parts together, a **captive nut/T-nut/Jesus nut** can be used. This is when a slot for a nut is cut into one part, while a bolt hole is cut into a part perpendicular to it. This allows two perpendicular parts to easily be held together without tapping the ends of material. The slots and tabs need to have a tolerance on them that allows them to fit together. Best practices would be to cut a test part to allow you to figure out the best fit. In many cases for waterjet parts, making the tab 0.005" smaller and/or the slot 0.005" bigger in both dimensions will allow the parts to fit together nicely.



A robot frame built using slots and tabs to create a pseudo-box tube from flat plates.

This sketch below shows the dimensions you might use for a captive 8-32 nut. The bottom section (dimensioned to 0.177) is the diameter of a clearance hole for an 8-32. The top section (dimensioned to 0.354) is the flat-to-flat size of an 8-32 nut, plus a tolerance of 0.01” to ensure the nut will fit in. Be sure to measure the nuts you have with calipers to make sure they’ll fit. Keep in mind the below sketch is for a part intended to be cut on a waterjet; if using a router, you would need to add reliefs at the 90 degree interior corners to make the part machinable.



3.6 3D-Printing

3D printing parts is extremely useful for 3D parts with complex geometry that is impossible or impractical to create on a mill.

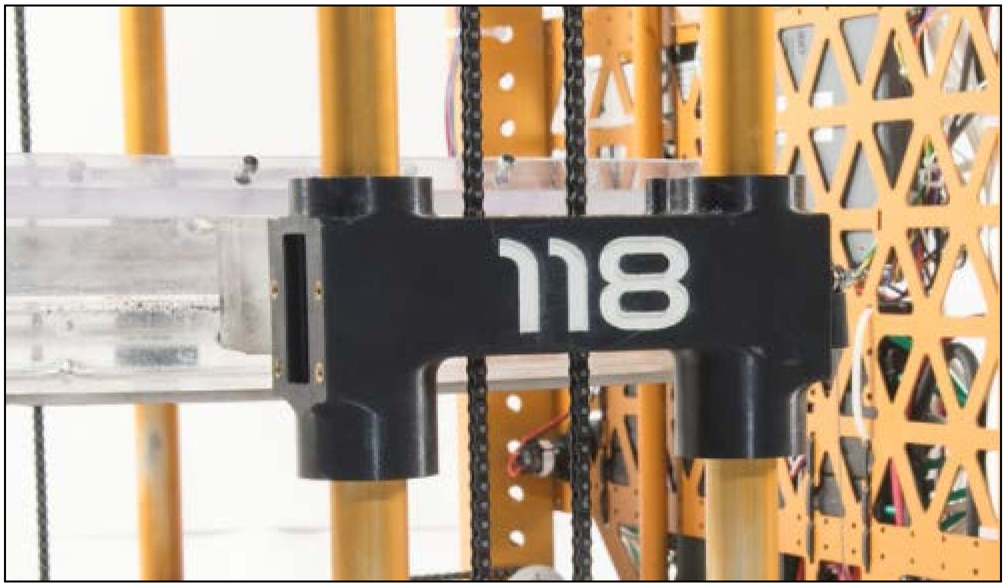
When designing parts to be 3D printed, try to minimize steep “overhangs” as the printer will have to use support material for each of these overhangs.

When you desire to have threaded holes in 3D printed parts it is best to use heat set inserts, which are brass threaded inserts that are pressed into plastic parts using a soldering iron with a special tip. Check McMaster for more info on the sizes of these inserts and what size holes to design in your plastic parts. In lower load applications, you can also use press-in/threaded inserts that do not require a soldering iron - this video has more information in comparing threaded inserts. Tapping 3D printed parts is possible, however it is not recommended.

Large amount of heat set inserts in a 3D printed ULTEM part.

Markforged has several blog posts listed below on designing for 3D printing.
Designing for Printing: Unit Tests and Tolerances
Designing for Printing: Part Warping and How to Stop it
Designing for Printing: Embedding Nuts in Printed Parts
Using Heat Set Inserts
Casting Custom Wheels Using 3D Printing
Embedding Components in 3D Printed Parts
Threaded Joints with 3D Printed Parts
3D Printed Joints: Simplifying Assembly

Team 118’s 3D printed frisbee hopper from their 2013 robot.



Team 118’s 2015 lift trucks printed in ULTEM.

3.7 Weight Savings

Most teams run into issues meeting the robot weight limit, which in recent years has been 125 pounds. There are a variety of ways to cut weight on a robot listed below.

The first is to simply build less robot. The best method to eliminate weight on a robot is to decide not to build or to remove a heavy mechanism that rarely gets used. Climbers in years such as 2016 where the climbing is not worth very many points are a good example of this. For many teams focusing on improving the main scoring mechanism is far more valuable than diverting resources from that mechanism into one that scores less points in a match. This comes down to strategic design, which is talked about extensively in this presentation by Karthik Kanagasabapathy.

It is almost always better to use thinner material without pocketing than using thicker material with pocketing. This is cheaper, takes far less machining time and usually gets similar strength and weight results as the pocketed part. Using thinner material and adding a flange or riveting and thin round tube across the load path can be used to create parts that are lighter and stronger than heavily pocketed thicker parts. Plastics or sometimes composites can often be used to replace aluminum parts to save weight.

Teams sometimes cut **lightening patterns** into parts to save weight. This is also called pocketing. When designing lightening patterns you need to be aware of the loads going through the part and support features, such as bearing holes and bolt holes.

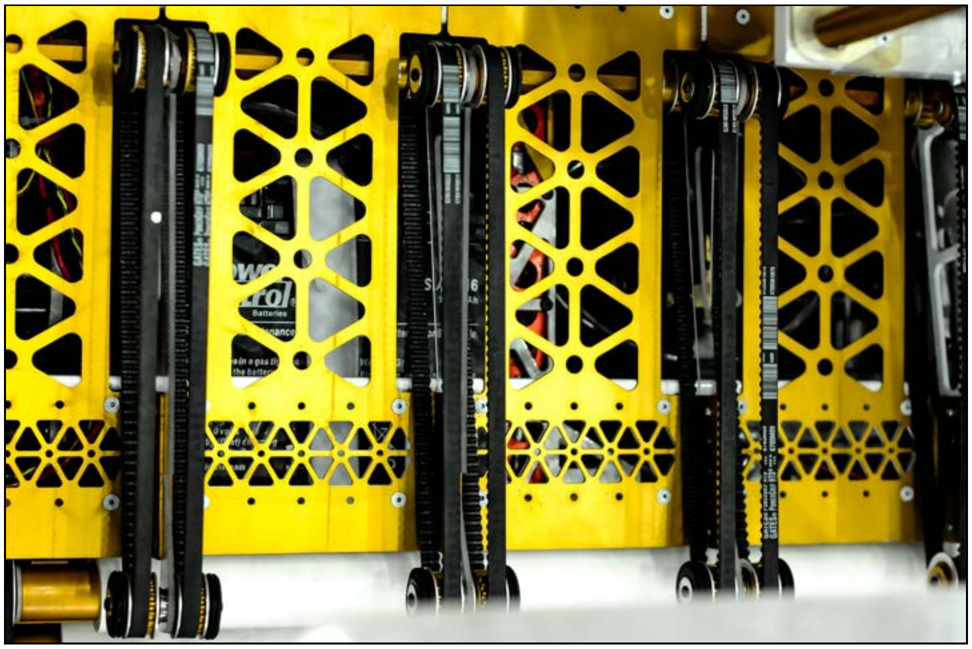
Designing for Lightening Patterns

When creating lightening patterns, remember that triangles are the strongest shape. It's always important to not have any sharp corners. All sharp corners should have rounds, otherwise the part is likely to tear or crack there. When cutting parts on a waterjet or a laser cutter, the rounds can be almost any size, 1/8” is a good radius for almost all parts, but on parts with smaller pockets, 1/16” or smaller radius is fine. On milled or parts cut on a router, you need to be aware of what size end mill will be used to cut it. Having radii of 1/8” technically allows you to machine your part with a 1/4” endmill, however it is best practice to make this radius slightly larger to improve tool life and decrease cut time.

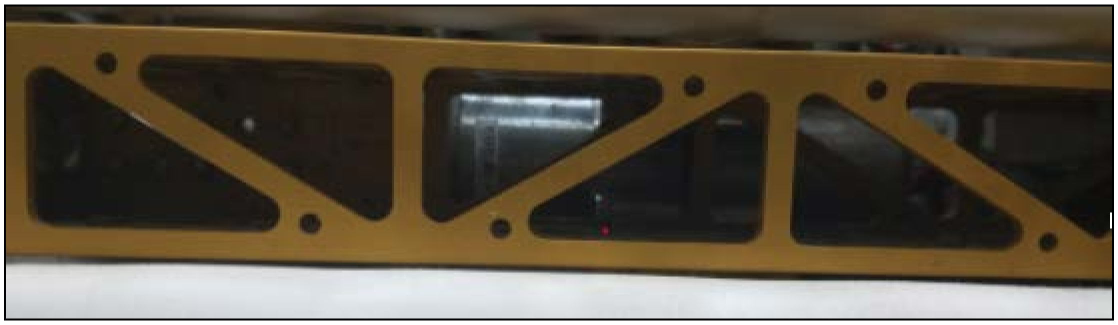
It is best to wait until the very end of your design to create lightening patterns, otherwise you will likely waste a lot of time when changing things on your parts.

Types of Lightening Patterns

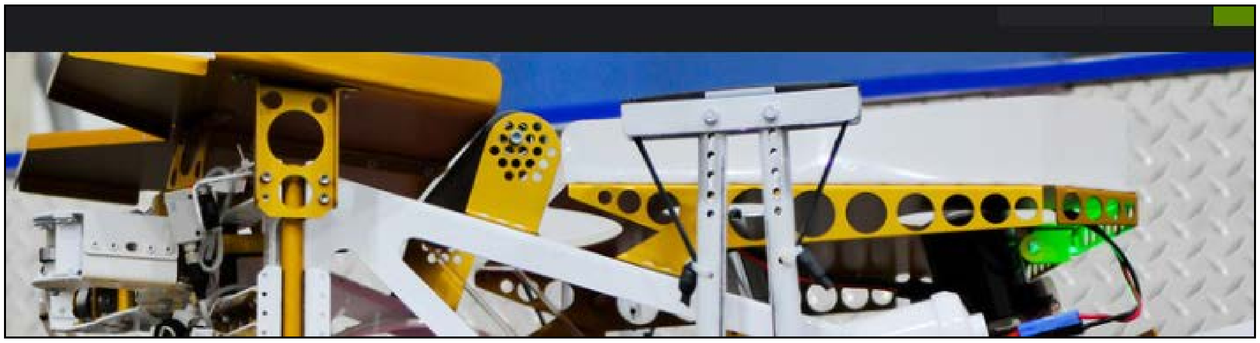
Isogrid: Isogrid is a lightening pattern that has its roots in spacecraft. It consists of 6 equilateral triangles surrounding a center point. One downside of isogrid is that it takes longer to machine than most other lightening patterns. Isogrid is usually only machined on larger tubing because the pattern consists of many smaller pockets, which would require very small endmills on tubing any smaller.


Isogrid on the hopper floor of Team 118's 2017 robot.

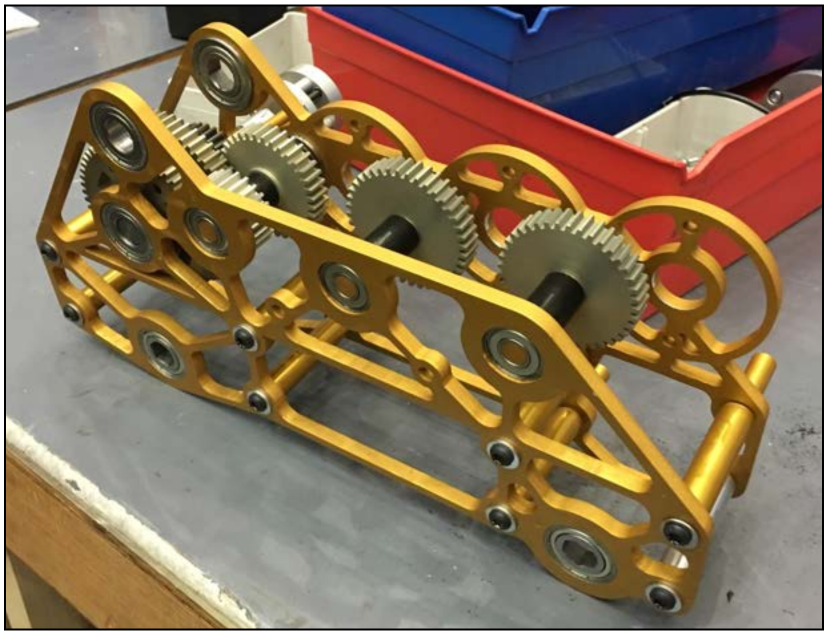
Triangles: as stated previously, triangles are the strongest shape. Triangle lightening patterns are quicker to cut on a waterjet or router, and quicker to mill than isogrid. Triangles are also typically used on box tube that is less than 2 inches in dimension since milling isogrid on parts smaller than that usually requires small endmills and increases machining time.


Triangles on the front ladder bar of Team 118's 2017 robot.

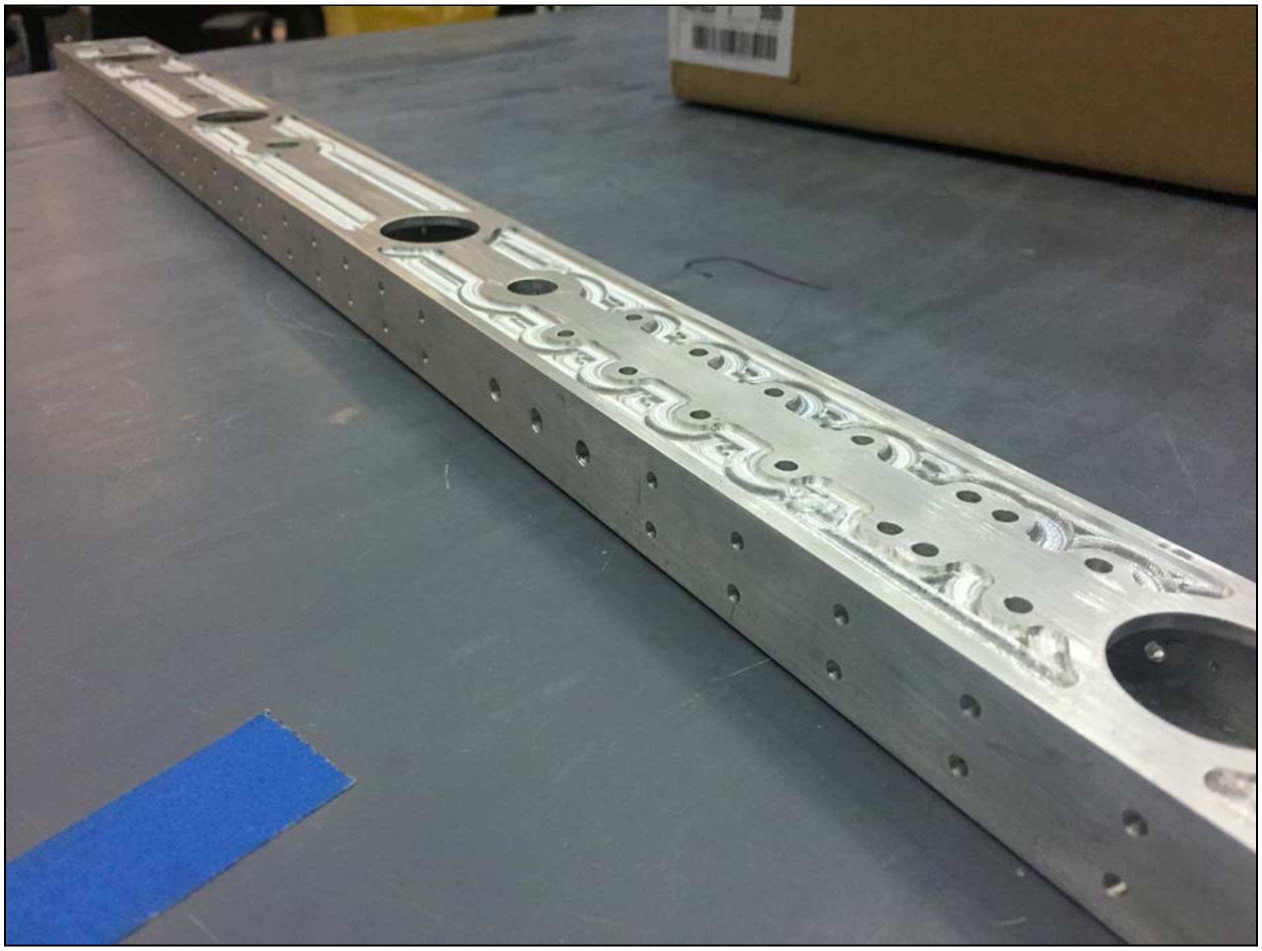
Circles: circles are the easiest lightening patterns to machine as they can be machined on a drill press, if necessary. Circles are also easy to draw and even make parametric. Many teams end up drilling circular holes in their robot late in the build season, when they realize their robot is overweight.


Circle lightening pockets on the frisbee loader of Team 118's 2013 robot.

Lightening non-standard parts and Gearboxes: for parts in which there are many bearing and bolt holes, such as in gearbox plates, you should connect the bearing holes and bolt/rivet holes to the rest of the part to distribute loads through these struts. It is important to leave enough material around the bearings, bolt holes, and the outer edge of the part to properly support the loads. It is ideal to connect all the holes with struts arranged in a triangular shape. If you have a large open rectangular area, it is best to add another strut separating it into two triangles. This is most applicable on parts that are over 3/16" thick. RAMP Greybots has a great video here on how to do gearbox pocketing.


The drivetrain gearbox from Team 118's 2016 robot.

Half Depth Cutting (Pocketing): on parts manufactured on a mill or a CNC router, it is possible to create lightening pockets that don't go all the way through the part. This is frequently done on NASA rover parts. The thin shell left behind adds a significant amount of strength compared to a complete cutout. In 2017 and 2018, Team 118 did half depth pocketing on their drive rails. This saved about half a pound of weight, but greatly increased machining time. Pocketing drive rails is not recommended.


Drive rail for Team 118's 2017 practice robot.

4. Power Transmission

4.1 Motors

Motors convert electrical power into rotating mechanical energy. There are multiple types of motors, including brushed, brushless, and stepper motors. Most mechanisms on FRC robots use motors to provide motion. Motors are COTS parts, the game rules manual contains a list of legal motors, this list is updated each year.

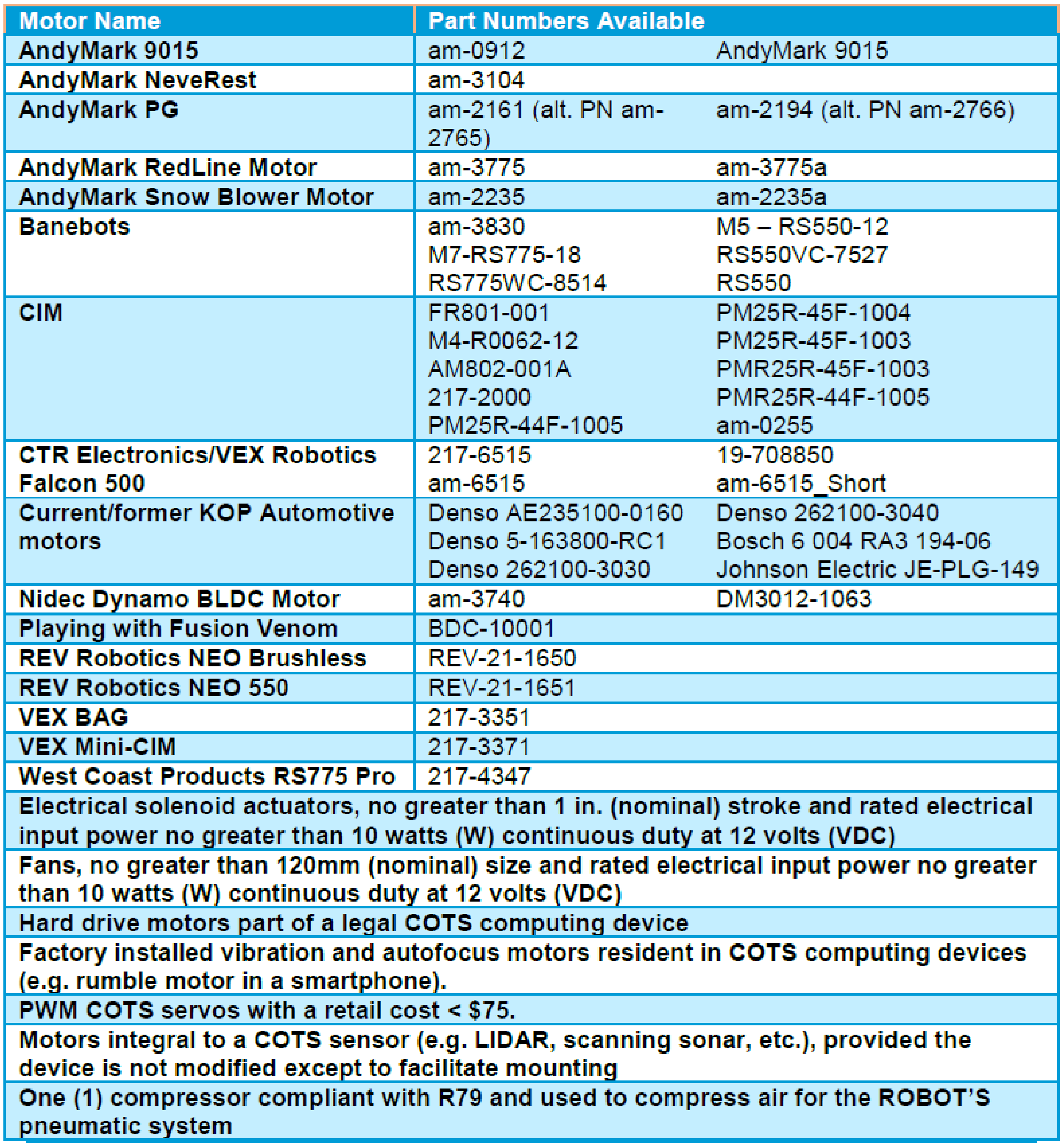
Brushed DC Motors are internally commutated motors that run on direct current. The brushes transfer electrical energy into the rotating coils through a mechanical commutator (explained here & here & here). Brushed motors can be rotated by connecting it directly to a battery. They are normally controlled with a speed controller but can be controlled with a relay. The brushes contacting the rotor mean that this type of motor is less efficient, is prone to high wear, and has more points of failure than a brushless motor.

Brushless Motors are externally commutated motors. They are generally more efficient and require less maintenance than brushed DC motors. Since they have no parts that contact other parts they have a much longer lifespan. Brushless motors require a special motor controller to make them rotate. This video explains how they work.

Stepper Motors are DC motors that move in discrete steps. They have many coils that can be energized in sequences to make the motor move one step at a time. At this time there are no stepper motors on the FRC list of approved motors.

Motors are generally selected based on how fast they turn and/or how much torque they generate. Consideration must also be given to how the motor is mounted and how to adapt the shaft to the mechanism.

The list of legal motors varies by year. The following is the section of the 2020 manual that listed what motors you could use on your robot:



Motor Name	Part Numbers Available	
AndyMark 9015	am-0912	AndyMark 9015
AndyMark NeveRest	am-3104	
AndyMark PG	am-2161 (alt. PN am-2765)	am-2194 (alt. PN am-2766)
AndyMark RedLine Motor	am-3775	am-3775a
AndyMark Snow Blower Motor	am-2235	am-2235a
Banebots	am-3830 M7-RS775-18 RS775WC-8514	M5 – RS550-12 RS550VC-7527 RS550
CIM	FR801-001 M4-R0062-12 AM802-001A 217-2000 PM25R-44F-1005	PM25R-45F-1004 PM25R-45F-1003 PMR25R-45F-1003 PMR25R-44F-1005 am-0255
CTR Electronics/VEX Robotics Falcon 500	217-6515 am-6515	19-708850 am-6515_Short
Current/former KOP Automotive motors	Denso AE235100-0160 Denso 5-163800-RC1 Denso 262100-3030	Denso 262100-3040 Bosch 6 004 RA3 194-06 Johnson Electric JE-PLG-149
Nidec Dynamo BLDC Motor	am-3740	DM3012-1063
Playing with Fusion Venom	BDC-10001	
REV Robotics NEO Brushless	REV-21-1650	
REV Robotics NEO 550	REV-21-1651	
VEX BAG	217-3351	
VEX Mini-CIM	217-3371	
West Coast Products RS775 Pro	217-4347	
Electrical solenoid actuators, no greater than 1 in. (nominal) stroke and rated electrical input power no greater than 10 watts (W) continuous duty at 12 volts (VDC)		
Fans, no greater than 120mm (nominal) size and rated electrical input power no greater than 10 watts (W) continuous duty at 12 volts (VDC)		
Hard drive motors part of a legal COTS computing device		
Factory installed vibration and autofocus motors resident in COTS computing devices (e.g. rumble motor in a smartphone).		
PWM COTS servos with a retail cost < $75.		
Motors integral to a COTS sensor (e.g. LIDAR, scanning sonar, etc.), provided the device is not modified except to facilitate mounting		
One (1) compressor compliant with R79 and used to compress air for the ROBOT'S pneumatic system		

(Chart from 2020 FRC Game Manual)

The CIM, 775 Pro, BAG, and mini-CIM are often overwhelmingly the most commonly used motors on FRC robots. Recently the new brushless Neo, Neo 550, and Falcon 500 have become commonly used. Each of these motors has different specifications, which can be found on their respective product pages from vendors such as VEX Pro and AndyMark. The most useful specs are often rotational speed (rpm), torque (Nm, oz-in., in-lbs), and power (Watts). The most common motors and some of their specifications and unique properties are summarized in the table below:

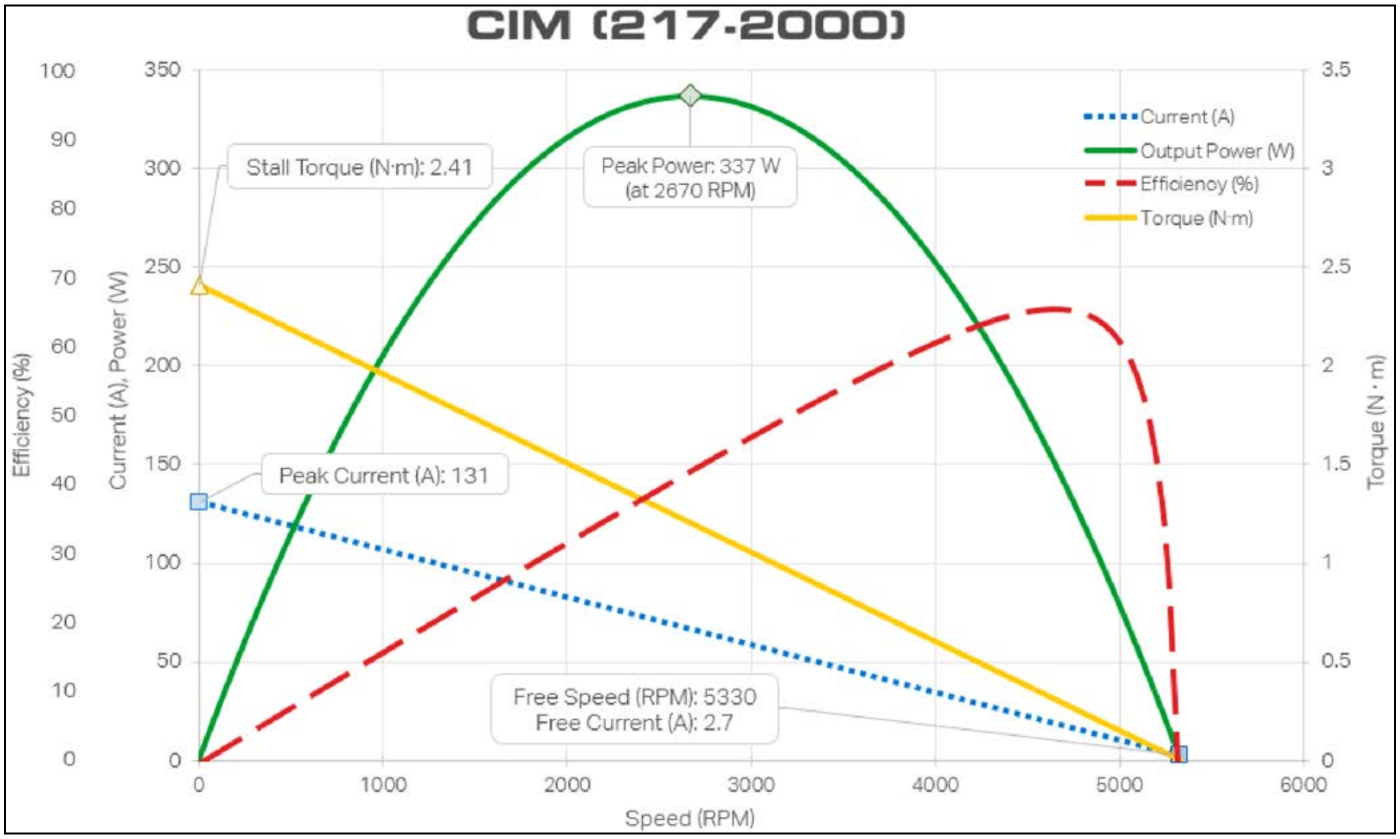
Motor	Speed (rpm)	Torque (in-lbs)	Peak Power (W)	Common Applications/ Special notes	Image
BAG	13,180	3.5	149	Intakes, small arms/elevators. Small and durable package similar to a CIM.	
mini-CIM	5,840	12.4	215	Drivetrains, arms/elevators. Designed as a drop-in, more efficient, slightly less powerful replacement for a CIM. Has the best properties for handling high temperature and stall torques.	
CIM	5,330	21.33	337	Drivetrains, arms/elevators. The only motor that has recently had a legal limit to how many you can use on a robot. Unlimited in 2018 on.	
775 Pro	18,730	6.28	347	Intakes, shooters, arms/elevators/ drivetrains (with caution). The 775 Pro is easier to damage when stalled, as it is only cooled when it's spinning. The most power dense brushed motor option.	
775a RedLine	21,020	~6.19	382	New motor for the 2019 season, is faster and more powerful than the 775 RedLine.	
REV NEO	5,676	23	406	New brushless motor for the 2019 season. Is designed as a drop-in replacement for a CIM.	
VEX Falcon 500	6,380	41.5	783	New brushless motor for the 2020 season. Highest power motor currently legal in FRC.	
Neo 550	11,000	~8.59	279	Current smallest brushless motor legal for FRC.	

4.1.1 Simplified Motor Selection

With so many options for motors, it becomes a challenge to select the proper motor for your application. This problem becomes more challenging outside of FRC when you have nearly endless motor options to choose from. The table above shows what you might want to consider. Often, you will have a mechanism in mind, with an approximate speed and torque already determined. You can do this step by drawing out your mechanism and doing some simple calculations for speed and forces using simplified geometry. Excel is a powerful tool for these calculations.

Through software and a motor controller, you can give a motor different voltage inputs up to 12V. Each type of motor will give you a different speed based on the load (torque) it experiences. You can find graphs showing this data for a specific motor at a specific voltage. VEX Pro has good data from motor testing. There is a lot of information involved in selecting the correct motors but it is most important to focus on torque and RPM, and power. To learn the basics of motor curves, watch this video from 973 RAMP.

Here's what the graph for a CIM at 12V looks like:



(Photo from https://motors.vex.com/vexpro-motors/cim-motor)

As a simplification, speed and torque are a tradeoff. As the torque increases (more load on the system), the speed decreases. Power is speed times torque. Somewhere in between stall (no rotation and maximum torque) and free speed (the speed a motor runs with no load, i.e. zero torque), there is a peak in power. The relationship between each of the graphed quantities above is different for each type of motor and different voltages.

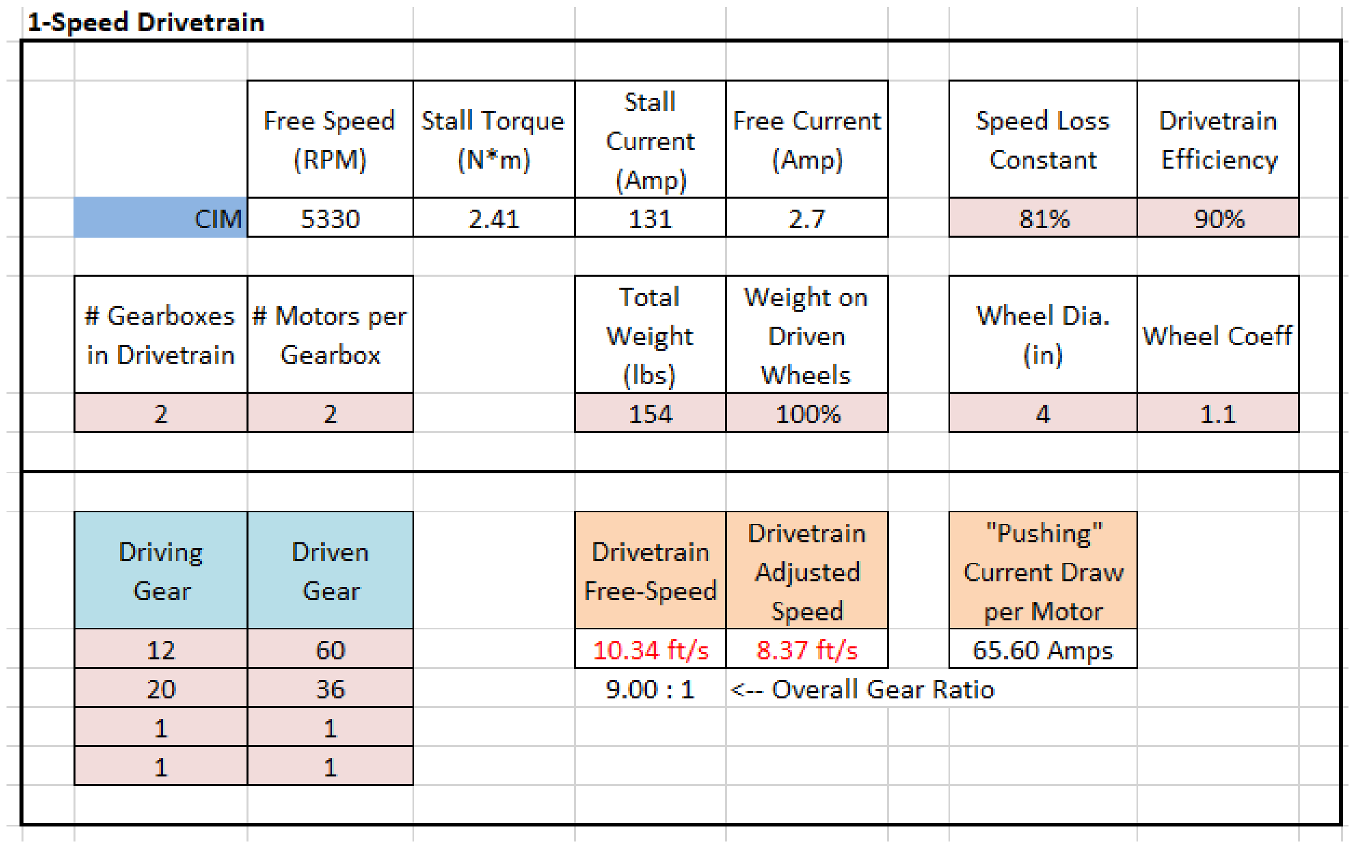
In order to design a good mechanism for an FRC robot, you can make some assumptions based on previous designs. For example, we know a roller with a 775pro can pick up just about anything. You can get away with just assuming you'll use a 775pro for this application, and move on to ratio selection. The same goes for CIMs/mini-CIMs on your drivetrain. One special note, however: If you put two motors on a mechanism instead of one, you get twice the torque and consequently twice the power, but the same speed (with greater acceleration, since acceleration is directly proportional to force, f=ma).

4.1.2 Ratio Selection

Most of the time, motors run faster than you need in your design. In order to slow a motor down, and consequently get more torque, you need some sort of **reduction**. This reduction can come from gears, belts, chain, etc. (any of the torque transfer methods described later). To pick the right ratio for your design, you'll need to do some math.

This is where Microsoft Excel comes in handy. This basically turns into a unit conversion problem, like dimensional analysis you have done/will do in chemistry or physics. Start with your motor's free speed from a chart on the product page. This is in rotations per minute (rpm). Your goal is to convert it to the speed of motion on your mechanism. This will likely be a linear speed, like how fast your drivetrain goes, how fast your intake sucks in a ball, or how fast your lift is. On lifts, it can be more helpful to see how long it will take the arm/elevator to go all the way up.

In the excel calculation below, an input speed, wheel diameter, and ratio (broken down into its stages) are entered, and a speed is output in feet per second. This could be for a drivetrain or an intake. The spreadsheet is parametric; your input numbers are parameters used in the calculations to get an output speed. It will update and recalculate as you change these parameters. It's also common practice to add a Speed Loss Constant to account for the fact that the motor will not run at full free-speed under load.



1-Speed Drivetrain

	Free Speed (RPM)	Stall Torque (N*m)	Stall Current (Amp)	Free Current (Amp)	Speed Loss Constant	Drivetrain Efficiency
CIM	5330	2.41	131	2.7	81%	90%

# Gearboxes in Drivetrain	# Motors per Gearbox	Total Weight (lbs)	Weight on Driven Wheels	Wheel Dia. (in)	Wheel Coeff
2	2	154	100%	4	1.1

Driving Gear	Driven Gear
12	60
20	36
1	1
1	1

Drivetrain Free-Speed	Drivetrain Adjusted Speed	"Pushing" Current Draw per Motor
10.34 ft/s	8.37 ft/s	65.60 Amps

9.00 : 1 <-- Overall Gear Ratio

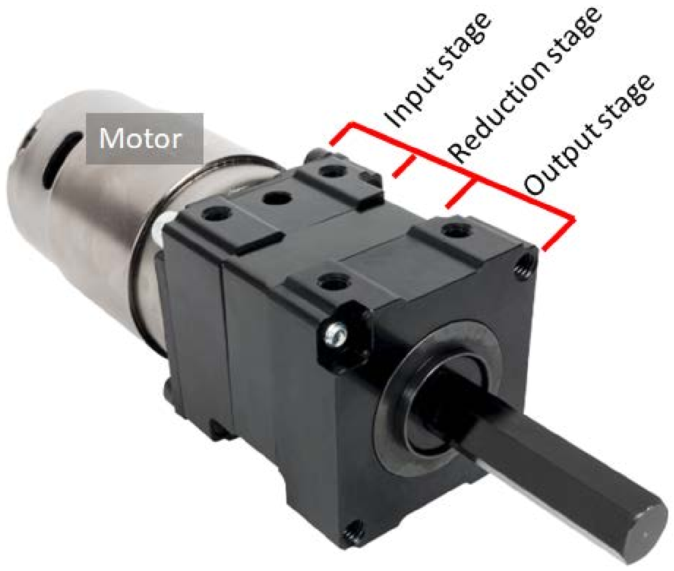
(Screen capture from JVN's Design Calculator)

You can create your own spreadsheets, or you can use the design calculator created by JVN. To see more of how these kinds of calculations are done in Excel, check out the RAMP 973 videos about it. Team 3847 Spectrum has a great robot design spreadsheet found here that has a calculator similar to the JVN calculator, and gives you more freedom in your variables.

4.1.3 Planetary Gearboxes

VersaPlanetary

The VersaPlanetary gearbox is one of the most popular and versatile methods of getting a reduction from a motor in FRC. A VersaPlanetary has three basic sections that you assemble together to get various ratios. There's an input stage, reduction stages, and an output stage, that stack up neatly. The flexibility to change each of these for different motors, ratios, and output shafts makes them a great choice for prototyping.



The input stage includes a plate onto which you bolt a motor. There are different plates for different motors. This plate is then bolted to an aluminum input stage block.

The reduction stage includes the gearset that changes the ratio. There are different gear sets that you can swap out. The ratios for a single stage can be between 3:1 and 10:1. You can have more than one reduction stage by just stacking up more stages. The total ratio is the product of the stages. There is a load rating guide on the VEX product page that tells you what ratios will work without damaging the gearbox. You can push these limits in low-load applications, when you just want something to go slower, with more precision, as Team 118 did on their 2013 robot's shot deflector (or "macker") which was a 1000:1 gear ratio.

The output stage includes the output shaft, supported by bearings in another aluminum block. You can install different output shafts, depending on the application. The most common output shaft used is 1/2" hex, but you can also use 3/8" hex or 1/2" round with a key, or 3/8" ThunderHex, among other options.

Keep in mind that when using versaPlanetaries, it's best to minimize the number of reduction stages in the gearbox. This maximizes efficiency. In addition, when you have different ratios stacked up to get a compound ratio, put the higher reduction stage closer to the motor, where it experiences less load. A 10:1 is the weakest gearset, so it should be put closer to the motor input. To learn how to assemble a VersaPlanetary gearbox, use the directions from VEX. These gearboxes are by far the best gearboxes for prototyping because of the large variety of reductions and motors that can be used with them.

UltraPlanetary Gearboxes

UltraPlanetary gearboxes are available from Rev and work with 550 series motors, such as the neo550. They are very small and lightweight, but don't offer the same number of ratios as the VersaPlanetary gearbox. The output shaft of the gearbox is currently sized for a 5 mm hex (UltraHex from Rev can be used to adapt it to a 1/2" hex). It is also possible to mount directly to the face of the output cartridge instead of using the 5 mm hex output. As can be seen from the load ratings guide, the UltraPlanetary Gearbox should be used in lower load applications.

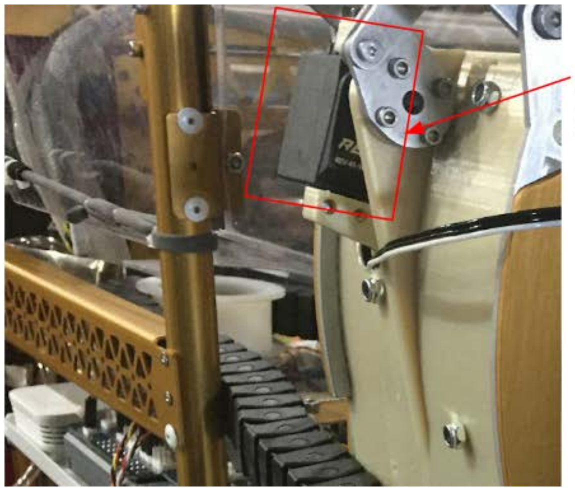
AndyMark Sport Gearboxes

The AndyMark sport gearbox is a heavier, beefier planetary gearbox option. The gearbox structurally benefits from having a single piece housing, but this does slightly limit its versatility. If you desire to increase the ratio of an existing gearbox, AndyMark provides an external 4:1 stage that can be added to any of the other ratios. However, this does forfeit some of the advantages of the standard one-piece housing. The Sport gearbox can be bought as either the CIM Sport (for CIM style motors), 57 Sport (for 550 or 775 series motors), or Falcon Sport (for the Falcon motors).

4.2 Servos

Servos are essentially small motors that can only rotate a set number of degrees. The REV Smart Robot Servo however, can rotate continuously, and can be used as a small motor. They have very little torque compared to most motors, but can be useful for small actuations. All servos used in FRC are controlled via PWM. Servos are useful for small actuations not requiring much torque. For example, Team 118 used a REV Smart Robot Servo to actuate the adjustable hood on their 2017 robot's shooter.

Linear servos are also available that can be used in places a small pneumatic cylinder would normally be used. They do not provide very much force but can be used for small mechanisms such as releases for spring loaded mechanisms for teams that want to avoid using pneumatics on their robot. Linear servos must also follow the rules in the game manual for normal rotary servos.



REV Smart Robot Servo driving an over-center linkage on Team 118's 2017 shooter hood.

4.3 Bearings

Bearings support rotating shafts and are used to reduce friction in power transmission mechanisms.

An assortment of specialty bearings.

4.3.1 Radial Ball Bearings

Radial Ball Bearings: used to support radial loads on shafts with minimal friction. Bearings should be retained in some way, whether that is pressing bearings in, using the heads of bolts to keep the bearing from falling out, or features on the shaft to retain the bearing. VEXpro and AndyMark stock most of the bearings typically used in FRC, and McMaster-Carr is a great source of less commonly used ball bearings in a large variety of sizes. McMaster-Carr also has the CAD files of all their bearings and lists all of the specs for their bearings, the most useful being max speed and load ratings.

4.3.2 Thrust Bearings

Thrust Bearings: support axial loads on shafts. Think of this as pushing on the face of the bearing. Thrust bearings are commonly used in swerve drives, as the bearing used for the rotation of the wheel module needs to support the weight of the robot. However many times a regular ball bearing is sufficient to support thrust loads, such as on a standard drivetrain. When turning with a standard WCD, the wheels on one side of the robot get pushed into the bearing race, This is an axial force, but is negligible, and standard VEXpro or AndyMark bearings can take this small thrust load. Forms of thrust bearings include needle roller thrust bearings and tapered roller thrust bearings. Many bearings such as tapered roller bearings, angular contact bearings, and four point contact bearings (X-bearings) can support thrust and radial loads. Kaydon Bearings has a good guide here if you want more information on the many types of bearings available.

Timken Tapered Roller Bearing from NASA’s Space Exploration Vehicle.

Fully assembled Timken tapered roller bearing with outer race.

4.3.3 Linear Bearings

Linear Bearings: used to support linear motion mechanisms, such as elevators. There are many types of linear bearings such as Delrin/Acetal Igus linear rails commonly included in the FRC Kit of Parts, linear ball bearings, and round linear rails and bearings frequently seen on CNC routers and other machine tools. One thing to note is that linear ball bearings such as the one pictured below should be used on steel as they will dig into soft aluminum. Linear bearings found on McMaster-Carr will list what materials with which they can be used. Typically bearings that can be used with aluminum do not have rollers and have a low friction plastic such as PTFE/Teflon or Delrin inside them.

Linear Roller Bearing with grooves for retaining clips.

4.3.4 One-Way Bearings

One-Way Bearing/Sprag Clutch: bearings which have internal features that cause them to lock and transmit torque when spinning in one direction, but free spin in the other direction. These can be used on FRC robots as a non-actuated Power Take-Off (PTO). Team 118 used a one-way bearing on their original climber in 2017, in which their climber shaft rotated when the robot drove backwards, and the bearing spun freely when the robot drove forwards. This mechanism can be seen here. Team 971 also had a similar climber in 2017 that was powered off their ball intake, using this one-way bearing.

One-way bearing used by Team 118 on their 2017 climber.

4.4 Bushings

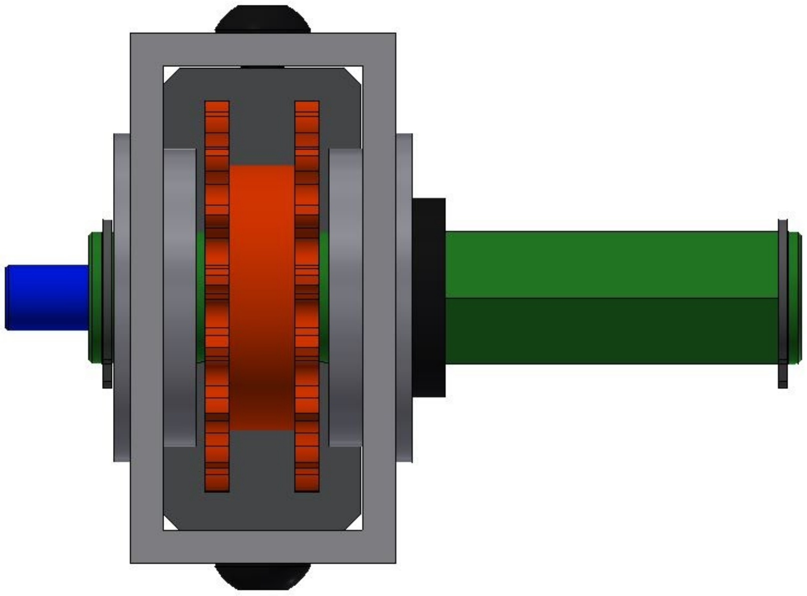
Bushings: similar to ball bearings, but have no roller elements and are usually plastic or bronze. They are most useful for low speed applications, as they have more friction than ball bearings, but can also handle much higher loads. They are also lighter and cheaper than ball bearings. Bushings are also known as **plain bearings.**

An assortment of bronze, plastic, and Oilite bushings.

4.5 Shafts

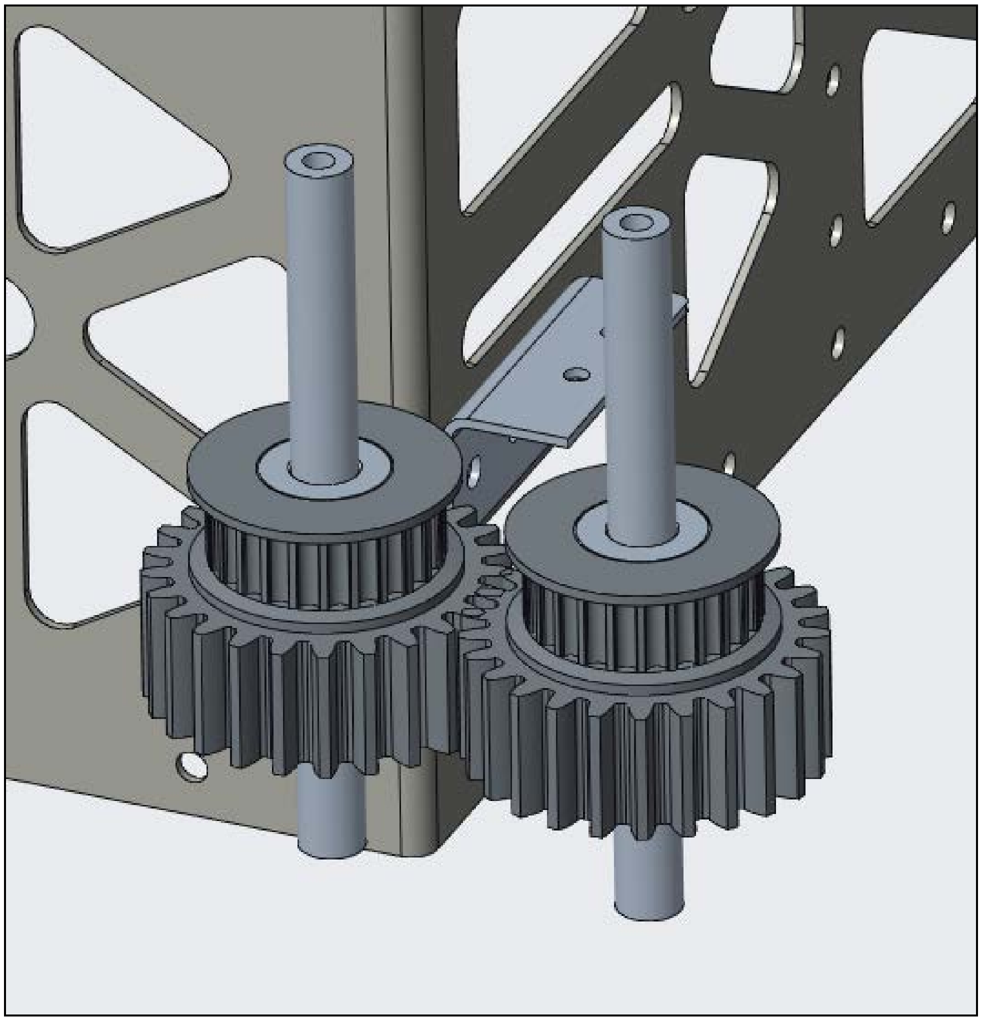
4.5.1 Live Axle vs. Dead Axle

Live axle: torque is transmitted through the shaft to a wheel, gear, sprocket, etc. This requires one of the shaft profiles listed below to transmit this torque. The shaft rotates.



An example of a live axle system in which torque is transmitted through the shaft.

Dead Axle: bearings are inside of the wheel, gear, sprocket, etc., and torque is transferred directly to the wheel by a sprocket or gear bolted to the wheel. Dead axles are more structural than live axles because they can be used as standoffs. The shaft does not rotate; something on it rotates.


2 dead axle gear/pulleys in the roller claw on Team 118’s 2018 robot.

4.5.2 Shaft Profiles

Methods for Torque Transfer-Shaft Profiles

To transmit torque from a shaft to a wheel, gear, pulley, sprocket, etc. in a **live axle** system, the shaft must have some kind of feature to allow it to transmit torque. There are many different ways to do this, with the most common in FRC being hex shaft, thunderhex shaft, and keyed round shaft.

Types of Shaft:

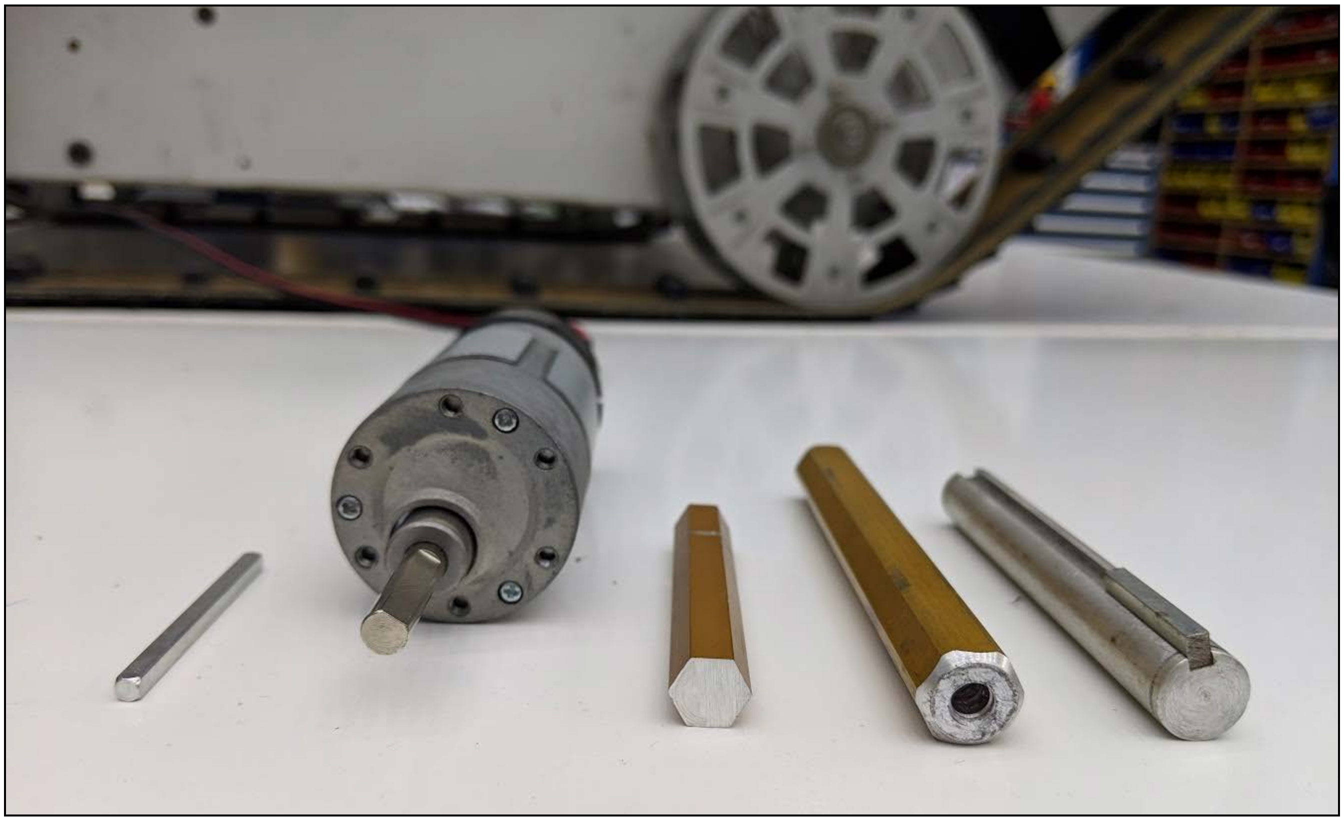
Hex shaft: hexagonal shaped. The two sizes typically used in FRC are 1/2” and 3/8”. It is recommended to purchase a broach in both these sizes to use on custom parts. A broach is pressed through a part to get a desired shape. A broach is the most precise way to get a good fit for a hex shaft in parts, however, a hex can be machined using other tools and methods.

ThunderHex: hex shaft with rounded corners that allow it to fit in a round bearing. This makes it much simpler to align with bearings. It comes in two sizes 3/8” and 1/2”. Half-inch (1/2”) ThunderHex has a .201” through hole that allows you to tap it for a 1/4-20 bolt and can fit in a 13.75 mm round bearing or bushing. Note that 13.75 mm (~0.54”) will appear close in size to a 0.5” round bearing. The 3/8” ThunderHex has a .165” hole to tap for a 10-32 bolt and can fit in a 10.25 mm round bearing or bushing. It is only sold by VEXpro.

Round Keyed: a keyed shaft has a slot cut into it that a machine key fits into to transmit torque. The key slot has to be broached into the part you are trying to transmit torque to. Due to the increased availability of gears, wheels, etc. that fit on hex shaft, keyed shaft is rarely used in FRC anymore. An example of where keyed shaft may be used would be on a shooter where high speed round bore bearings are needed. CIM, mini-CIM, and REV Neo motors have an 8mm round keyed output shaft.

D profile: a round shaft that is placed on a mill to cut a flat into it. This makes the shaft “D” shaped. This method isn’t ideal for FRC robots, however there are D shaft universal female output shafts for VersaPlanetaries and VersaHex adapters. These are mostly aimed at the FTC market, where D profile shafts are common.

Square: this is the profile used by VEX EDR hardware. It is exclusive to VEX EDR and certain tools. The VEX Robotics System can be useful for prototyping FIRST robots. There are 1/4” square universal female output shafts for VersaPlanetaries and VersaHex adapters that allow you to adapt typical FRC hardware to high strength VEX Edr shaft along with ThunderSquare bearings to fit the high strength VEX EDR shaft.



1/8” Square, 6mm D-shaft, 3/8” Hex Shaft, 1/2” ThunderHex Shaft, and 1/2” Keyed Shaft with a Machine Key.

4.6 Methods for Torque Transfer

4.6.1 Gears

Gears: basically rotating wheels with teeth that mesh to transfer torque, and can be used to reduce the speed and increase the torque of a system. They can also be used to increase the speed of a system, however that is rarely done in FRC. Two meshing gears will rotate in opposite directions.

Gear Terms

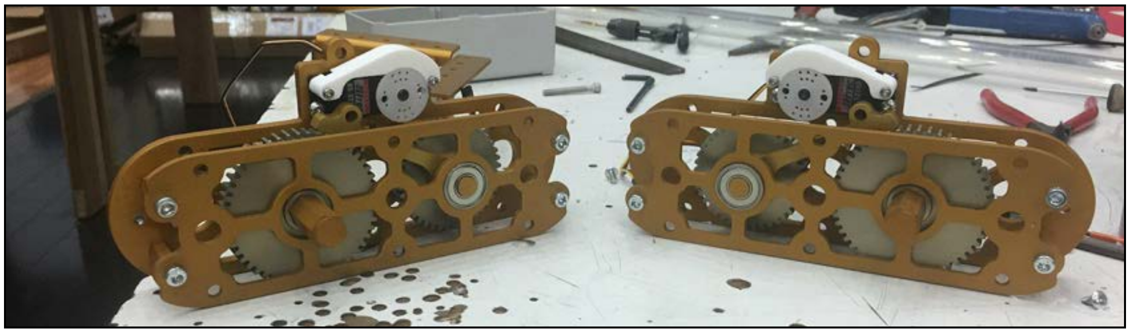
Pressure Angle: is the angle between the direction the teeth exert a force on each other and the line joining the centers of the two gears. There are 2 pressure angles commonly used, 14.5 degrees and 20 degrees. Two meshing gears must have the same pressure angle.

Diametral Pitch (DP): the ratio of the number of teeth per inch of the gears pitch diameter. Typical COTS FRC gears are 20DP or 32DP. A higher DP means finer, smaller teeth. Two meshing gears must have the same pitch.

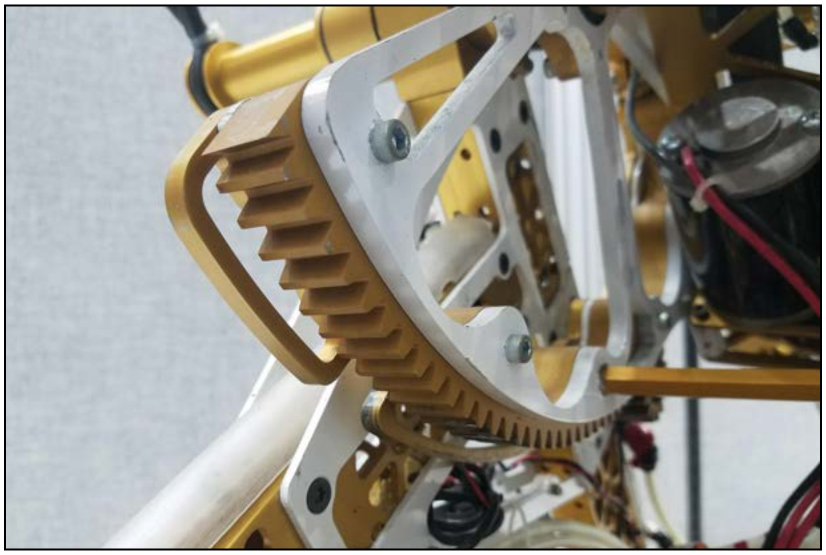
The commonly used VEXPro line of gears use 20dp and a 14.5° pressure angle.

Types of Gears

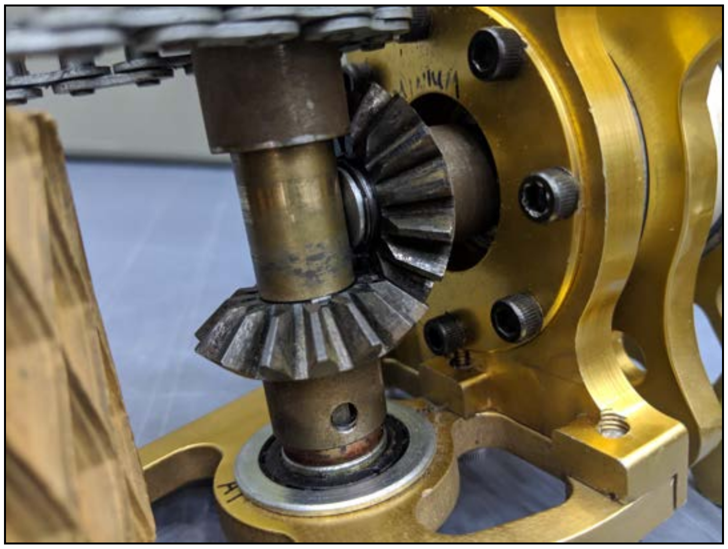
Spur gears: (the most basic types of gears) the teeth are parallel to the axis of rotation. Spur gears can only be used on shafts that are parallel. A small spur gear that is the driving gear, usually on the output of a motor, is called a **pinion**. A large gear that a pinion meshes with is sometimes referred to as a **bull gear.** VEXpro and AndyMark sell gears that are 20DP and have a 14.5 degree pressure angle. VEXpro and AndyMark also sell gears that are 32DP and have a 20 degree pressure angle.


Team 118’s 2017 Drivetrain Gearboxes.

Sector Gears: spur gears that are not complete circles and are used for mechanisms that do not need to rotate a full 360 degrees. An example of their use is on arms that only need to rotate less than 360 degrees. Sector gears can be created in a variety of methods, such as cutting a COTS gear to the right sector angle, taking the outline of a COTS gear and only machining the area you need, or designing a custom gear all together using the design tools on Rushgears.com.


Custom sector gear for the shooter head pivot on Team 118’s 2014 robot.

Bevel Gears: conical shaped gears with most of their tips cut off. If you extended the cone, the vertices of the gears must coincide. They can be used to transfer torque between shafts angled at anywhere from 1 to 179 degrees from each other. When 2 bevel gears are on shafts that are 90 degrees to each other, and each gear has the same number of teeth, they are called miter gears. Usually the vendor will supply drawings showing the center distances needed for the gears to properly mesh and at what angles they mesh. For example here is the drawing for VEXpro bevel gears.

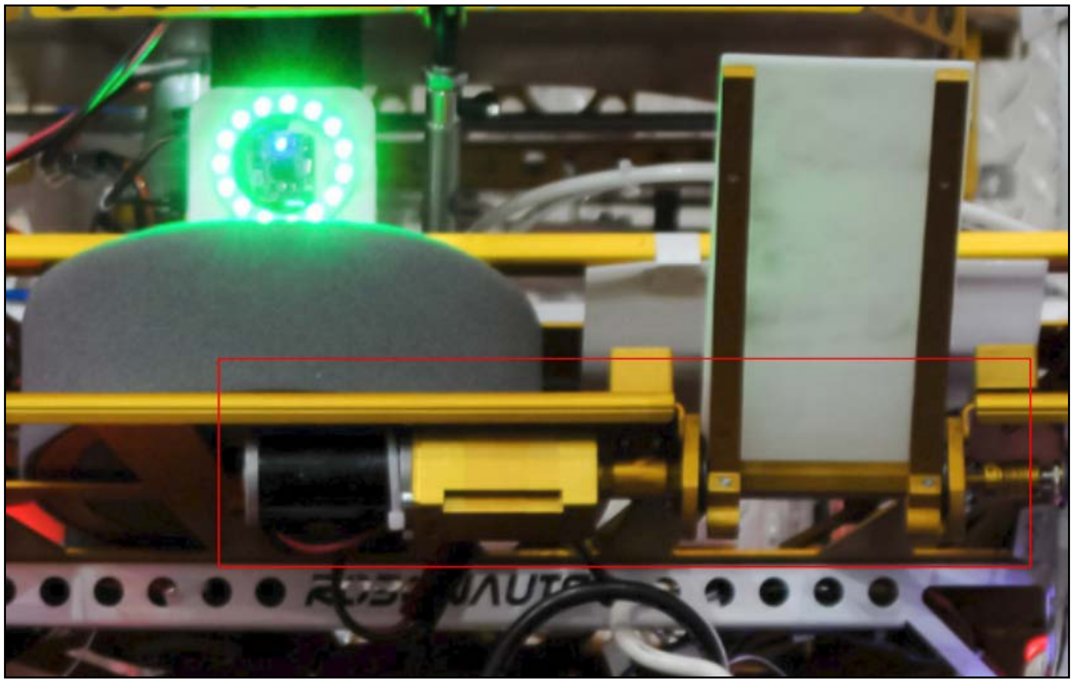

Bevel gears in Team 118’s 2008 crab module.

Worm Gears: used to get very high reductions in a compact space and transmit torque between shafts that are at a 90 degree angle to each other (perpendicular). There are two components, the worm and the worm wheel. A worm looks very similar to a screw and the worm wheel looks very similar to a spur gear. The big advantage of a worm gear is that it is difficult to back drive. The worm can drive the worm wheel but the worm wheel takes a large force to drive the worm. This allows mechanisms using them to be self-locking and hold their position.

Rack gear: a linear bar with teeth that allows for linear motion. Racks are driven by pinions.

Planetary Gear Sets: used to get very large gear reductions in a very compact space. They are composed of a sun gear at the center of the assembly, which meshes with usually 3 to 4 planet gears, which then mesh with a ring gear. Planetary gear sets are used in VersaPlanetaries to get reductions of 3:1 to 100:1 in a very compact volume. While higher ratios are possible by adding more stages, they tend to bind or break often. It’s best to design for their use within this range. It is possible to carefully design outside the recommended range. In 2013, Team 118 used a 1000:1 versaplanetary, however this was only acceptable due to almost no load on the gearbox.

Planetary gear set from a VersaPlanetary.


Team 118's 1000:1 Planetary gearbox.

Strain Wave Gears (Harmonic Drives): a type of gearbox that consists of a wave generator in the center that flexes a flex spline, and a spline on the outside that it meshes with called the circular spline. Harmonic drives can get very high reductions in a very light package. The other notable feature of harmonic drives is that they have no backlash. The largest reason they are almost never used in FRC is that they are very expensive, priced around $600 on average, and the benefits are just not worth it. The two main manufacturers of strain wave gear sets are Harmonic Drive and Cone Drive. There are some designs out there for 3D printable harmonic drives, however they usually cannot handle high torque loads.

Harmonic Drive used in NASA's Space Exploration Vehicle.

Cycloidal Gearboxes: cycloidal gearboxes consist of an input shaft driving an eccentric bearing that is in a cycloid disc. The cycloidal disc meshes with the stationary ring gear, just like a planetary gearbox. The cycloidal disc has holes in it, which move pins or rollers that are attached to the output plate. Cycloidal gearboxes have been used on FRC robots, however they usually are not necessary. They are not very efficient, especially if just using pins without rollers on the output plate. Another disadvantage of cycloidal drives is that due to their eccentricity, the gearbox tends to vibrate, especially at high speeds. This will cause the components in the gearbox to experience accelerated wear. Cycloidal gearboxes are frequently believed to be non-backdrivable, however this is NOT true. The only thing that would keep a cycloidal gearbox from backdriving is it's inefficiency. Under a high enough load, a cycloidal gearbox will backdrive.

Designing For Gears

Center-to-Center: as with most methods for torque transfer, you need to use a calculated center-to-center distance to effectively and efficiently transmit torque. You can use the design calculator here to calculate the center-to-center for 20DP or 32DP gears.

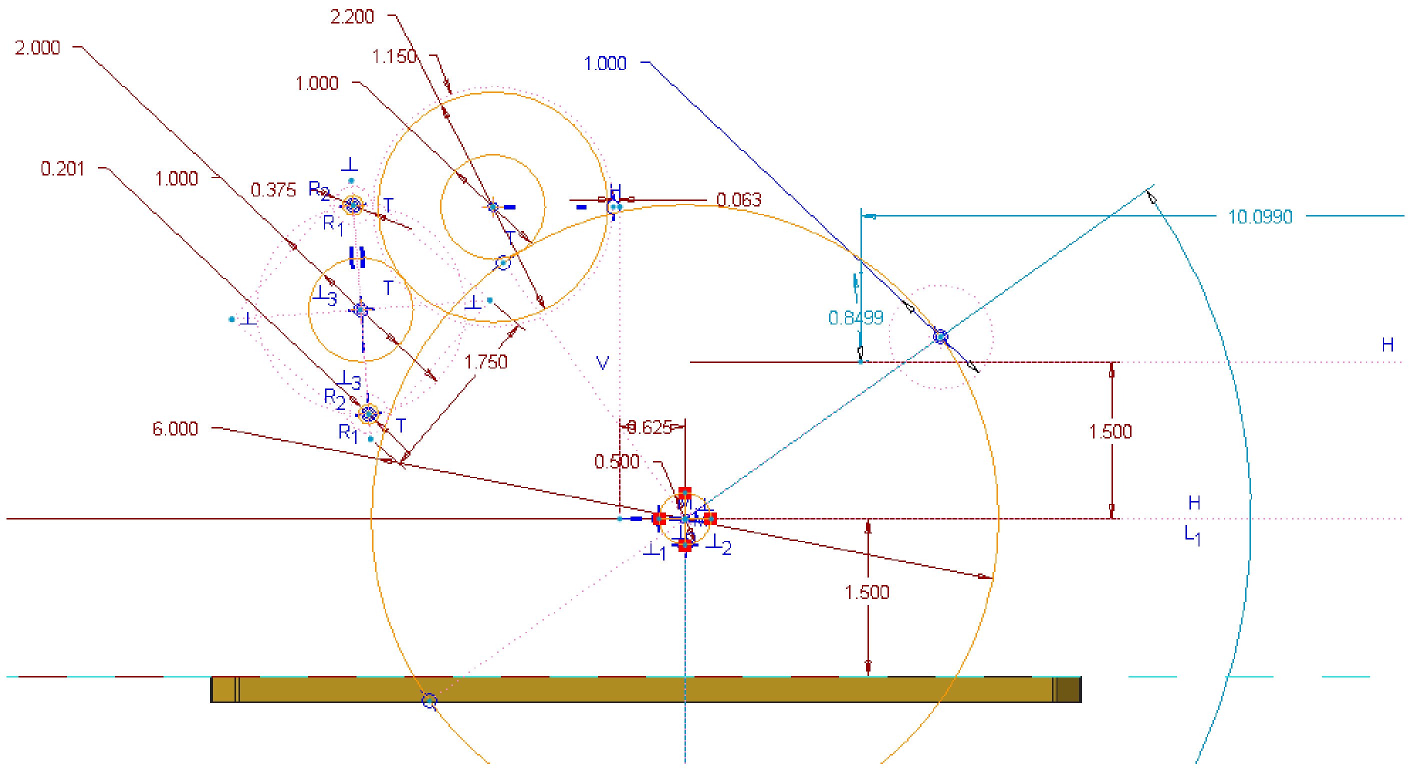
You can also do these calculations by hand, by using the equation: $C = \frac{N_1 + N_2}{2P}$, where C is the center distance, N_1 and N_2 are the number of teeth on each gear, and P is the diametral pitch of the gears. Note: this means that for a calculated center-to-center, only the sum of the teeth on the two gears matters; a 50t to a 50t has the same center-to-center as a 60t and a 40t.

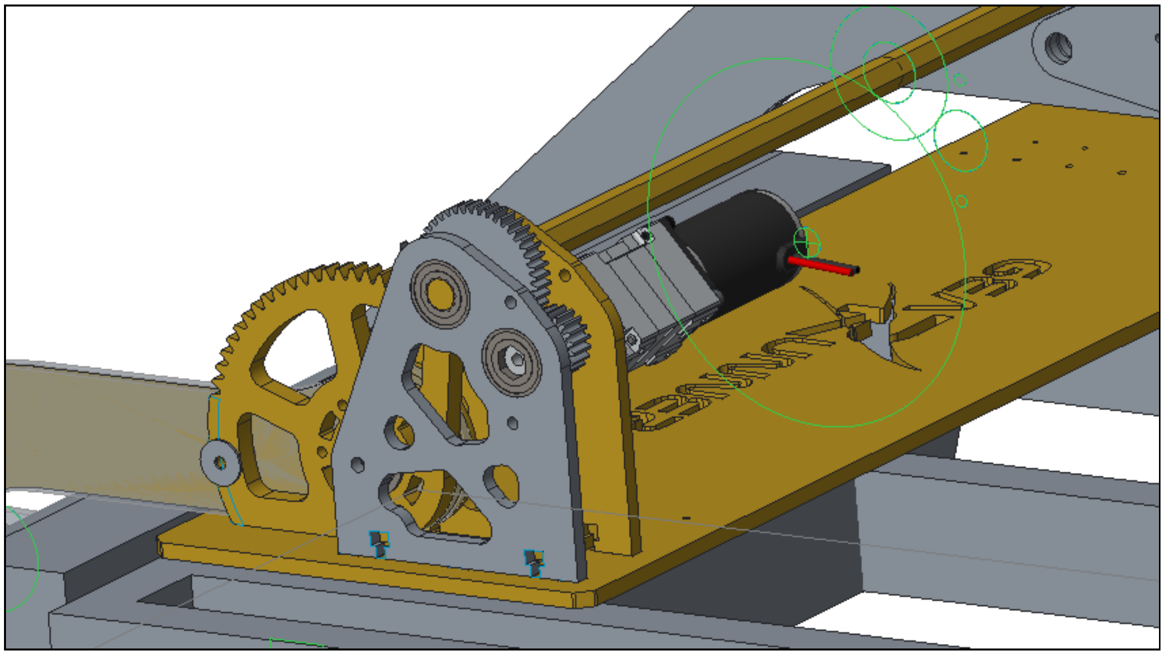
Often, a small amount of tolerance is added by increasing this distance by 0.003” or so (done automatically in most center-to-center calculators like the one above). The purpose of doing so is to decrease friction and increase efficiency, with the addition of a small amount of **backlash**. Backlash is “slop”; a lag or dead spot between the input and output. It can make precise control of such a mechanism more difficult, but is acceptable (to an extent) in fast mechanisms, like drivetrains and intakes.

Pitch Diameter: Another useful way to design with gears is by drawing them in your sketch. You can model the exact distance between two gears by drawing two circles tangent to each other. You should dimension your circles to what is called the **pitch diameter**. The pitch diameter is the diameter at which a gear interacts with other gears. This diameter is listed in the product specifications for a gear.

The equation for the pitch diameter is $P_d = \frac{\#\,Teeth}{Diametral\ Pitch}$ By drawing the gear in your sketches, you can get a more complete picture of a mechanism at the sketch-level, letting you more effectively plan out where to place other components, like standoffs. If you sketch a gearbox using pitch diameters, you will have to use creative construction geometry if you wish to add a 0.003” tolerance. It may also be useful to draw the outer diameter (OD) of the gear as well.

The two images below show the single sketch used to lay out the gearing for the intake arm on Team 118’s driver training robots.





Designing Custom Gears

If you want to design a mechanism with gears that are not commercially available, you can create custom gears using the design tools on RushGears.com. The custom gear tool on RushGears.com will produce a 3D CAD file of your custom gear after you input the diametral pitch, pressure angle, number of teeth, gear thickness, bore, and keyway location. Once you download the CAD file of the gear, you can modify it as you wish by adding features such as bolt patterns and lightening holes. Team 118 has successfully used the CAD generated from Rush Gears for custom turret gears on some of their past robots.

Designing Sector Gears

If you wish to design a sector gear that meshes with standard COTS FRC gears, you can import the step file of a COTS gear and create a new gear by projecting the teeth that are needed.

Another way to design a sector gear would be to create a custom gear on RushGears.com using the method mentioned above, and then cut away the teeth that are not needed. This is how the arms in the above Team 118 driver training robots were done.

4.6.2 Chain

Chain Drive: sprockets with a certain number of teeth attached to a shaft in a live axle system or to a wheel on a dead axle system. The sprockets are then connected with chain. This is the type of drive system used on bicycles. There are two types of chain typically used in FRC, #25 and #35 ANSI roller chain. #25 is lighter, while #35 is capable of transferring higher loads. Teams, such as 1619, have successfully used bicycle chain, which is lighter and smaller than #25 chain. Chain is ideal for transferring power over a large distance with high loads.

Types of Chain Links:

Master Links: a master link is a chain link that allows you to connect two ends of a chain run without using a chain tool. This makes it very easy to put together chain runs quickly, however they should be avoided on final robots, due to being much weaker than normal chain. A chain run with a master link will always fail at the master link. To avoid this use the DarkSoul, Rev, or VexPro chain break to create your chain runs.

DarkSoul chain break for #25 chain.

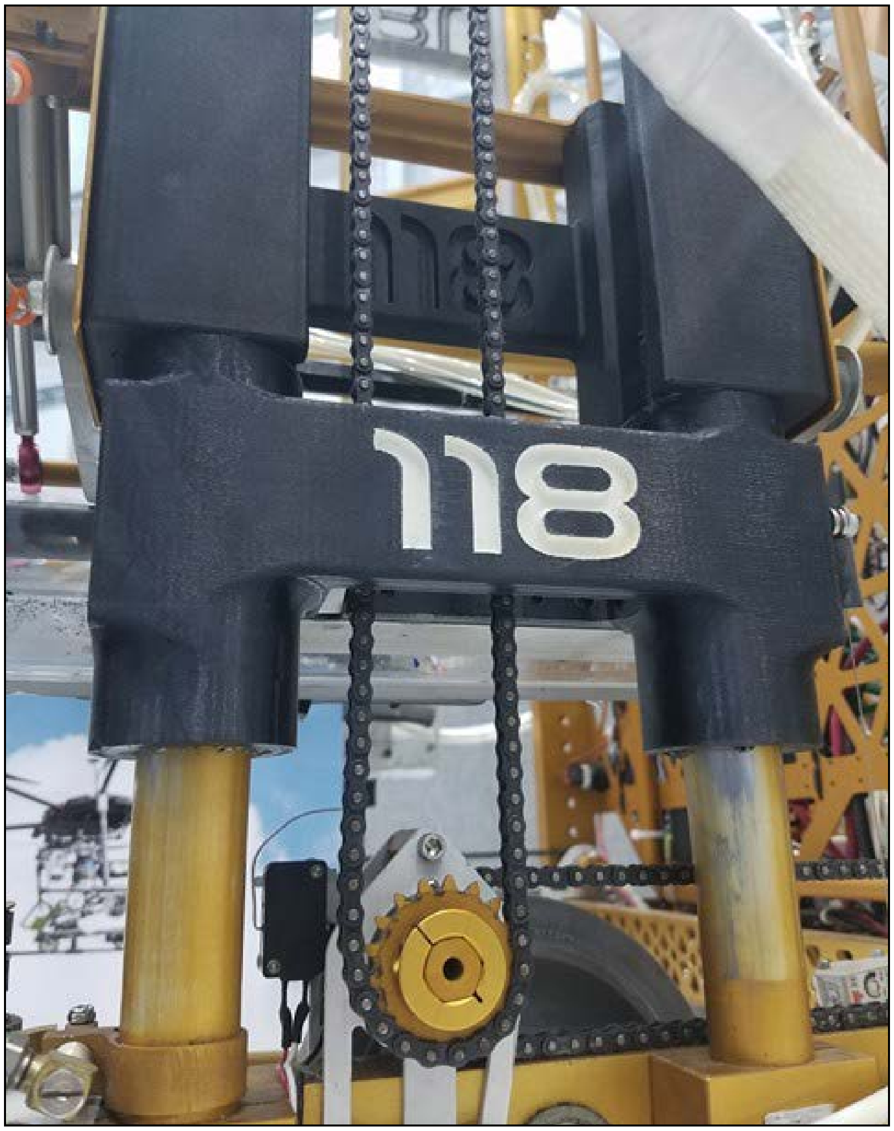
Half Links: allow you to create a chain run with an odd number of links. However just as with master links, they are weaker than normal chain, so try to avoid them by designing for an even link center to center.

Types of Chain:

#25 Chain: most commonly used on robots. #25 chain is also known as 1/4 pitch due to each link being 1/4” long. The most common place chain is used is on drivetrains, where belts may not be strong enough or are too large, and gears are impractical to transmit torque to each wheel due to the distance between the wheels. Since 2013, Team 118 has put their drivetrain chain inside a 1”x2” box tube. This is called “Chain in Tube.” The largest sprocket that can be used in a Chain in Tube drivetrain with a 2” tall tube that has 1/8” thick walls is a 17-tooth sprocket. A stronger alternative to #25 chain is #25H chain, which has thicker plates. #25H is a drop-in for #25, meaning no changes need to be made to switch between them. However, some double sprockets, specifically COTS options, do not fit #25H, so verify what chain you are using before attempting to use these sprockets.

#35 Chain: larger, stronger, but also significantly heavier than #25 chain. #35 chain is also known as 3/8 pitch because each link is 3/8” long. #25 chain is usually sufficient for FRC purposes, however #35 chain has its uses occasionally. Because #35 chain is more forgiving to misalignment and nearly impossible to break in FRC, it is great for teams without precision manufacturing available.

Bike Chain: bike chain is very strong and is narrower than #25 chain. It comes in a variety of widths from 3/32”-3/16” internal widths. Teams, such as 1619, have successfully used bike chain in their drivetrains in the past. Bike chain is not as simple to use as the above chain types because sprockets with shaft bores commonly used in FRC are not readily available (such as 1/2” hex). Most bike chain has a pitch of 1/2”.



#25 chain run at the bottom of Team 118’s elevator from 2015.

Designing for Chain and Sprockets

When designing a mechanism that uses chain to transmit torque you should calculate the correct center-to-center for the sprockets. This is the distance from the center of one sprocket to the center of another. Use the botlanta chain distance calculator to get that number. The number of links should always be even so there are no half links in your chain run, as these are quite weak and will cause your chain to fail under large loads. If the sprockets on either end of the chain run are the same, then the calculation for center to center simplifies to multiples of 0.25” for #25 chain and multiples of 0.375” for #35 chain.

Many teams add length to their Center-to-Center distances to make the chain run more efficient and effective. The amount of center distance to add varies based on who you ask, but for #25 chain it is generally around .018”, and for #35 chain it is around .012”. These are the values that Paul Copioli, former president of VEX Robotics, recommends. To get an absolute best value, you can make test parts with center distances increasing by a few thousandths each, and use the one that appears to work the best. Generally, if you use the above values, you will have no problems.

With that said, Team 118 has successfully run drivetrains with no center add since 2013 with no problems using their chain-in-tube setup. It is best to experiment yourself to find what works best for your mechanism and your machining resources.

Sometimes the COTS sprockets just aren't ideal for your mechanism. If this is the case, you can machine custom sprockets. For example, Team 118 waterjets the 17-tooth double sprockets they use in their chain-in-tube drivetrains, broaches them, and turns them down on a lathe. When designing custom sprockets, you should be aware of the thickness the sprocket needs to fit inside the chain. For #25 chain, the thickness should be .125”; for #35 chain, the thickness should be .1875”.

For teams that do not have precision machining resources the VEXpro VersaBlocks can be used to implement chain runs that have optimal chain tension. These work by clamping around box tubes and just need a large clearance hole for the shaft drilled into the tube and can be done with a hand drill. For low to medium torque applications the clamping force from the blocks is usually enough to keep the shafts in the correct position. For higher impact mechanisms such as drivetrains the WCP Cam can be used to ensure proper chain tension without worrying about the bearing block moving.

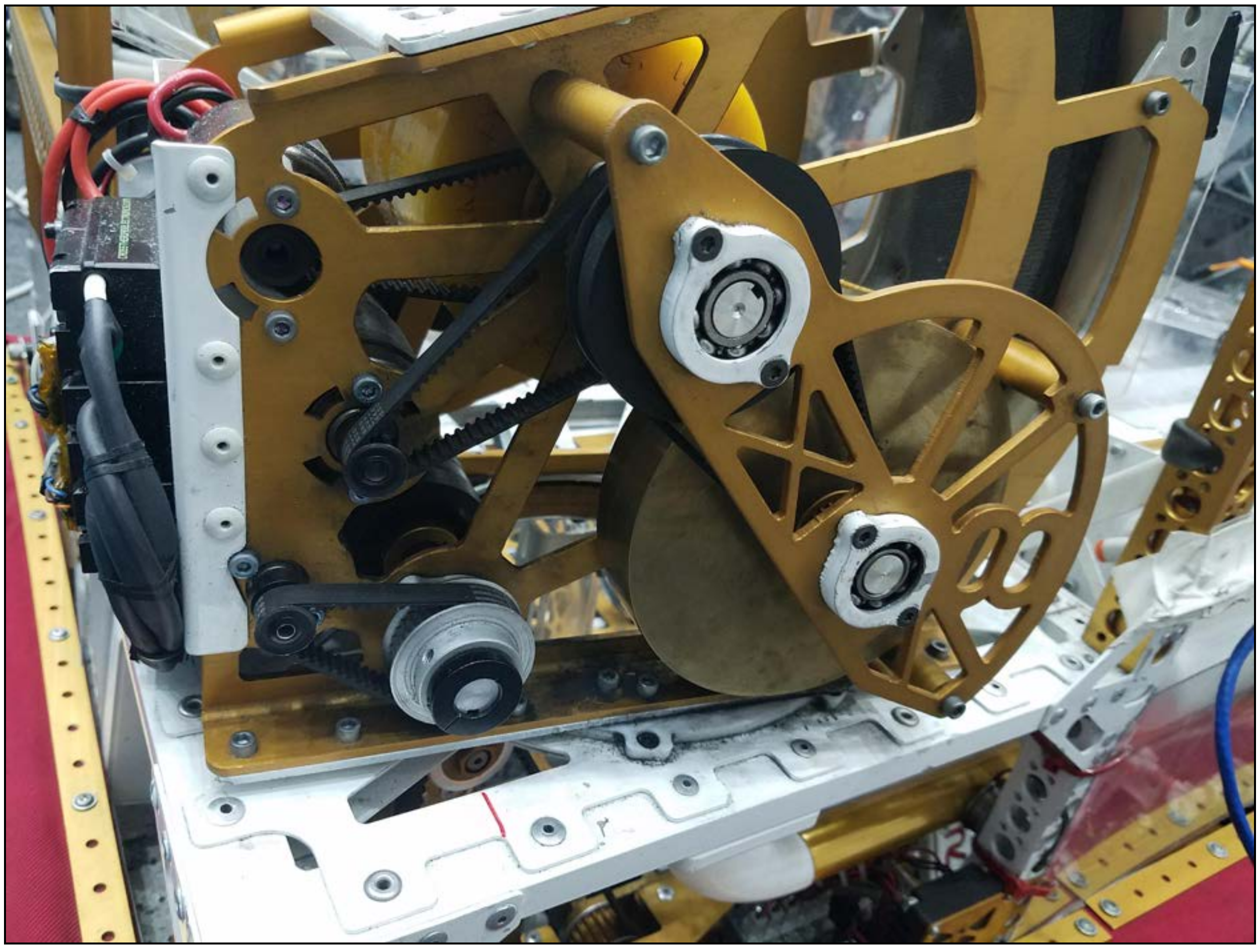
4.6.3 Belts

Belt Drive: belt drive is very similar to chain drive, however belts are used in place of chain and pulleys are used in place of sprockets. Belts are typically used to transmit lighter torque loads. They can be used in high load applications, such as drivetrains, however belts capable of handling the same loads as chain are much wider than chain. As with chain, there are many different types of belts. The ones typically used in FRC are 5mm HTD and 3mm GT2. If purchasing outside of the FRC supplier ecosystem dominated with 5mm HTD and 3mm GT2, it may be worth purchasing 5mm GT2 belts and pulleys as they have a higher torque capacity than the other two profiles.

Types of Belts:

5mm HTD: these belts have round teeth that allow for higher loads and a pitch of 5mm. They come in two widths 9mm and 15mm from FRC suppliers. The 9mm belts are usually general use belts and the 15mm belts are used almost exclusively on drivetrains and other high load applications. Other width belts are available from other suppliers, such as Gates, Stock Drive Products, or B&B Manufacturing. Double-sided belts are also available from these suppliers.

3mm GT2: these belts also have rounded teeth that allow for higher loads. They have a pitch of 3mm and typically come in a width of 9mm. Other widths, along with double-sided belts, are available from other suppliers such as those named above. Compared to 5mm HTD, GT2 belts are for lighter load applications, are more efficient, and are lighter in weight. VEXpro sells 12t 3mm GT2 pinion pulleys that can be pressed onto the output shaft of RS 550 motors and 775pros.


GT2 belts used on Team 118's 2017 shooter.

Designing for Belts and Pulleys

As with chain, belts need to have the correct center-to-center to properly transmit torque. Belts, however, come in discrete lengths. To calculate the center-to-center for a certain belt length, or to determine the belt length for a certain distance use the WCP How To: Belts calculator. In certain efficiency sensitive applications, it may be advantageous to shorten the center-to-center by .005" or greater. Team 254 shortened their belt runs by .060" on the feeder of their 2017 robot. This applies to fast mechanisms, such as feeders and intakes with low torque transfer and high sensitivity to efficiency losses; this does not apply to slow mechanisms, such as arms, elevators, or drivetrains, as the higher load could cause the belt to skip if the pulleys are not spaced properly.

Another way to design for timing belts is to use the pitch length of the belt. To calculate the pitch length of a belt, you simply multiply the number of teeth by the pitch of the belt.
$Pitch\ Length = Pitch * Number\ of\ teeth$

The belt you use needs to be coincident with a pulley of a known pitch diameter.
$Pitch\ Diameter = (Pitch * Number\ of\ teeth)/\pi$
You can also usually look up the pitch diameter of a pulley on the supplier's website. For a simple belt path between two pulleys, you can draw the belt path with both pulleys drawn at their pitch diameter, with tangent lines connecting them. You can then dimension the path length to the pitch length. WCP explains how to do basic belt paths in this video.

Something cool about timing belts is that you can connect multiple shafts with them using a complex belt path. Designing a complex belt path is similar to drawing a simple belt path above, but more time consuming. WCP also has videos on how to do complex belt paths here and here.

Sometimes the COTS pulleys just aren't ideal for your mechanism, or you want to modify tension in the belt by adding or removing a tooth or two in a pulley to something not readily purchasable. If this is the case, then it is possible to 3D print custom pulleys. Pulleys printed in material such as NylonX or Onyx are usually strong enough for most applications, and are lighter than aluminum pulleys from VEXpro. Printing pulleys also allows you to integrate custom features into the pulley such as roller end caps or unique drive bores or bearing bores. You can also print pulleys that have a tighter hex bore than VEXpro pulleys to minimize backlash between the shaft and the pulley. It is also possible to purchase aluminum or steel pulley stock from suppliers, such as B&B Manufacturing, and make your own custom pulleys. There are many places from where you can purchase belts. VEXpro sells 5mm HTD and 3mm

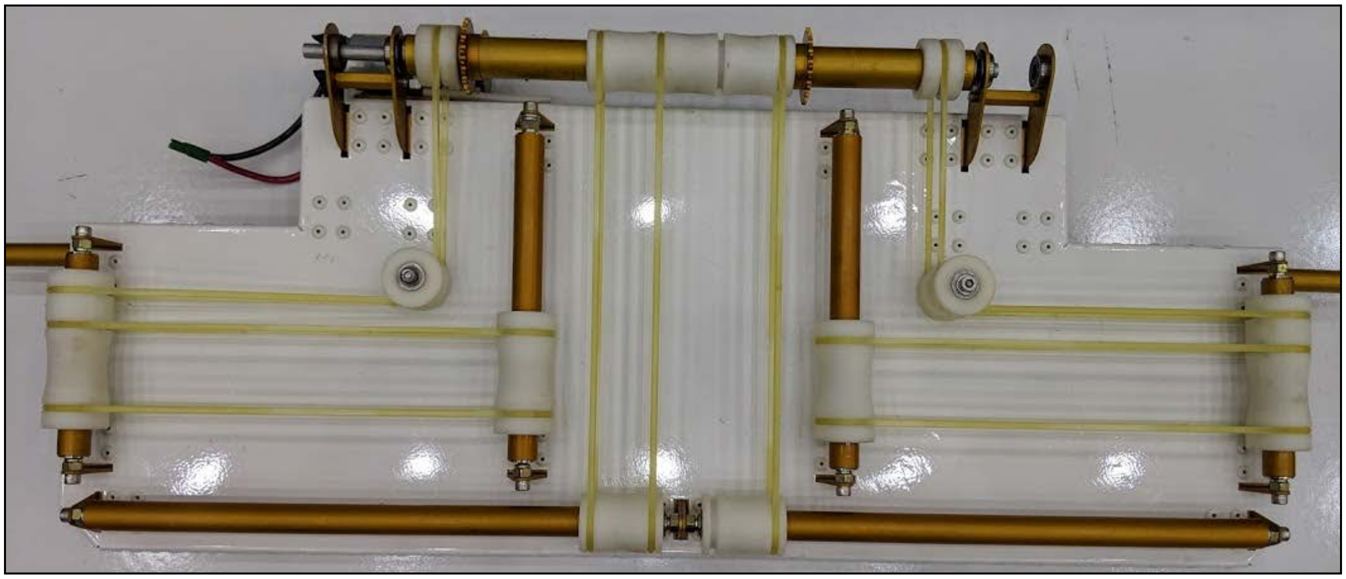
GT2, as does WCP. Gates sells a large assortment of belts, including many not mentioned here. Stock Drive Products sells GT2, HTD, XL, and MXL, along with many more. V-belt guys is another vendor of a wide variety of belts. Stock Drive also has CAD files for most of their products, which is useful for modeling your own pulleys to 3D print. You can also use a tool, such as the Parametric Pulley generator on Thingverse or the configurable pulley generator in MKCad on Onshape, to generate CAD files of custom pulleys.

4.6.4 Polycord

Eagle Tubing/Polycord: Eagle tubing (more commonly known as polycord) is a type of cut-to-length tubing used to transmit light torque loads. It is made from polyurethane, and comes in many different diameters. It is less efficient than timing belts, gears, or chain. Sizes typically used on FRC robots include 3/16” and 1/4”. It is important to note that complex polycord assemblies such as the intake below are low efficiency systems and have a lot of drag torque.

Designing for Polycord

Polycord is by far the easiest method to use when designing torque transmission; however, it is the least efficient method. It relies on the tubing being too short to transmit torque and pulls the shafts closer together. This puts uneven loads on bearings and adds a lot of friction to the system. To use polycord, the distance between pulleys doesn’t matter since the tubing can be cut to any desired length. When cutting the tubing, it should be cut around 10% shorter than the desired length to properly transmit torque. The tubing is then welded together using a tool that looks like a soldering iron with a large flat surface on the end. To weld the tubing together, you place both ends in the welder, and place the iron in between. Once heated, remove the iron, close the clamp, and hold for a few seconds. You should be able to pull on a properly welded polycord run with quite a lot of force without it breaking. An interesting feature of polycord is it acts as a clutch when stalled and just slips on the pulleys, this protects the motors that power it. Polycord pulleys can be 3D printed, however if both pulleys are plastic, the polycord will weld itself to the pulleys when stalled and fail. To prevent this, use 1 plastic pulley and 1 metal pulley, with the metal pulley as the input. To machine metal pulleys, you can grind a piece of tool steel to the right shape and use it on a lathe.


Complex polycord run that centers balls on an intake built for Team 118's 2012 robot.

4.6.5 Flat Urethane Belting

Flat Urethane Belting: useful for creating conveyor systems or intakes. It is used in a similar way to polycord; it is welded together using heat. The best pulley shape for flat belting is a convex crown pulley. This keeps the belt tracking straight without falling off the pulley. For more information on why convex pulleys are best, read this document. Flat urethane belting is sold by VEXpro here. Pulleys can be 3D printed or machined.

4.7 Motor Driven Linear Actuators

There are many different methods of getting linear motion for a mechanism. The easiest way is to use pneumatic cylinders, however there are other methods that allow you to convert the rotational motion of a motor to linear motion.

There are a few COTS solutions for linear actuators besides pneumatics. One of those is the DART Linear Actuator, and the others are COTS linear servos.

DART Linear Actuator: a COTS linear actuator sold by AndyMark and designed by iR3 Creative. It is designed to be driven by a CIM motor that drives an ACME lead screw. It is available in 3 different ratios and 2 different lengths. Another option to the DART linear

Actuator is to take a different COTS linear actuator, remove the motor, and replace it with an FRC legal motor.

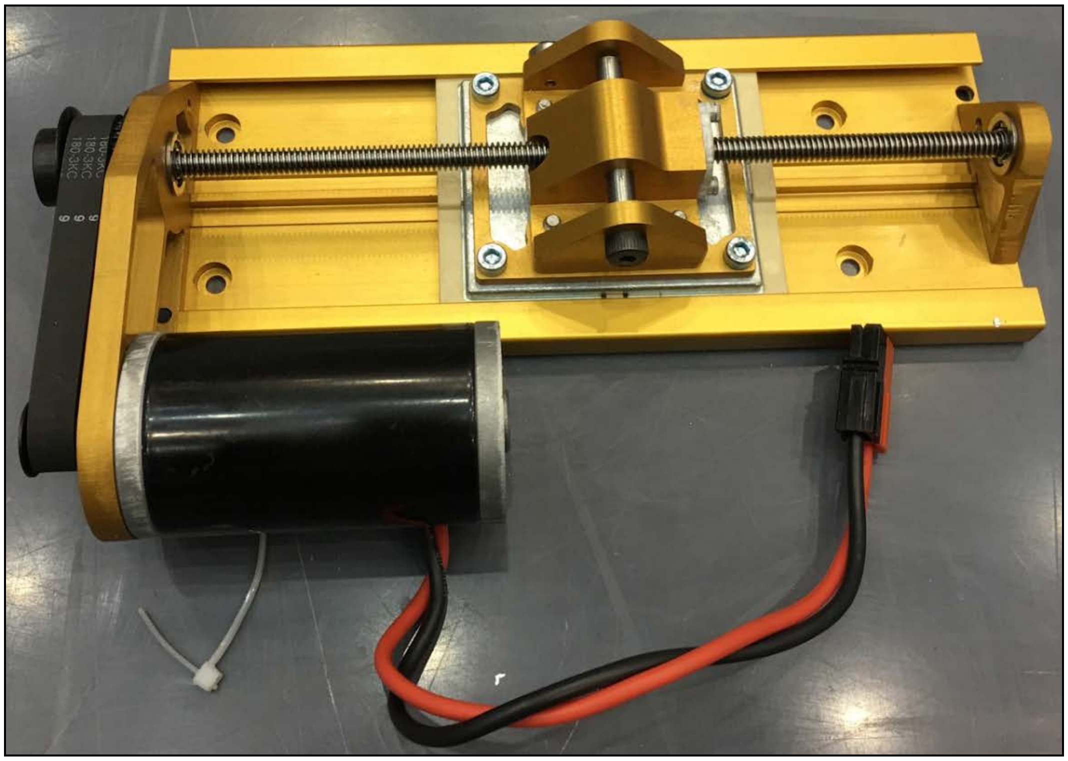
Linear Servos: useful for linear actuations requiring little force. There are many COTS linear servos available from a variety of suppliers such as AndyMark and ServoCity. Linear servos can be used in place of small pneumatic cylinders for teams trying to avoid using pneumatics. Check the FRC current year's servo rules before purchasing a COTS servo.

4.7.1 Lead Screws

Lead Screw: is a threaded rod that is spun to drive a nut. Most lead screws use ACME thread, however a standard bolt thread can technically be used as a lead screw with a large efficiency loss. Lead screws with more threads per inch, by design, automatically give you a large reduction. A lead screw torque and force calculator can be found here. Lead screws with fewer threads per inch, some of which have multiple start leads, are more efficient. Multi start threads have multiple independent threads on the shaft, this means that you can get the same linear travel with a slower rotating screw, which can help with controllability and minimize issues with high speed shaft whip. Lead screws usually need to be combined with some sort of linear bearing setup to support the mechanism. Lead screws, like bolts, are designated by their diameter and their pitch.

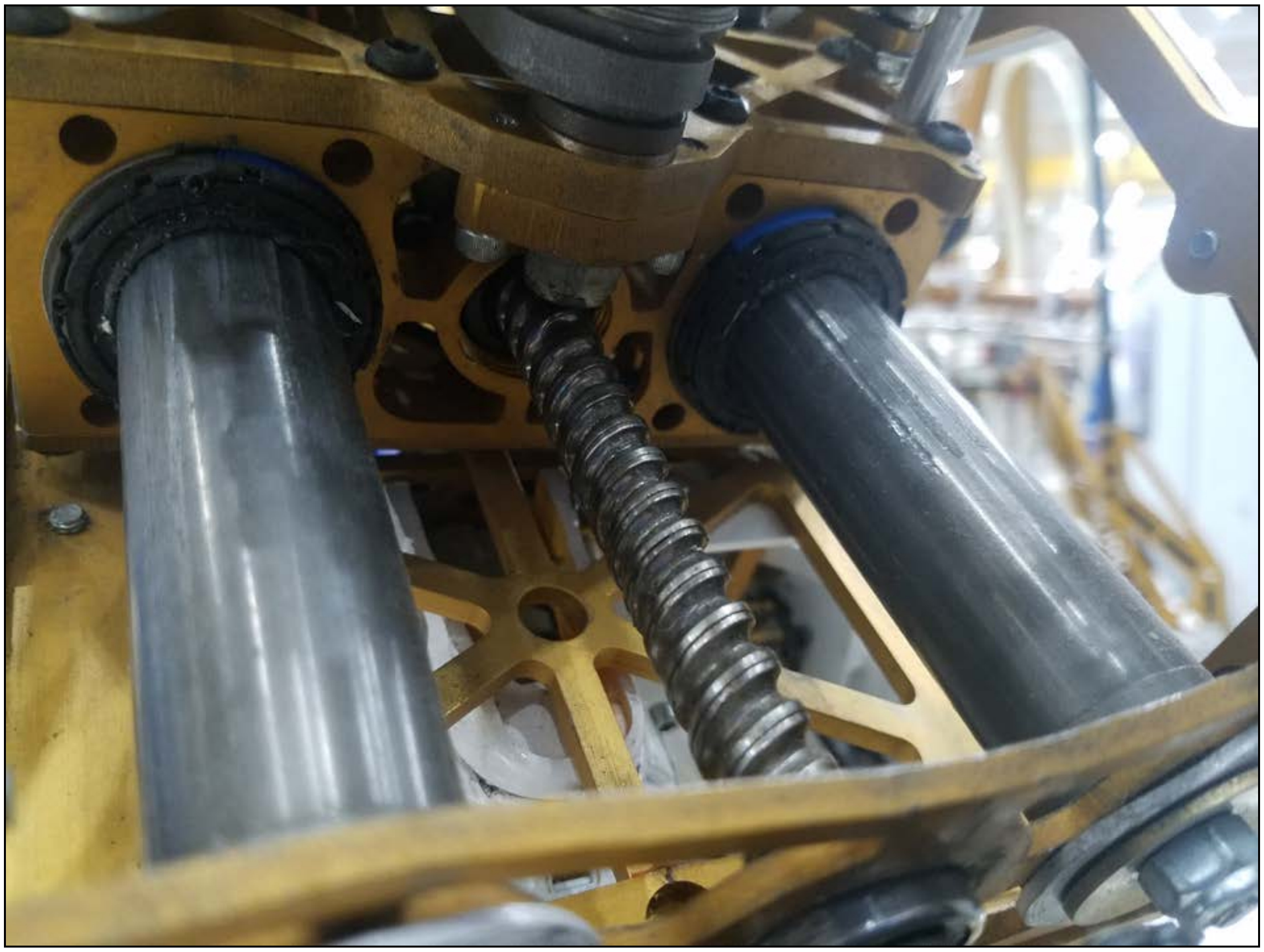
Lead Screw Basics

A common lead screw size is 1/4-16. The nut used is frequently made of bronze, or an engineering plastic, depending on the load case. Since there is sliding friction between the screw and the nut, the material the nut is made of affects the efficiency of the system. Lead screws also have a critical speed, above which the screw will reach its natural frequency and begin to suffer severe vibrations, often called shaft whip. The diameter, length, and type of end supports affect the critical speed. The formula for the critical speed of a lead screw is $Critical\ Speed = ((4.76 * 10^6)D * C)/L^2$ where L is the length between bearing supports, D is the root diameter of the screw, and C is the constant representing the type of end supports. A calculator for the critical speed of a lead screw and a list of the types of end supports can be found here. Below is a photo of team 118's shot adjustment mechanism mounted to an IGUS linear rail driven by a lead screw.


Team 118’s lead screw driven shot adjustment mechanism from their 2016 robot.

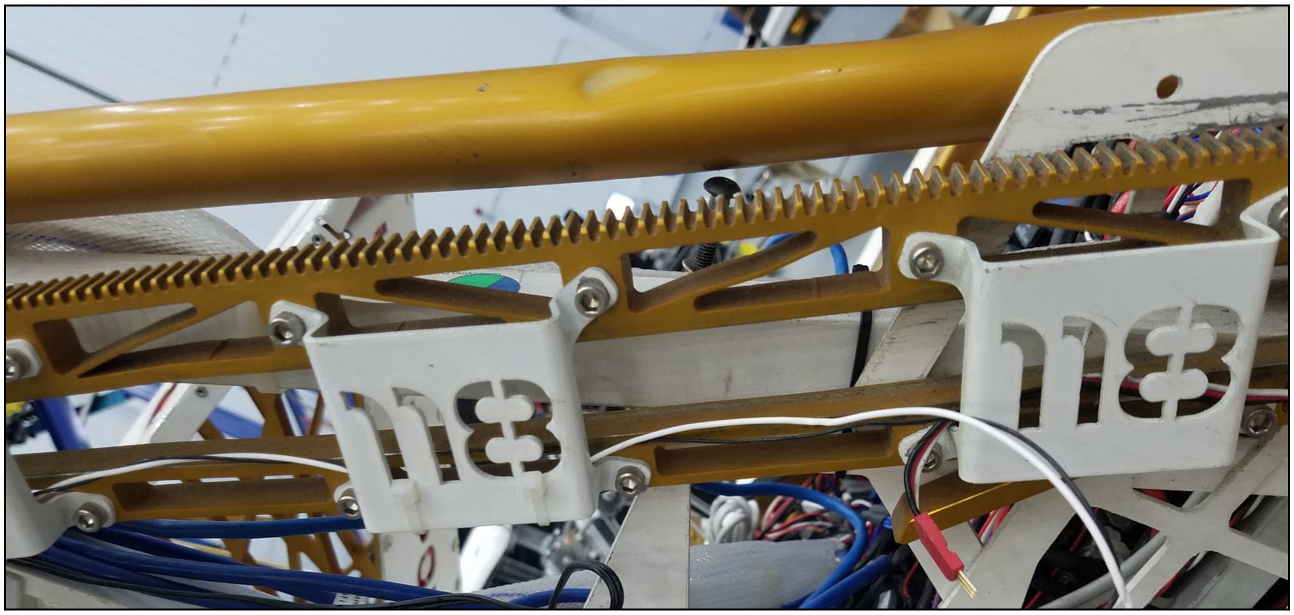
4.7.2 Ball Screw

Ball Screw: used in the same way as a lead screw, however it uses ball bearings, rolling in a helical raceway on the shaft. Ball screws are more efficient than lead screws, and can handle higher thrust loads than lead screws. Ball screws are around 90% efficient, whereas lead screws of the same size are only around 20% efficient. Ball screws are, however, more expensive and larger than lead screws.

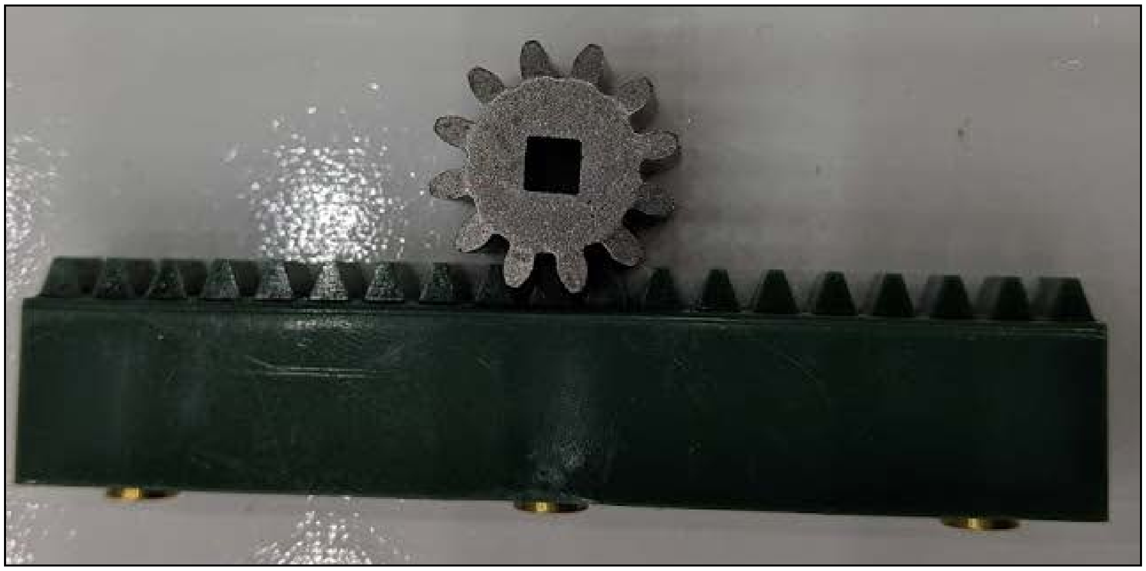

Ball screw with linear ball bearings in Team 118's 2014 shooter mechanism.

4.7.3 Rack and Pinion

Rack and Pinion: a rack and pinion consists of a pinion gear that meshes with a rack, which is basically a straight gear. Rack and pinion systems usually require some sort of linear bearing setup, in addition to the gears themselves.



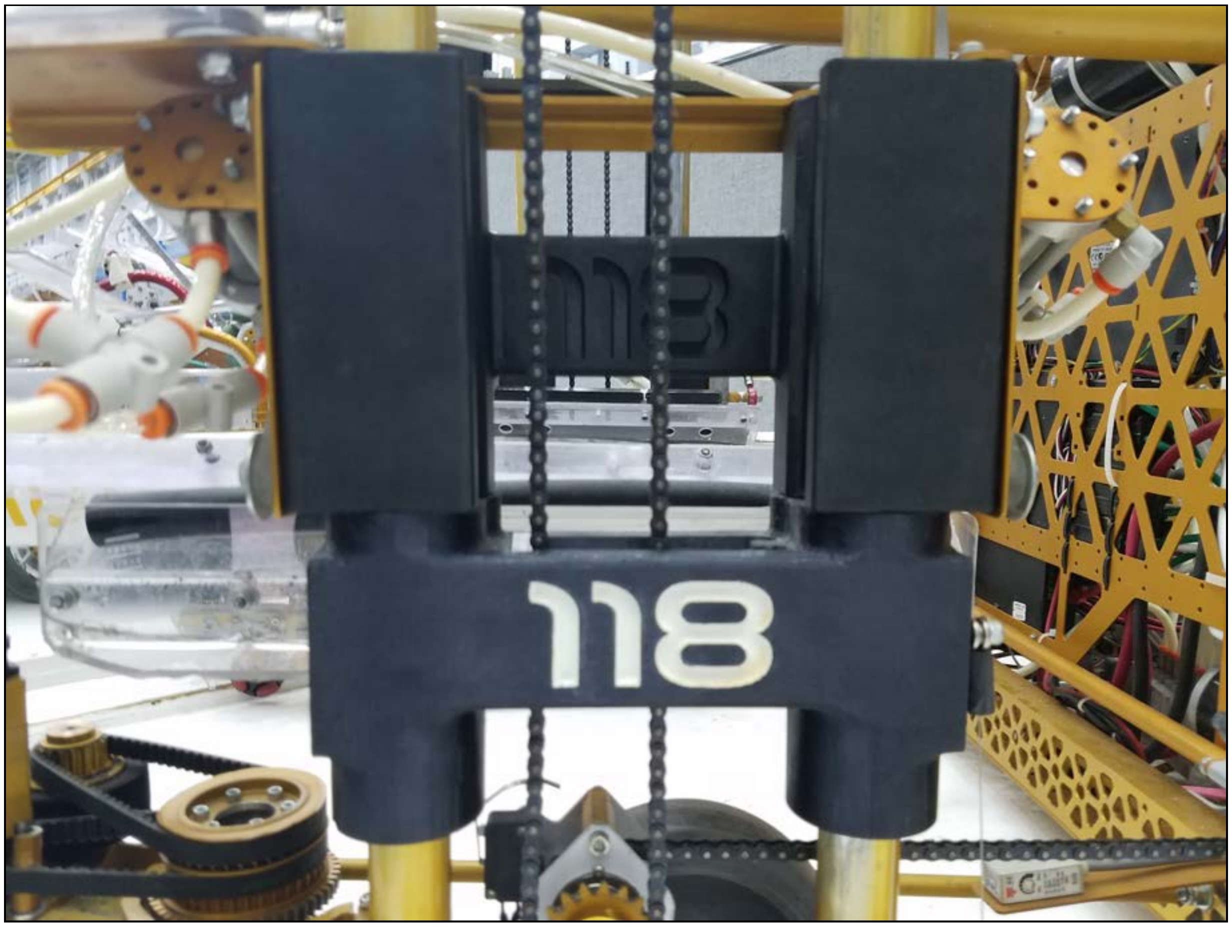
Custom rack gear on the stinger mechanism on Team 118’s 2012 robot.


VEX EDR Rack and Pinion Set.

4.7.4 Linear Elevator

Linear Elevator: typically built using box tubing, or less frequently using round tubing. In addition to the tubing, they have bearings that are attached to lift trucks. The bearing and cable setup varies depending on the number of stages, however the bearings almost always ride on the tubing. Four bearings per tube are needed to properly constrain the elevator.

Elevators are typically driven using either belts, chain, or cable, but some teams have used rack and pinions or lead screws/ball screws. Team 973's RAMP Greybots channel has great videos on how to design a cascading elevator here, here, and here. They also have a video here on how to choose the right motor for an elevator. Below is the elevator on Team 118's 2015 robot that uses 3D printed lift trucks, which are also used as the bearings.



Team 118's chain driven elevator with 3D printed elevator blocks, running on round tubing, on their 2015 robot.

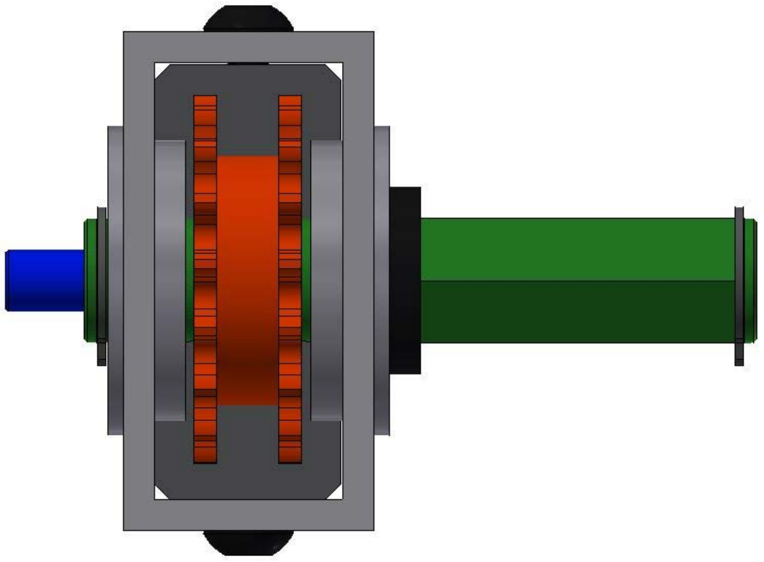
5. Mechanism Design

5.1 Drivetrains

Drivetrain: the most important subsystem on the robot. If your drivetrain fails, most likely you won't be able to effectively complete any other game tasks. This is why it is best to have a simple and reliable drivetrain. It is best to strive to never compromise the drivetrain.

5.1.1 Drivetrain Terms

Chain-in-tube: when the chain that connects all the wheels in a drivetrain is inside a box tube. The purpose for this is protecting the chain from debris, and is supposed to be worry-free. The box tube constrains the chain, preventing it from jumping off the sprockets. Chain-in-tube is not exclusive to drivetrains; it can be used on other mechanisms, as well. The downside of chain-in-tube is that it is more difficult to diagnose problems and perform maintenance and repairs. The ideal sprocket size for #25 chain in a 1"x2" 1/8" wall tube is a 17-tooth sprocket; this is the largest size that can fit, and makes it nearly impossible for the chain to jump off the sprocket. These were formerly available from 221 Robotics, and are now available from AndyMark.



Cross section of a chain-in-tube drive rail using a 17-tooth double sprocket.

Drop center: used on many robots to allow them to turn effectively. It refers to the center wheel(s) being lower than the wheels at the corners of the drivetrain. In a 6 wheel drop center, the center wheel is usually 1/16” to 3/16” lower than the outer two wheels. This means that at any time the robot is only on 4 wheels, thus reducing “turning scrub” (friction from the wheels dragging sideways).

Wheel track: the distance between the left side wheels and right side wheels. The larger the wheel track, the better the robot turns. A short wide robot will turn better than a long narrow robot.

Artificial Drop Center: with chain-in-tube, you can only achieve a minor drop center because the chain would run into the inside of the box tube if using 17-tooth sprockets. To get the effect of a larger drop center, teams have used smaller diameter wheels on the corners of the robot. Usually these wheels have to be turned down on a lathe, as the size difference in the wheels needs to be around 1/4” or less and there usually is not COTS wheels available in these specialized sizes.

West Coast Drive (WCD): has the wheels cantilevered from the frame, using live axles, and has the gearbox output shaft directly driving a wheel on each side. They are usually built using a box tube construction. However, Team 148, the Robowranglers, has made their own pseudo box tube using sheet metal parts. Typically a WCD uses bearing blocks with chain tensioners, however some teams press the bearings directly into the box tube and do not use chain tensioners. The term originates from teams in California such as 254 making them popular in the mid-to-late 2000’s.

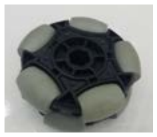
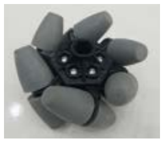
Inverted WCD: identical to WCD, except the wheels are inside the frame instead of on the outside. This makes mounting bumpers much easier, and allows you to maximize your interior volume. The one downside of this type of drivetrain is that it makes for a narrower wheel track, which makes the robot’s turning scrub slightly worse. Team 118 did this style of drivetrain in 2017 and 2018 after being inspired by Team 971’s 2016 offseason robot.

Holonomic Drivetrain: a 3 degree of freedom drivetrain. It can drive forwards and backwards, rotate, and strafe left and right. Holonomic drivetrains are highly maneuverable.

Articulating drivetrain: can switch from one drivetrain type to another, typically by using pneumatic cylinders.

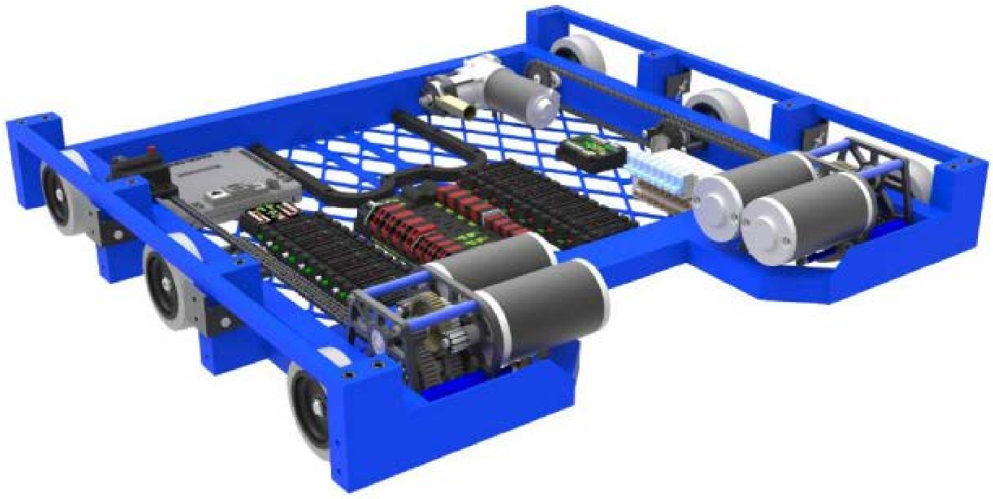
5.1.2 Wheels

Types of Wheels for Drivetrains

Wheel	Description	Photo
Colson Wheels	A standard traction wheel in FRC. They have pretty high traction, are durable, lightweight, and come in a large variety of diameters and widths.	
VersaWheels	Traction wheels sold by VexPro. They are high traction, but wear down faster than Colson wheels.	
Plaction Wheels	Wheels that have replaceable tread. They have high traction, but require more work to maintain than the wheels listed above.	
Rubber Treaded Wheels	Wheels sold by AndyMark. AKA the FRC kit of part wheels.	
Molded Wheels	Teams have molded their own custom wheels that were high traction and lightweight. Team 118 did this in 2012 and 2013.	
Pneumatic Wheels	Wheels that have tires with air filled tubes inside. Many teams used them in 2016 for crossing the obstacles on the field.	
Omni Wheels	Wheels with passive rollers that allow the robot to turn with almost no turning scrub. Used in many holonomic drivetrains.	
Mecanum Wheels	Wheels with passive rollers 45 degrees to the wheel. Allows for holonomic motion. These have proven benefits in intakes for centering game pieces.	

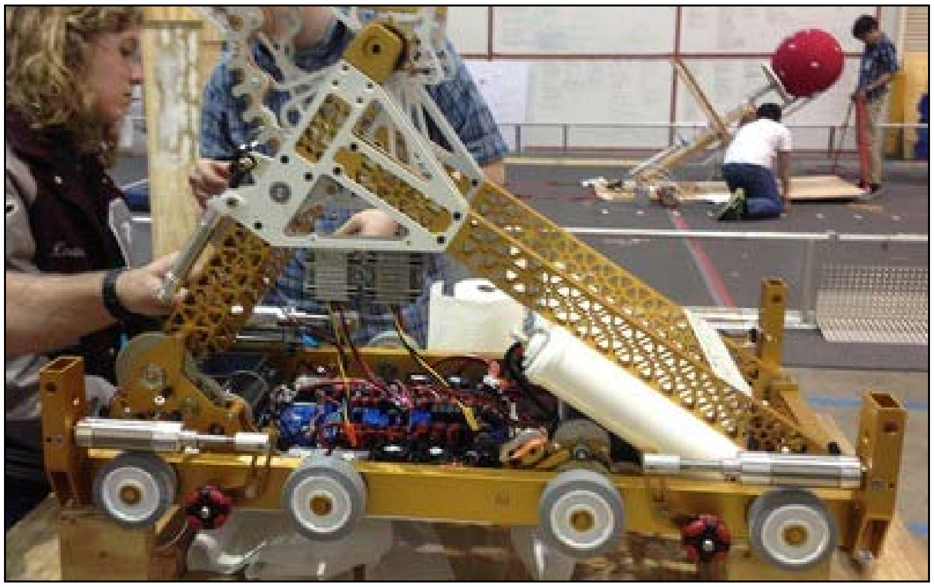
5.1.3 Types of Drivetrains

6 Wheel Drop Center: a type of tank drive; one of the most basic drivetrains. The AndyMark kit of parts drivetrain is a 6 wheel drop. One of the few cons is the drivetrain can rock a lot depending on the size of the drop, which could affect mechanisms outside the frame and autonomous reliability.



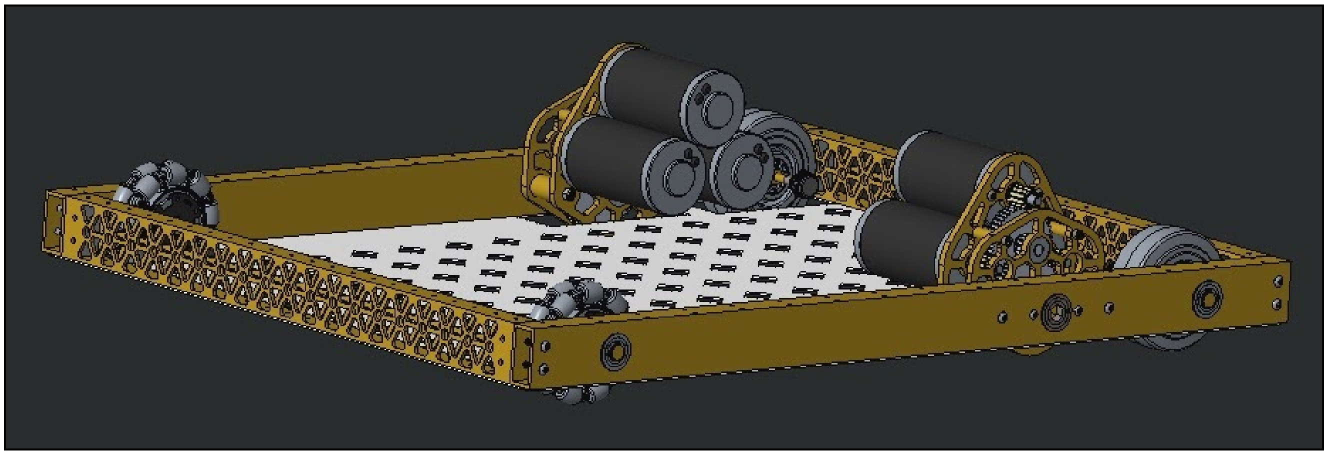
Team 254's 6WD WCD from 2017.

8 Wheel Drop Center: fundamentally similar to a 6 wheel drop, however the center 4 wheels are lower than the outer wheels. This makes the rocking less extreme, but adds turning scrub.

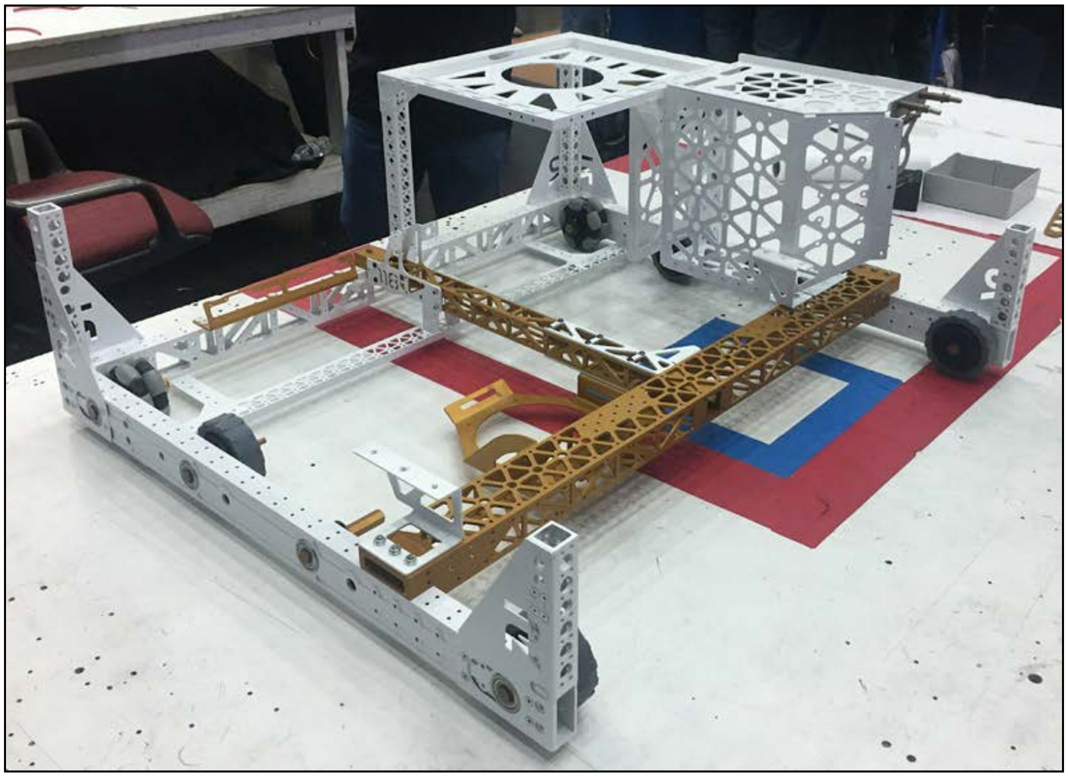


Team 118's 8WD WCD from 2014.

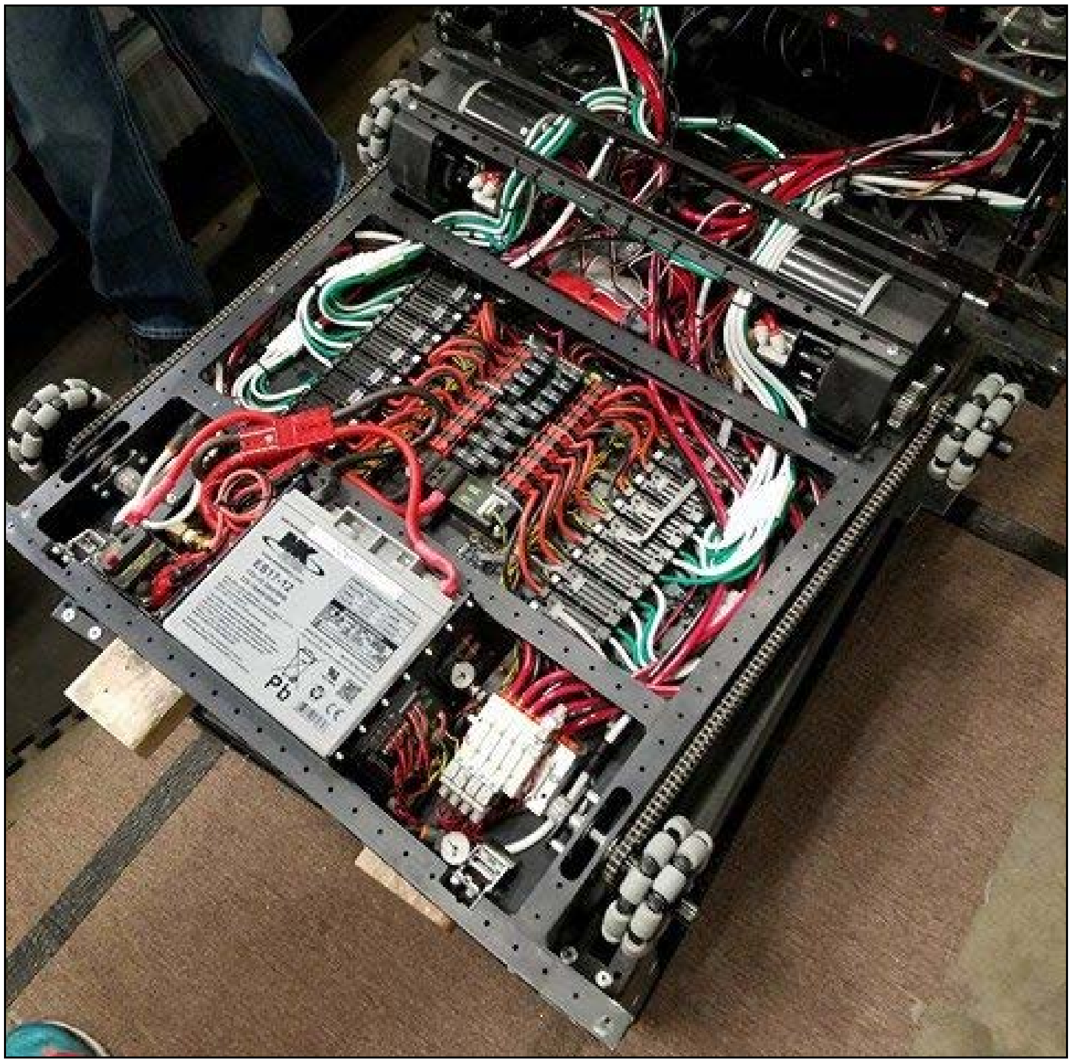
2+2 Drive: has 2 traction wheels and 2 omni wheels. This makes it difficult to be pinned, however can be difficult to control at high speeds. It requires the center of gravity (CG) to be placed over the traction wheels in order to be easily controllable. Generally, the center of rotation will be at, or close to, the axis of the traction wheels, depending on the robot's CG.


A 2+2 drivetrain designed by Team 118 in the 2016 offseason.

4+2 Drive: has 4 traction wheels and 2 omni wheels. This makes the robot easier to drive than a 2+2 and resists being pushed sideways more than a standard 2+2. You can do this to change the robot's center of rotation.

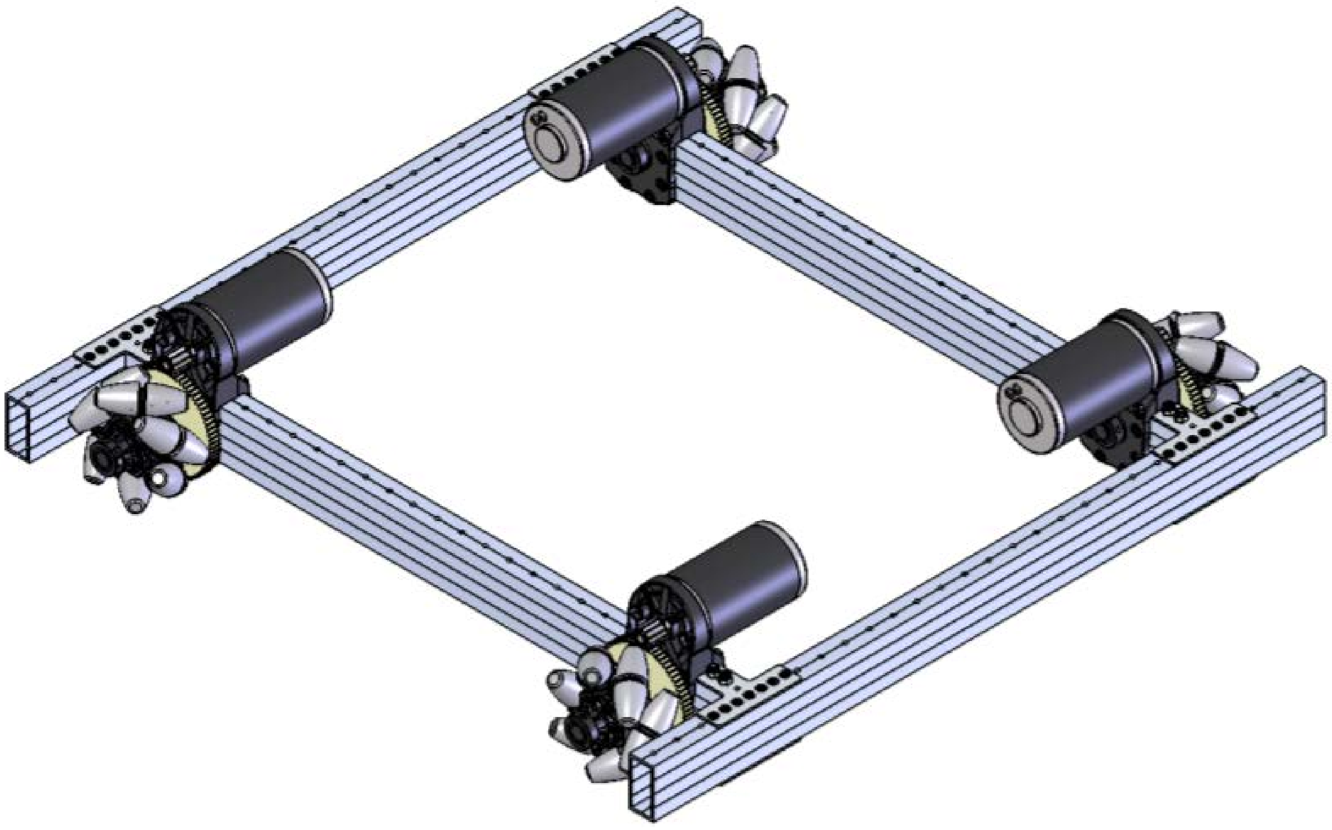

Team 118's 4+2 Drivetrain from 2017.

All Omni Drive: a drivetrain that has 4 omni wheels. This drivetrain is very difficult to drive, and can be pushed around very easily. However, it is the hardest to pin of all the drivetrains, and tends to roll off of defense. Team 3310 is well known for this drivetrain type. Team 33 used to use it, as well.



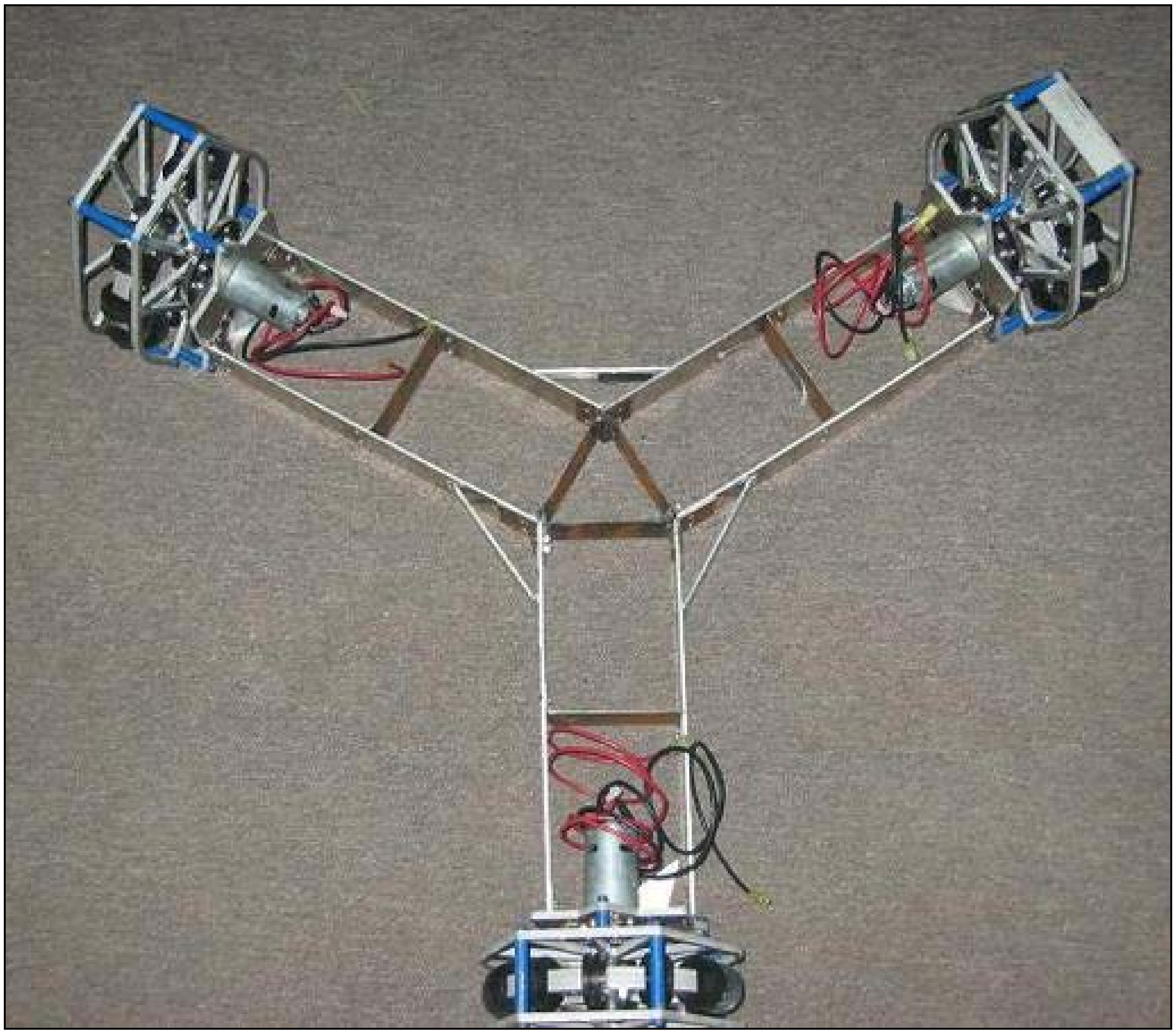
3310's All omni drivetrain from 2017.

Mecanum Drive: uses 4 mecanum wheels providing holonomic motion. It is typically the least efficient of all the holonomic drivetrains. They also require 4 separate gearboxes, compared to a tank drive that only requires 2. Most successful teams avoid using mecanum drivetrains due to there being no real advantage in most games, and having many disadvantages, including lack of friction, lack of "power", high cost, etc.

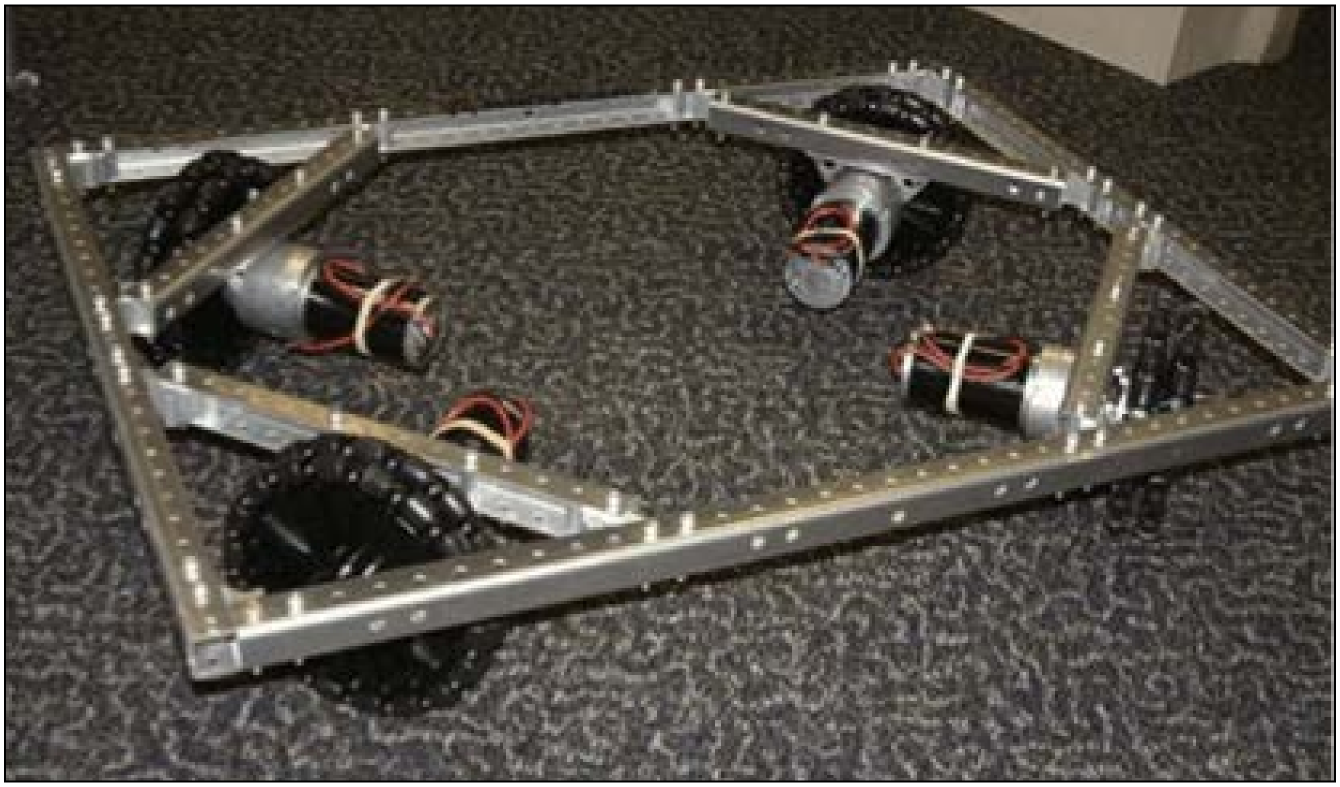


VEXpro VersaChassis mecanum drive.

Kiwi Drive: has 3 omnis in a triangular configuration. This allows you to have a holonomic drivetrain with only 3 motors/gearboxes. It also means that all your wheels will always be on the ground. It was most notably used by 1114 in 2015. Justin Ridley and Anthony Lapp built the first kiwi in FIRST while on team 857 in 2002.

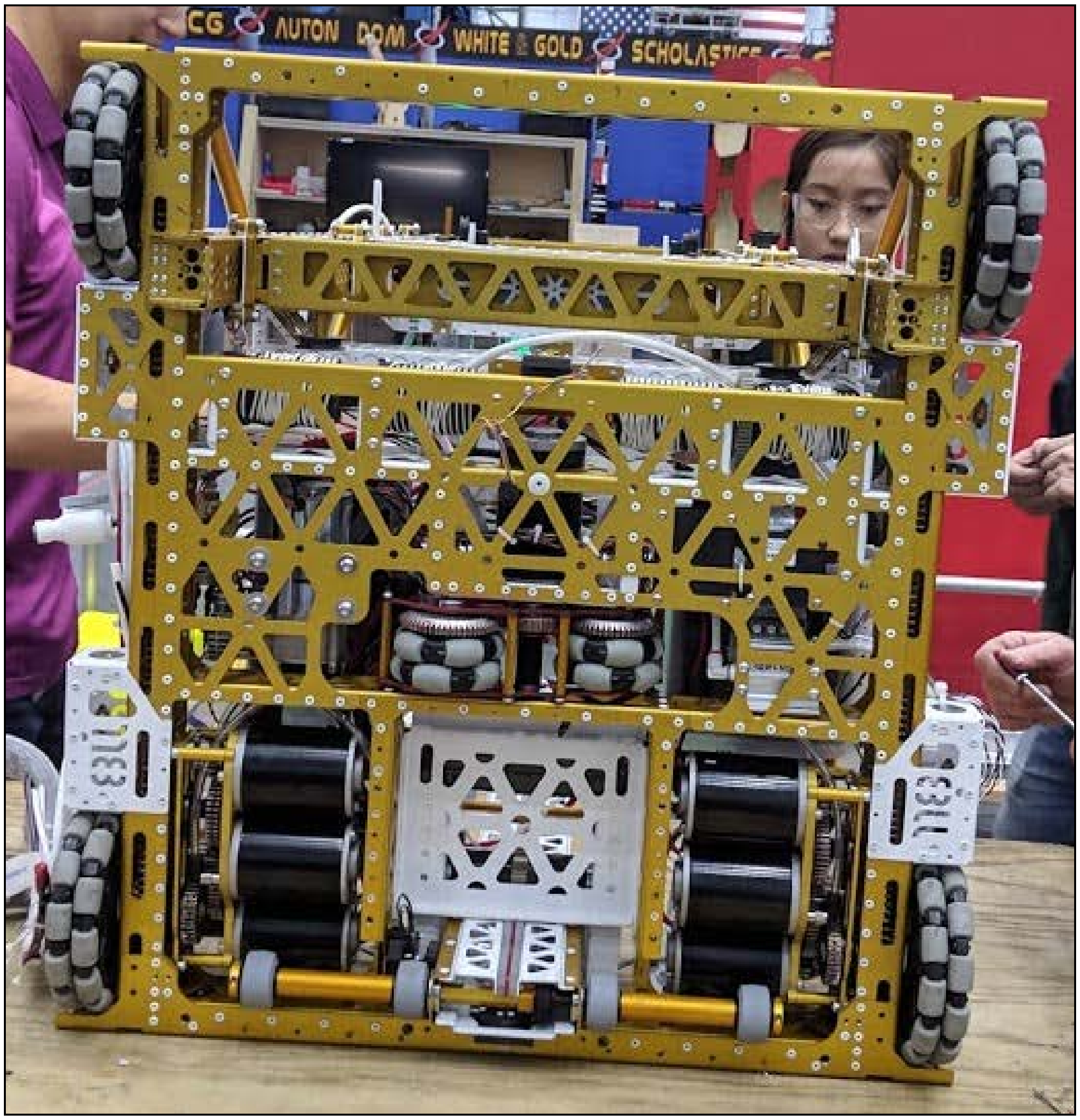

Team 857's Kiwi drive from 2002.

X-Drive: a drivetrain with 4 omni wheels at a 45 degree angle to the frame rails. An X-drive, in theory, goes around 1.4 times faster than a tank drivetrain, using the same output speed gearboxes. Like mecanum, it also requires 4 separate gearboxes. It is very rarely used in FRC. More information on why X-drives go faster than tank drives can be found here.



An FRC style X-drive.

H-Drive: has the same drivetrain as an all-omni drive plus 1 to 2 omnis in the center of the robot, perpendicular to the normal drive wheels. This allows for holonomic motion, however is usually the heaviest of the non-articulating holonomic drivetrains. An H-drive requires at least 3 gearboxes. The center omnis need to be forced into the ground to be most effective. Team 148 did this most effectively in 2014 with their pivoting strafing wheels which has been copied by many teams since, including on Team 118's 2019 robot below.

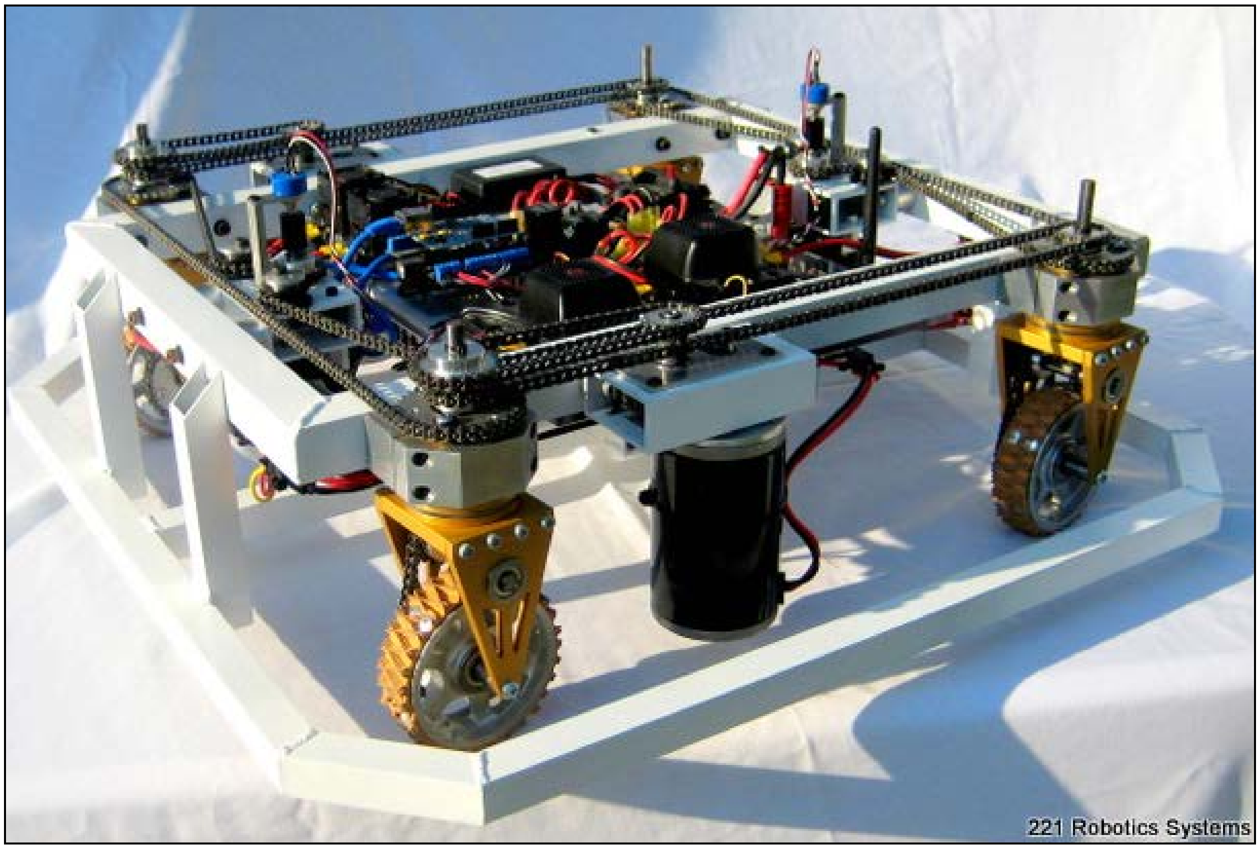


An H-drive on Team 118's 2019 robot.

Swerve Drive: a type of holonomic drivetrain in which each of the 4 (or sometimes 3) wheels are powered and steered independently. This gives a highly maneuverable drivetrain with the downsides of being heavy, difficult to program, and having a high part count. Some of the best teams at swerve are 16, 1323, 1717(RIP), 2767, and 2910. Some more advanced types of swerve drives include differential swerve, in which 2 motors power the wheels but are also used to steer the module. These have not been used in competition yet, but several prototypes have been built in the offseason with varying levels of success. Another advanced swerve architecture is motor-in-wheel swerve, in which the propulsion motor is inside the wheel. Team 2451 has built the most well known example of this.

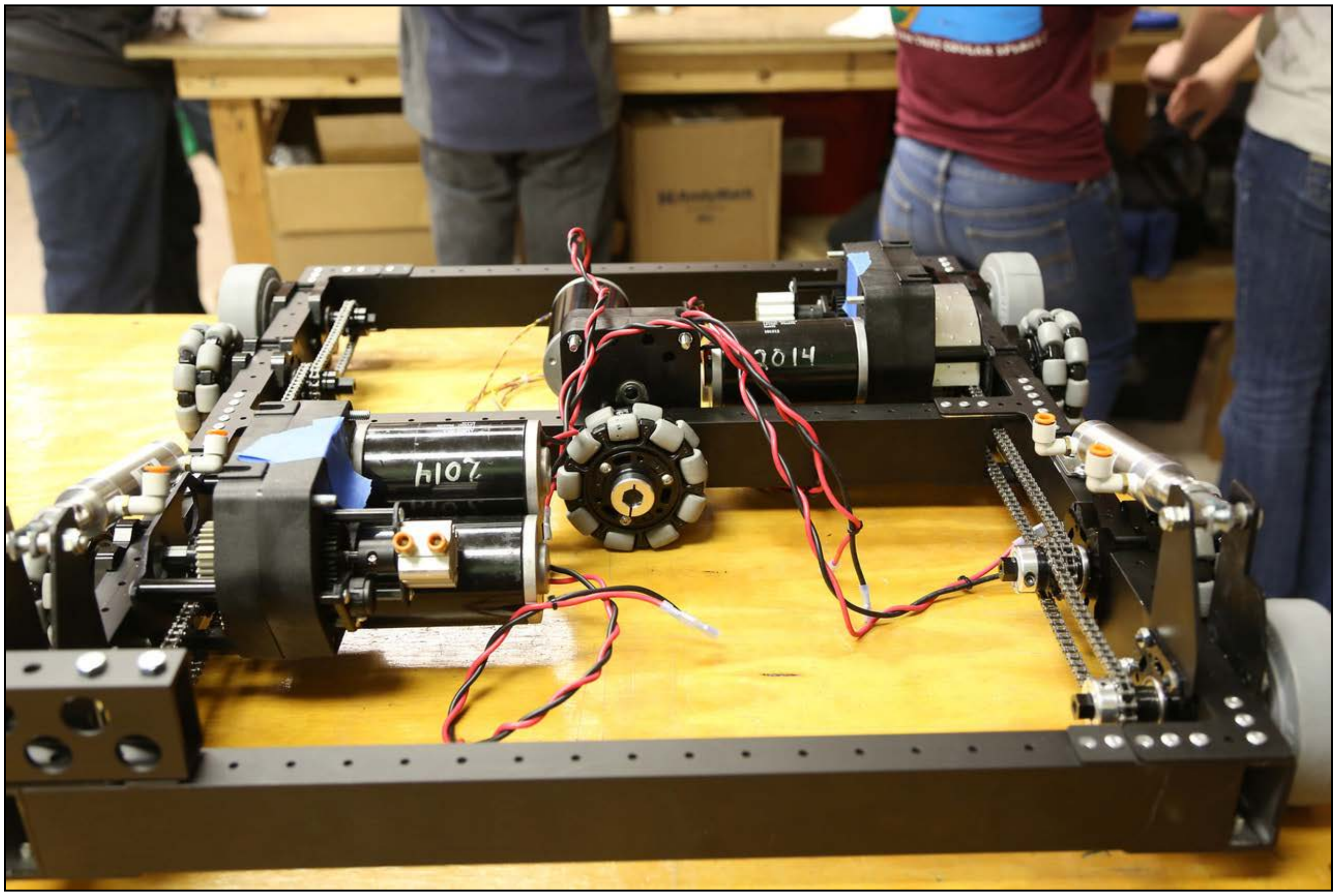
Swerve drive designed by 221 Robotic Systems.

Crab Drive: very similar to a swerve drive except all 4 (or sometimes 3 -Team 148 did this in 2008) wheels are steered together, making it slightly simpler than a swerve drive. This means that the robot cannot rotate, which is why all of Team 118's robots from 2005-2009 had turrets to rotate the robot upper body. Crab drives are very heavy and complex and have mostly faded away into FRC history with very few teams using them today. Many teams opt for lighter, simpler drivetrains or put a few more resources into their drivetrain to build a swerve drive.

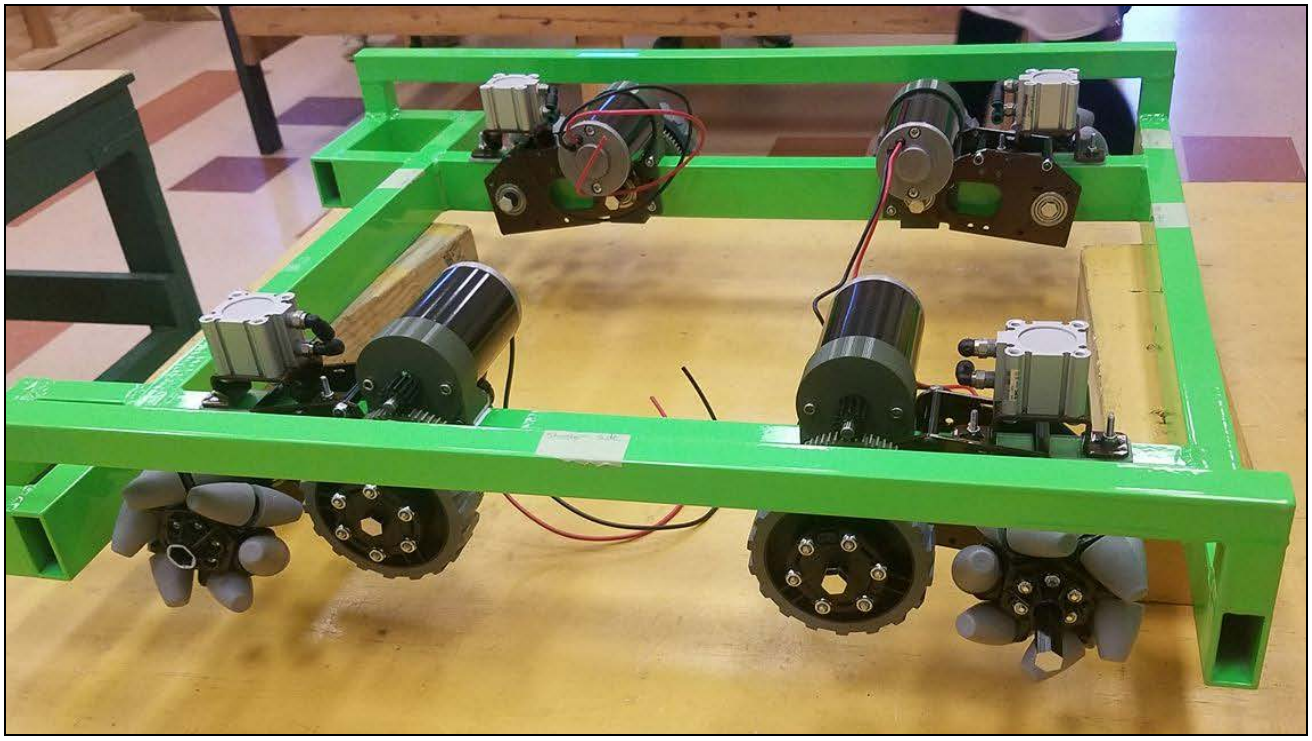


A crab drive built by 221 Robotics.

Butterfly Drive: an H-drive that can switch to 4 traction wheels. Team 148 has built multiple robots with a butterfly drive. A variation of this is the grasshopper drivetrain, in which only one side actuates. Team 624 is known for their grasshopper drives. This drivetrain is relatively heavy compared to other drivetrain types.

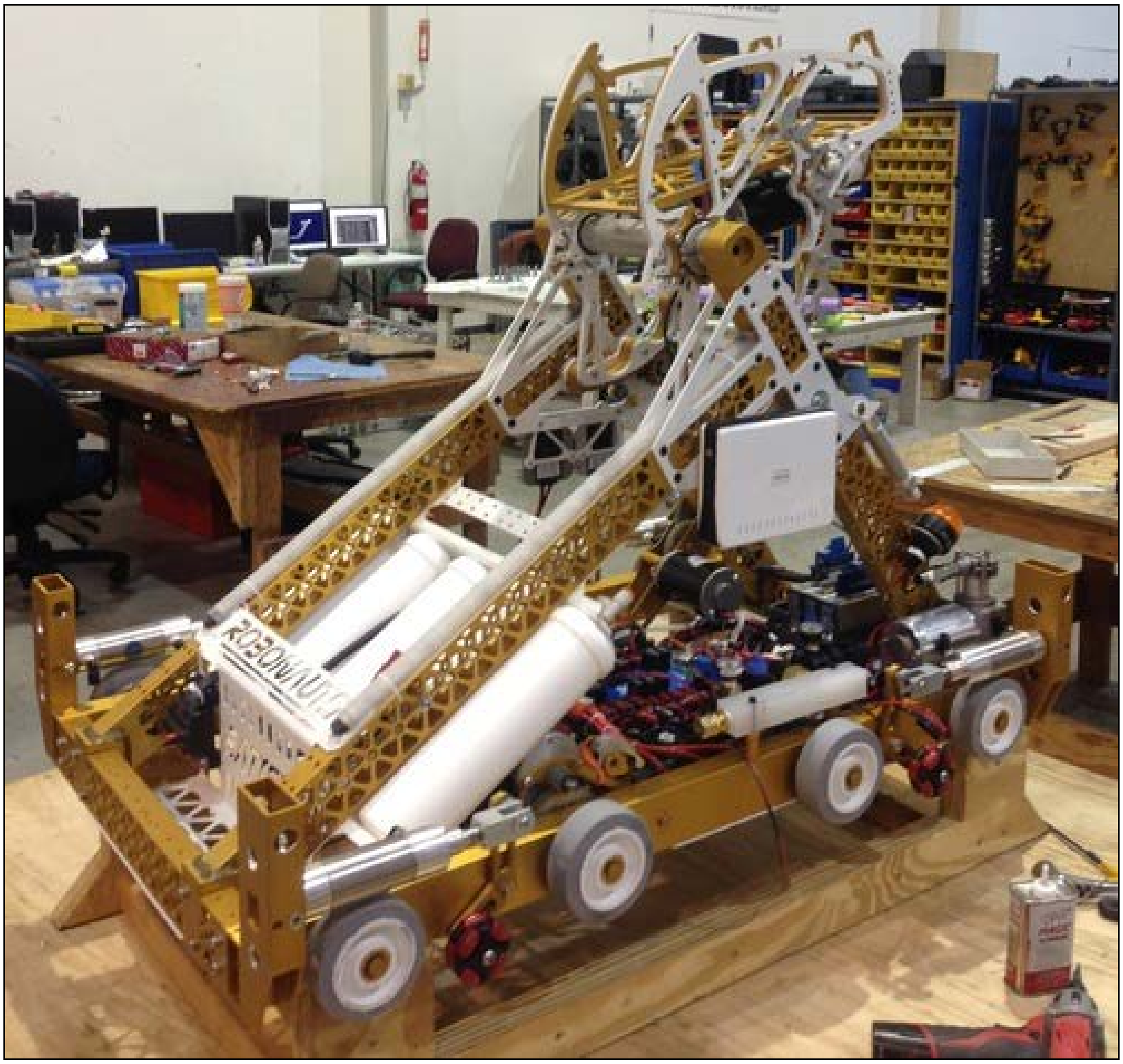

Team 624's 2014 Grasshopper drive.

Octocanum Drive: a type of actuated drivetrain that switches between 4 tractions and 4 mecanum wheels.



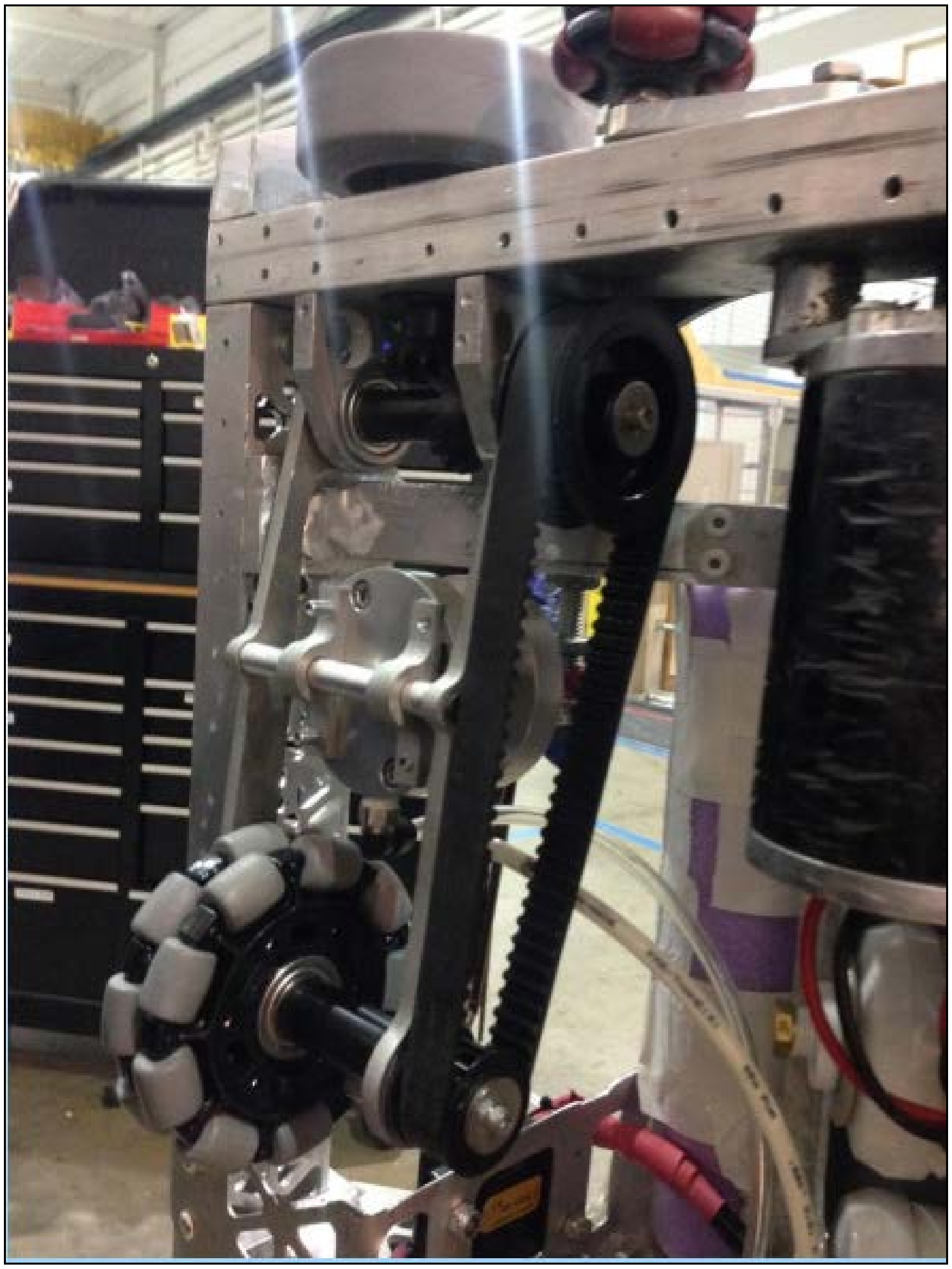
Team 5414’s 2017 Octocanum drivetrain.

Swag Drive: used on Team 118's 2014 robot, Recoil. A swag drive has drop down omnis that allow the drivetrain to essentially become a 2+2, when the omnis are lowered. This makes it more difficult to be pinned and allows the robot to rotate much faster than with a standard tank drive.



Team 118's 2014 robot with it's drop down omni swag drive.

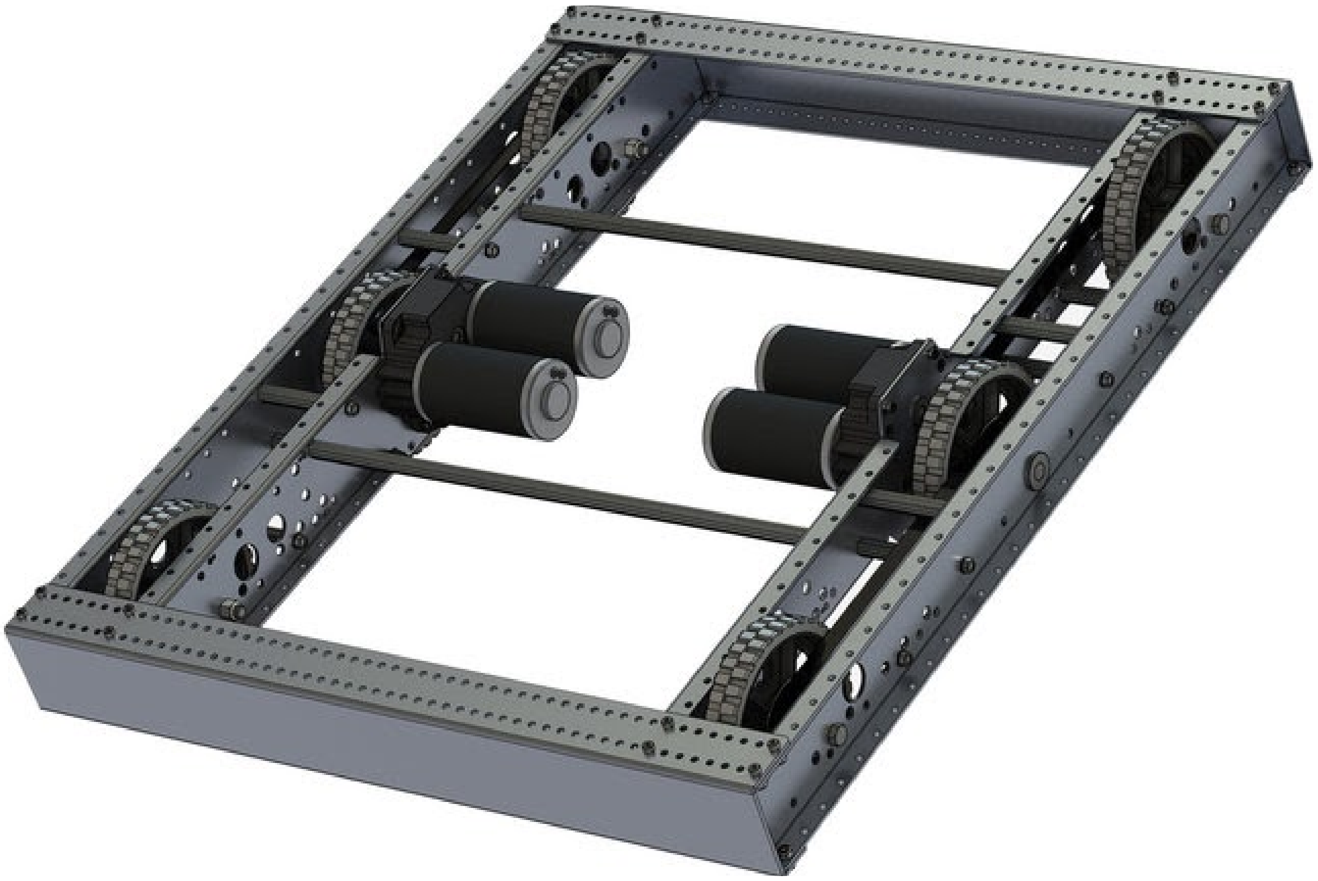
Powered Swag (Perpendicular Drop Drive): a drivetrain created by Team 118 in the 2014 offseason that has a drop down omni wheel at the front of the robot that is powered off of one side of the drivetrain using bevel gears. This has the same benefits as swag drive, but also allows the robot to rotate faster and allows more turning authority.


Team 118's Powered Swag from the 2014 offseason.

5.1.4 Drivetrain Structure

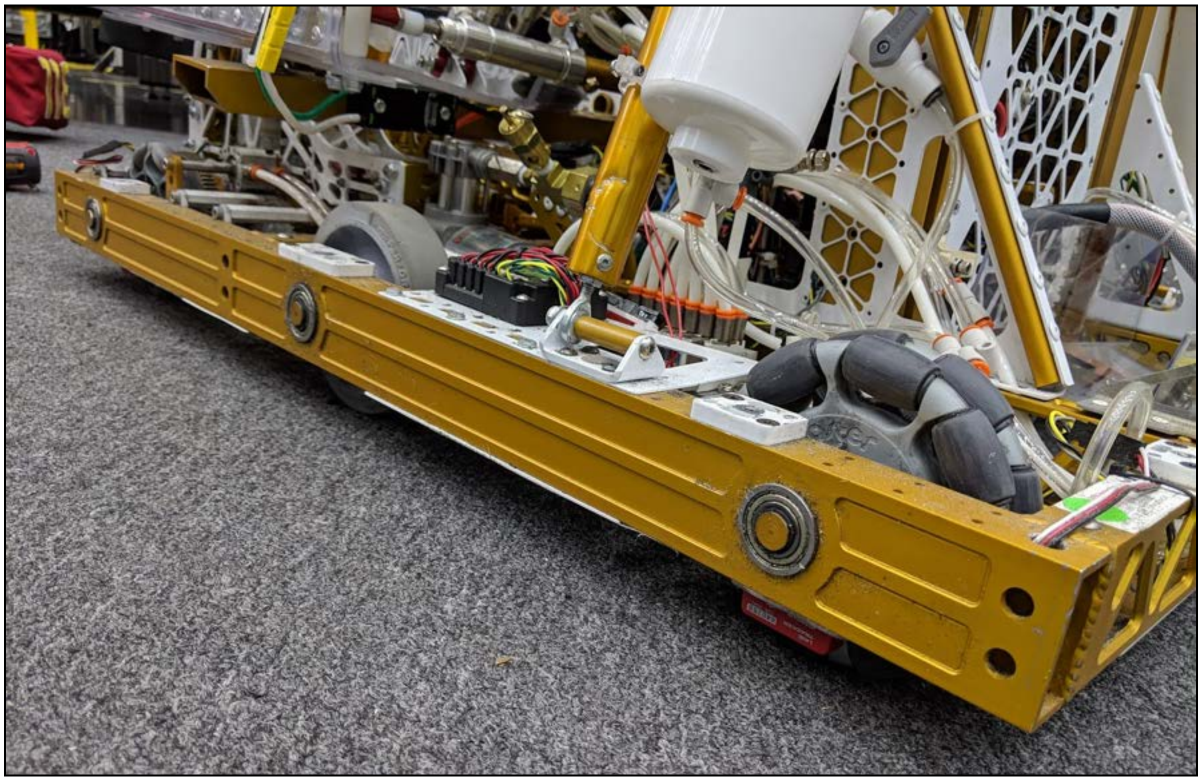
The drivetrain should be the most mechanically robust part of your robot. If your drivetrain fails, the rest of your mechanisms most likely won't be effective so choosing the right drivetrain structure is a crucial part of finding success on the field. Below is a list of the most common drivetrain design styles.

Parallel Plate: this is a drivetrain structure in which the wheels of the drivetrain are mounted between two parallel sheet metal plates. This style of drivetrain can be dead or live axle, or a combination of the two. The best example of this style of drivetrain is the AndyMark drivetrain that comes in the kit of parts. It is somewhat challenging to design and build your own parallel plate drivetrain; your team may prefer, or be better off, investing your resources into game specific mechanisms, rather than constructing a custom drivetrain..

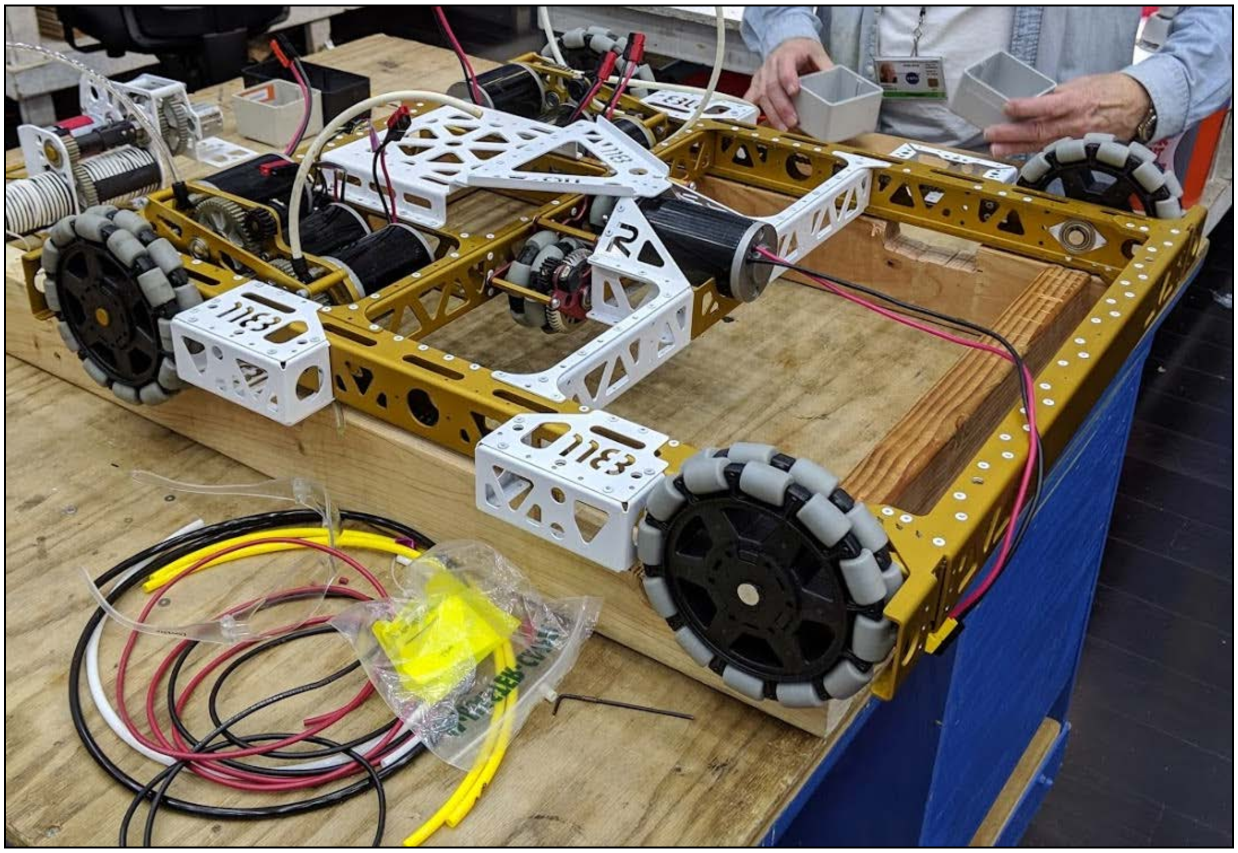


AM-14U4 robot chassis.

Box Tube Construction: most commonly used on West Coast Drive style drivetrains but it is very versatile and can be used on many other drivetrain configurations, such as kiwi or swerve. Many custom box tube drivetrains are built using 1"x2" 1/8" wall aluminum tubing, but VEXpro offers 1"x2" tubing with a .100" wall thickness, as well. Box tube drivetrains usually use chain or belts to transmit torque to each wheel, and use either bearings directly pressed into the tubing with exact center-to-centers, or use sliding bearing blocks driven by a cam that tensions the chain. By using the VEXpro/WCP bearing blocks and VEXpro VersaFrame stock, a team can easily build a "VersaChassis" using nothing more than a saw and a hand drill. VEXpro has several examples of their VersaChassis here.


Box tube drivetrain on Team 118's 2018 robot.

Sheet Metal "Box Tube": a design style in which a pseudo box tube is created with sheet metal C-channels. Team 148 is well known for using this design style since 2015. The advantage of using this drivetrain structure instead of standard box tube is that it can be manufactured on 2D machine tools such as a waterjet, high power laser cutter, or turret punch. These are all highly specialized machine tools, making this drivetrain out of the reach of many teams. It also requires precision sheet metal bending. A sheet metal drive in this style can be lighter and quicker to manufacture a standard box tube drivetrain. It is the most difficult drivetrain style to design and requires skilled operators to bend the parts. If your team has a sheet metal sponsor, this style of drivetrain may be within your reach. Team 148's first robot built using this design style was their X009 offseason robot.



Sheet metal chain-in-tube drive train from Team 118's 2019 robot.

5.1.5 Designing a WCD Style Drivetrain

This example shows how Team 118 formerly designed drivetrains; other teams have different practices. For example, Team 118 does not add center distance to their chain runs, but Team 254 does, the details of which are described in the torque transfer method section below. Also the following design exercise is shown in PTC Creo/Pro Engineer, which is the CAD software Team 118 uses. Many other teams use software such as Solidworks, OnShape, or Autodesk Inventor, however the basic principles still apply.

RAMP Greybots has great videos here and here on how to design a West Coast Drivetrain. A great tutorial on how to do this in OnShape can be found here.

Torque Transfer Method

The two types of torque transfer methods used for West Coast Drivetrains are timing belts and chain. Most teams use chain with #25 chain being more common than #35, a few teams have used 15mm wide 5mm HTD timing belts. For teams with precision machining tools such as a mill, router, or waterjet bearing bores can be machined directly into the drive rails using direct center-to-center calculations or by adding an empirically derived center distance add intended to reduce slack in the chanin and backlash in the system. The details of designing chain runs are described in section 4.6.2. Another method commonly used by teams with or without precision machining resources is to use sliding bearing blocks such as the VEXpro VersaBlocks. These allow the chain to be optimally tensioned to eliminate as much backlash as possible which can help make autonomous modes be more reliable and repeatable. Some teams such as team 971 have made their own sliding bearing blocks that can be tensioned. Another benefit of using separate bearing blocks is that the center drop can be changed without re machining the drive rails by just replacing the bearing blocks.

Frame

The frame is the foundation for the rest of the robot and should be designed to be as strong as possible within reason. The two main parts of the frame are the **drive rails** and the **ladder bars.**

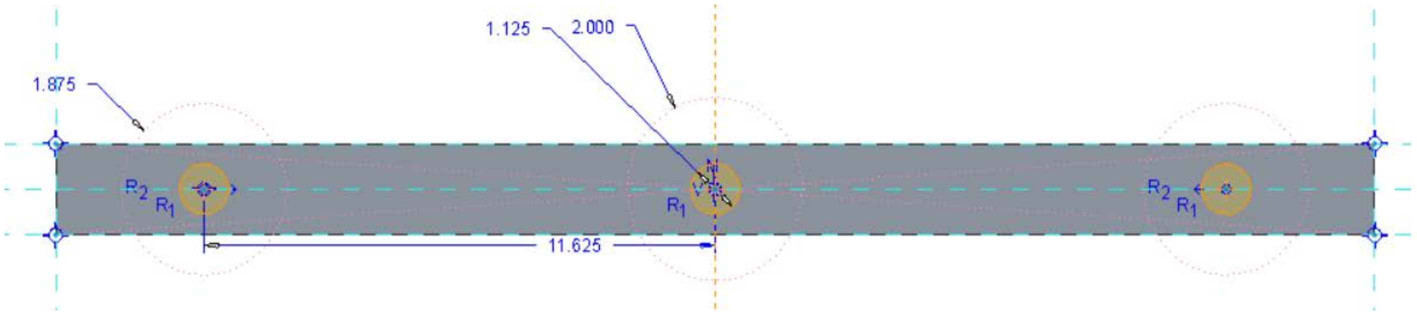
Drive Rails: the structure on the sides of the robot that contains all the bearings or bearing blocks for the drive wheels.

Ladder bars: the members, usually 1x2 box tube, that connect the drive rails together. You need to have at least 2 ideally at the front and back of the robot. Sometimes other robot mechanisms require the ladder bars to be pushed towards the middle of the robot.

Designing a Drive Rail

Drive rails are simpler to design than it seems for non-articulated drivetrains. In this example, we will design a 6 wheel drive, chain-in-tube drivetrain with an artificial drop center using 1”x2” 1/8” wall tubing. Once you determine the spacing you want between the wheels, you can start designing the drive rail.

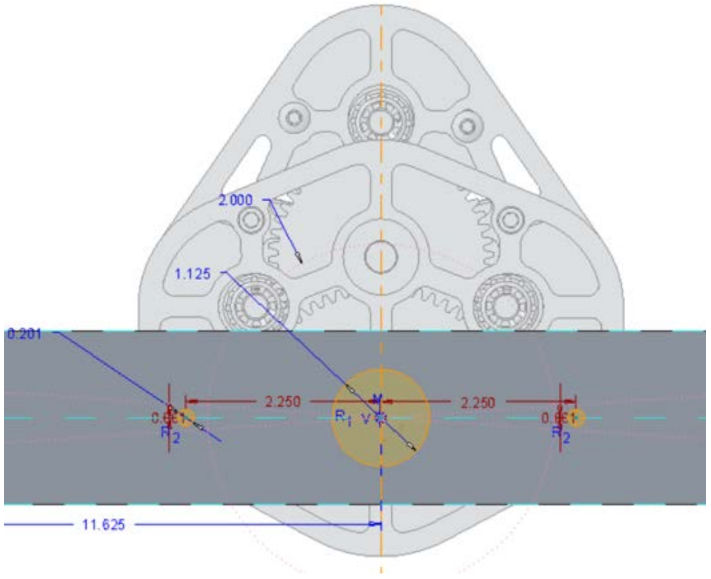
1. Sketch a 1”x2” box tube with 1/8” thick walls. It is possible to use tubing with thinner walls than this, such as the .100” wall 1x2 Versaframe tube. Using .100” tube allows you to fit 18-tooth sprockets.
2. Extrude to the desired drive rail length. In this example it’s 30 inches.
3. Create a sketch on the side of the rail. It’s good practice to have the major features including bearing holes, gearbox mounting holes, and chassis connecting block mounting holes in this sketch.
4. First we will sketch the bearing holes and then later sketch the other mounting holes. Since we are doing a 6 wheel drive, we will have one wheel in the very center and the other two are going to be equidistant apart. One way of getting a center point in a rectangle is to draw construction lines diagonally across the part. Where they intersect is the center point. At the center point, we will draw a 1.125” bearing hole and then draw a bearing hole on each end that is about 11.5” from the center. However, this isn’t a perfect center-to-center needed for our chain to transmit torque properly. 11.5 inches is only 109 links, and we want to have an even number of links, so if we add .125” we will have 110 links and a distance of 11.625.”
   The following sketch also has a 4” diameter wheel in the center with 3.75” wheels on the outside drawn with construction circles.



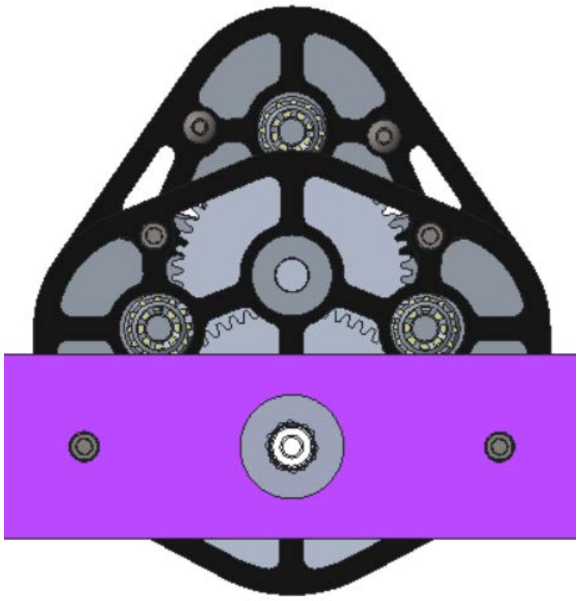
5. Now that we have our bearing holes we need to add the gearbox mounting holes. For this example we will use a COTS gearbox, the WCP SS Gearbox. Once you download the CAD file for this gearbox from WCP, you need to import its STEP file into your drive rail assembly. This goes for all the COTS FRC gearboxes available. Once the gearbox is constrained to the drive rail properly, the mounting holes can be added. The easiest way to do this is to go into the bearing hole sketch and reference the mounting holes on the gearbox. However, once the circles are drawn, the references need to be

deleted and dimensions locked. If the references to other parts (called external references) are left, they can cause major problems down the road, and can break assemblies.

6. Unless we run the bolts all the way through the box tube, we need to make the holes on the outside of the drive rail larger so that the 10-32 bolt head fits through them. McMaster says the head diameter of a 10-32 cap head bolt is .312". We will go ahead and make the clearance hole on the outside of the driverail .325". This needs to be done in a separate sketch because the clearance hole is not going all the way through the tube; the hole just goes through one side.
   To the right is what the drive rail should look like after this step.



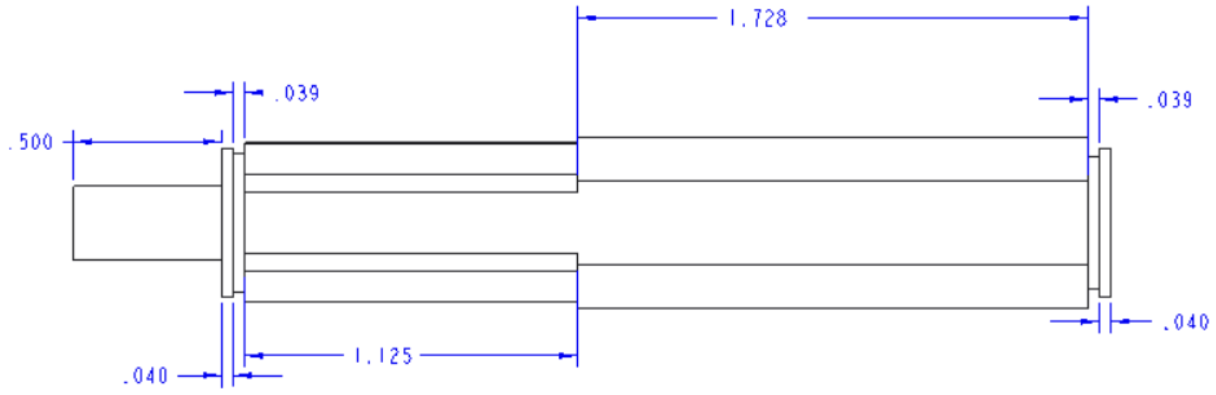
7. Now the bearings can be added. It is ideal to not have more than 2 bearings supporting one shaft, but the gearbox already has two in it. It is possible to remove the one touching the drive rail and press it into the outside wall of the drive rail. Custom gearboxes can be designed to only have one bearing on the backside of them. On the gearbox output shaft we will use a 1/2" hex bearing and we will use a 1/2" thunderhex bearing on the rest.

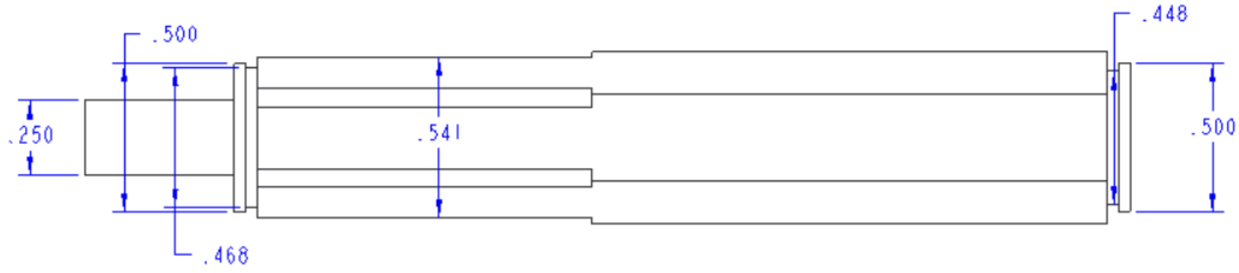


8. Now we can add spacers and wheels. It is not always required to have a spacer between the bearing and the wheel, depending on what wheel is being used, but for this example we will put a 1/16" spacer there. We can also add the wheels. For this example, we will use 4 inch colsons, with the outer wheels turned down to 3.75."
9. Once the wheels are added, the shafts can now be designed. The reason we put thunderhex bearings in the box tube is so we can turn down a 1/2" hex shaft to thunderhex. This prevents the shaft from being pushed in too far. Retaining ring grooves will be cut into the shaft right past the wheel on the outside of the robot, and

right past the inside bearing on the inside of the robot. On the outsides of these grooves, the shaft will be turned down to a 1/2” round to make installing the retaining ring easier. Lastly, the shaft will be turned down to a 1/4” round on the inside of the robot to allow for an encoder to be coupled to the shaft. The shaft should be slightly longer than needed to allow for any variation in the wheel width, spacer length, or any machining tolerance. Adding .01” to both the hex section of the shaft and the thunderhex section should be sufficient. This extra length can be designed in CAD or just added when creating the shaft drawing. For the retaining ring slots, we will be using 1/2” rings. The width of the slot and diameter of the slot can be found on McMaster.

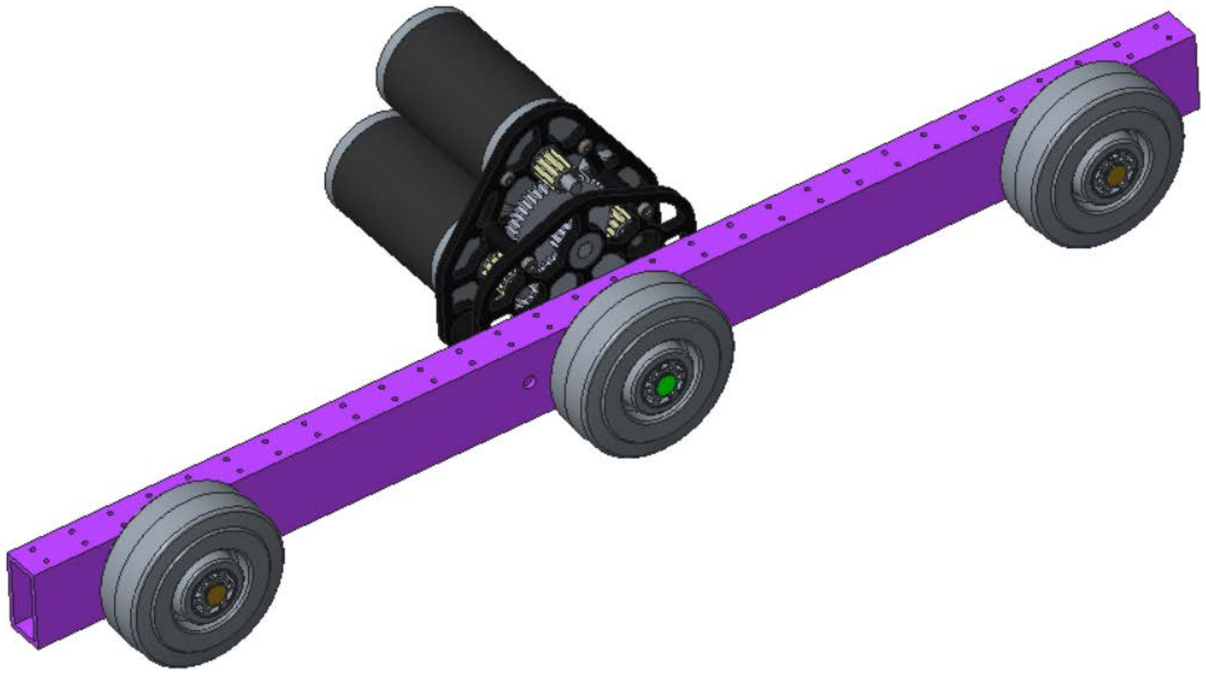
The following drawings show what the shaft lengths and diameters are for each section.

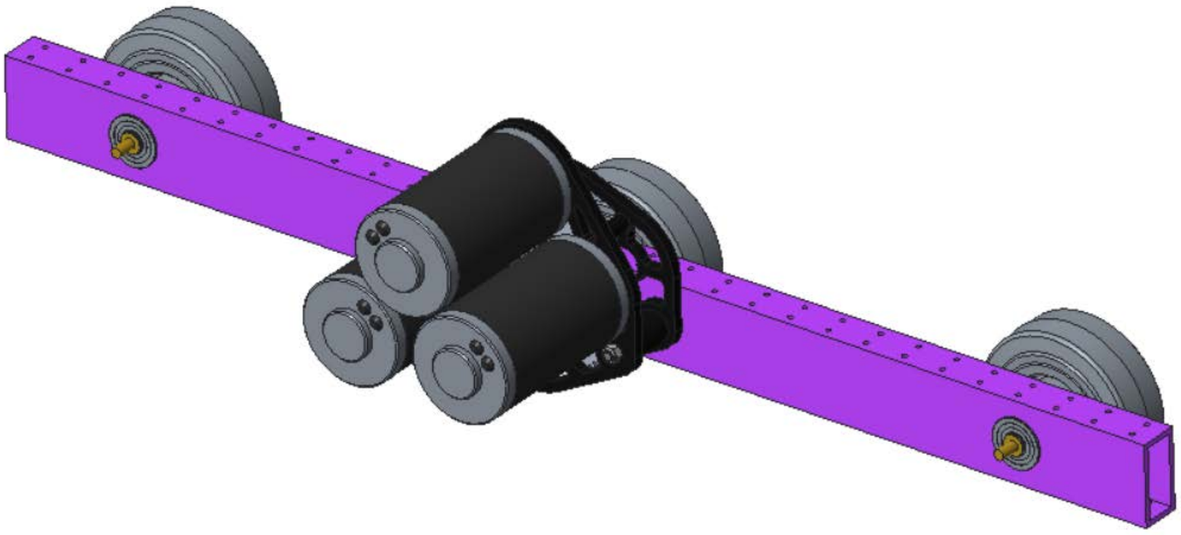




10. Next, the sprockets can be added. Since this is a chain-in-tube drivetrain we will use the 17-tooth double sprockets from 221 Robotic Systems, however these sprockets are no longer commercially available.
11. Lastly, we usually design a bolt/rivet pattern on both sides of the drive rail to allow for the mounting of other mechanisms. Usually, these are 1/8” rivet holes, which can also be tapped for 8-32 bolts.

Below are screenshots of both sides of our drive rail. Once ladder bars are designed, the mounting holes for 221 chassis blocks or gussets can be added.





Belly Pan

The belly pan goes at the bottom of the robot, and usually connects the drive rails and ladder bars. The belly pan often serves as a mounting point for electronics, the battery, the compressor, air tanks, and other robot mechanisms. Usually belly pans are 1/16”-1/8” thick, and are sometimes pocketed. However it is usually better to use those machining and design resources on other parts of the robot and just use a thin aluminum, polycarbonate or even wood sheet without any pocketing as the robot’s bellypan.

You can use rivets to attach the belly pan to the drive rails, but this means with a chain-in-tube drivetrain you have to layout the chain in a specific way, and rivet on the belly pan in a specific way to prevent the rivet tails from hitting the chain. If rivets need to be replaced, then the chain needs to be cleaned out because drilled out rivets can jam between the sprocket and the chain and break the chain if left inside the tubes. You can also drill and tap holes in the tube and bolt the belly pan on. This is more work up front, but is easier to maintain in the long run.

5.2 Elevators

Elevators are linear mechanisms that are typically used to get a game piece from the ground to some higher scoring location. They usually consist of structural elements, most frequently tubing that extend with bearings between them. There are several different ways to build an elevator, to be covered.

Sensors frequently used on an elevator usually consist of one sensor used to measure the position of the elevator, such as an encoder or string potentiometer, and two limit sensors, such as limit switches, beam breaks, or hall effect sensors. One limit sensor at the top and one at the bottom.

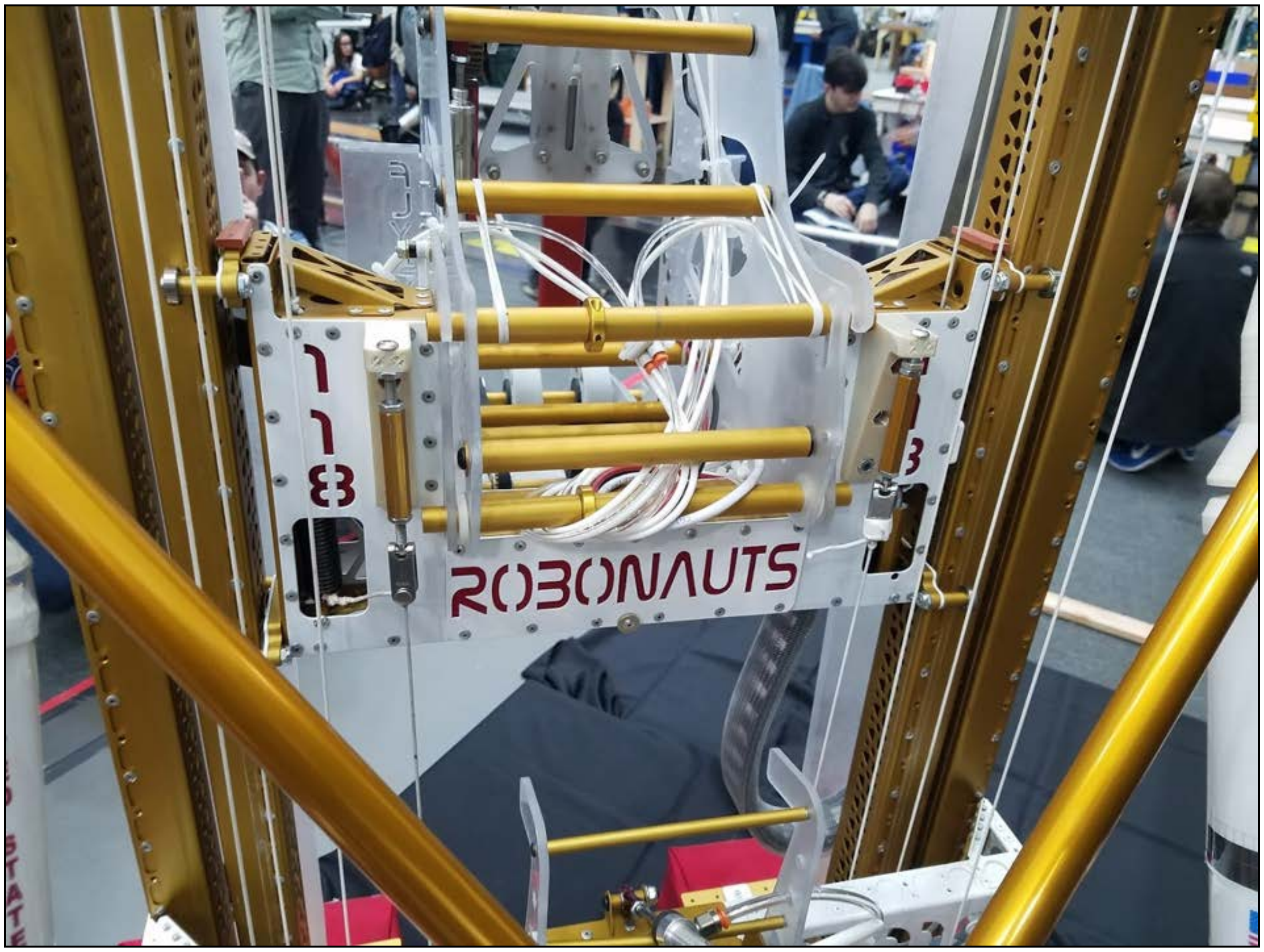
The two ways to actuate multi-stage elevators are cascading and continuous. A visualizer to help understand the differences can be found here.

5.2.1 Elevator Stages

The number of stages on an elevator varies depending on who you ask. Some consider the stationary bottom segment of an elevator a stage, while others consider only the moving segments stages. In this guide we will only consider the moving segments stages.

Single Stage Elevator: a single stage elevator is the simplest to implement. Single stage elevators only have one moving segment, requiring only one loop of chain, belt, or cable to move. Single stage elevators allow a robot to come close to doubling in height, and are many times sufficient to play the game effectively.

Multi-stage Elevator: much more complex than a single stage elevator, specifically when it comes to rigging is with chain, belts, or cable. Multi-stage elevators are difficult to build without excess slop. A larger bearing spread is needed for the bottom stage than is needed for the top stage.



Two stage elevator on Team 118's 2019 robot.

5.2.2 Elevator Rigging

Continuous: continuous rigged elevators raise one stage at a time. The top stage raises first, and the bottom stage last. Each stage raises and lowers at the same speed. The rigging for a continuous elevator is more complex than for a cascading elevator.

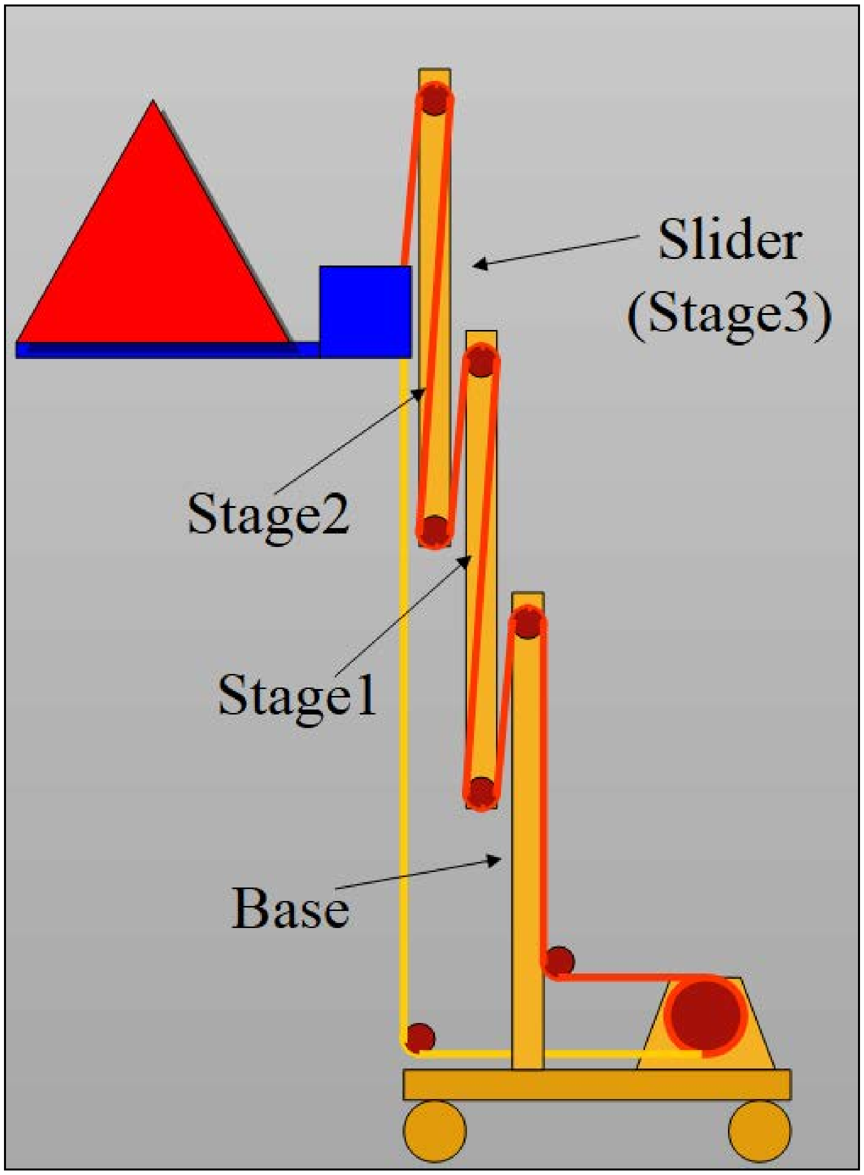


Diagram of continuous rigging from Andy Baker's Manipulators Presentation.

Cascade: cascade rigged elevators raise each stage at once the same amount. Cascades are simpler for running the chain, belt, or cable than continuous rigged elevators. Cascade rigged elevators make running wires to the top of the elevator easier. They also require more torque to raise the lower stages but are more predictable to control because they are deterministic.

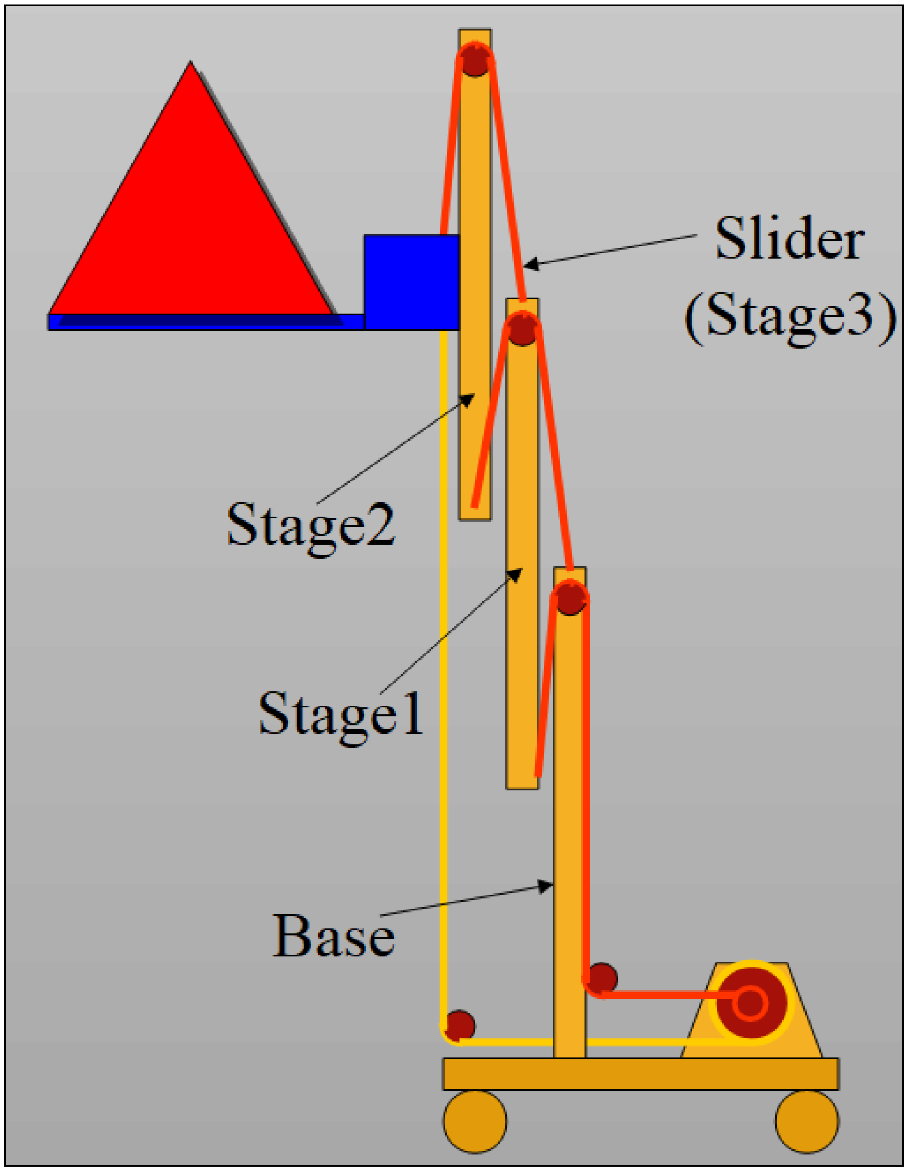


Diagram of a cascading elevator from Andy Baker's Manipulators Presentation.

5.2.3 Elevator Rigging Material

Timing Belts: almost always strong enough for elevators. They are lightweight and allow for the elevator to easily be powered up and down. It is also easy to design a tensioner for timing belts in an elevator.

Chain: can be used for elevators lifting very heavy objects. Chain is almost always overkill for elevators, but can be very useful when your team doesn't have access to precision machining resources.

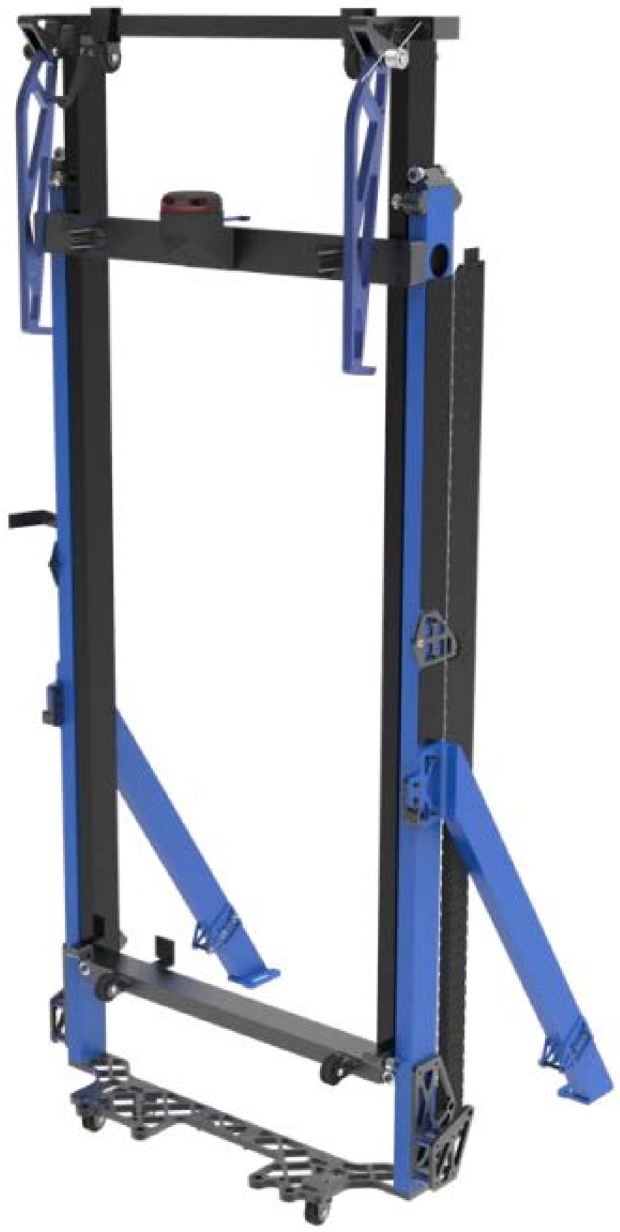
Cable: a very lightweight way of rigging an elevator. High strength, lightweight cable such as Spectra cable or Dyneema rope is what is typically used.

Rack and Pinion: have been used to raise single stage elevators with the motor and pinion on the moving stage, meshing with a rack gear mounted to the fixed segment. Team 1251 did this in 2018.

Lead Screws/Ball Screws: have been used for single stage elevators. Some disadvantages are the cost, their maximum speed (due to the whip in the shaft), and their heavy weight. Team 624 has notably used lead screws for elevators in 2015 and 2018.

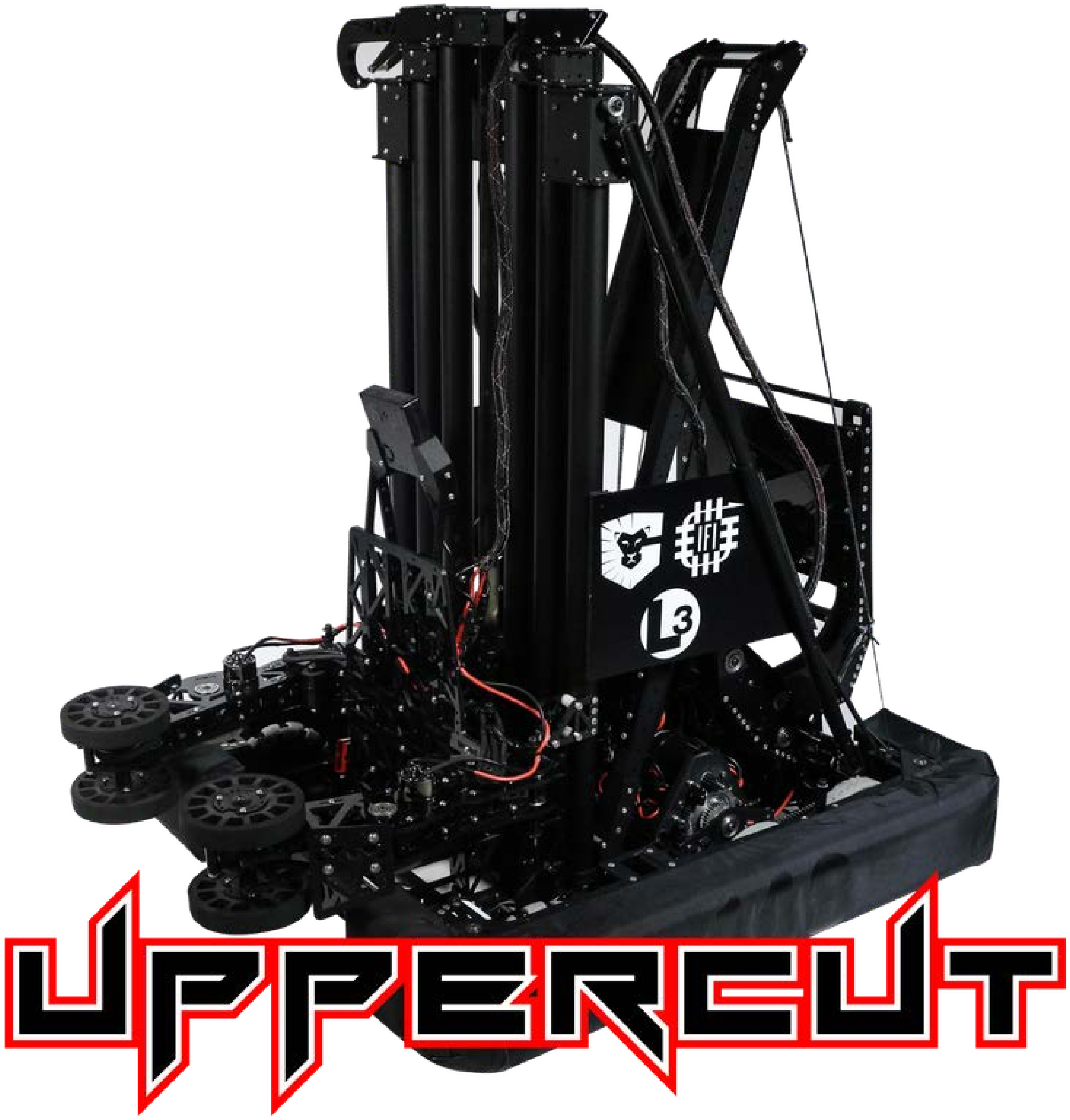
5.2.4 Elevator Structure

Box Tube: the most robust elevator structure and is easiest to implement. The VEXpro VersaFrame elevator system is designed to be used with either 1”x1” or 1”x2” box tube.



Box tube elevator from Team 254’s 2018 robot.

Round Tube: can be used to create very lightweight, yet very strong, elevators. They are more difficult to design and machine, and typically use 3D printed concave rollers with bearings inside, instead of the traditional roller bearings rolling on box tubing. Team 148 built a stellar example of a round tube elevator in 2018 and the CAD of it can be found here.



Team 148’s 2018 robot with a round tube elevator.

5.2.5 Elevator Counterbalance

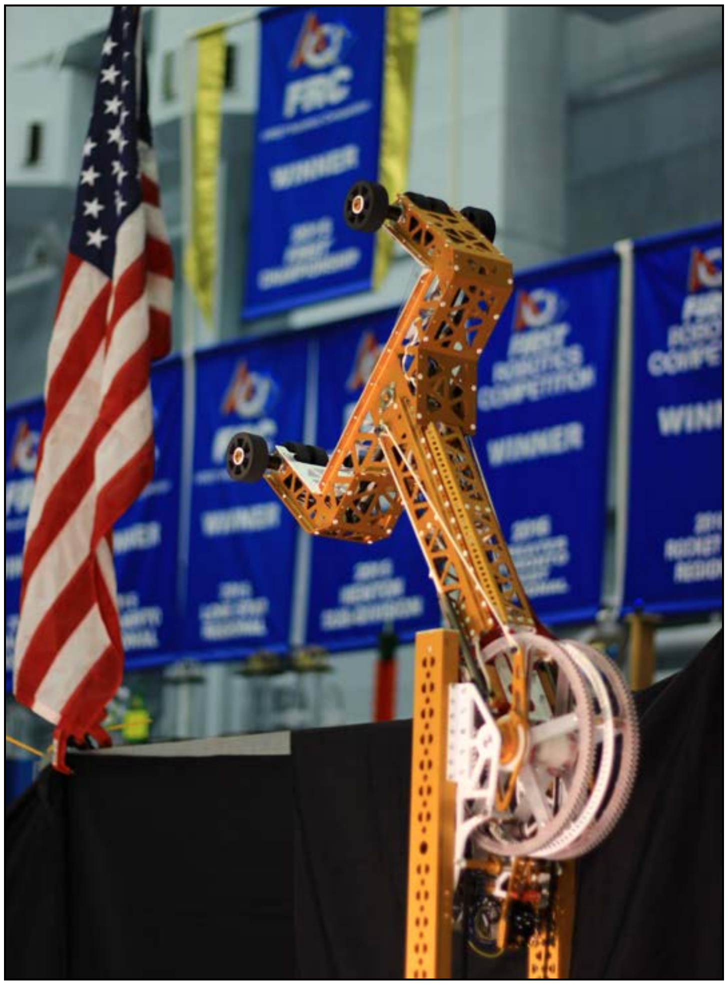
Elevators can be counterbalanced by using constant force springs or latex surgical tubing. Constant force springs are quite easy to implement, however they can be dangerous if not handled properly. Constant force springs can be purchased from many different vendors, such as Century Spring, which has quite a large inventory. To entirely offload an elevator, choose constant force springs that sum to the entire weight of what is being lifted. When you counterbalance an elevator, you can gear it faster because the motor is having to lift less weight, therefore needing less torque.

5.3 Arms

Arms are a very common manipulator in FRC, due to being simpler to design and build than elevators. Arms have two main uses, raising a game object to score high and rotating an arm all the way around to score on the opposite side of the robot on which you acquire the game object. Sometimes an arm is used to do both of these tasks.

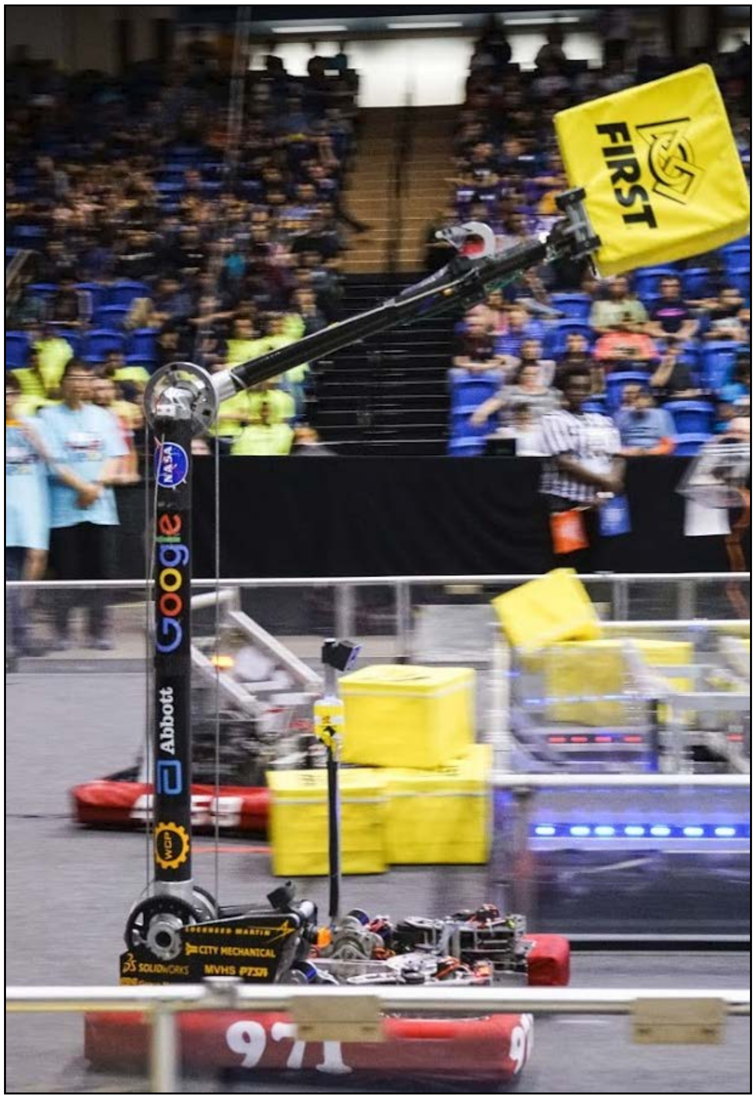
5.3.1 Types of Arms

Single-Joint Arm: the simplest form of arm that has only one axis of rotation. Single joint, simple arms do not keep the end effector in the same orientation throughout its rotation. This needs to be kept in mind when designing a single joint arm.



The sheet metal, single joint arm from Team 118's 2018 robot.

Multi-Joint Arm: allow your robot to reach farther and higher than with a single joint arm. Multi-joint arms also allow you to have control of the end effector orientation, when designed to do so. A big disadvantage of multi-jointed arms is that when both joints are driven by motors, it becomes very difficult to write software to control them. Many robots have added a joint near the end of the arm, actuated by a pneumatic cylinder, making it much simpler to control.

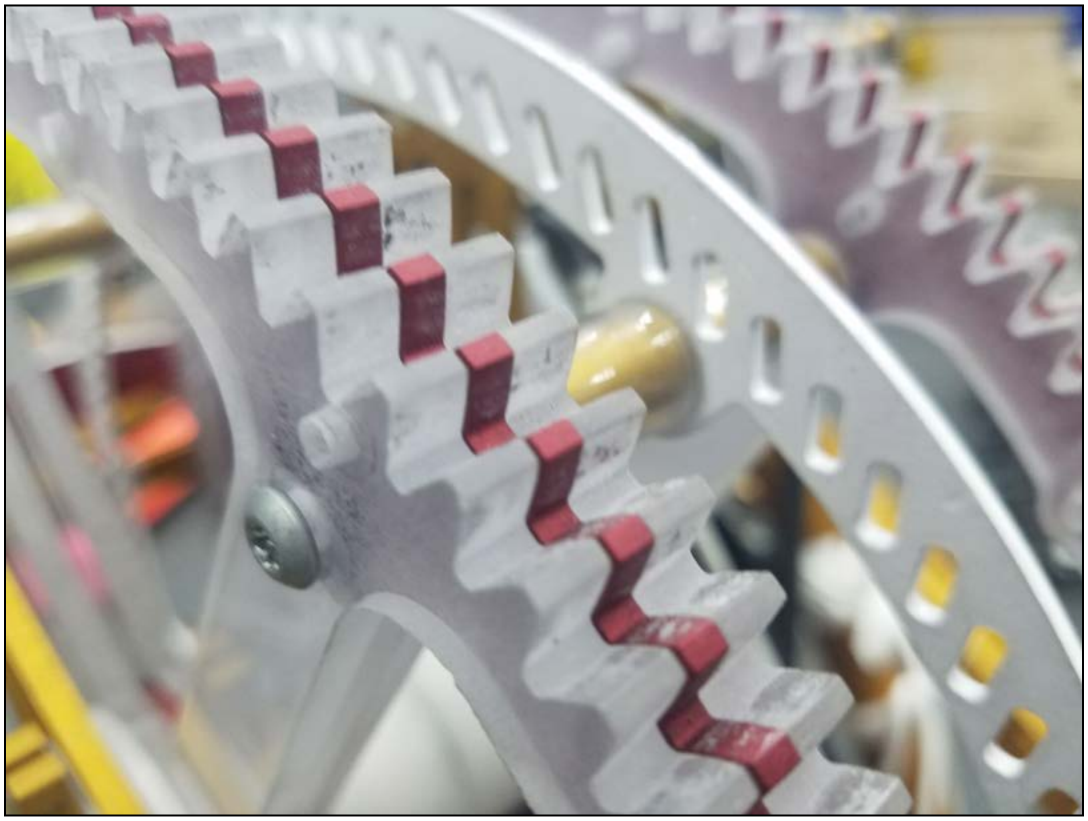


One of the best examples of a multi-joint arm ever built in FRC on Team 971's 2018 robot.

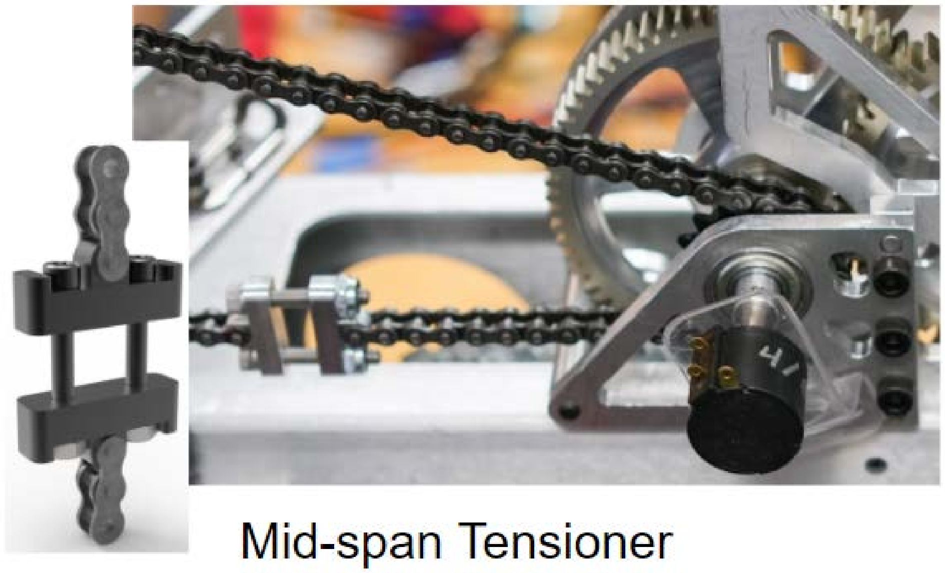
5.3.2 Arm Design Recommendations

When designing arms, it is beneficial to make the end of the arm as light as possible. It also makes the arm easier to move and control if heavy components, such as motors that drive an end effector, are placed as close as possible to the axis of rotation. It takes more torque to lift a motor at the end of an arm than it takes to lift a motor at the pivot of the arm.

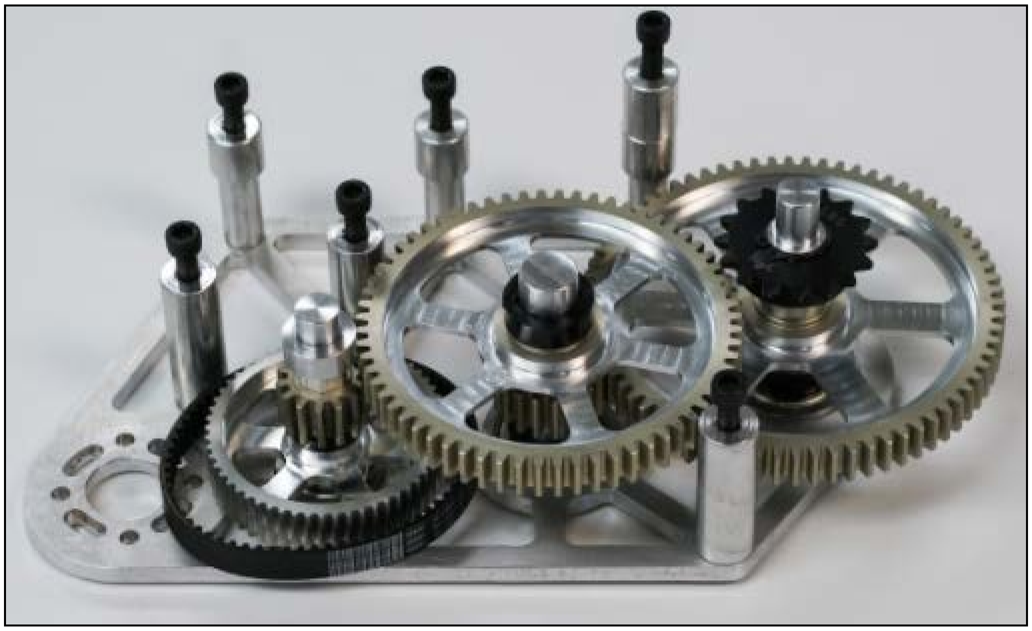
Arms also suffer from backlash just like any other mechanism using a standard method of power transmission. One way to eliminate basically all the backlash in an arm that is gear driven is to have offset gears on the arm. On the next page is an example of the offset gears on the arm on Team 118's 2018 robot. The red center gear has slots in it where it is mounted. The white polycarbonate gear is positioned so that it is touching one side of one tooth on the pinon. Then the red gear is slid until it is touching the next tooth. The offset takes up the backlash that would normally be present where two gears mesh.


Offset gears on the arm of Team 118's 2018 robot used to eliminate backlash.

Another way to eliminate backlash in an arm mechanism is to use a tensioned chain run as the final reduction to the arm. Team 971 has also used custom made oversized hex shafts to eliminate backlash between the sprockets and the shafts. In 2020 team 118 did something similar in their hood drive gearbox by epoxying the gears to the shafts to eliminate backlash between the shaft and gears. For gears that needed to be removable set screws were used to push the gear to one side of the shaft slop.



Tensioned Chain Run on a Team 971 Arm Gearbox.



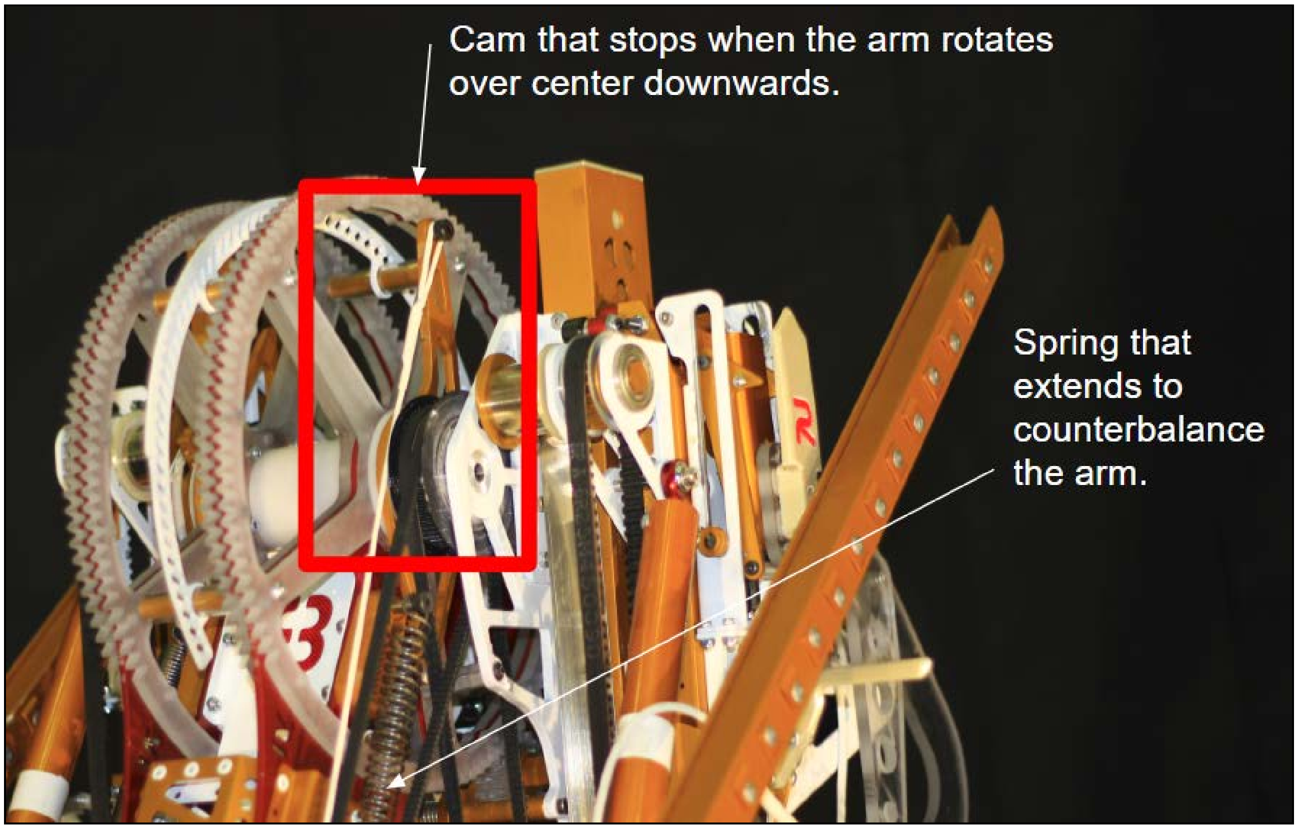
Team 971 Arm Gearbox with Custom Machined Press Fit Hex Shafts.

It is also best to get as high a reduction as possible at the arm pivot. Also mounting the sensor you are using close to the pivot will make the arm easier to control, and will eliminate much of the error that comes from backlash. Sensors commonly used to control arms are potentiometers or absolute encoders. Rotary encoders with a sensor that allows the robot to know or find the arms position on boot up can also be used. Team 971 has used rotary encoders for positioning with a potentiometer to know the location of the arm when the robot boot-up.

5.3.3 Counterbalancing Arms

Counterbalancing arms that do not “over center” (pass straight vertical or straight down) is relatively easy. The most common method is to have the arm extend slightly past the point of rotation, attach springs (or elastics) to that end of the arm and attach the other end of that spring (or elastics) to the robot’s chassis or other fixed structure.

Counterbalancing arms is much harder when the arm goes “over center.” When the arm goes over center the direction the counterbalance springs need to pull changes. Team 118 used a unique counterbalance mechanism on the arm of their 2018 robot which made the arm easier to control. The system below shows this mechanism that consists of 2 extension springs, 2 cam plates, and 2 standoffs fixed to the plates on which the arm pivot is mounted. When the arm is pointing straight up, neither spring is extended. When the arm begins to lower on the front side of the robot, one of the cam plates hits a fixed standoff and stops rotating with the arm. This causes one of the springs to start extending and offloading the arm. When the arm rotates back towards straight up, the spring retracts. The same thing happens when the arm rotates past straight up to the back of the robot- the other cam plate hits the other standoff and causes the other spring to start to extend and offload the arm.


Cam that stops when the arm rotates over center downwards.
Spring that extends to counterbalance the arm.

Counterbalance setup on the over-centering arm on Team 118's 2018 robot.

5.4 Linkages

Linkages are assemblies of rigid links connected by pin joints. Most linkages used in FRC take an input from an actuator, either a motor/gearbox or a pneumatic cylinder, and output a complex motion depending on the geometry of the linkage.

5.4.1 Types of Linkages

Four-Bar Linkage: the simplest movable, closed linkage (a closed three-bar linkage would just be a fixed triangle). Most four-bar linkages used in FRC are parallel linkages. For a four-bar linkage to provide parallel motion, each set of parallel links must have the same center distance between their pin joints. Four-bar linkages can be used to get many other shaped motions, depending on the length of segments and which points are fixed. These other types of motions are, however, rarely used in FRC. Some examples of these other four-bar linkages can be found here.

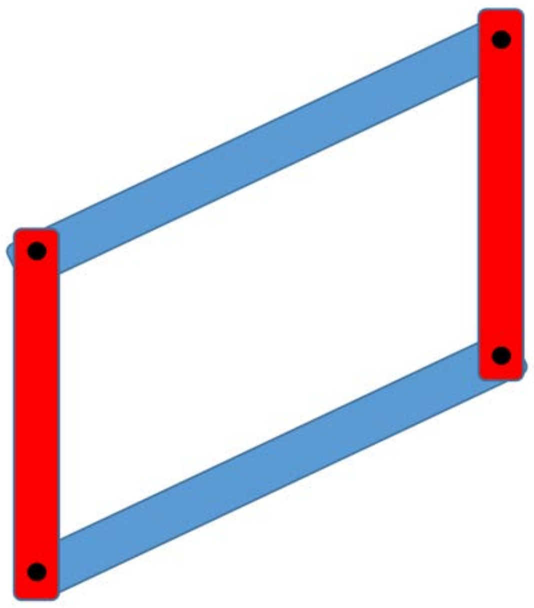


Diagram of a four-bar linkage, pin joints in black.

Double Reverse Four-Bar Linkage: a type of linkage that started in the VEX Robotics Competition, but several FRC teams used this linkage during the 2018 FRC season, most notably Team 33. The linkage consists of a parallel four-bar with another parallel four-bar mounted facing the opposite direction to the vertical link at the end of the lower parallel four-bar. The upper link in the lower four-bar has a gear mounted to it that meshes with a gear mounted to the bottom link of the upper four-bar. This allows both four-bars to raise simultaneously. The biggest downsides to this linkage are that it has a very high part count, it is heavy and takes up a large amount of space, and it sways side to side at the top. The reason the linkage became popular in VEX is that the linear elevator parts are heavy and had high friction, so the alternative method of getting linear motion was to build a double reverse four bar. Due to elevators being relatively simple to build, they almost always are not the best design to get linear motion.

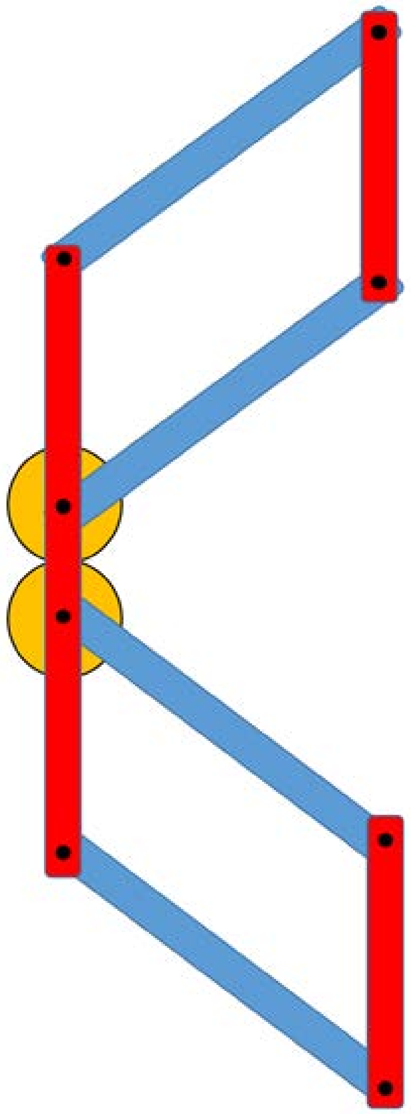


Diagram of a double reverse four-bar with pin joints in black and gears in gold.

Six-Bar Linkage: consists of 2 four-bars that are on top of each other. The Top link in the lower four-bar is also the bottom link of the upper four-bar. The vertical link at the end of the lower four-bar is also the vertical link at the beginning of the upper four-bar. These two shared links make this “double four-bar” a six-bar instead of an eight-bar. You can continue to “stack” four-bars on top of each other sharing 2 links between each four-bar to get an eight-bar, ten-bar or N-bar.

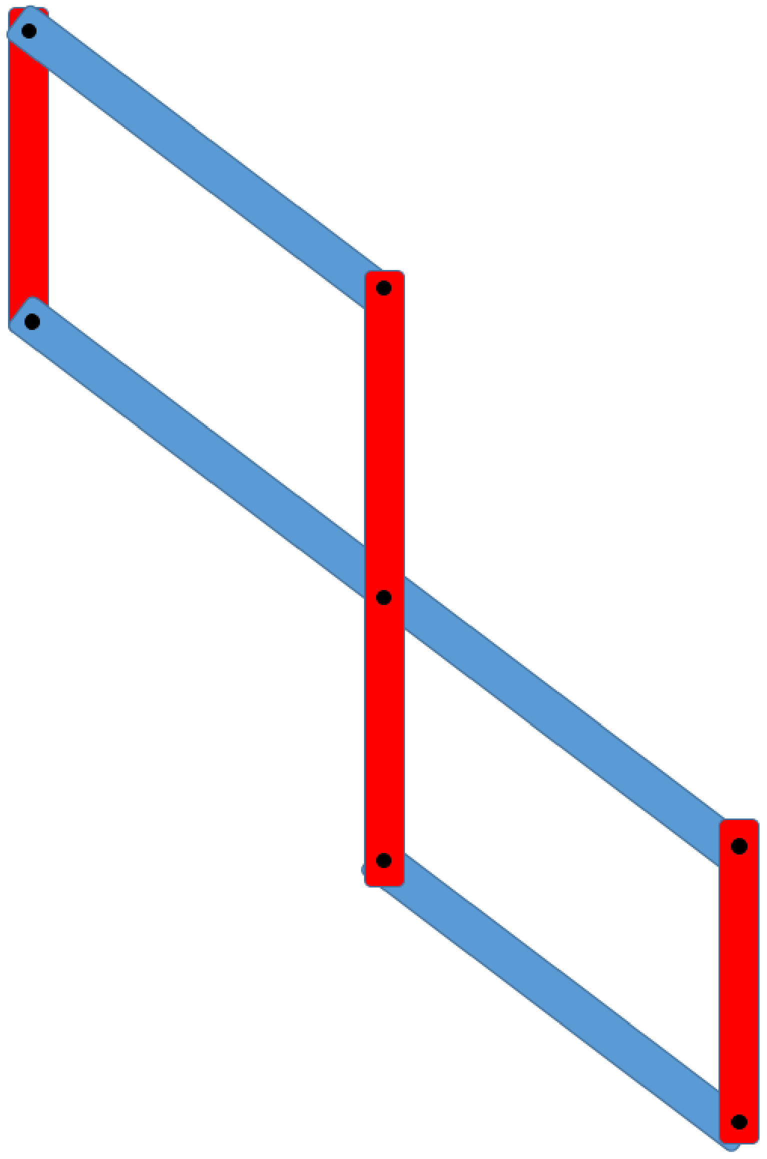
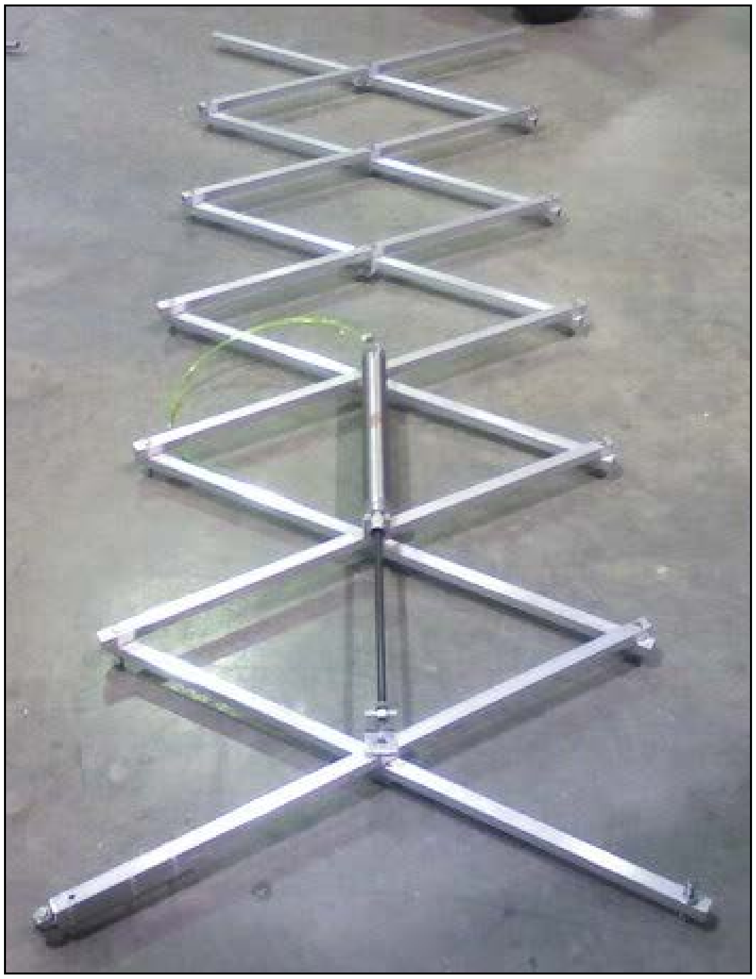


Diagram of a six-bar with pin joints in black.

Scissor Lift Linkage: a complex linkage used to get linear motion. They are very complex, very heavy, have a very high part count, are difficult to smoothly raise depending on the method of actuation, and sway an extreme amount at the top. We recommend avoiding them.

If you ever get the chance to look at a scissor lift used to lift people, look at the size of its bearings, bushings, and pin joints, and you will be able to see why, in industry, scissor lifts are usually only used on heavy equipment.



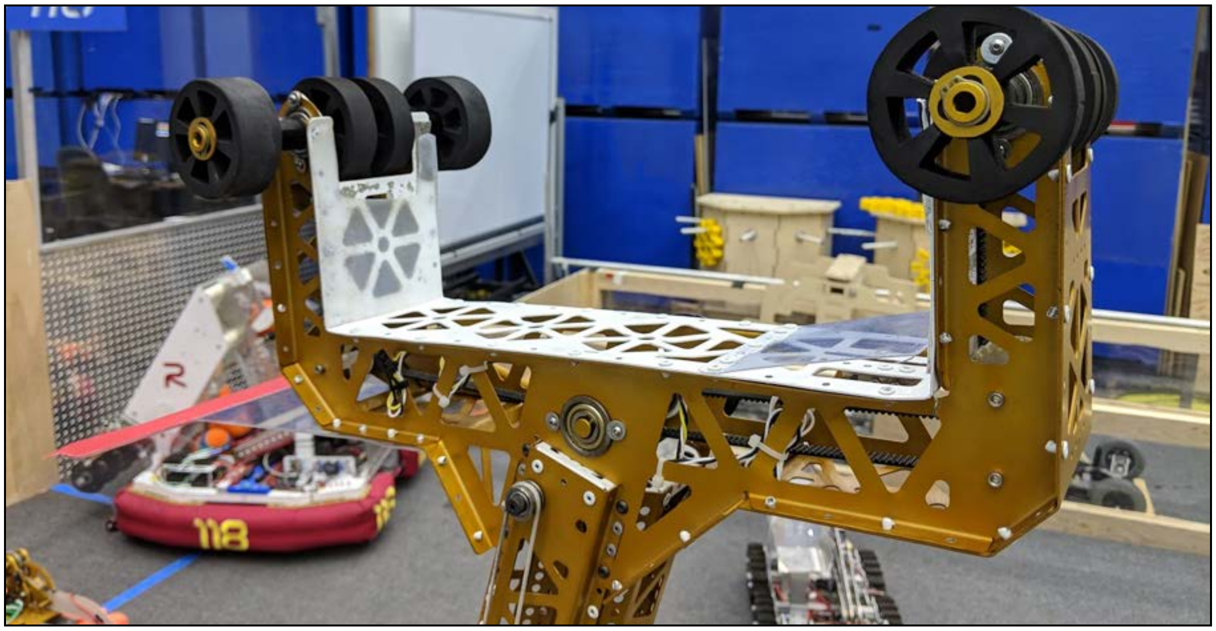
Pneumatically extended scissor lift built by Team 399.

5.5 Intakes

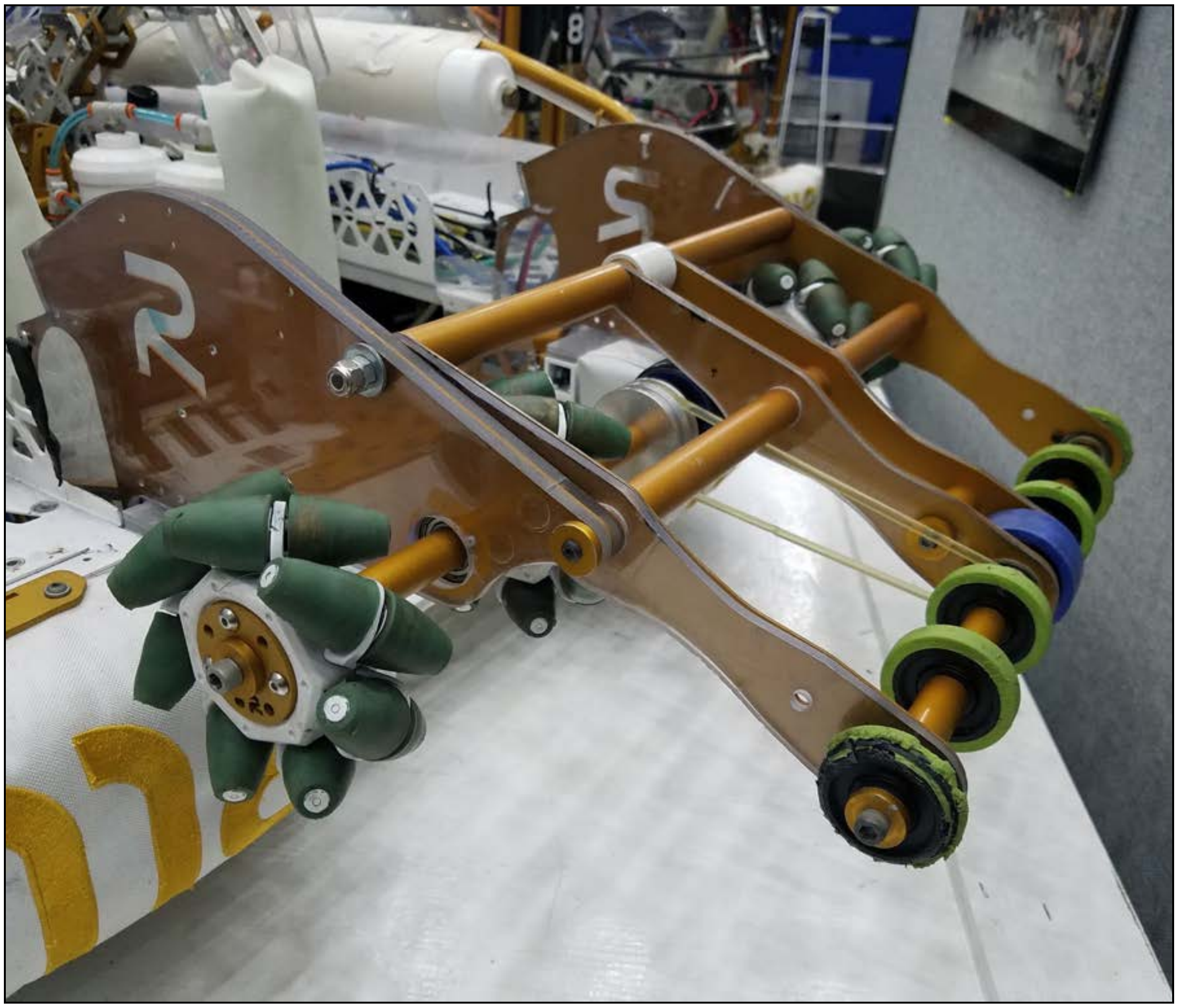
Having an effective intake is critical to playing the game at a high level. Almost always the best intake will have some sort of powered roller incorporated in it that allows the intake to be "Touch it, Own it." However many teams do find success with non-roller intakes, such as pinchers, and some robots are even successful with no intake at all such as 330 and 3310 in 2017, which had no ball intake. The list of intake architectures below is not a comprehensive list, however it covers all major intake architectures used in FRC.

5.5.1 Intake Architectures

Top and Bottom Roller (Roller Claw): a roller intake architecture in which there are two counter-rotating rollers rotating on an axis parallel to the ground, in which a game object is sucked in between. Frequently used to intake inflatable game objects, such as those from the 2007 and 2011 FRC games. This style requires the game object to be kicked up over the bottom roller and then is sucked into the claw. It tends to be more effective on game objects with a larger radius on the bottom edge than it is on more square objects.

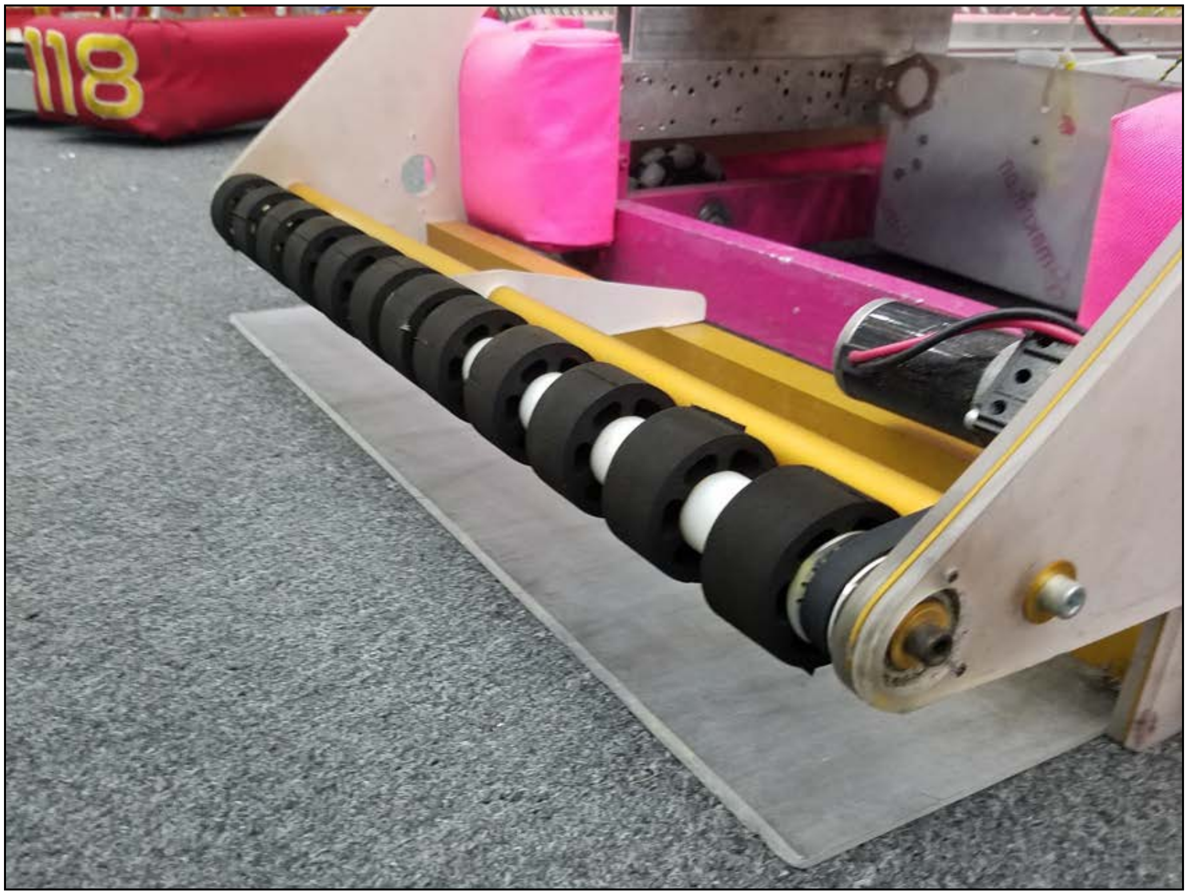

Roller claw on Team 118's 2018 robot.

Top Roller: a roller intake architecture that consists of at least one horizontal roller rotating on an axis parallel to the ground. It is often used to intake spherical game objects. Many times this style consists of more horizontal rollers behind the main front roller that guide the game objects into the correct part of the robot.

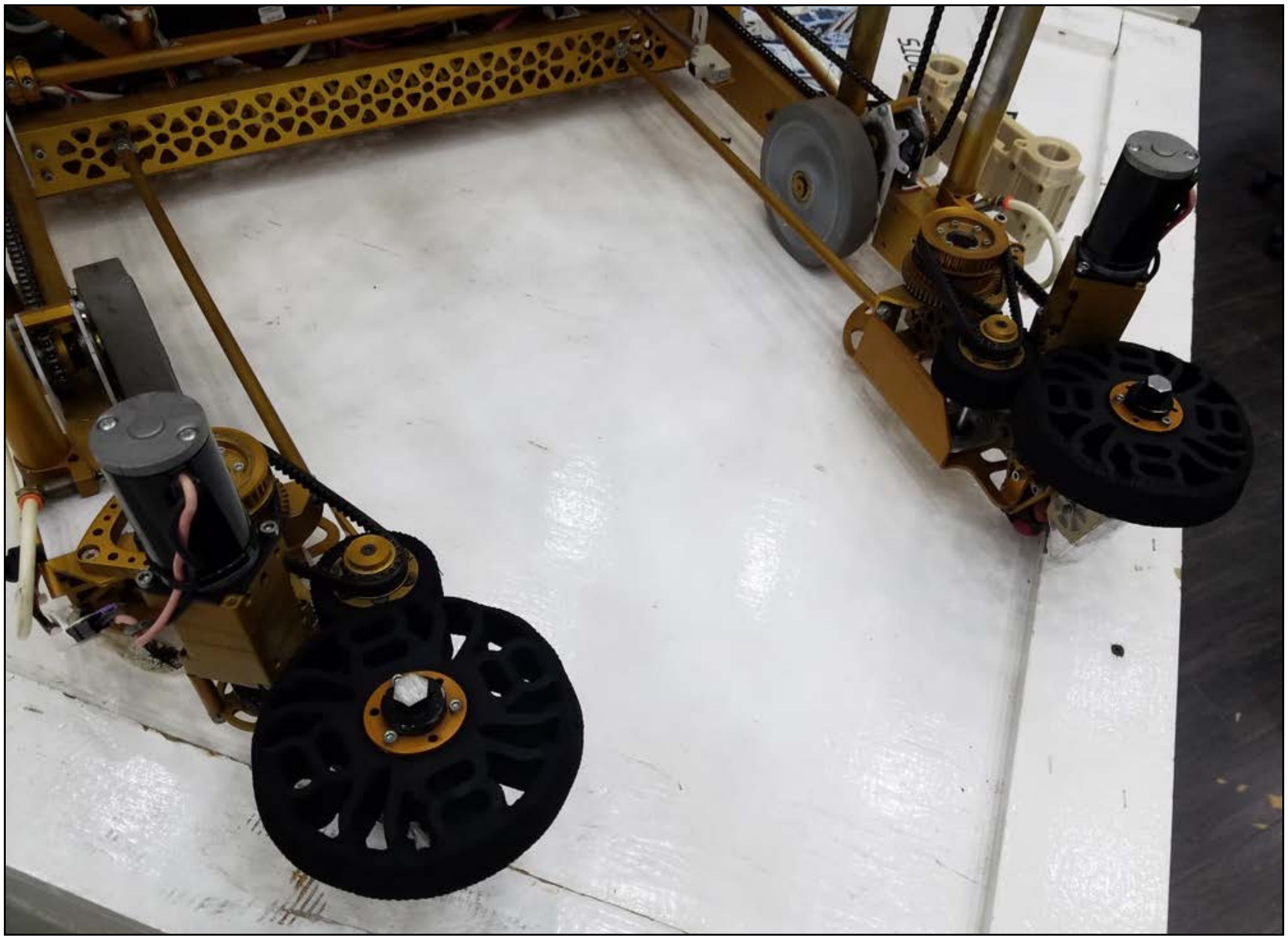


Team 118's top roller intake with mecanum wheels on their 2016 robot.

Top Roller and Dustpan: a roller intake architecture similar to the roller claw, except instead of a bottom roller there is a very thin piece of material, often chamfered, that rides on, or very close to, the carpet. The top roller comes in contact with the game object first and sucks it up onto the dust pan. This allows you to have a positive grip on the game object to manipulate it without losing control of the object. This intake architecture was used by many teams in the 2017 game for acquiring gears and by a few teams to acquire cubes in the 2018 game.

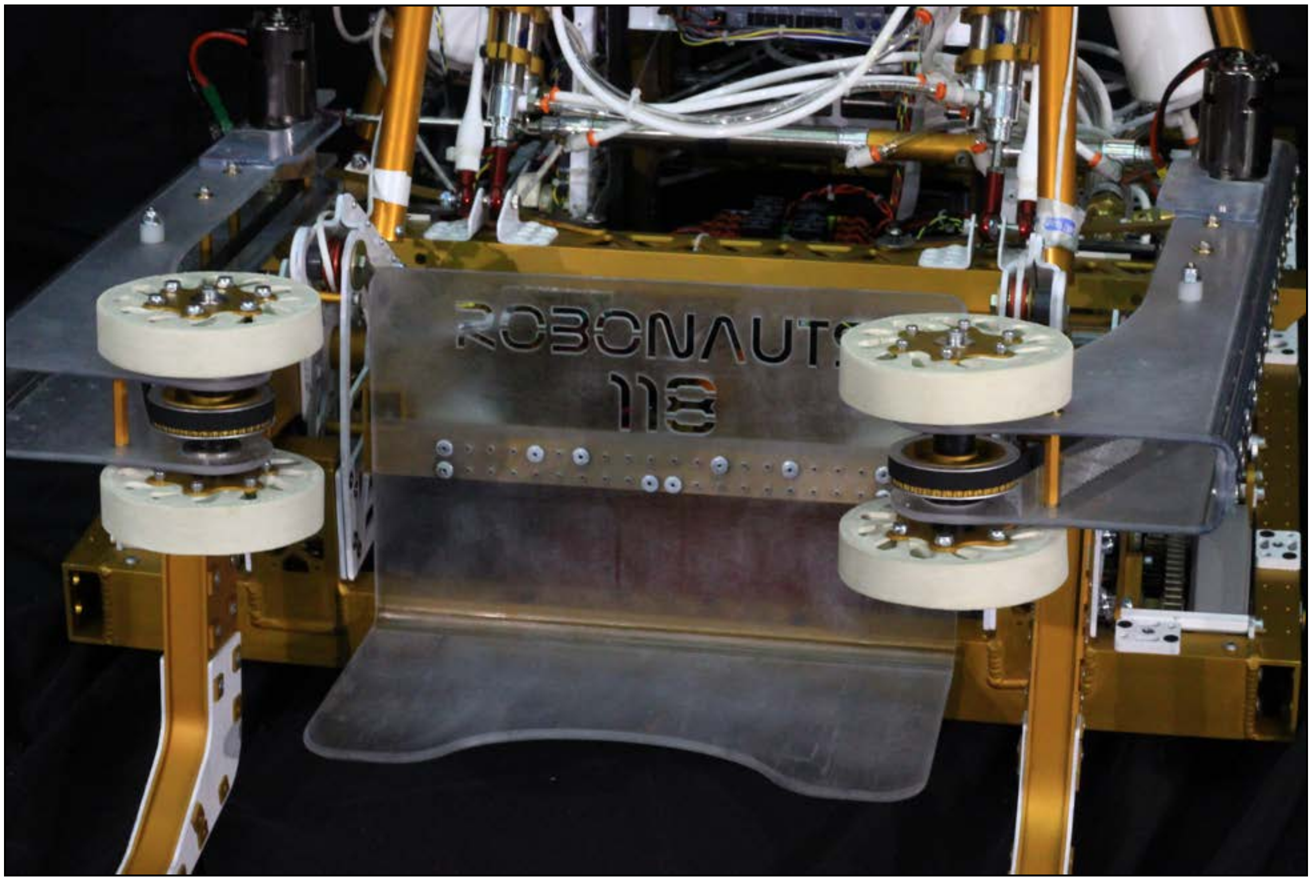

Top roller and dustpan intake on Team 118's driver training robots; designed to pick up gears from the 2017 game.

Side Rollers: a roller intake architecture consisting of two or more sideways rollers rotating on an axis nominally perpendicular to the ground that intakes a game object into a robot. This style was used by many teams in the 2015 game to intake totes and recycling containers.



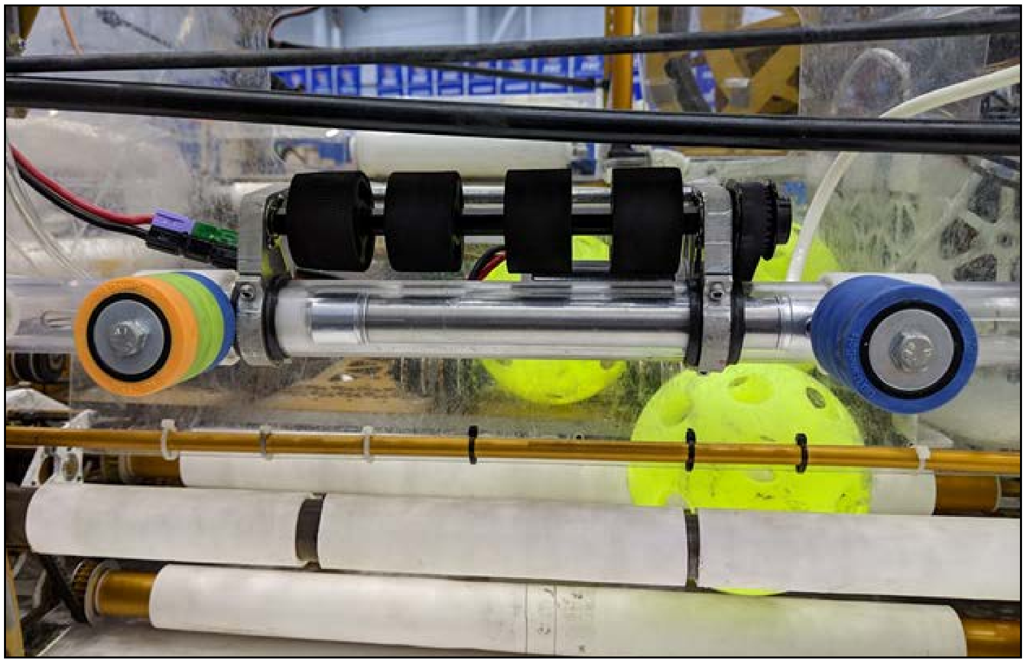
Side roller intake on team 118’s 2015 robot.

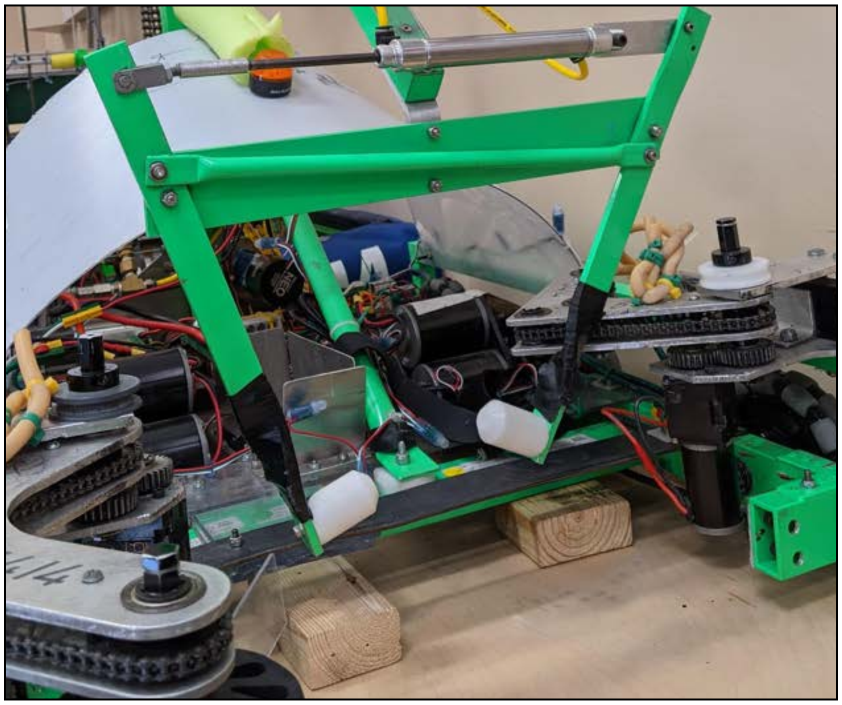
Side Roller and Dustpan: a roller intake architecture rarely used in FRC until the 2018 game. It consists of two or more sideways rollers rotating on an axis nominally perpendicular to the ground. The wheels contact the game object on the sides and suck game objects onto a dustpan, allowing them to be manipulated off the ground. Notably used by Team 118 during the 2018 season.



Team 118's side roller and dustpan intake from their 2018 robot.

Pincher: a non-roller intake architecture in which a claw-like mechanism grabs a game object and picks it up. This style is usually less effective than a roller intake due to not being a “Touch it, Own it” intake. However, pincher intakes are usually much simpler to build, and can be as simple as just a pneumatic cylinder pinching a game object with bolts.


Pneumatic pincher with roller on Team 118’s 2017 robot.


Pincher intake on Team 5414’s 2018 robot.

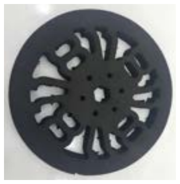
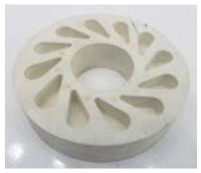
5.5.2 Types of Intake Wheels

There are dozens of different types of intake wheels available made specifically for FRC use and countless others that can be modified for use in FRC.

Many wheel descriptions include the durometer (usually a number followed by the letter A, i.e. 30A) or Shore. The lower the number the softer the wheel, thus the easier it is to compress. However, a 45A wheel in one material may be easier to compress than a 30A wheel in another material. The material, along with the shape of the wheel all have an effect on how well a wheel will pick up a game piece.

Intake material greatly affects how well your intake works. The amount your intake compresses the game object or amount your game object compresses your rollers also affects how well your intake works. To find the best combination we recommend prototyping as many different configurations as possible, choosing the best one, and designing the intake on your competition robot around that data. Below is a list of some of the most commonly used intake wheels.

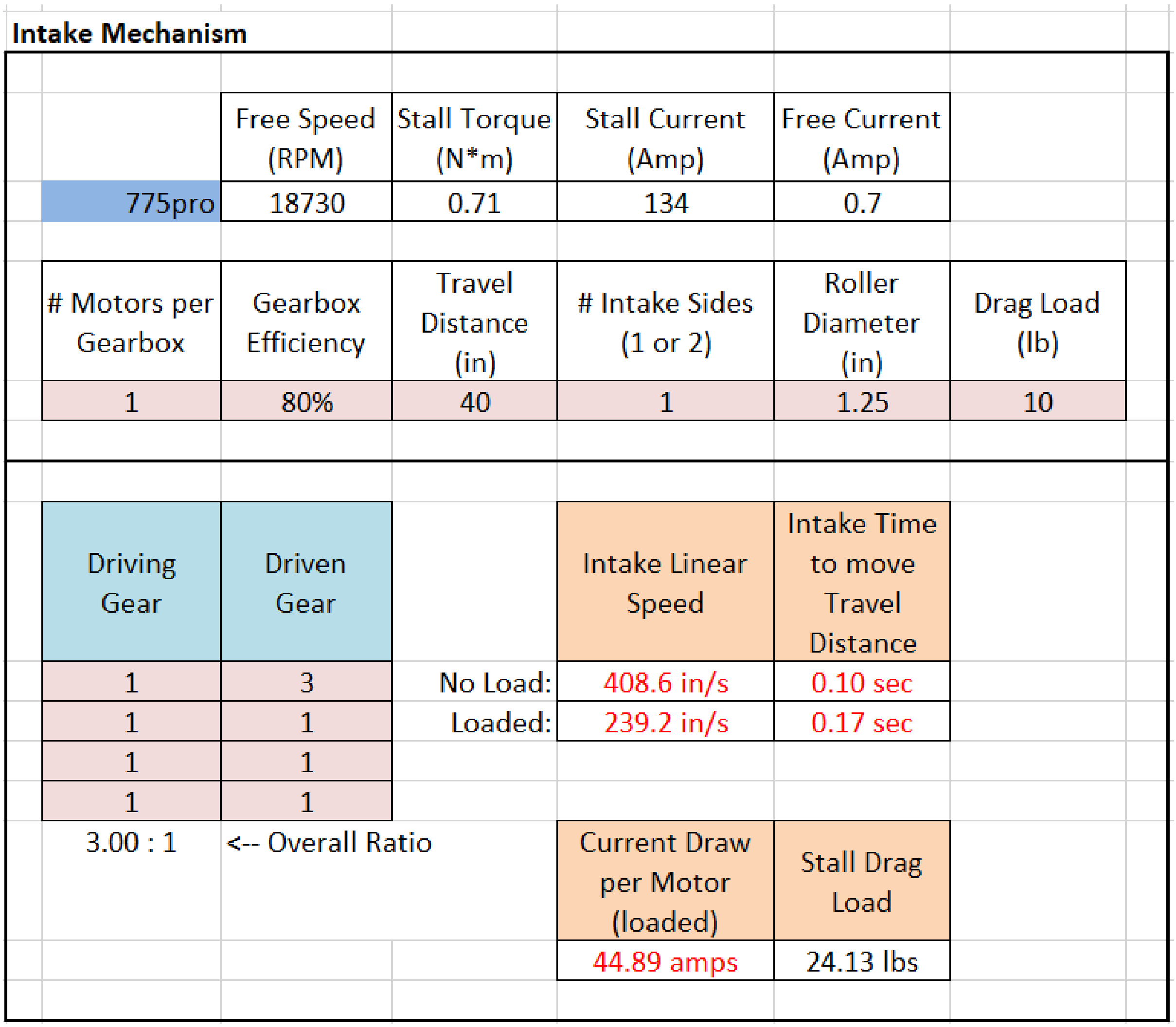
Wheel	Description	Photo
Compliant Wheels	Soft rubber wheels sold by Andymark in a variety of sizes and Durometers. In general they are great intake wheels for many game objects.	
Flex Wheels	VEXpro's brand name for compliant wheels. Come a variety of durometers and diameters. Also legal in the VEX Robotics Competition as of 2020.	
Stealth Wheels	Hard plastic wheels covered in a thin, grippy TPUlayer. They also come in a variety of diameters and durometers.	
Colson Wheels	Often used in drivetrains, but can be useful in intakes as well. They come in a large variety of diameters and are 65A.	

Wheel	Description	Photo
Mecanum Wheels	Mecanum wheels have become popular on intake because they are very effective at centering spherical game objects, and even footballs.	
Andymark HiGrip Wheels	Often used in drivetrains, and are also useful for intakes. Come in various diameters and durometers.	
Space Wheels	A type of custom intake wheel originally developed by Team 118 during the 2015 season. They are waterjetted out of neoprene rubber. WCP now sells derivatives of the Space Wheels.	
BaneBots Wheels	Thin intake wheels that come in a variety of odd diameters and durometers.They are very soft and therefore, wear quite quickly.	
RC Car Wheels	Many rubber wheels, sometimes filled with foam, are available for RC cars. These have frequently been modified for FRC use by Team 973.	
Durasoft Wheels	Heavy rubber wheels sold by Fairlane Products that have been used on intakes. They come in urethane, neoprene, and nitrile.	
Surgical Tubing Rollers	Custom rollers in which surgical tubing is forced over aluminum or polycarbonate tubing to create sticky rubber rollers. Frequently used by Team 254.	

5.5.3 Intake Ratio Selection

In general, you want the linear speed of your intake roller to be at least faster than the linear speed of your drivetrain. Some teams design their intake rollers to run at twice the linear speed of the robot's drivetrain. If you drive full speed into a game object with an intake roller that is running at a linear speed slower than your drivetrain, your intake won't be able to acquire the game piece;, it will just push the game piece along the field.

The "Intake Mechanism" tab of the JVN Design Calculator can be used to calculate the linear speed of your intake roller, and is great for choosing a ratio that is fast enough to intake while driving at full speed as well as having enough torque to acquire the game piece.
The screenshot below shows an example intake ratio calculation.



Intake Mechanism

	Free Speed (RPM)	Stall Torque (N*m)	Stall Current (Amp)	Free Current (Amp)
775pro	18730	0.71	134	0.7

# Motors per Gearbox	Gearbox Efficiency	Travel Distance (in)	# Intake Sides (1 or 2)	Roller Diameter (in)	Drag Load (lb)
1	80%	40	1	1.25	10

Driving Gear	Driven Gear		Intake Linear Speed	Intake Time to move Travel Distance
1	3	No Load:	408.6 in/s	0.10 sec
1	1	Loaded:	239.2 in/s	0.17 sec
1	1			
1	1			
3.00 : 1	<-- Overall Ratio			

Current Draw per Motor (loaded)	Stall Drag Load
44.89 amps	24.13 lbs

(Screen capture from JVN's Design Calculator)

When selecting a motor for an intake, you need to keep in mind that stalling certain motors, such as the 775pro, will very quickly destroy them, while a BAG motor can run at stall for longer before being destroyed. Some teams design their mechanisms to run 775pros at a lower voltage, protecting the motor. If the roller on your intake slips on the game piece when the game piece is sucked in all the way, it may be reasonable to use a fan-cooled motor, such as a 775pro or a 550. But if your intake rollers stop moving and lock up when the game piece is sucked in all the way, it may be wiser to use a motor that can survive being stalled for a short period of time, such as a BAG motor, Mini-Cim, or Cim, or to integrate current limiting in your control software to protect the motor.

5.6 Shooters

Shooters are frequently used in FRC to launch a game object into a scoring element. The three main categories of shooters are flywheels, catapults, and linear punchers.

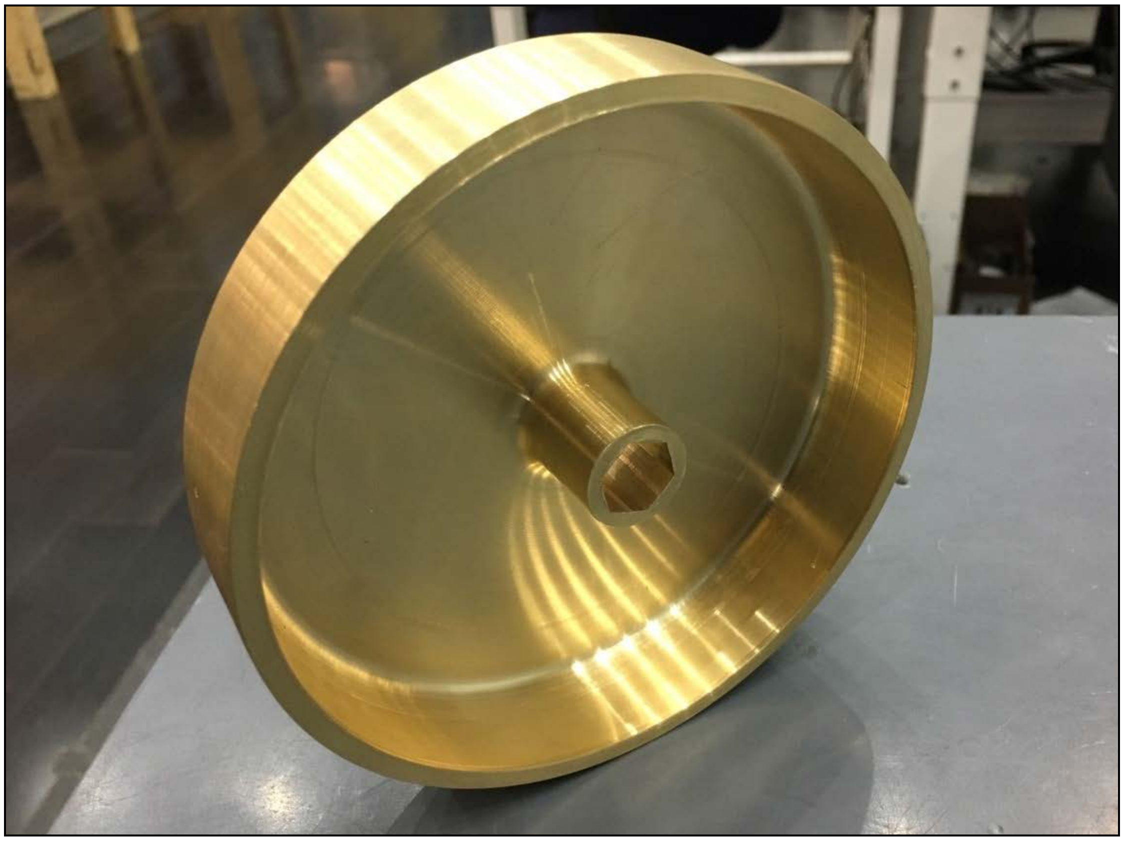
5.6.1 Flywheel Shooters

Flywheel shooters consist of a wheel, or wheels, spinning at several thousand RPM. The game object is then fed through the wheel, or wheels, and accelerates until it leaves the shooter.

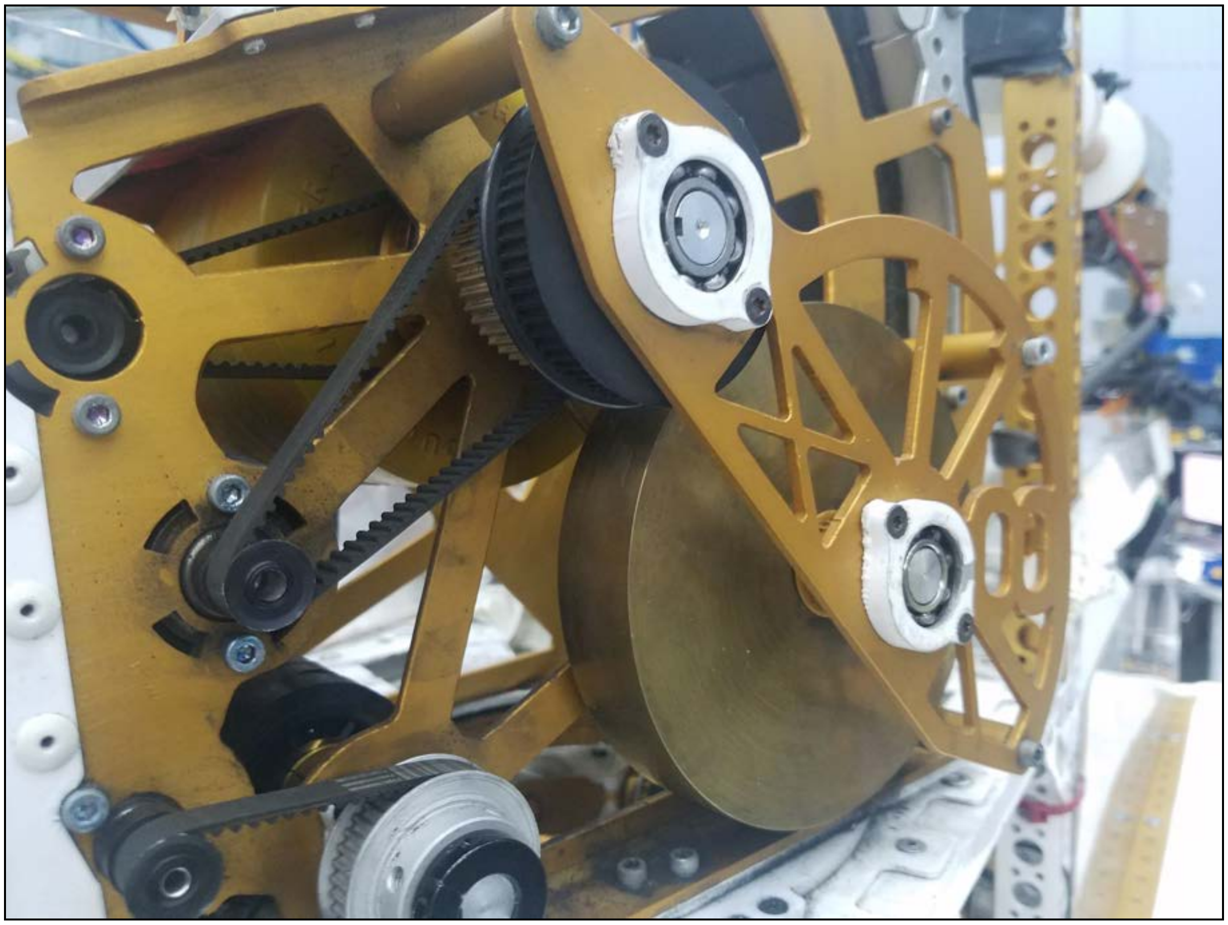
When a game object goes through a flywheel shooter, the wheel slows down. The wheel slowing down too much can cause your next few shots to be inaccurate. If it is a game where you can only control one game object at once, like in 2016, this isn't that big of an issue because there is plenty of time to get the flywheel back up to speed. In a game like 2017, where game objects need to be launched in very quick succession, the wheel slowing down can cause you to miss most of your shots. A mechanical way to solve this is to add more inertia to the system. It can also be solved by sizing the motor to accelerate the wheel back up to speed very quickly between shots, this requires a robust and highly accurate software control loop.

To increase the inertia of the system, you can increase the speed and mass of the wheel. Increasing the speed usually will make your shooter no longer perform the way you want, and increasing mass when you have a 120 pound weight limit, isn't ideal. To increase inertia without increasing mass by much, teams such as 118 in 2017 and 2020 had a seperate wheel geared to rotate much faster than the shooter wheel. This keeps the shooter performing like you expect, and increases the inertia of the system greatly. The 1.5 pound inertia wheel spinning at a much higher speed was equivalent to a 15 pound wheel spinning at the speed of the shooter wheel. The advantage to this is that Team 118 saved 13.5 pounds on their shooter, while still being able to shoot at a rate of 10 balls per second.

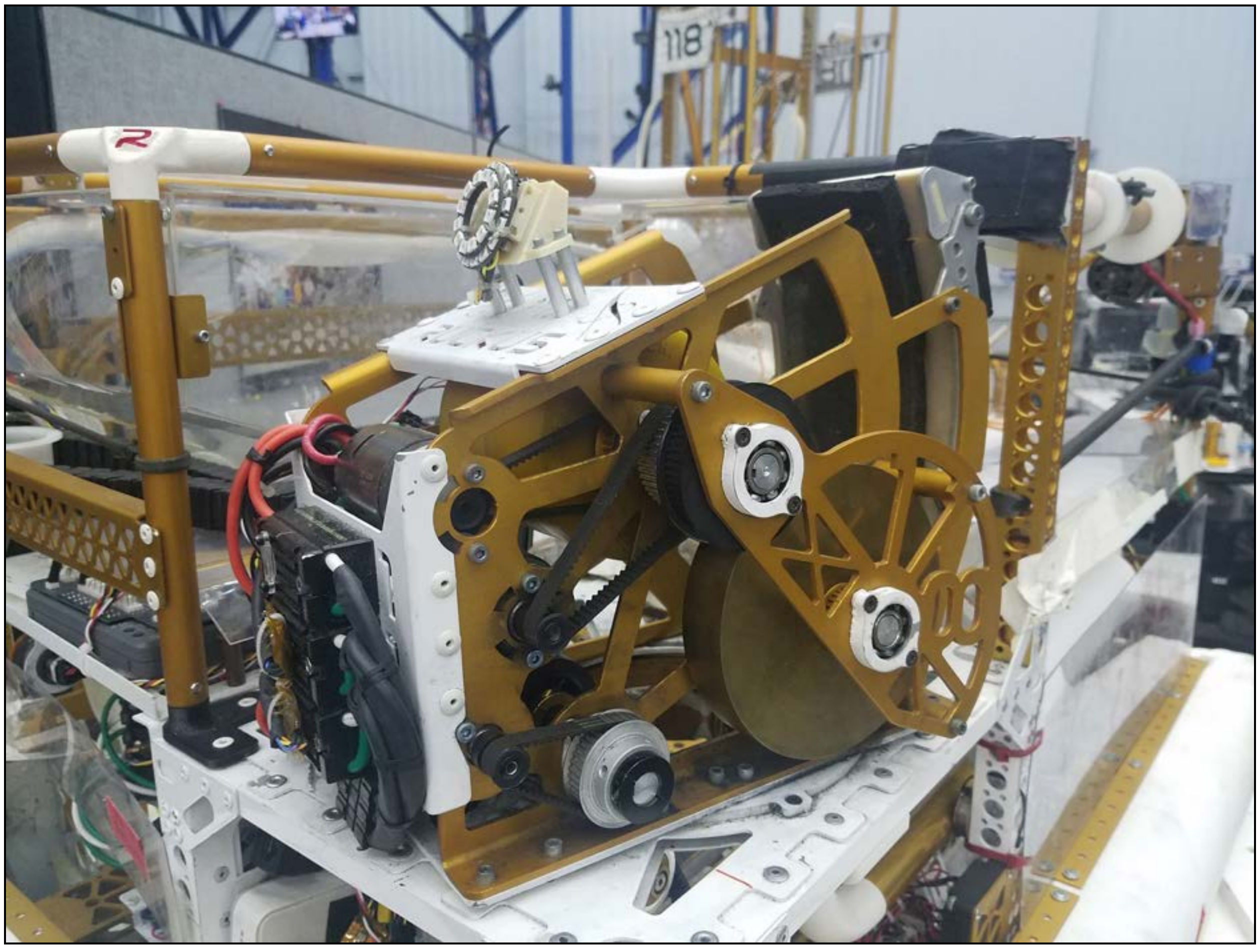
Mass near the edge of the wheel will provide more inertia than mass near the axis of rotation. For this reason the inertia wheel on Team 118's 2017 robot is milled out on the inside, just leaving a thin rim of material around the edge. The wheel is also made of brass, as brass is more dense than aluminum. Other teams, such as Team 1678, pressed an aluminum hub into a brass rim using differential heating to create a wheel with high inertia.


The brass 1.5 pound inertia wheel from Team 118's 2017 robot.

Care should be taken that the bearings being used are rated for the speeds at which an inertia wheel is spinning, as well as the force that it is generating. Hex bearings have a "sloppier" fit than most round bearings. This can cause severe vibrations at high speeds and cause bearings to prematurely fail. Thunderhex or round bore bearings are a better choice for very high speed mechanisms. McMaster also sells bearings rated for very high speeds such as these, used by Team 118 on their 2017 shooter.

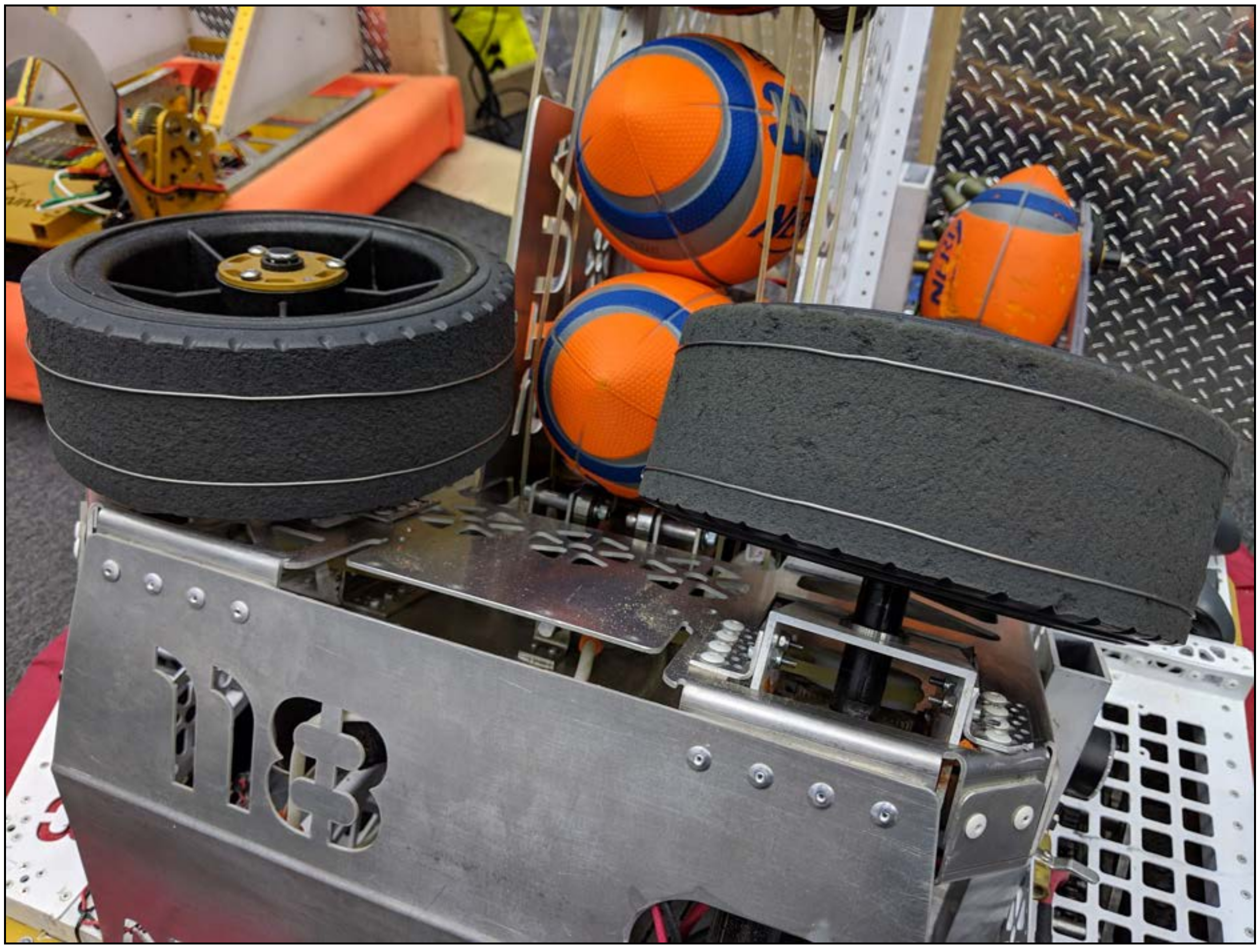

The inertia wheel assembled in the shooter on Team 118’s 2017 robot.

Single Flywheel Shooter: consist of a shooter wheel and a shooter hood. The ball is compressed between the wheel and the hood to accelerate it. Depending on the game object, the compression can come from the wheel, the game object, or the hood. Single flywheel shooters are usually the most compact flywheel shooter, as well as the easiest to integrate into a turret. Single flywheels can adjust their shot by changing the speed of the wheel and by changing the geometry of the hood. A hood that wraps further around the wheel will produce a flatter shot.



The single flywheel shooter with inertia wheel on Team 118's 2017 robot.

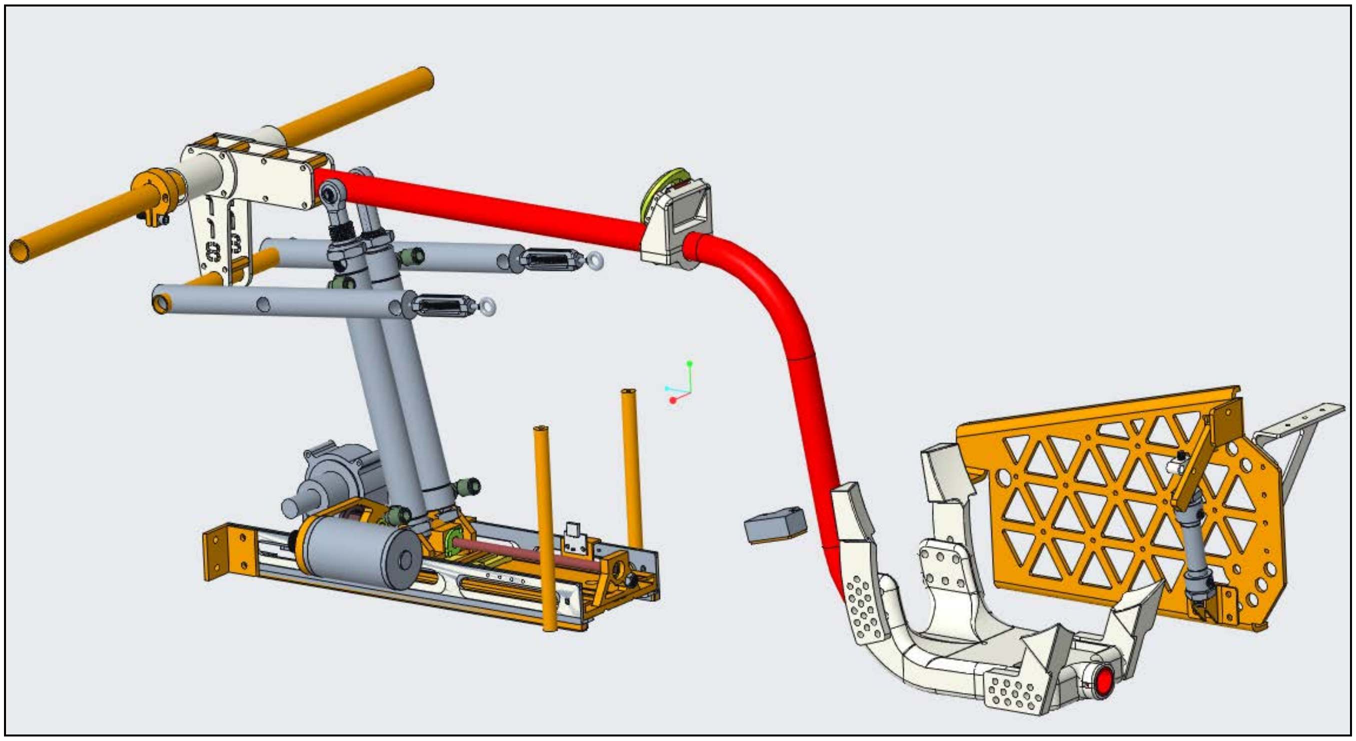
Dual Flywheel Shooter: consists of two counter-rotating wheels a game object is sent through. Just like a single flywheel, compression is needed to accelerate the game object, and this compression can come from either the game object or the wheels. Best practice is to link both of the wheels with belts, chain, or gears to keep them running at the same speed. To adjust the shot from a dual flywheel shooter, the speed can be adjusted.


Dual flywheel shooter on Team 118's football shooting robot.

5.6.2 Catapult Shooters

Catapult shooters are great for game objects that are not consistent in regards to their compression, as a catapult does not rely on compression to accelerate the game object. Catapults can also be used to launch irregular shaped objects that would be impossible to launch with a flywheel. There are a variety of different launch mechanisms, such as pneumatics, slip gear launchers, drop cams, or choo-choo linkages.

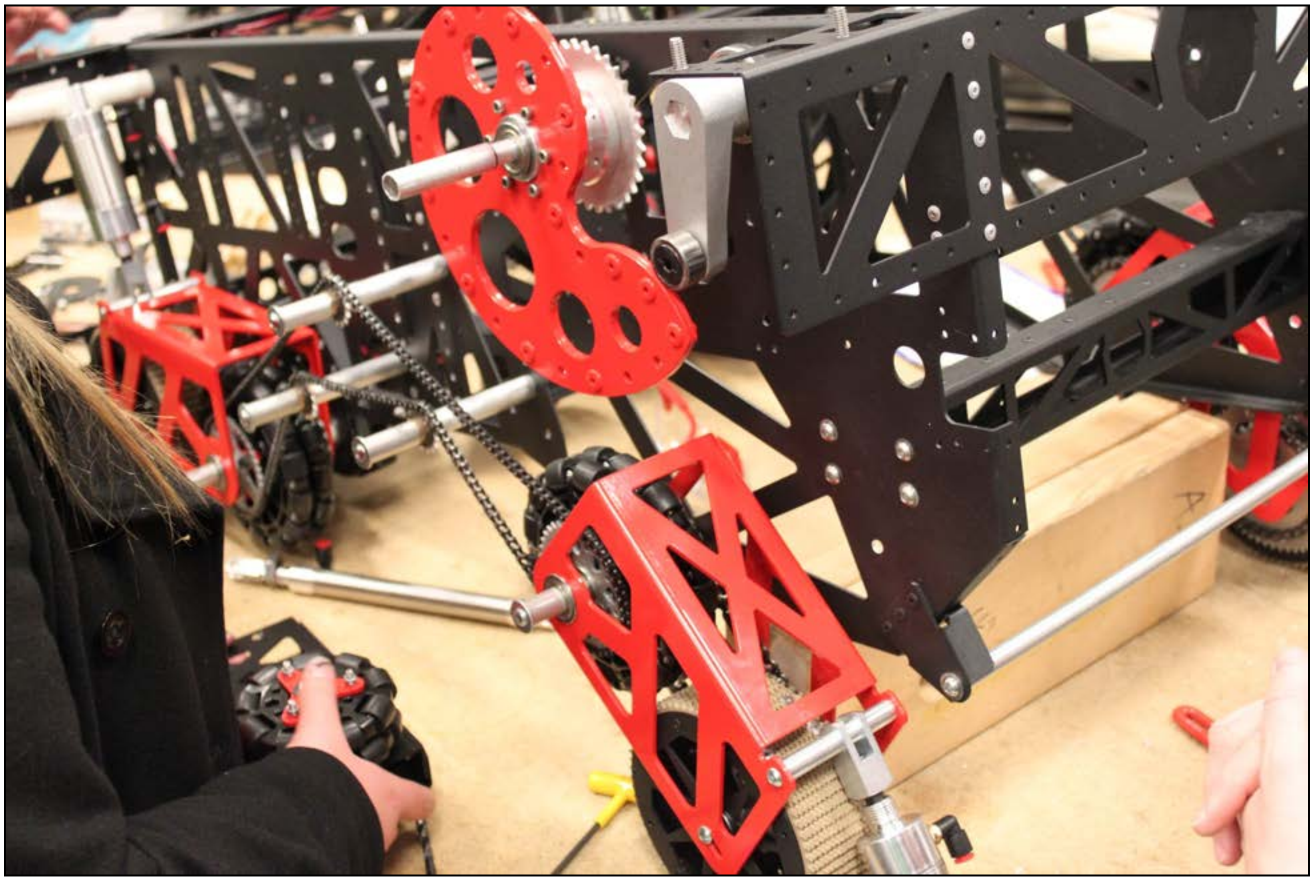
Pneumatic Catapults: launch game objects by firing pneumatic cylinders attached near the pivot point of a catapult arm. Pneumatics can be used in conjunction with springs or elastics to launch heavier game objects. Things to consider when designing a pneumatic catapult is that when the pressure in the pneumatic system changes, the shot accuracy can be affected. Ways to adjust shots are not using all of the cylinders when a shot with a lower force is desired. Another, more complex, method Team 118 used on their 2016 robot was to move the base of where the pneumatic cylinders were mounted. This changed the angle at which the force was put into the catapult arm, affecting how much force with which the catapult launched.



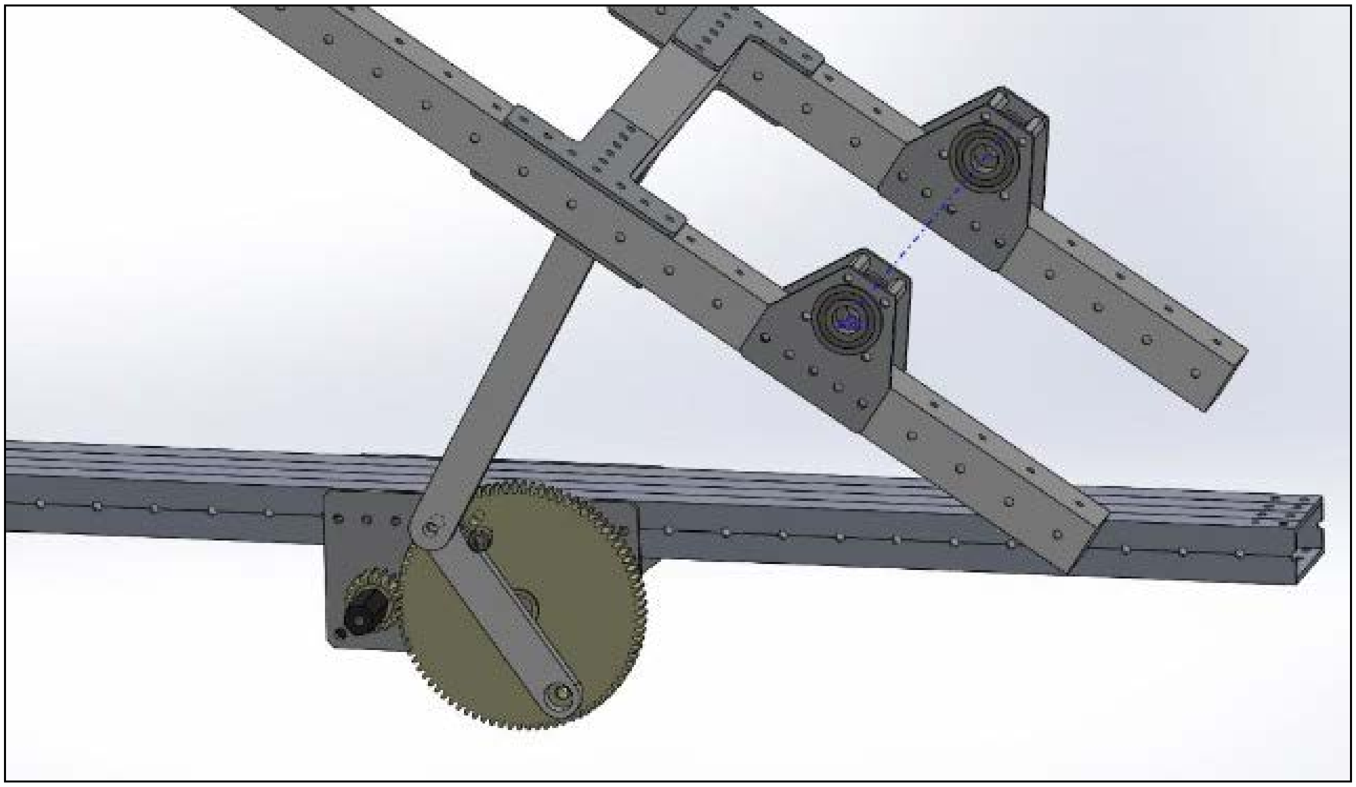
Pneumatic catapult from Team 118’s 2016 robot.

Slip Gear Catapults: consist of a pull-back gear in the gearbox that is missing teeth which meshes with a gear mounted to the catapult arm that has all its teeth. When the pull-back gear is meshing with the catapult arm gear, the arm is being retracted, which extends springs or elastics loading the arm. When the pull-back gear rotates to where there are no teeth on the gear, the catapult fires and the retraction process starts over. A hard stop is needed to stop the catapult at the end of its travel. This type of catapult is more popular in VEX than in FRC, however it has been done in FRC. Some disadvantages are that right before the catapult fires, all the load of the catapult is held back with just one tooth, which could damage the gear depending on the load.

Drop Cam Catapult: an eccentric wheel that smoothly retracts a catapult. The catapult has a follower, which is usually a bearing that rolls on the cam. The catapult arm has elastics or springs attached to it that extend while the catapult is retracted. You can learn more about how cams work here.


Red drop cam with bearing follower on Team 148's 2010 robot.

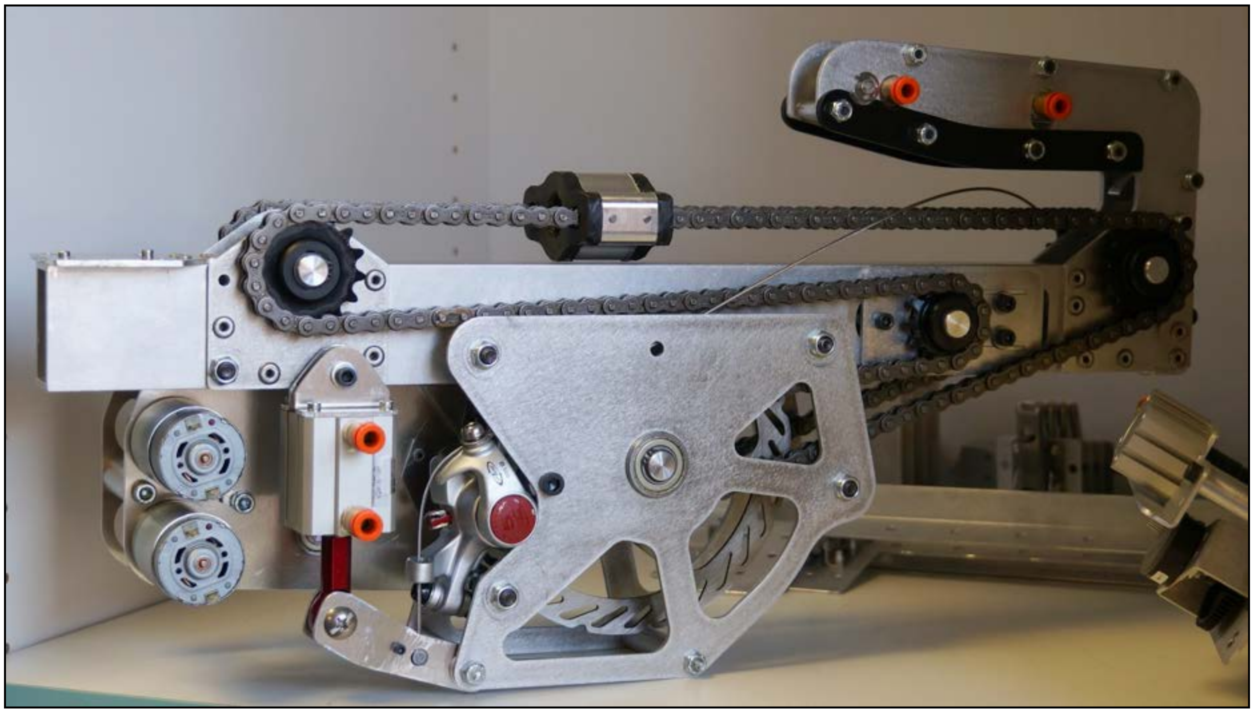
Choo-Choo Catapult: uses a choo-choo linkage to retract a catapult arm that extends springs or elastics. The choo-choo mechanism works by essentially rolling up 2 links to retract a catapult. The links need to be rigid and get loaded in tension. Was made popular by JVN's 2014 Build Blitz team.

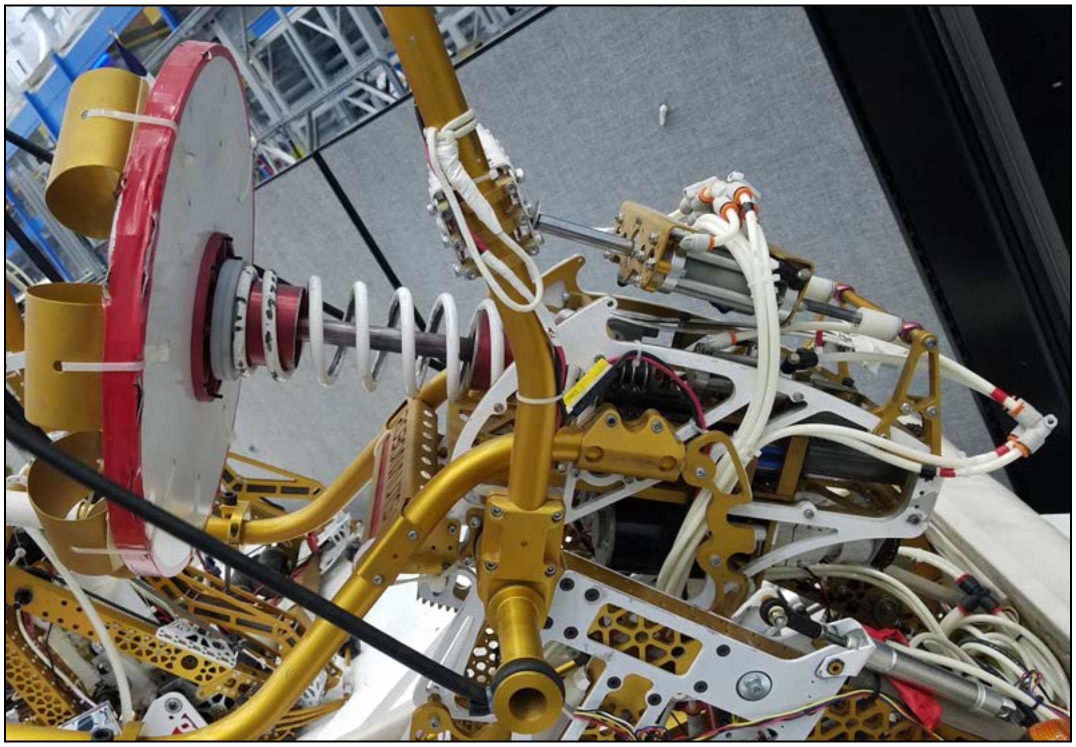

Choo-Choo catapult from Team JVN's 2014 Build Blitz robot.

5.6.3 Linear Punchers

A linear punch retracts and extends linearly, and pushes and accelerates the object it is launching. There are several different methods of actuating a linear punch, such as a slip gear, drop cam, custom gearbox, and pneumatics.

Spring-Loaded Linear Puncher: use a pull-back mechanism to compress a compression spring or extend an extension spring to store energy in the mechanism. A variety of pull back mechanisms could be used. On Team 118's 2014 robot, a ball screw with a Sea-Catch is used to pull back a shooter plate and compress a compression spring. On team 971's 2014 robot, a carriage driven by a chain pulls the shooter plate back to the desired position where it is held in place with a disc brake. When the shooter is fired, the latch releases, and the process starts over. The carriage can vary the shot power by pulling back the carriage different distances.


The shooter gearbox from Team 971's 2014 robot.


Team 118’s spring-loaded linear punch on their 2014 robot.

Pneumatic Linear Punch: consists of pneumatic actuators that push a game object out of a robot. They can be useful for launching lightweight objects, or for launching short distances. They can also be assisted by elastics or springs. Several teams used these in 2018 to launch power cubes from the end of an arm into the scale. Team 4613 notably used a pneumatic punch on their 2018 robot to launch cubes into the scale. Their pneumatic puncher can be seen here.

5.7 Turrets

Turrets are commonly used to mount either shooters onto or a manipulator such as an arm, elevator, or both onto to allow scoring without rotating the drivetrain. A turret is a huge complexity to add to a robot, especially when used with manipulators that will put a large moment load into the turret bearing. For most games they are unnecessary for building a competitive robot and take a large amount of machining resources, programming resources, add cost and use up a motor slot to drive them. Armabot and WCP sell COTS turrets.

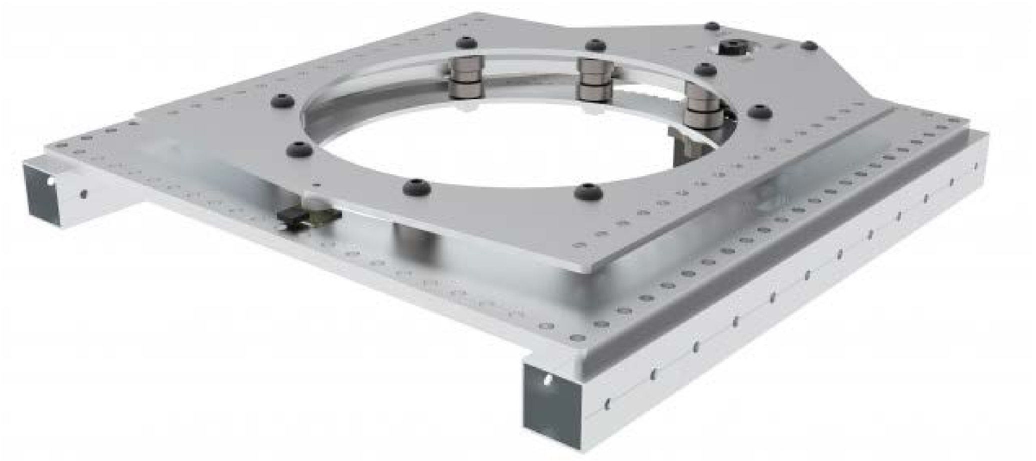
5.7.1 Turret Bearing Types

The most fundamental part of the turret is the bearing that the mechanism rides on. Most FRC turrets use one of the 4 types of bearing setups listed below.

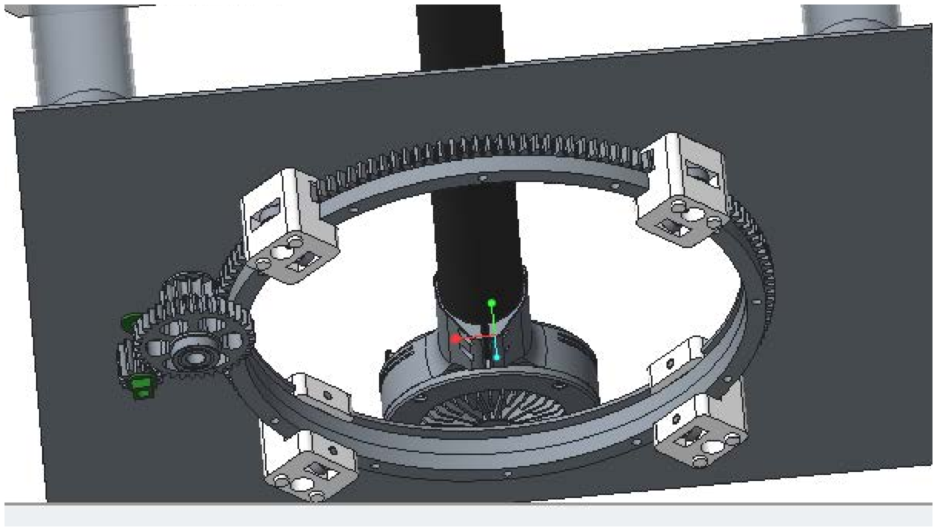
Lazy Susan: consists of a COTS lazy susan bearing that the mechanism mounts to. This is the simplest, and usually cheapest way to implement a custom turret solution. Most lazy susans are plenty strong enough to support the weight of a shooter but the loading from impacts can cause them to separate, spilling ball bearings all over the field. Many COTS lazy susans will struggle to support the moment loading from manipulators mounted to them. This lazy susan has been used by FRC teams in the past and variants of it have been used by Team 118 for non-FRC projects.

Lazy Susan Bearing.

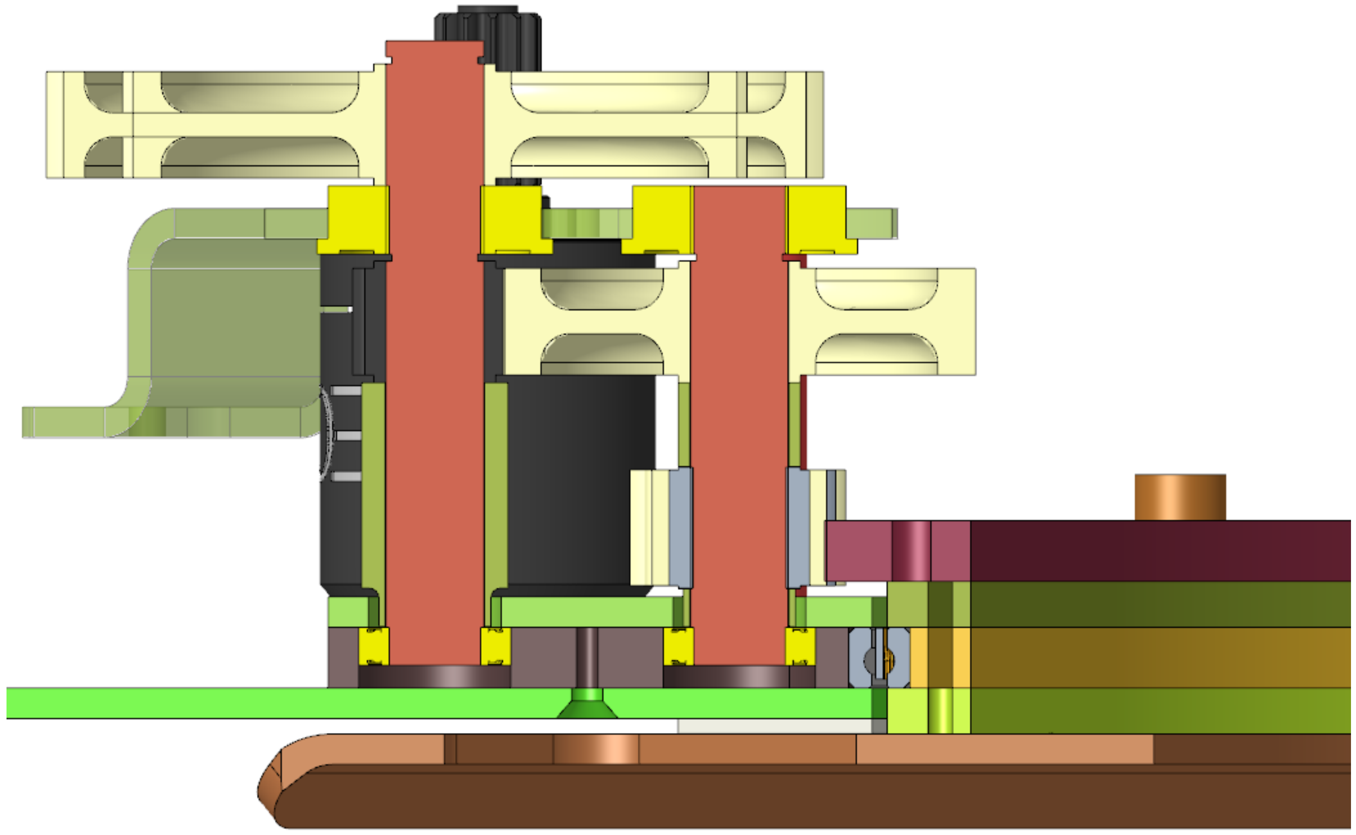
Shoulder Bolts with Bearings: this is the most common way for teams to implement custom turrets on their robots. It consists of several stacks of roller bearings around a turret ring that support the driven turret ring. This method works really well with shooter turrets but since the top and bottom bearings can see moment loads they aren't designed for this method doesn’t handle very high moment loads. While this method does have a high part count it allows more flexibility in turret diameter than lazy susans and is relatively inexpensive. This type of turret bearing setup has shoulder bolts perpendicular to the plane of the turret that have 3 bearings on each bolt. The top and bottom bearings are the same size and the turret ring sits in between the outer races of these bearings. The middle bearing rolls against the turret ring itself. This can be done with the bearing stacks on either the inside or outside of the turret as long as there is at least 3 to constrain the turret ring, 6-8 is recommended and will provide much better support to the turret. The WCP Greyt Turret uses this bearing implementation shown in the photo below.


WCP GreyT Turret.

3D Blocks with Bearings Riding on All Surfaces: this method is similar to the method above in that it uses a large amount of small bearings except in this one the outer race of all the bearings are contacting the turret ring. This is technically better as all bearings will be seeing radial loads and not moment or axial loads they are not designed for. It is more complex as the structure needed to support these bearings needs to be complex since each bearing is on its own axis. These blocks can be CNC machined however for many applications 3D printed blocks can be strong enough.


Turret CAD showing 4 3D printed Blocks that each support 3 bearings that the turret rides on.

Singular X-Bearing: a turret that is supported by one large, thin section X-bearing. X-bearings support radial, thrust (axial), and moment loads. This method is the best for mechanisms with large moment loads as the bearings used specify the moment loading capacity of the bearing. This is also the method commonly used to support swerve modules. Popular aerospace industry suppliers are Kaydon and Silverthin however bearings from these companies, especially in the larger sizes are too expensive to be used in FRC. However, cheaper vendors such as Lily Bearing Manufacturing or The Thrifty Bot sell much cheaper bearings that are equivalent to those offered by Kaydon and Silverthin. The bearing needs to have either the inner or outer race clamped to the fixed structure of the robot while the moving part of the turret is clamped to the opposite race. Team 118 favors this bearing implementation as it is the most robust and has used large, thin section X-bearings on their 2012, 2017, and 2020 robots. The CAD screenshot below from Team 118's 2020 turret shows this implementation. The outer race is clamped to the fixed robot structure while the moving part of the turret is clamped to the inner race of the bearing and has the drive gear (shown in maroon) attached to it.

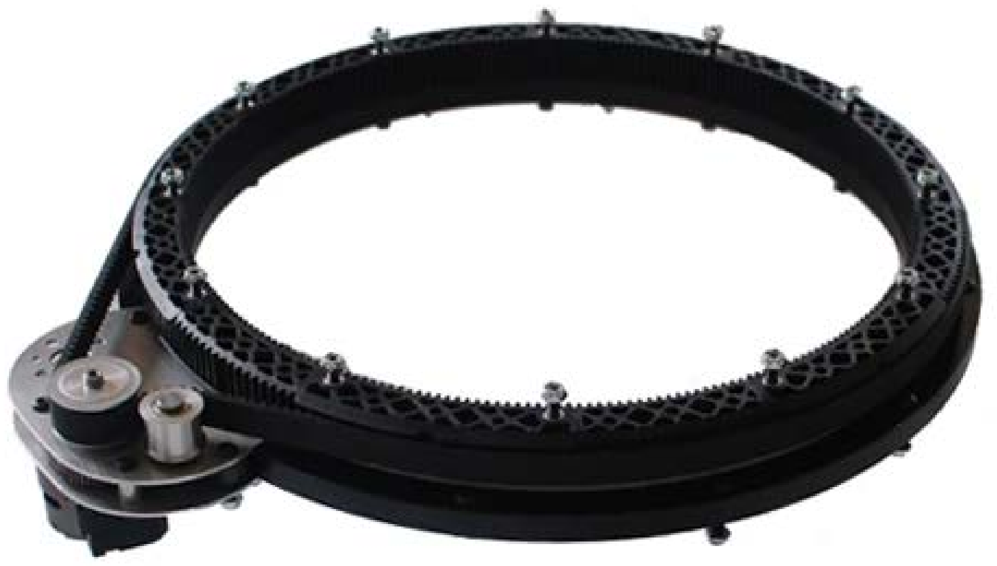


Cross-Section of Team 118's 2020 Robot's Turret Gearbox.

5.7.2 Turret Drive Types

The 3 most common types of turret drive mechanisms are chain, belt, and gear driven.

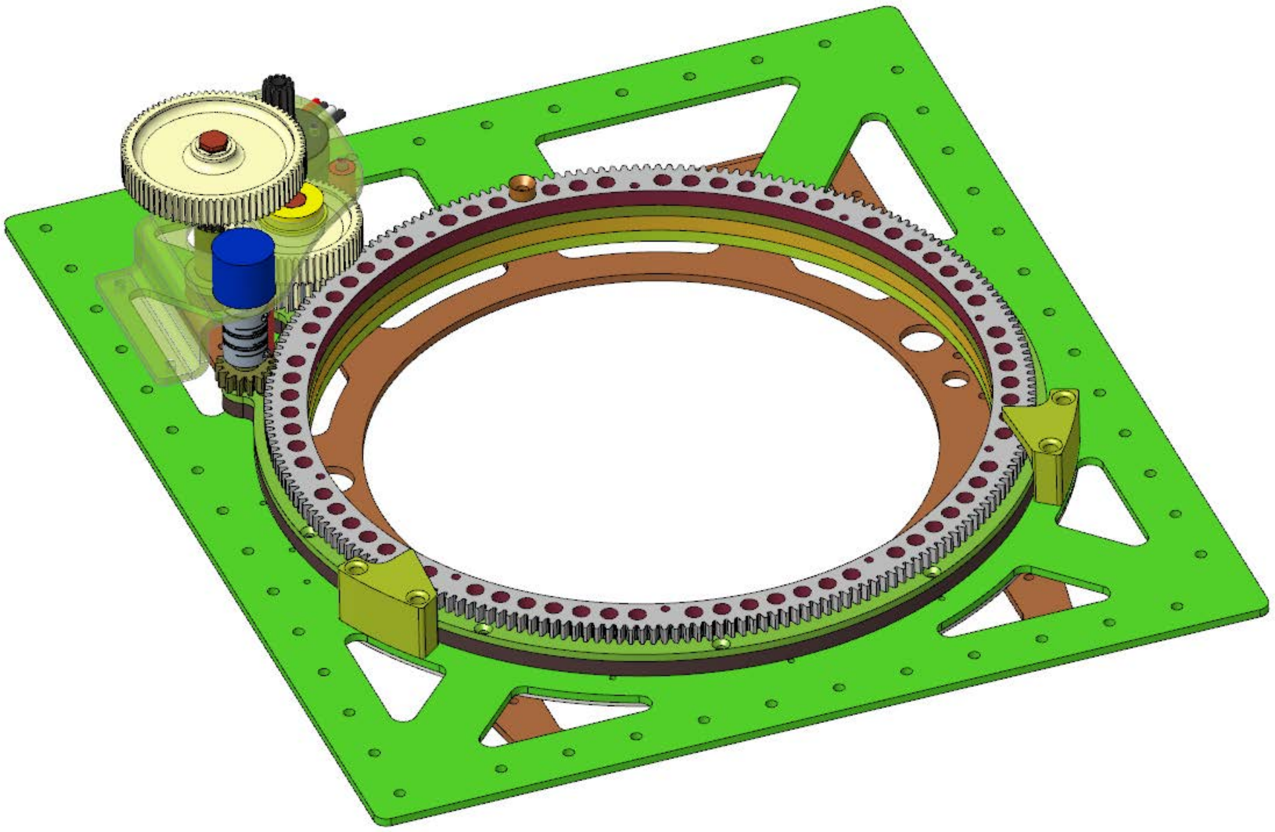
Belt Driven: belt driven turrets can be very simple if they rotate less than 360 degrees. If the turret rotates less than 360 degrees the belt can simply be attached to the turret drive plate using bolts or rivets. The point on the turret ring the belt attaches to will be a dead zone where the drive pulley will not be able to pass the bolted section. If the turret needs to rotate more than 360 degrees pulley teeth can be machined onto the turret ring or can even be 3D printed for low torque applications. The Armabot Turret240 is a belt driven turret.



Armabot Turret240 Belt Driven Turret.

Chain Driven: chain driven turrets are very similar to belt driven turrets. The turret sprocket can be machined on a CNC router or waterjet or a large COTS sprocket can be used. The WCP Greyt turret is a chain driven turret.

Gear Driven: the most complex turret drive method is a gear driven turret. The complexity comes from having to design a large gear and machine it. Gear driven turrets can be the most compact turret drive method and have been favored by Team 118 who has used them on their 2012, 2017, and 2020 robots.



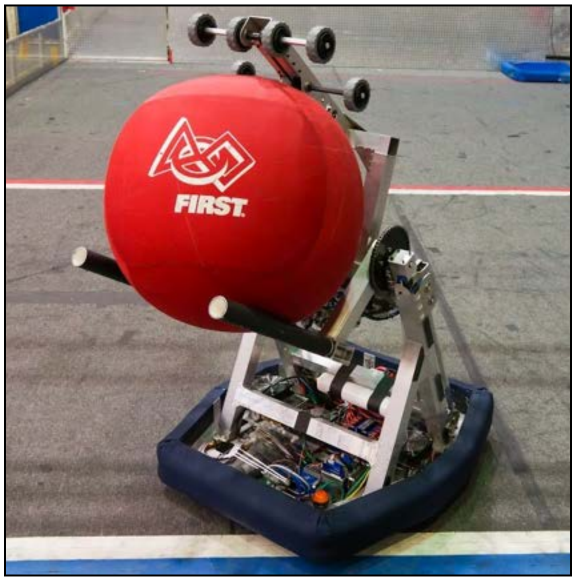
Gear Driven Turret CAD from Team 118’s 2020 Robot.

5.8 Bumpers

Bumpers are required in the FRC rules and prevent robots from damaging or being damaged by other robots and field elements. However, the mechanism design of bumpers from shape, height, and materials has strategic value. This section specifically contains bumper and frame design decisions that have an effect during matches and affect robot play. For a more general bumpers guide, including materials selection, bumper types, construction, and bumper maintenance, the "FRC Robot Bumpers Guide" by Kelsey Draus is an excellent resource.

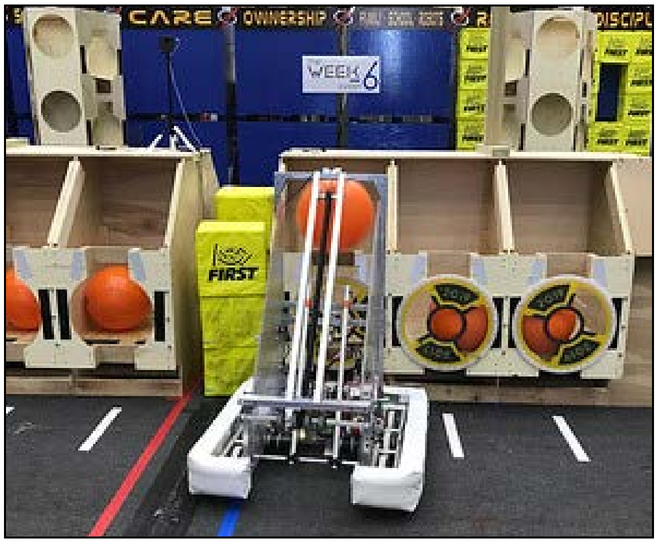
5.8.1 Bumper Shape

Hexagonal/Octagonal: Rectangular bumpers are the most used bumper shape because of their comparative ease of build; however, other design choices provide certain advantages. For example, a hexagonal or octagonal shape makes it much more difficult for other robots to use a T-bone pin on you. Because of the angled shape of the sides of the bumpers, the force applied by the opposing robot on those angles is never perpendicular to the direction of your wheels. This means that you can spin out of the T-bone pin when you previously could not with a traditional rectangular bumper design. Additionally, even if an opposing robot hits the small hitbox of your bumper that is parallel with your wheels, that bumper surface is smaller and drops off at a larger angle sooner, so it is easier to spin out of that pin. Rounded bumpers have a similar effect when it comes to T-bone pins, however, a lack of front and back flat sides also means that you lack the large surface area for a controlled directional push. The hexagonal/octagonal shape still allows for this.

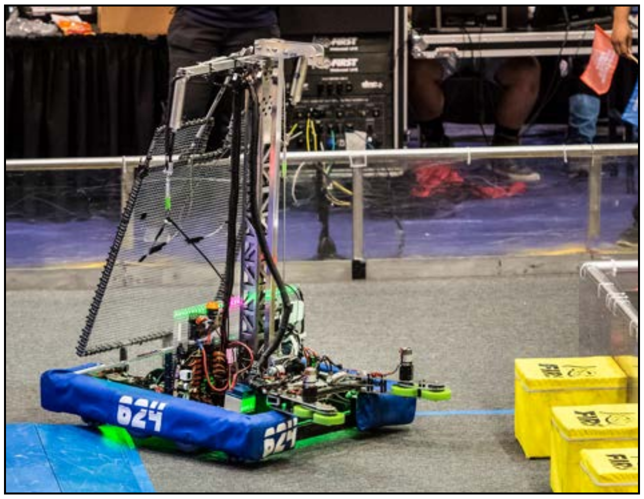


Team 971's hexagonal/octagonal bumpers in 2014.

Bumper/Frame Gap: Having a bumper or frame gap often allows you to have mechanisms that can more easily interact with game pieces. Having a bumper gap also has advantages and disadvantages in terms of interactions with other robots. For instance, a robot with a frame or bumper gap can be easily 'hooked' by another robot that also has a gap. Because of this hooking maneuver, having a frame or bumper gap can be potentially advantageous to a robot looking to play defense. Conversely, having a frame or bumper gap can be a disadvantage for robots who don't want to get hooked and pinned. Additionally, a robot looking to play defense with a bumper and frame gap can push the gap into a corner of an opponent robot and pin the trapped robot against walls and field elements. These maneuvers involve close contact between robots, often areas that are within frame perimeter, so be wary of potential damage both to your robot and another's robot.



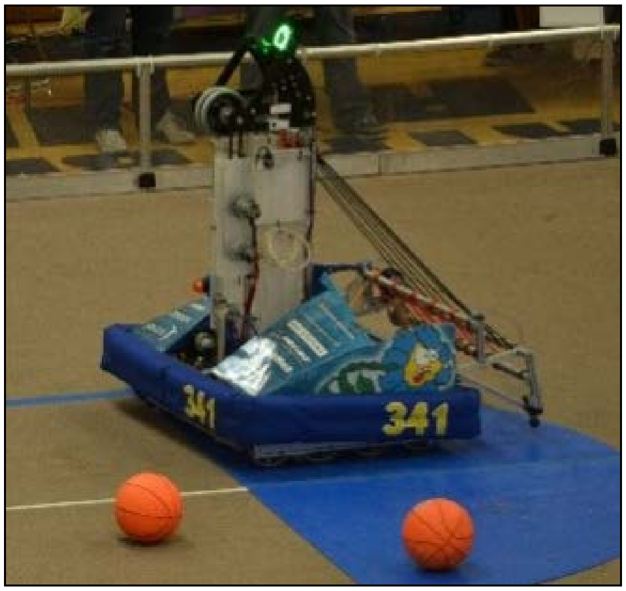
2019 Everybot's bumper and frame gap.



Team 624's 2018 robot with a bumper gap and no frame gap.

5.8.2 Bumper Height

Interaction with Game Pieces and Field Elements: The bumper zone is a range, so the specific bumper height is a design decision. You might want to have higher bumpers to, for instance, have the clearance to go over berms or other field obstacles. You may want to have lower bumpers to prevent game pieces from going under your bumpers. You could also have higher bumpers if you wanted to build an under-the-bumper game piece intake. In 2012, Team 341 angled their bumpers. On one side of their robot, the bumpers were low to the ground so that the balls would go into their intake and over their bumper, and not under their robot. On the other side of their robot, their bumper was raised so that they could easily drive over the center barrier.

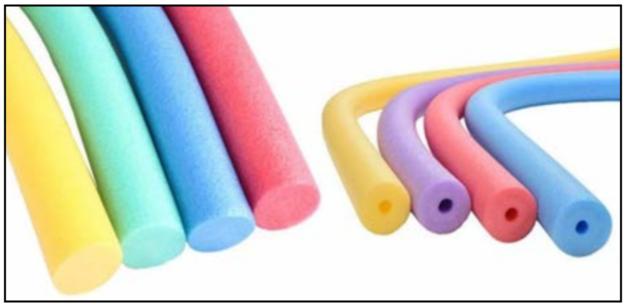


Team 341's angled bumpers in 2012.

Interaction with other robots: Having lower bumpers than another robot means you will be heavily favored in a pushing match. Suppose you are in a head-to-head pushing match with another robot and your bumpers are lower than theirs. As you push against them, your bumper wedges under theirs, applying an upward force on their bumper as well as the horizontal force that pushes the robot. That upward force applied to their bumper counteracts their weight force, pushing the front of their robot up. This decreases their traction and can even push far enough to where their front wheels are no longer on the ground. Now you have a huge advantage and can easily push against them. This also applies if you are pushing against the side of said opponent robot. Having lower bumpers than other robots also prevents other teams from doing this to you.

5.8.3 Pool Noodles

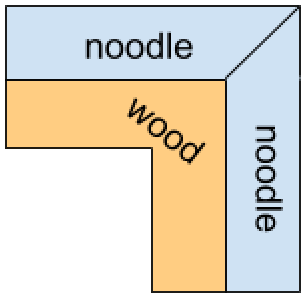
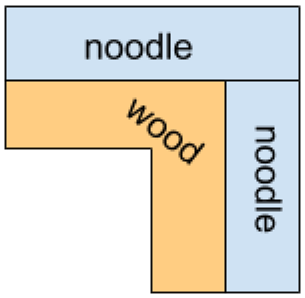
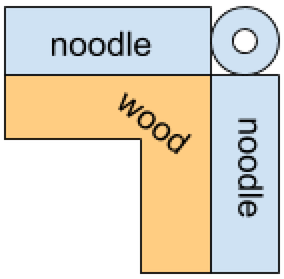
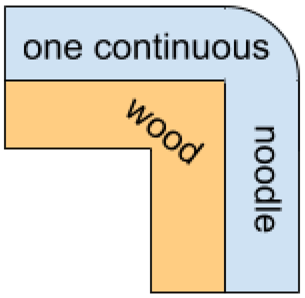
Pool noodle shape: The design decision here is mainly between a solid pool noodle or a hollow pool noodle.



solid vs hollow pool noodles.

Hollow pool noodles compress more. This means they can absorb more impact when you collide with another bumper. Because your bumpers compress more, that compressed pool noodle has friction because of the surface area increase. A solid pool noodle is the opposite. It can absorb less impact but there is less friction when pressing against the bumpers of another robot because the pool noodle doesn't compress as much. The larger the hole in the hollow noodle, the more it can compress. Alternative noodle styles may have different effects. For instance, a hollow flower shaped pool noodle may have greater friction due to both a more compressible structure as well as the surface area on the outside due to the ridges of the flower petals.

Pool noodle corner constructions: There are 4 primary methods to creating bumper corners with the pool noodles. The main difference in these corner constructions are how easily you can get around other robots and field elements and how well other robots can get around you. Usually, teams looking to play defense and prevent robots from scoring want to make it so other robots can’t get past them as easily, so a larger profile makes sense. Conversely, if you want to be able to get around field elements and other robots easily, a smaller profile may be better.

Bumper Corner Construction Method	Factors Driving Corner Construction Decisions
45 degree angle cut  **90 degree extension cut** 	These bumpers create a sharp corner. This makes it more difficult for you to get around other robots and field elements because the corner can get caught on things. This also makes it harder for other robots to get around you.
Vertical pool noodle 	These bumpers create a rounded corner. This makes it easier to get around other robots and field elements. This also makes it easier for robots to get around you.
Wrap around 	This is the bumper shape that has the smallest profile and is easiest to get around other robots and field elements, especially if a hollow pool noodle is used.

5.8.4 Fabric

Fabric: There are many different types of fabrics. Smoother fabrics like sailcloth or slick nylon have less friction. Fabrics like heavy duty nylon have more friction. The tighter the fabric is, the less friction there is when you interact with other robot bumpers. The looser the fabric, the more friction there is. If you want to be able to get around other robots and avoid defense, you would generally want less friction. If you want to impede other robots' motion, like in a T-bone pin, you would generally want to have more friction.

See the section titles, "Materials Selection- Fabric" in the "FRC Robot Bumpers Guide" referenced in Section 5.7 for suggestions of where to purchase bumper fabric.

6. Electronics

6.1 FRC Electronics System

Electronics System: the collection of electrical/electronic and pneumatic components that make the robot run. It consists of the following major components, plus a few additional.

roboRIO: the robot “brain”, or main computer, where most of the software magic happens.

Radio: the component that provides wifi communication between a robot and a driver station. Can be powered through POE or 5mm barrel jack.

Driver station: a laptop, switchboard, and hand controllers used to operate the robot.

Battery: the 12 volt, lead acid, rechargeable battery that powers the robot.

Main Breaker: turns on and off the robot, connects the battery to the power distribution panel. Has an Integrated circuit breaker rated at 120 Amps.

Power Distribution Panel (PDP): distributes voltage from the battery to the motor controllers and other onboard devices. It uses resettable circuit breakers for overcurrent protection. It can also provide voltage and current readings for each channel.

Motor controllers (Speed Controllers): controls the direction the motor spins and how much voltage it is given. More expensive motor controllers, such as the Talon SRX, can do more complicated tasks and are basically their own little computer.

Voltage Regulator Module (VRM): takes power from the PDP and provides regulated voltage to 12 volt and 5 volt devices, such as the radio, sensors, cameras, and custom circuits.

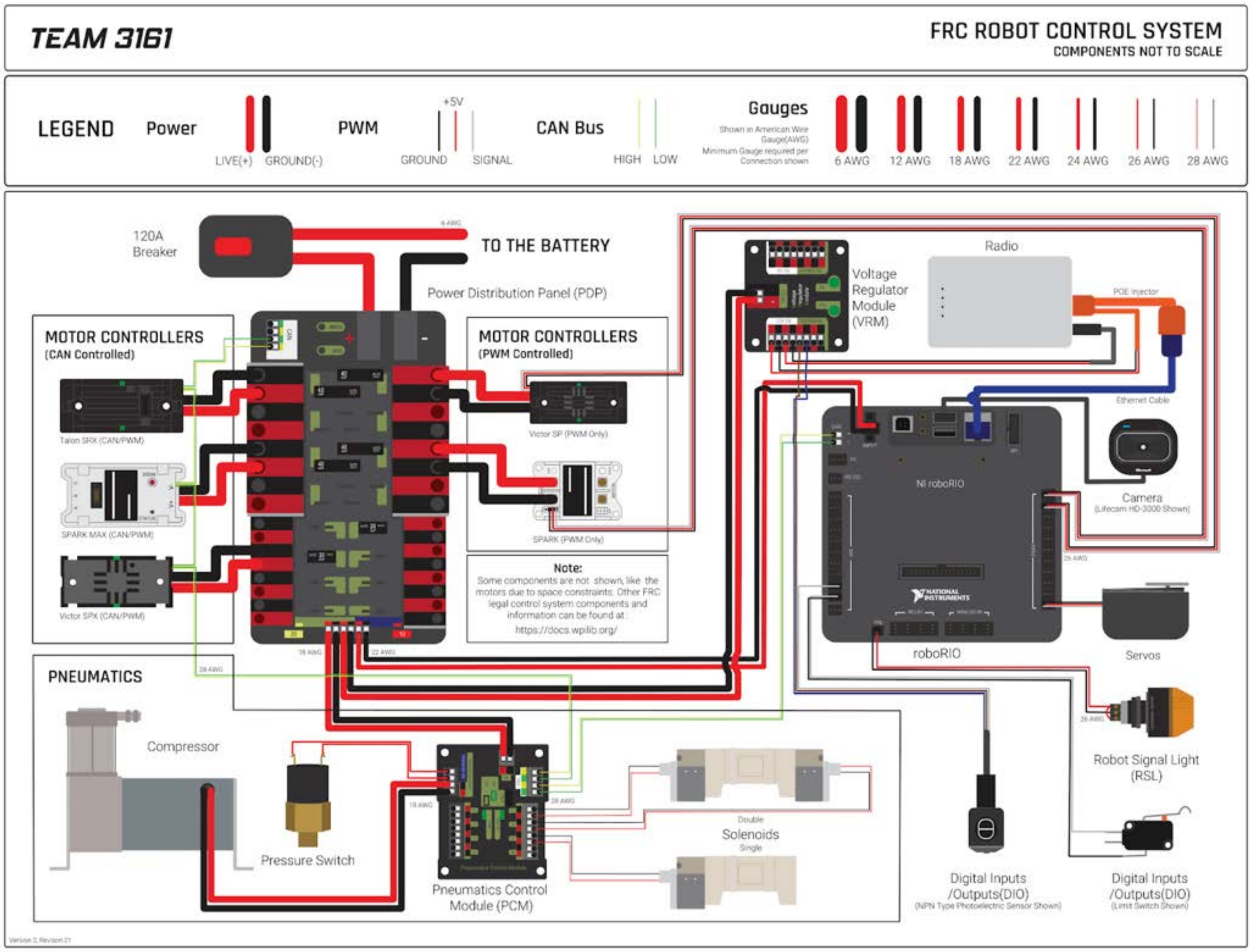
Pneumatic Control Module (PCM): controls the robot’s pneumatic compressor and provides on/off commands to pneumatic solenoids.

Solenoid: an electrically operated valve that delivers air to extend and retract pneumatic actuators.

Compressor: provides compressed air at 120 psi for the pneumatic system. High pressure is normally stored in air tanks or reservoirs.

Pneumatic Regulator: intakes high pressure air from the compressor or air storage and delivers low pressure, 60 psi air, to the pneumatic control components. This is considered a safe "working pressure" for FRC robots.

Robot Signal Light (RSL): is used to communicate the robot's state. If it is solid, the robot is disabled; if it is blinking, the robot is enabled.



TEAM 3161
FRC ROBOT CONTROL SYSTEM
COMPONENTS NOT TO SCALE
LEGEND
Power
LIVE(+)
GROUND(-)
PWM
+5V
GROUND
SIGNAL
CAN Bus
HIGH
LOW
Gauges
6 AWG
12 AWG
18 AWG
22 AWG
24 AWG
26 AWG
28 AWG
120A Breaker
TO THE BATTERY
Power Distribution Panel (PDP)
MOTOR CONTROLLERS (CAN Controlled)
Talon SRX (CAN/PWM)
SPARK MAX (CAN/PWM)
Victor SPX (CAN/PWM)
MOTOR CONTROLLERS (PWM Controlled)
Victor SP (PWM Only)
SPARK (PWM Only)
Note: Some components are not shown, like the motors due to space constraints. Other FRC legal control system components and information can be found at: https://docs.wpilib.org/
Voltage Regulator Module (VRM)
Radio
POE Injector
Ethernet Cable
NI roboRIO
roboRIO
Camera (Lifecam HD-3000 Shown)
Servos
Robot Signal Light (RSL)
PNEUMATICS
Compressor
Pressure Switch
Pneumatics Control Module (PCM)
Double
Solenoids
Single
Digital Inputs /Outputs(DIO) (NPN Type Photoelectric Sensor Shown)
Digital Inputs /Outputs(DIO) (Limit Switch Shown)

(Photo from FRC 3161 and Stefen Acceptance)

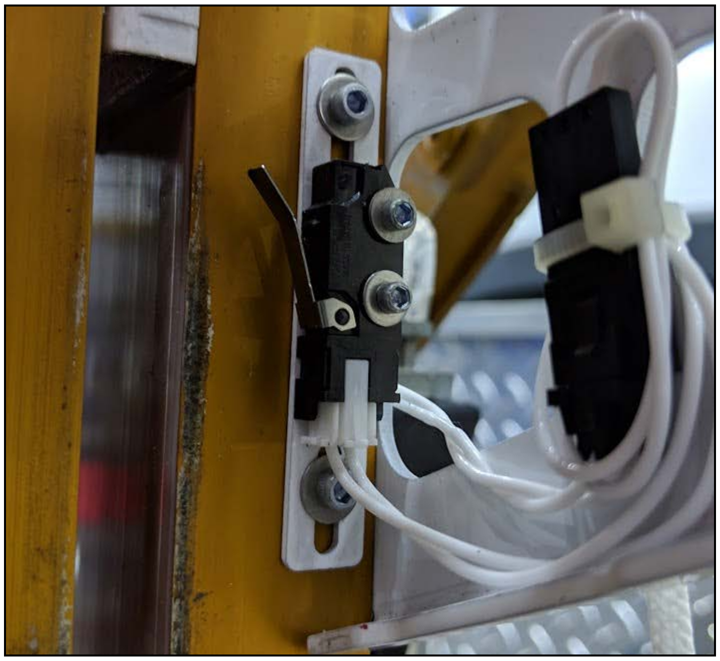
6.2 Sensors

Sensors are used to detect events or changes in an environment. Sensors allow your software to know where your mechanism is, what it is doing, and how it is performing. Choosing the right sensor can make writing software and controlling a mechanism much simpler. Sensors commonly used in FRC are listed below.

Digital sensors output discrete values, and are connected to the digital ports on the roboRIO. One type of digital sensor has outputs of 0 and 1, AKA binary. These usually indicate if a condition is met (i.e. when a limit switch is pressed, the output is 1; when it is not pressed, the output is 0). These types of sensors are usually simpler to write software for, but are not as flexible. Other digital sensors are used to count events (i.e. digital encoders), or otherwise provide discrete values to the roboRIO. Digital sensors might connect to the DIO, I2C, or SPI ports of a roboRIO. Often these types of sensors present analog values to the software.

Analog sensors output a range of values, and are connected to the analog ports of the roboRIO. For example, potentiometers allow you to know the specific angle of a mechanism as it is adjusted through different positions. Analog inputs to the roboRIO are always converted to a number through an analog-to-digital converter.

One design tip when integrating sensors in a mechanism is to never let a sensor, such as a limit switch, be the hard stop for a mechanism. It is often advantageous to have some amount of adjustability with your sensor mounts, allowing you to tune your mechanism to where it trips the limit switch before it hits a hard stop, but hits the hard stop before hitting the body of the limit switch.


Adjustable limit switch mount on Team 118's 2018 robot.

Rotary Encoder: used to measure rotations and can rotate continuously. Rotary encoders can be either digital or analog sensors, but the values presented to the software are analog values.

Magnetic (Mag) Encoder: a type of rotary encoder that measures rotations of a shaft by sensing the magnetic field of a diametrically polarized magnet that is on the shaft. The CTRE Mag Encoder is a digital sensor.

Potentiometer (Pot): an analog sensor used to measure limited rotations, such as on an arm or turret. Most cannot rotate continuously.

Absolute Position Sensor: an analog sensor used to measure the absolute position of a mechanism. They are usually used with swerve drive steering gearboxes to know what angle the module is at. It can be thought of as a continuous rotation pot.

String Potentiometer (String Pot): an analog potentiometer used to measure a linear distance.

Limit Switch: a digital sensor used to detect when something has reached its limit. It is most commonly used to stop commanding the motor when a mechanism reaches its limit.

Hall Effect Sensor: uses magnets to detect when something is near the sensor. It doesn't require the mechanism to contact the sensor. It can be used as a tachometer on a flywheel. A hall effect sensor can be an analog or a digital sensor however those commonly used in FRC are digital.

Hall Switch (AKA Reed Switch): essentially a non-contact limit switch. It detects when a magnet gets within a certain proximity of the sensor. This is a digital version of a hall effect sensor.

Ultrasonic Sensor: an analog sensor used to measure distance using ultrasonic sound waves.

Accelerometer: an analog sensor used to measure acceleration.

Gyroscope: an analog sensor used to measure the change in rotational angle per unit of time, usually for the entire robot. The largest downside is the reading will drift throughout an FRC match.

Magnetometer (Digital Compass): an analog sensor that works just like a compass. It can be used to determine the robot's orientation relative to magnetic north. Magnetic fields such as those in motors can interfere with the sensor, as well as high current wiring, such as battery or motor leads.

Inertial Measurement Unit (IMU): a sensor that combines the accelerometer, gyroscope, and sometimes magnetometers. It can be used to know where your robot is on the field. IMUs can also suffer from drift over the course of a match, or be affected by magnetic fields on a robot such as from motors.

Lidar: a scanning sensor used for 3D mapping. Lidar fires a laser or light at a target and measures the reflected pulse with an integrated sensor. It is relatively new to FRC. One example can be found here. Non sweeping Lidar sensors such as this one can be used as a distance measuring sensor like an ultrasonic sensor would be used.

Light Sensor: can be used to detect when an object is in front of the sensor. Many light sensors have an analog adjustment dial on them, but the signal sent to the roboRIO is digital.

Beam Break Sensor: a digital sensor used to detect when an object crosses a beam. They can be used as non-contact limit switches. They can also be used as a tachometer to measure wheel rotations.

6.3 Cameras

Cameras are great to use as a sensor, such as in computer vision tracking or a game object identification system. They are also useful for providing video feedback to the driver's station.

Computer Vision: in FRC, computer vision usually refers to using a camera to track something such as a game object or a scoring target. Many teams write their own vision tracking software, such as Teams 118 and 254, however there are many COTS solutions to vision tracking out there that are relatively easy to implement. Some of these COTS solutions include the LimeLight camera, the PixyCam, Gloworm, and the JeVois Smart Machine Vision Camera.

The LimeLight camera is a "plug and play" camera developed by FRC Team 987 to be used for computer vision in FRC. It can also be used to stream video back to the driver's station.

The PixyCam is a computer vision camera designed to interface with a Raspberry Pi or an Arduino. PixyCam is not as "plug and play" as the LimeLight, but it has a large library of open source software, giving teams a starting point for developing their own computer vision system.

The Gloworm is an open-source hardware solution built around allowing FRC teams to develop their custom computer vision software.

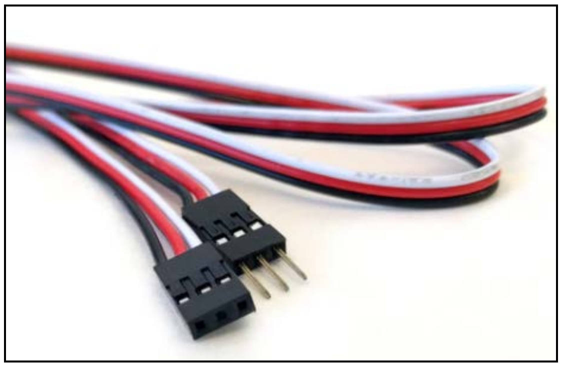
The JeVois Smart Machine Vision Camera is a tiny self-contained computer vision package with a video sensor, quad-core CPU, USB port, and serial port. The JeVois Smart Machine Camera is less commonly used than the above computer vision systems, however it is still a powerful system.

Video Feedback: almost any USB webcam can be used to provide video feedback to the driver's station. Many teams use small external monitors on which to display this video feed. USB webcams can be plugged into the roboRIO and set up to stream to the driver's station with relative ease. This can be especially helpful in years where there are large field obstacles. Typically, the resolution and the framerate needs to be decreased to fit within the FRC bandwidth limitations. Switching to black and white video can also help stay below these limitations. Team 118 has used this 120 degree wide angle camera in 2017 and 2018 for driver feedback.

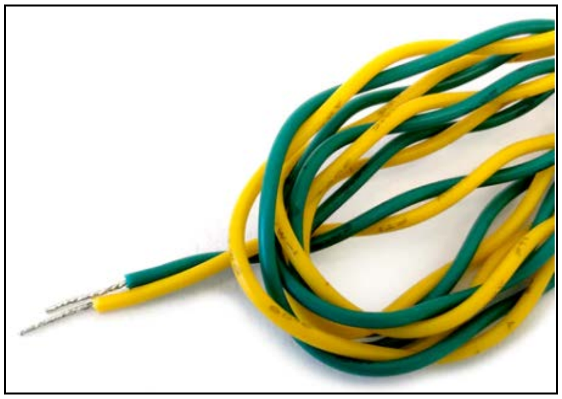
6.4 Communication Method

There are two main controls system communication methods used in FRC, PWM and CAN.

PWM (Pulse Width Modulation): a communication method using electrical signals sent in pulses. FRC PWM is based on hobby RC (remote control) motor controllers. The PWM signal is sent on a repeating cycle of 20 milliseconds (ms) with pulse widths ranging from 1ms to 2ms. With each cycle, a 5 volt pulse of varying length is sent to a control device. A 1.5ms pulse equates to zero output of a motor controller. Anything greater than 1.5ms causes the motor controller to output a variable positive voltage to the motor, with 2ms being 100% forward. Anything less than 1.5ms causes the motor controller to output a variable negative voltage to the motor, with 1ms being 100% reverse. A PWM signal cable is one of two ways to connect a motor controller to the roboRIO.


PWM Cable.

CAN (Control Area Network): a communications network originally developed for the automotive industry to simplify wiring harnesses; this is the same effect it has on FRC robots. Instead of requiring a separate signal cable to run from every motor controller to the roboRIO, the motor controllers are "daisy chained" together. Generally, the chain originates at the roboRIO and ends at the Power Distribution Panel. One downside is that a break in the chain will cause any device after the break to fail. The Pneumatic Control Module is also a CAN device. CAN wiring requires a 120 ohm terminating resistor at each end of the wiring chain. The PDP and roboRIO contain an internal terminating resistor, so the resistor is only needed if it ends at a device other than the PDP. The PDP contains a jumper for selecting or deselecting the terminating resistor.


CAN Wire.

6.5 Motor Controllers

Motor controllers are used to control how much voltage is sent to a motor, and controls which direction the motor rotates. Some motor controllers allow sensors to be plugged into them and can run their own control loops.

Motor Controllers Allowed for the 2020 Season

"Modern" Controllers (Currently in production)

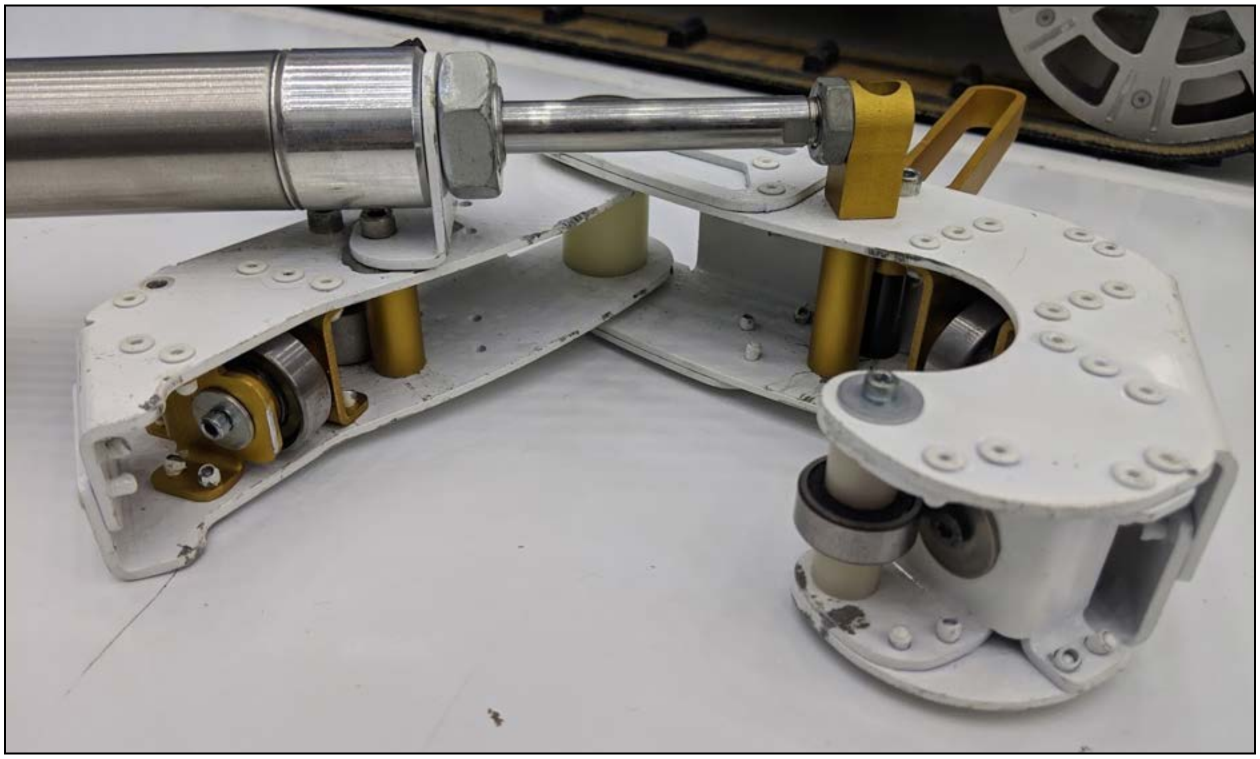
Name	Communication Method	Notes	Photo
Talon SRX	CAN/PWM	Can run internal control loops when sensor is plugged in.	
Victor SPX	CAN/PWM	Similar to Talon SRX, but does not have internal control features or allow sensor input. It is lighter, smaller, and cheaper than a Talon.	
Spark	PWM	One of the cheapest FRC motor controllers.	
Spark MAX	CAN/PWM/USB	Only motor controller for the REV NEO brushless motor.	
Nidec Controller	PWM	Motor controller is integrated into the Nidec brushless motor.	
TalonFX	CAN/PWM	Motor controller is integrated with the Falcon 500 motor	

“Legacy” Motor Controllers (Out of production/Uncommon)

Name	Communication Method	Notes	Photo
Talon SR	PWM	No longer in production, but still a useful motor controller.	
Victor SP	PWM	No longer in production.	
Victor 888/884	PWM	Subject to the failure of the cooling fans.	
Jaguar	PWM/Formerly CAN	Very large and expensive.	
DMC 60	CAN/PWM	Very Similar to the Talon SRX.	
SD540	CAN/PWM depending on the model	Rarely used by any teams.	

6.6 Pneumatics

A **pneumatic system** uses stored, compressed air to power cylinders, air motors, or other pneumatic devices. Pneumatic cylinders are usually what is referred to when talking about pneumatics in FRC. Air is stored in accumulators (air tanks) and is sent to cylinders by solenoids (air valves).


Pneumatic claw from Team 118's 2013 climbing mechanism.

6.6.1 Pneumatic Components and Terms

Accumulators (Air Tanks): plastic or metal tanks that store pressurized air. Tanks must be rated for at least 125 psi. Multiple tanks can be used together to increase air capacity.

Pneumatic Control Module (PCM): controls the robot's pneumatic compressor and provides on/off commands to pneumatic solenoids.

Solenoid: an electrically operated valve that receives a command from the PCM and delivers air to extend and retract pneumatic cylinders. Solenoids can be single-acting or double-acting. A single-acting solenoid will always return to the same position when the robot is disabled. A double-acting solenoid will remain in the last commanded position when the robot is disabled. Double-acting should be used when a valve needs to maintain its last position when a robot is disabled, or maintain either position for starting configuration.

Compressor: provides compressed air at 120 psi for the pneumatic system. In past seasons, a compressor could be either onboard the robot or offboard.

Pneumatic Regulator: intakes high pressure air from the compressor, or air storage, and delivers a lower pressure to the pneumatic control components. This is considered a safe “working pressure” for FRC robots.

Working Pressure: the pressure the actuators use, which is lower than the pressure in the accumulators. Historically, the maximum working pressure has been 60 psi, however refer to the current year's FRC rules for the official maximum legal working pressure.

Pressure Relief Valve: a spring-operated valve that releases all air from the pneumatic system. These valves are usually pre-set to 125 psi, the maximum system pressure allowed by FRC.

Pneumatic Tubing: usually a polyurethane tubing that connects the components of the pneumatic system that use air. Tubing must be certified for 125 psi.

Pneumatic Cylinder: linear actuator that extends and retracts using pressurized air.

Single-Acting Pneumatic Cylinder: a cylinder that is extended with air, but retracted with a spring (or vice versa). These are more compact, and can fit into smaller places, as they only require one pneumatic hose. A single-acting cylinder can be controlled by either a single-acting or double-acting solenoid.

Double-Acting Pneumatic Cylinder: a cylinder that is extended and retracted using air. These require two pneumatic hoses run to the cylinder. A double-acting cylinder can be controlled by either a single-acting or double-acting solenoid.

Cylinder Bore: the internal diameter of the cylinder.

Cylinder Stroke: the travel length of the cylinder rod.

Cylinder Rod: the threaded rod in the cylinder that extends and retracts from the cylinder body.

Rod End: a threaded joint that sometimes includes a ball joint that allows you to attach the end of the cylinder rod to a mechanism. McMaster-Carr has a large assortment of rod ends with various features here.

Flow Regulator: an air flow restrictor that can be put inline with cylinder air tubing and will control how fast air enters or exits a pneumatic cylinder. These can be very useful for tuning cylinders used on intakes.

Fittings: can be a tubing-to-device connector, which screws into a device and receives tubing via a push-in connector. There are also tubing-to-tubing connectors in the form of straight couplers, tee fittings, right angle fittings, and wye or splitter connectors. Tubing is simply inserted into the fitting. Removal requires holding down a release ring, then pulling the tube out.

6.6.2 Pneumatic System Tips

Pneumatics are great for mechanisms requiring just two positions, such as drivetrain shifters, deploying and stowing intakes, two position shooter hoods, and much more. Pneumatics are very useful when you do not have enough ports on the Power Distribution Panel and you want to add another mechanism. Pneumatic systems have a large up front weight cost, due to the compressor. When designed and operated properly, pneumatics actuate very fast and have compliance in the axis parallel to the rod. Perpendicular forces will usually bend the rod without external bearing support.

Keep in mind that if your robot has an onboard compressor, it will drain your robot’s battery, and can affect robot performance. The compressor can also be damaged from rough games, such as the 2016 game, and can be damaged from overheating. The 2016 game was so rough that Team 118 had to replace their compressor 4 times throughout the season. If your robot sits for a long time before you play a match, check the pressure of your pneumatic systems. Small leaks can slowly drain your stored air. Tether to your robot when in queue to refill your pneumatic system.

To find leaks in the pneumatic system, soapy water in a squirt bottle can be sprayed onto suspect fittings or tubing. Bubbles will form at the location of the air leak. The root causes of leaks are often incorrect application of teflon tape to threads, tubes that are not cut squarely, or tubes that are not securely seated in the PTC connectors.

A great troubleshooting aid is to use one color of tubing for the extend line, a different color for the retract line, and a third color for the stored air lines (from the compressor to the solenoid manifold or solenoid splitters).

Spectrum has a document on how to increase the flow for mechanisms such as catapults and punchers.

6.6.3 Pneumatic Force Calculations

The equation for the force of a pneumatic cylinder is
$Force = Pressure * Cross\ Section\ Area\ of\ Cylinder$
For example, a 1 inch diameter (which is slightly large for most FRC robot mechanisms) cylinder will extend with a force of about 94 pounds. (Area is 1.57 square inches X 60 psi working pressure.)

Note that the force exerted by the cylinder will be a little greater when the cylinder is extending than when the cylinder is retracting because the cylinder rod reduces the cross sectional area. You must consider how much air your cylinder is using. A larger bore cylinder provides more force but also uses much more air.

6.7 Electrical Connectors

Connectors are an integral part of FRC electronics. Having one fail could possibly result in losing a match. It is important to utilize connectors with a latching mechanism or a method of adding a device to prevent the connector from failing or disconnecting. Connectors are sized based on the voltage and current passing through them. The 3 major classes of connectors on the robot are categorized by their current carrying capacity.

6.7.1 High Power Connections

This is the most critical connection on the robot, because a failure here means a dead robot on the field. FRC has made this selection simple by only allowing two legal connectors. Each robot must have a set of these connectors attached to the robot and battery. These connectors must carry the entire power load of the robot and therefore are the largest.

Legal APP (Anderson Power Pole) Connectors

Name	Photo
SB-50	(Can be either red or pink)
SB-120	

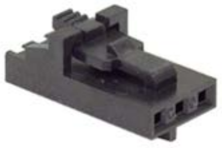
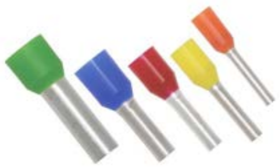
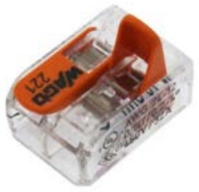
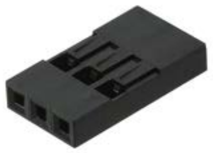
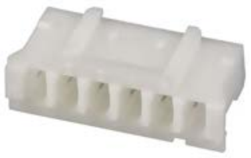
6.7.2 Medium Power Connections

The next tier is power connections used for motors and non sensor applications. Typically these connectors are seen on power connections on the PDP. We recommend standardizing with a single connector family.

Name	Type	Description	Photo
Anderson Power Poles	Crimp	Wide range of colors that can be mated together to form different combinations of pin outs. Different sized crimps can be used to match a wide range of wire gauges.	
XT - 60	Solder	These connectors are more common in RC hobby products and are typically cheaper and easier to source.	

6.7.3 Low Power Connections

These connectors are typically used on signal or low power applications. There are usually more individual wires with smaller gauges. This power category is still important because it covers connections to the robot radio. Failure here means disconnection from the FMS (Field Management System).

Name	Type	Description	Photo
Molex SL	Crimp	Offers polarity protection and has a locking feature. Housings have specific genders.	
Ferrules	Crimp	Prevents the individual strands from fraying apart. Really useful to use on the Weidmuller connectors.	
WAGO Lever Nuts	Clamping	Great for prototyping. No soldering or crimp needed.	
Dupont	Crimp	The most generic connector, used a lot in RC hobby products. This connector offers no reverse polarity protection or latching properties.	
JST	Crimp	Smallest connector currently used in FRC. Not recommended for applications that require frequent connection and disconnection cycles.	

6.8 Insulation

While providing a good conductive path is important, it is equally important to prevent unwanted conductive paths. This is done with different types of insulation. Heat shrink is the general “go to” for insulating cables. Electrical tape is only used for emergency situations because it is faster to apply.

Name	Description	Photo
Heat shrink	Usually a plastic tube that, when heated, shrinks around the exposed connection for insulation.	
Electrical tape	Should only be used in emergencies. Can leave a sticky residue and is not abrasion resistant.	
Kapton/Polyimide tape	A more expensive option of electrical tape. Does not leave a residue and is self-adherent.	

6.9 Cable Management

Cable runs should be taken into consideration during mechanical design. Cable runs should be routed close to pivot points. In addition, passive cable management, such as snake skin and energy chains, should be utilized on mechanisms that could be linear, such as elevators. These methods and tools should also be applied to pneumatic tubes.

Name	Description	Photo
Zip ties	Quick and easy method to bundle wires and secure them in place.	
Zip tie mounts	Provides a mounting feature for objects to be secured to. Can be used with double stick tape on the bottom.	

Name	Description	Photo
Snakeskin/cable sheath/cable sleeve	Wraps around a bundle to provide abrasion resistance, organization, as well as prevent snagging.	
Energy chain	A more rigid version of snake skin. Used when the cable travel path is defined and consistent.	
Nylon lace	Used in tight spaces where zip ties aren't optimal. Can tie wire bundles into non-circular shapes.	

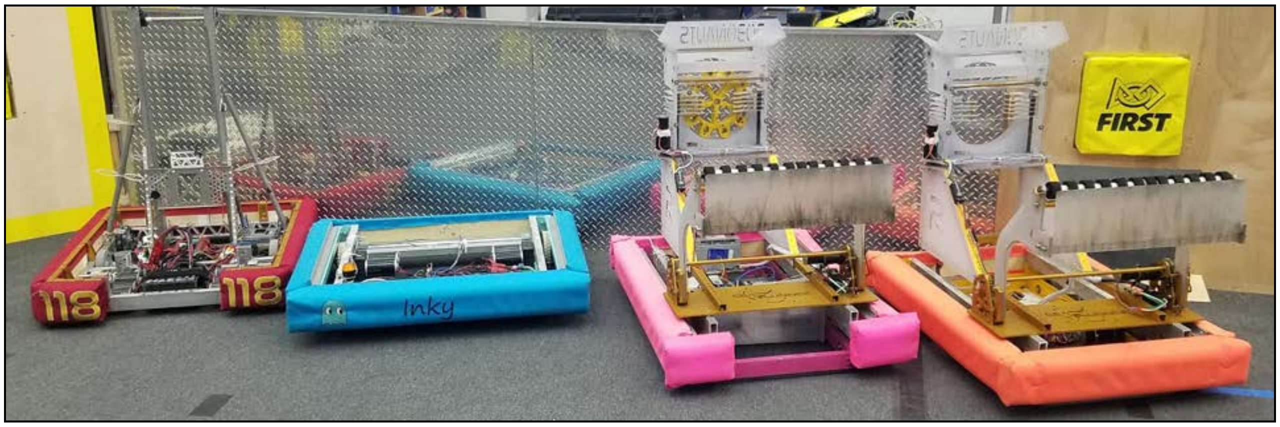
7. Prototyping

7.1 Prototyping Resources

Prototyping is immensely helpful when it comes to designing your competition robot. It allows you to flesh out ideas and prove their viability before building your competition robot. NASA's Systems Engineering Handbook describes prototyping as a way to "wring out the design solution so that experience gained from the prototype can be fed back into design changes that will improve the manufacture, integration, and maintainability of a single flight item." This is directly applicable to FIRST robots, where what you learn from prototyping will make building your competition robot much easier and will help make the robot much more effective.

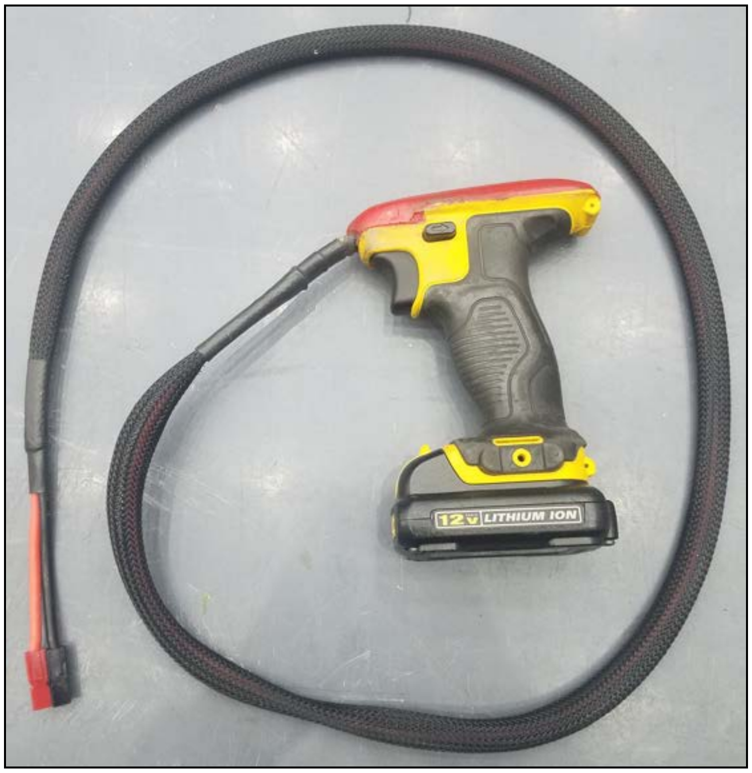
Prototyping Resources

PACBots: PACBots are practice and prototyping chassis used by Team 118, each named after a PAC-MAN ghost. They use these drivetrains immensely throughout the build and competition season to test prototype mechanisms, write software, and train drivers. We recommend having a set of drivetrains for prototyping, these don't need to be a new set of robots, you can use past robot drivetrains, old practice robots, or kit chassis. Just having some way of driving around prototypes is very valuable for building the best robot you can. To control the PACBots, Team 118 uses PACBot Software, a modular code base that can easily be modified in the configuration file. Team 118 openly shares this software with teams, and it is available on their website, www.robonauts.org/ under the "Robots" tab and can be found under the "Resources" header for the most recent year's featured robot. From there, the code can be downloaded, and includes a PACBot Software user's guide, which instructs teams on how to get PACBot Software onto their robots.



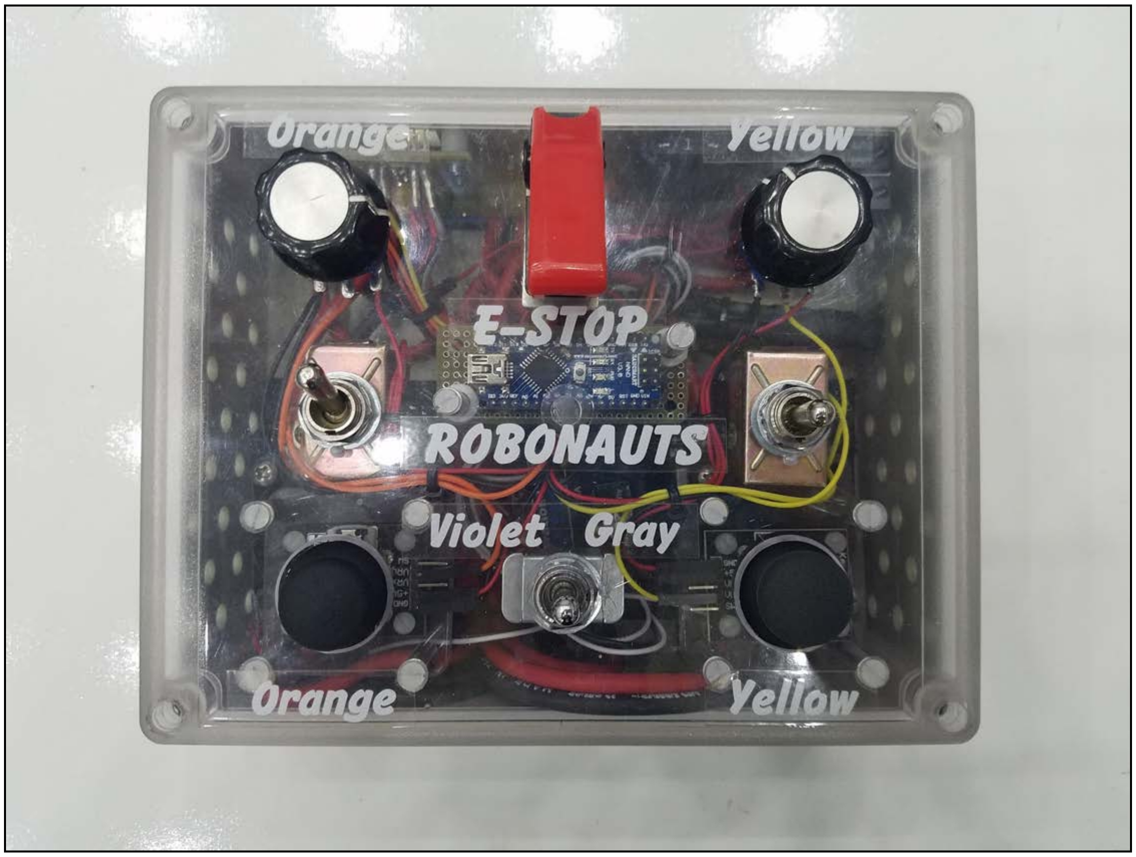
Team 118's PACBots with their driver training system mounted to two of them.

Power Gun: a battery-powered hand drill which has had the motor, gearbox, and chuck removed, and has quick disconnect power connectors, such as Andersons, with which you can power an FRC motor, without the use of a motor controller or any software. The appeal of a power gun compared to plugging an FRC battery directly into a motor is the variable speed you can get with the trigger of a power gun and the small form factor.



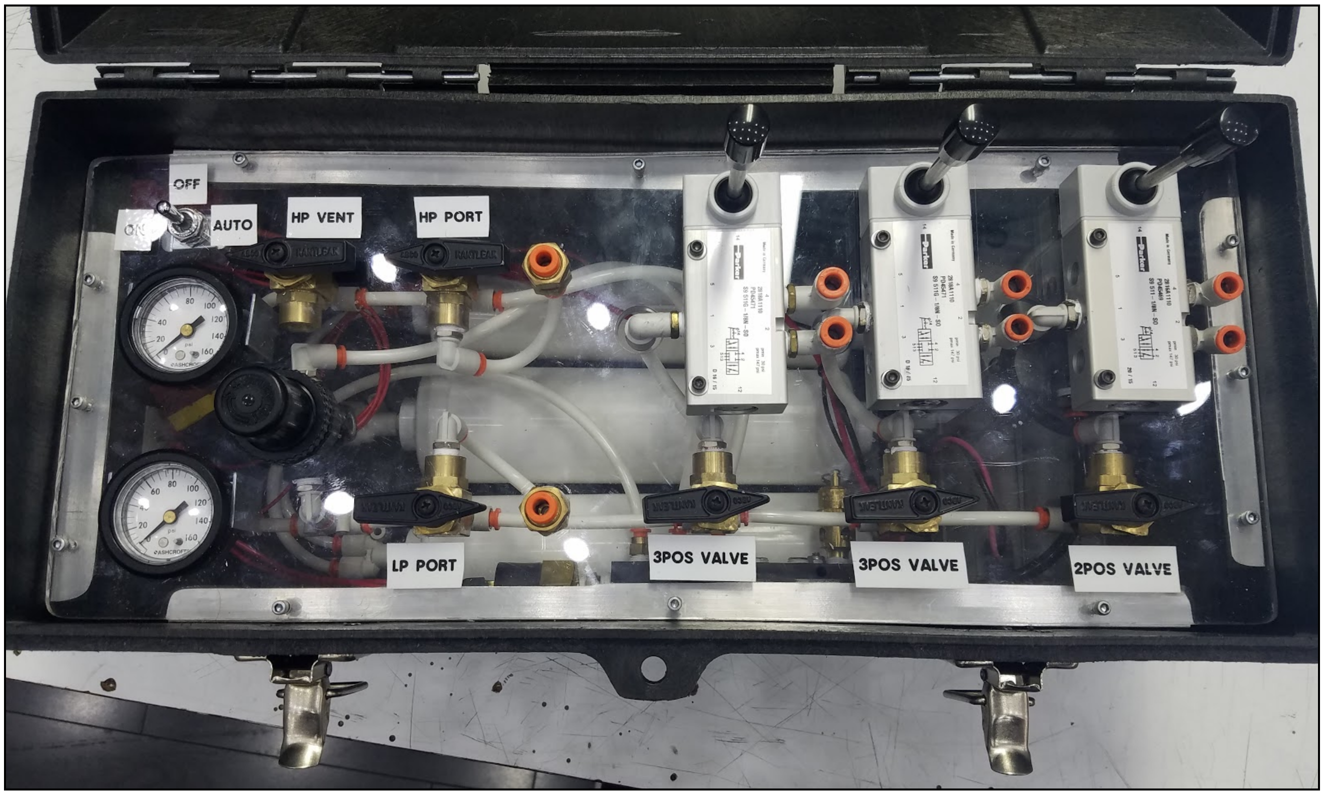
A 12 volt DeWalt drill modified to be a power gun.

Prototype Controller: many teams struggle with quickly testing their prototypes because of the complexity of the FRC control system. One way to solve this problem is to build a prototype controller. Team 118 has built several handheld prototype controllers which allow the team to test prototypes quickly. These controllers have analog sticks to drive mechanisms, along with potentiometers which allow a mechanism to be run at a constant speed. The Robonauts, Prototyping Controller uses an Arduino, along with the old Talon motor controllers, however we encourage you to develop your own prototyping controls system. Team 148, The Robowranglers often use VEX EDR Cortexes for prototypes and demo robots. R/C controllers can also be used to drive PWM motor controllers. A guide on how to build a Robonauts Prototyping Controller can be found here.



A Robonauts' Prototyping Controller.

Pneumatics Test Box: allows you to quickly integrate pneumatics into a prototype and test them without an FRC controls system. Team 118 has developed a pneumatics test box with integrated solenoids, accumulators, and a compressor that only needs an FRC battery plugged into it to charge the system. A link to the schematic for Team 118's Pneumatics Test Box can be found here, and the bill of materials can be found here.



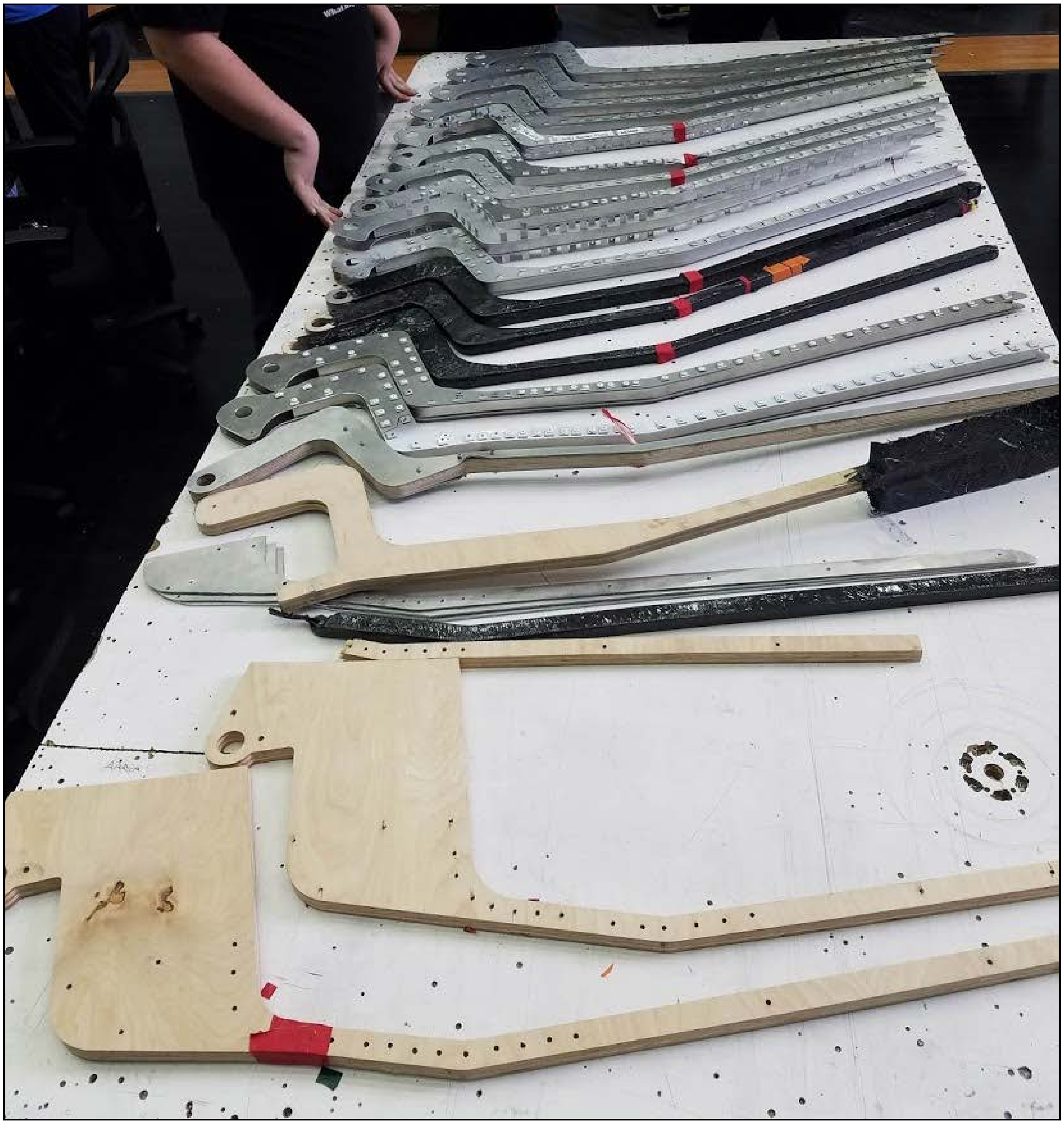
Team 118's Pneumatics Test Box.

7.2 Prototyping Examples

The team build blogs listed below are great resources for examples of prototypes of different mechanisms from different games.

Team 118's Build blogs:
2014
2015
2016
2017
2018
2019

Team 254's Build Blogs:
2011
2013
2014
2017
2018

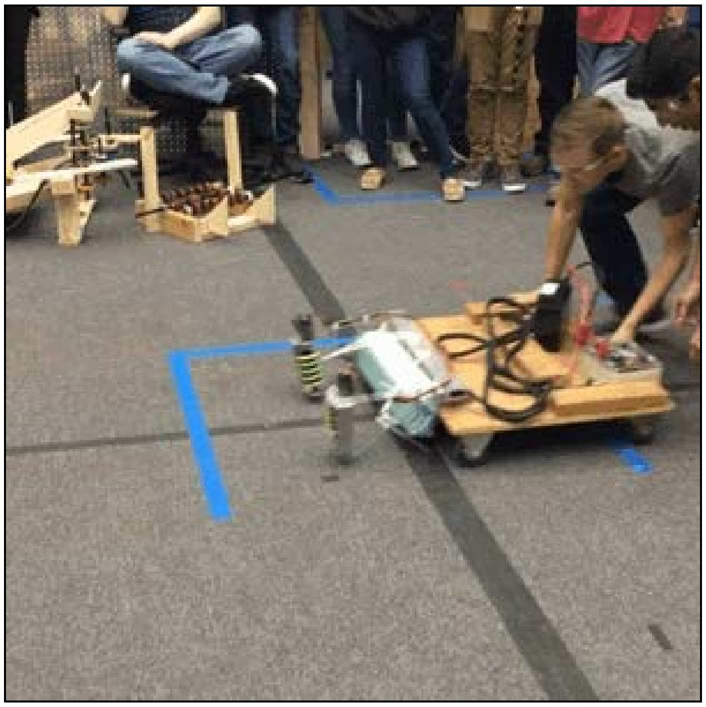

Collection of Prototype and Iterations of Climbing Tines from Team 118's 2018 Robot.

Below are images of Team 118's intake prototyping efforts from the 2018 season. In the beginning many different prototypes are built to validate ideas. These prototypes are simple, often use wood with hand drills as motors, and usually can be used to determine if an idea is viable. Then some of these prototypes are made higher fidelity, using appropriate motors and materials. Eventually a concept is settled on for the competition robot and a prototype is built so that the mechanism can be tuned. For example, Team 118 used their high fidelity intake prototype to determine which intake wheels worked the best and what angles the intake needed to reach to hand the cube off to the arm, as well as get cubes from the portal. These high fidelity prototypes can also be used for driver training by mounting them to a robot chassis.

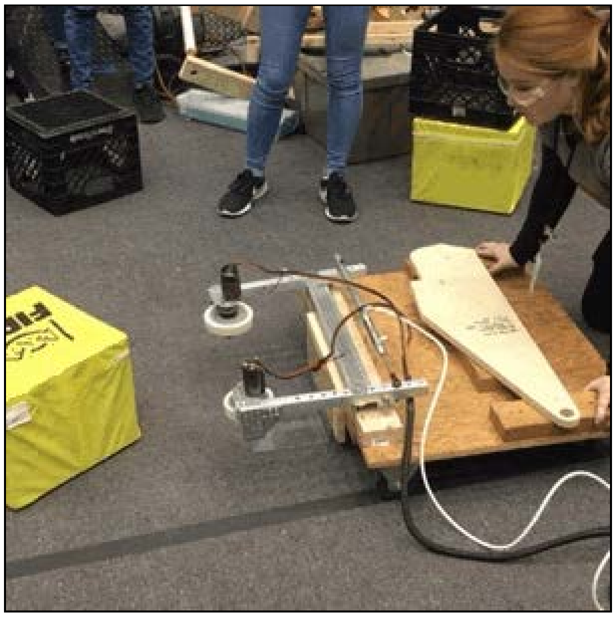
2018 intake prototype from day 3 of build season. This prototype was built very quickly using just battery-operated power drills and spare intake rollers from Team 118's 2017 robot.

Another 2018 intake prototype built on day 3 of the season.

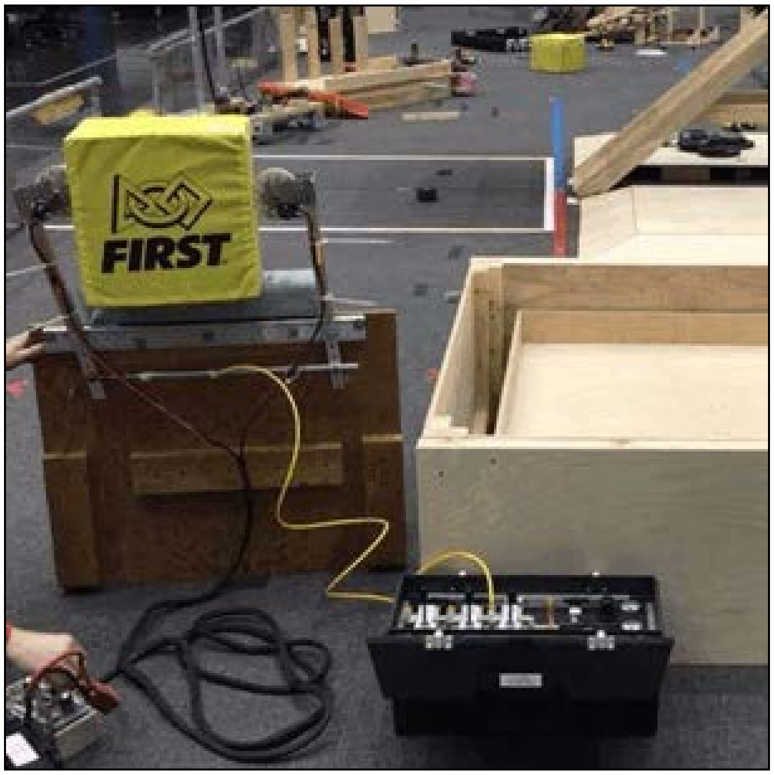
Intake prototype built on day 5 of build season.



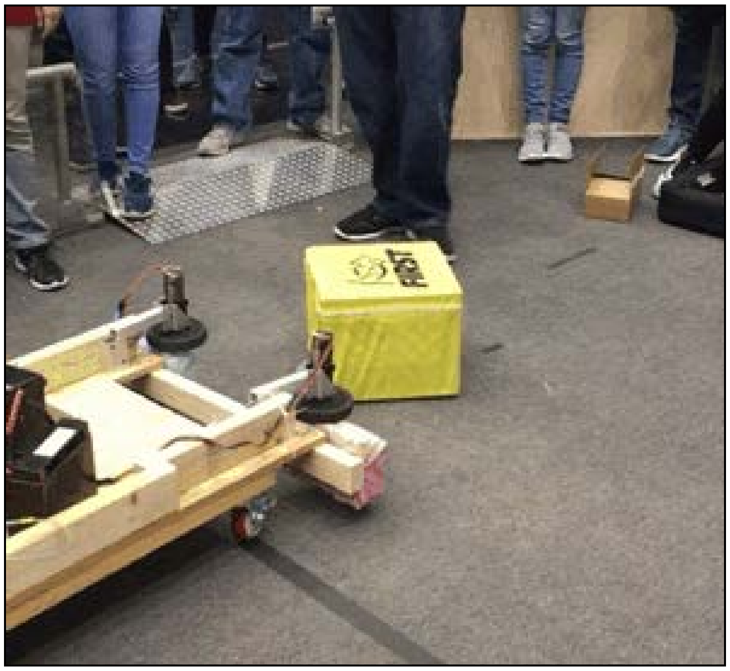
2018 intake prototype from day 10 of the build season. This uses the Robonauts Prototype Controller to control the motors.



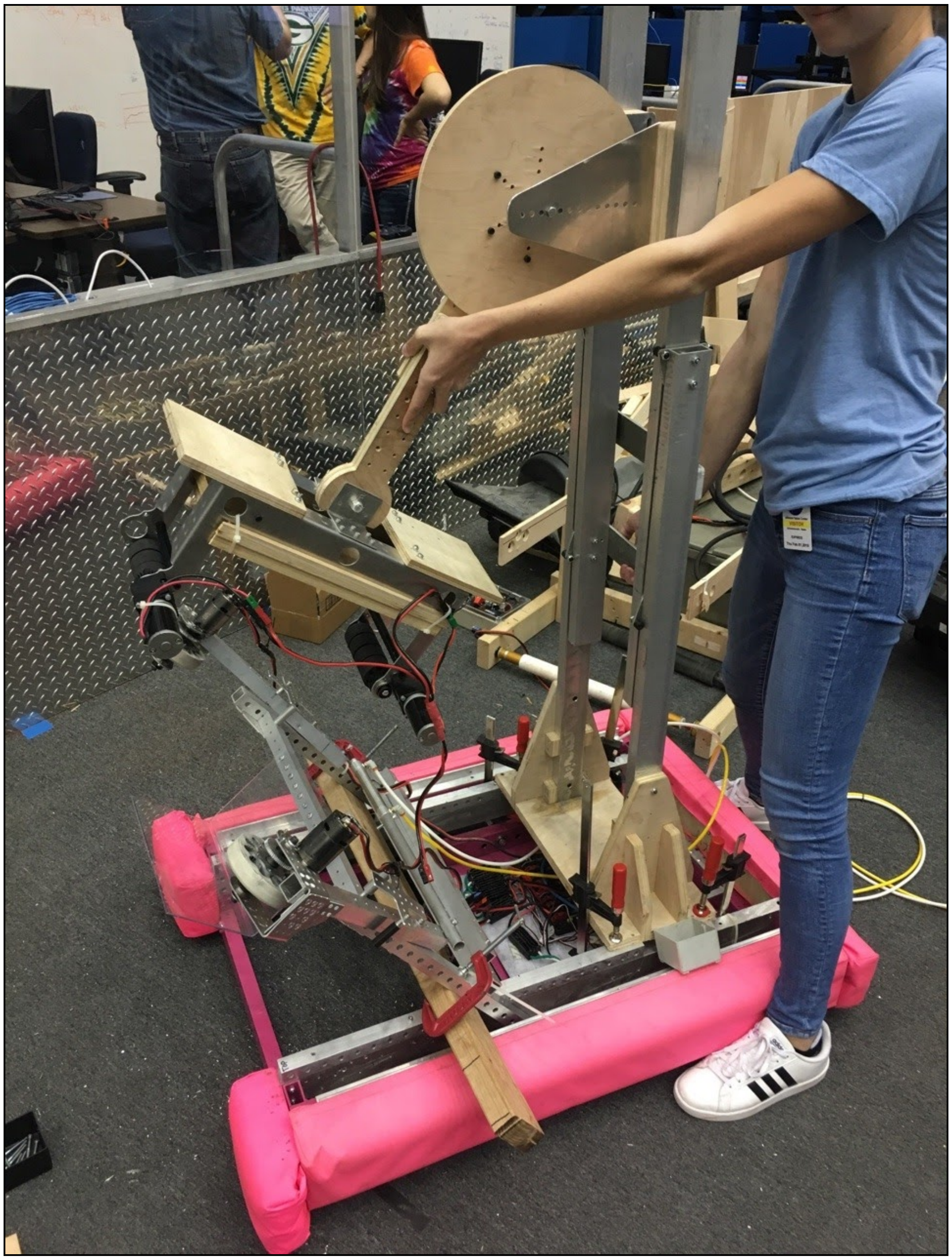
2018 intake prototype from day 13 of the build season. This uses a Pneumatics Test Box and a Power Gun.

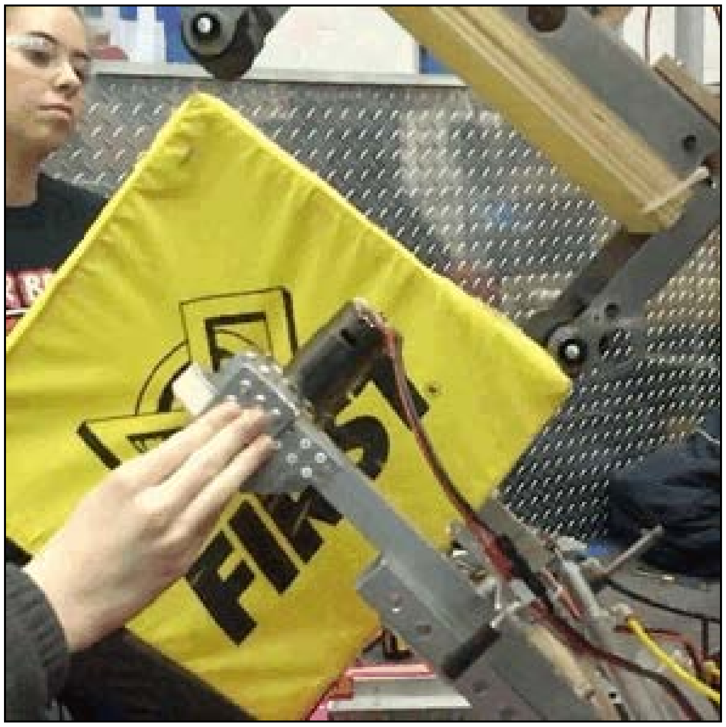


2018 intake prototype from day 13 of the build season. This is being used to test if when the intake is stowed, a cube could be launched into the switch in autonomous. Note the pneumatics test box and Robonauts Prototype Controller being used to test it.

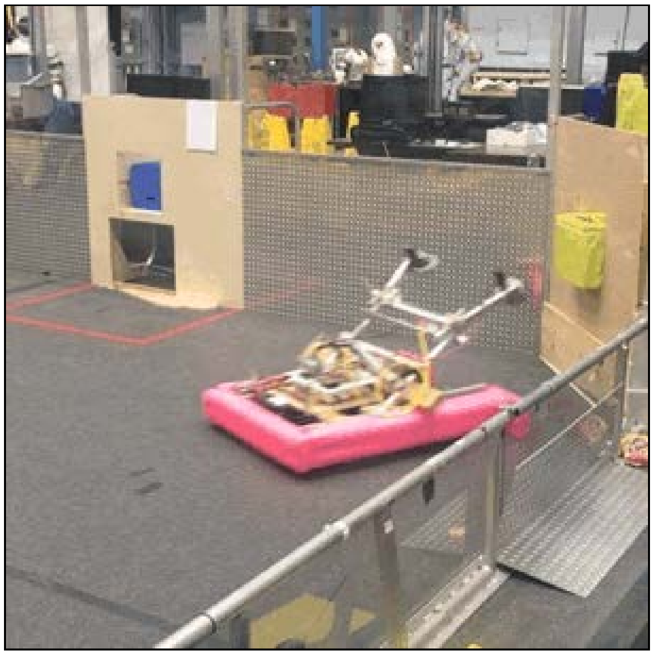


2018 intake prototype from day 13 of the build season. This prototype uses a Pneumatics Test Box and a Power Gun for testing.


2018 integrated prototype mounted to a PACBot, used to determine the perfect geometry for the robot’s intake, arm, and elevator.



2018 integrated prototype being used to find the perfect geometry for the cube hand-off.



2018 intake prototype mounted to a PACBot being used for driver training on day 21 of build season.

8. CAD

CAD (Computer Aided Design) is a great tool for prototyping and designing robots. Designing robots in CAD allows for you to more efficiently lay out your robot's mechanisms and allows you to design parts to be cut on CNC machines or 3D-printed. With CAD, you can design more advanced custom components and mechanisms.

There are a large number of CAD programs out there, many of which can be downloaded for free by FIRST teams. There is an extensive list here that compares the features of almost every CAD software package out there.

8.1 CAD Software Packages

Dassault SolidWorks: one of the most popular CAD programs used in FIRST and in industry. It is relatively simple to learn how to use and is pretty forgiving for beginners. It has all the features needed for designing FIRST robots. There are SolidWorks tutorials by FRC Team 1114 and Team 973 that are fantastic. SolidWorks has an integrated file management system called Product Data Management (PDM). SolidWorks does struggle with exceptionally large assemblies, but works fine for designing FRC robots.

OnShape: a web-based CAD software that has great tutorials for beginners. Its file management is unrivaled and is very similar to Google Drive. It lacks some of the more advanced features of some CAD programs but has enough functionality for designing FIRST robots and is frequently updated to add more capabilities. Team 973 has begun creating tutorial videos for OnShape as a part of their RAMP Greybots program. FRC teams 1678 and 973 recently switched to OnShape due to its superior file management system. The ability for multiple team members to be able to easily work on the same design simultaneously as long as they have an internet connection, without needing a high end computer, is another reason some teams have switched to OnShape. MKCad has an easily accessible FRC parts library for use with OnShape. An example of CAD standards for OnShape can be found on Pearadox's website.

PTC Creo/ProEngineer: is a very powerful CAD software that works well with very large assemblies. It is more difficult to learn and use than most other CAD packages used by FIRST teams. It requires its users to be very deliberate with their use and follow good modeling practices. It is used in industry, notably by NASA, and is used by a small number of FRC teams, such as Team 118. PTC has their own integrated file management system called PTC

Windchill. As this is a NASA document, the CAD tutorials in sections 8.3, 8.4, and 8.5 are specific to PTC Creo. Older versions of PTC Creo are rather difficult to use but Creo 4.0 and 5.0 have become more similar to SolidWorks in their user interface.

Autodesk Inventor: frequently used in high schools, especially those teaching Project Lead The Way curriculum. It is used in industry, however less frequently than Solidworks or PTC Creo. It is considered by many to be less powerful than Solidworks or PTC Creo, but has all the features necessary to design FIRST robots.

Autodesk Fusion 360: very popular with the hobbyist and combat robotics communities due to being free for these users. As a result, there are a large number of great tutorials out there. It has most of the functionality of Autodesk Inventor, however it has a user interface that is generally simpler and easier to use. It uses cloud saving, similar to Google Drive which helps prevent work from being lost. A notable feature of Fusion 360 is that it will natively run on Mac OS, unlike most other CAD programs. However, it does not work as well as Inventor does with large assemblies.

8.2 Independent File Management Programs

GrabCAD: GrabCAD is a fantastic file management system with free unlimited storage. It works best when GrabCAD Workbench is installed on the machine on which you are CADing. GrabCAD allows you to manage projects in an organization and give members of the organization different file management rights.

Custom Server: some teams manage their CAD files on a custom server. This is more complicated than just using GrabCAD workbench, however some teams prefer to not have to rely on an internet connection to update their files. It also is more difficult to prevent users from saving over each others' work.

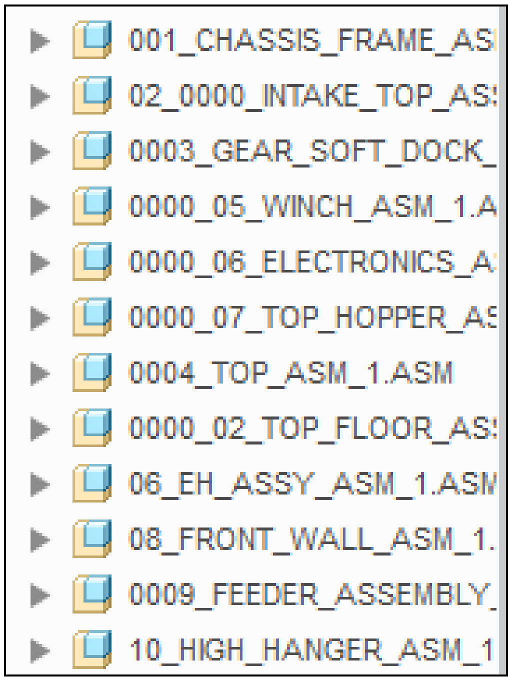
8.3 File Naming Scheme

Robot CAD Naming Scheme

Team 118 uses the following naming scheme for assemblies
0000_Top_Robot
0001_Chassis
Followed by each additional robot mechanism such as:
0002_Intake
0003_Climber
0004_Shooter

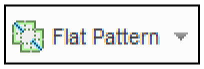
Usually each part within the assembly starts with the same numbers as the assembly, unless it is a COTS part. For example a drive rail may be named 0001_drive_rail.

Team 118 tries to follow the naming scheme shown, however sometimes assemblies do get mixed up.

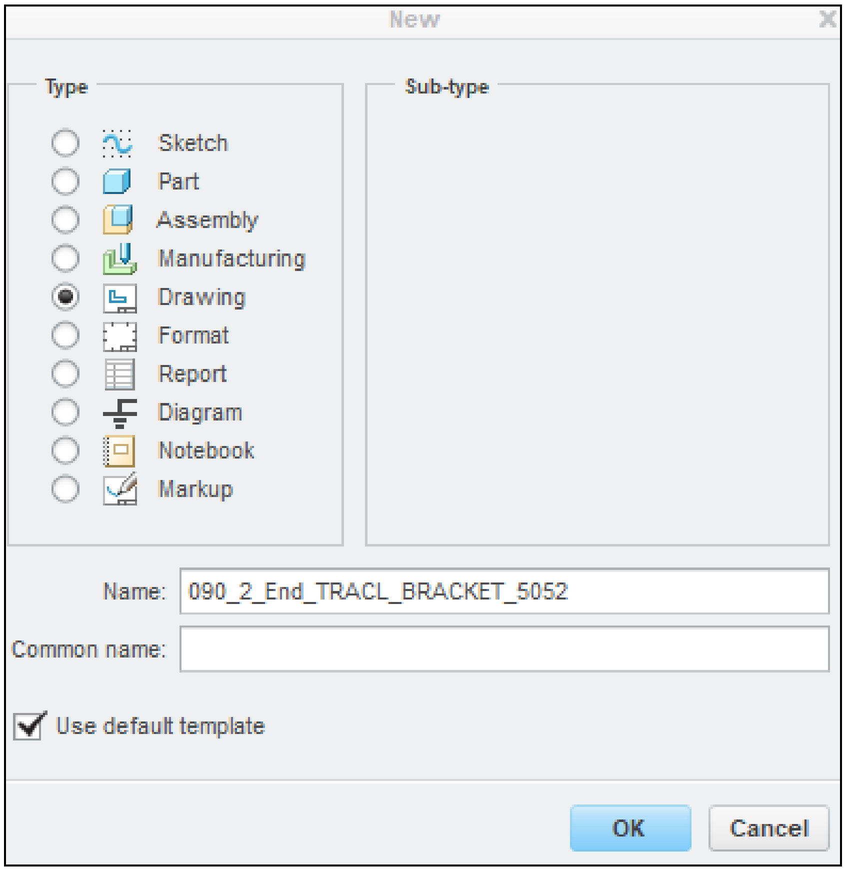


001_CHASSIS_FRAME_AS
02_0000_INTAKE_TOP_AS
0003_GEAR_SOFT_DOCK_
0000_05_WINCH_ASM_1.A
0000_06_ELECTRONICS_A
0000_07_TOP_HOPPER_AS
0004_TOP_ASM_1.ASM
0000_02_TOP_FLOOR_AS
06_EH_ASSY_ASM_1.ASM
08_FRONT_WALL_ASM_1.
0009_FEEDER_ASSEMBLY_
10_HIGH_HANGER_ASM_1

8.4 Creating DXF's (PTC Creo/Pro Engineer)

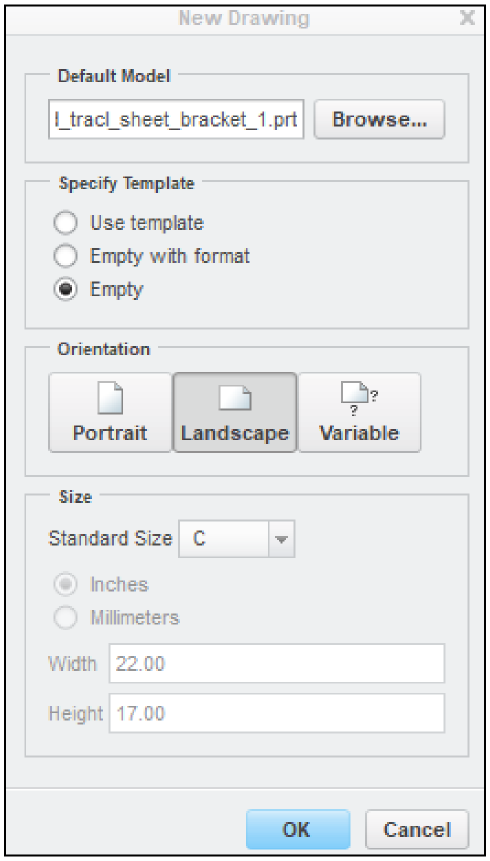
1. Open the part.
   If the part is a bent sheet metal part,
   select **Flat Pattern**  and click the green check. (You can suppress the flat pattern feature in the model tree after you've made your drawing to get it to show up as a bent part again.)

2. Click **New** and select **Drawing**, name the drawing, and click **OK.** The naming scheme Team 118 uses is *MaterialThickness_Quantity_PartName_Material*



Drawing: [✓]
Name: 090_2_End_TRACL_BRACKET_5052
Use default template: [✓]

3. Select **Empty** under 'Specify Template' in the next window and click **OK**.



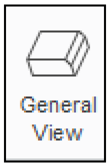
New Drawing

Default Model: l_tracl_sheet_bracket_1.prt

Specify Template
- ○ Use template
- ○ Empty with format
- ● Empty

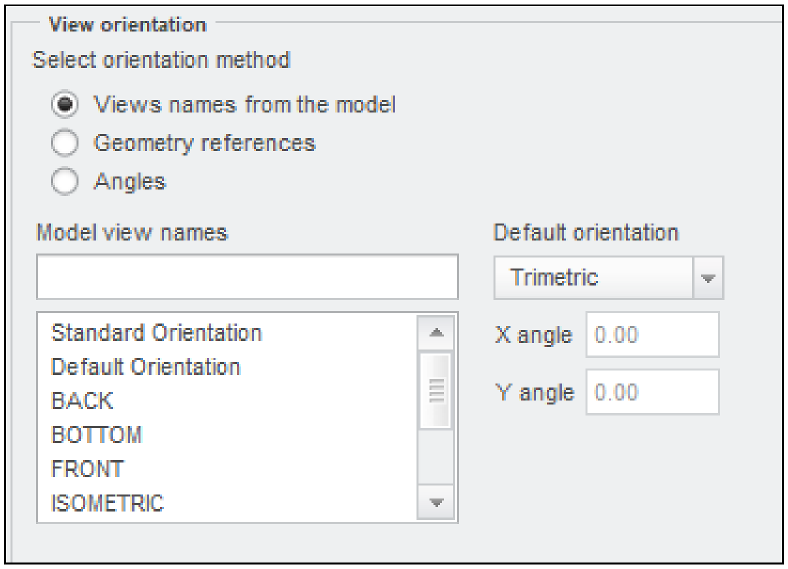
Orientation: Landscape

Size
- Standard Size: C
- ● Inches
- ○ Millimeters
- Width: 22.00
- Height: 17.00

4. Click **General View** 

and click **OK** when prompted to select a combined state. Then click anywhere on the sheet.

5. Select the desired orientation from the **View orientation** menu.



View orientation

Select orientation method

- (●) Views names from the model
- ( ) Geometry references
- ( ) Angles

Model view names

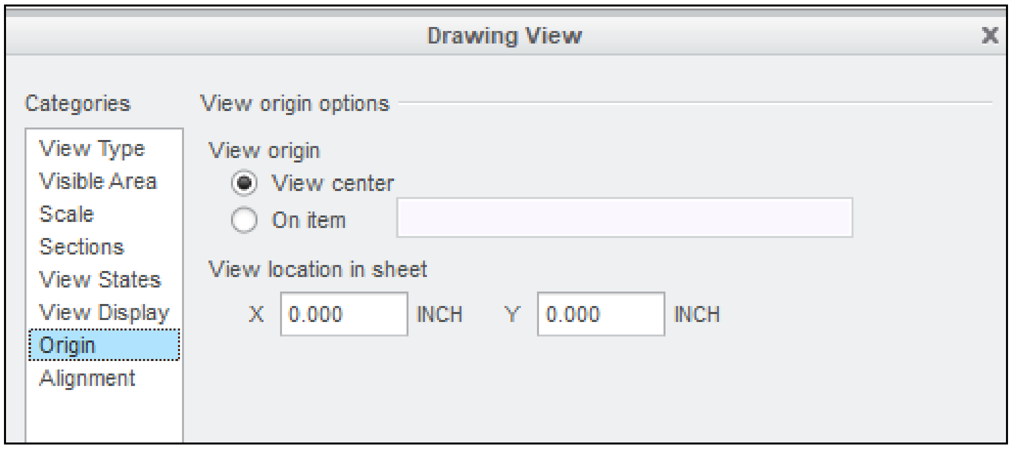
Standard Orientation
Default Orientation
BACK
BOTTOM
FRONT
ISOMETRIC

Default orientation: Trimetric

X angle: 0.00

Y angle: 0.00

6. Click on **Origin** under 'Categories', and set both the X and Y values under 'View location in sheet' to **0.**



Drawing View

Categories: View Type, Visible Area, Scale, Sections, View States, View Display, Origin, Alignment

View origin options

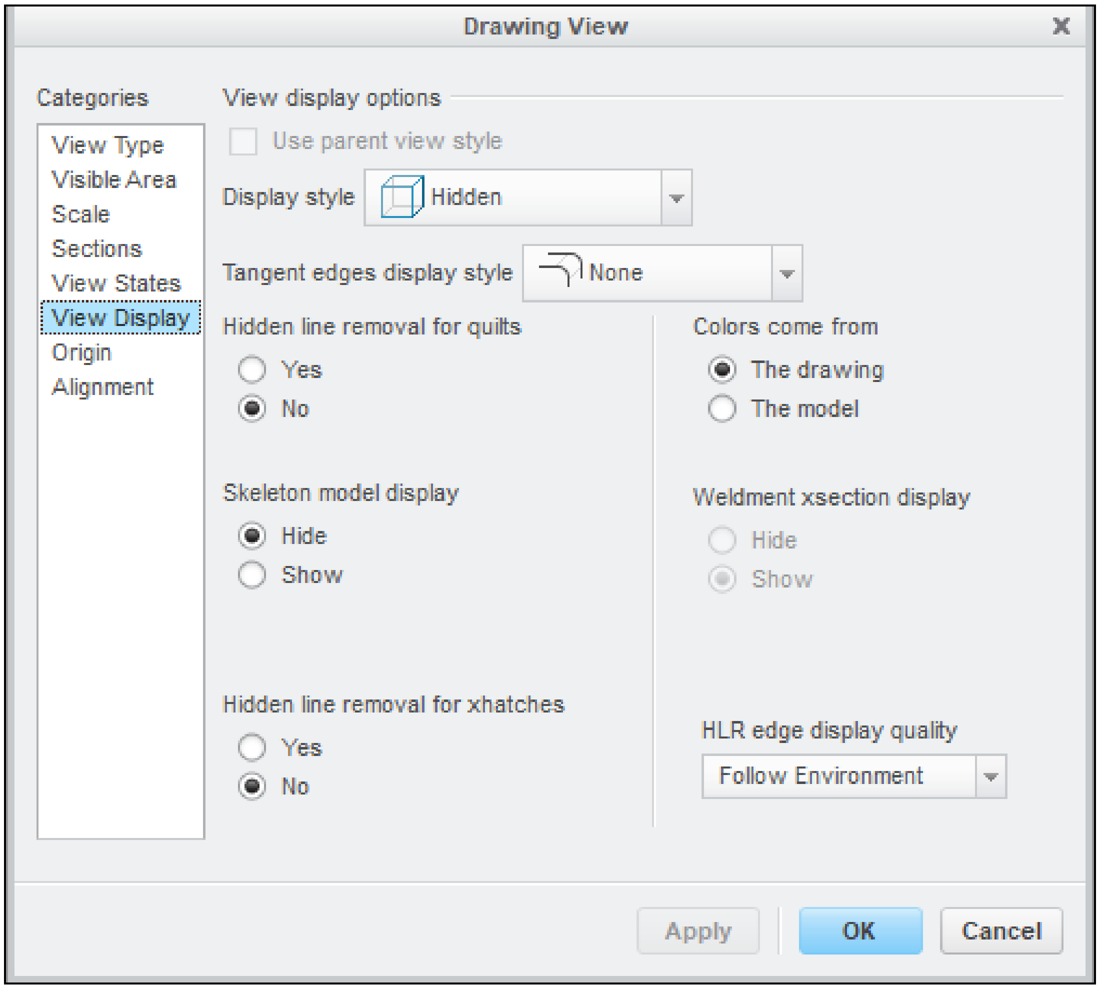
View origin

- (●) View center
- ( ) On item

View location in sheet

X 0.000 INCH   Y 0.000 INCH

7. For bent sheet metal parts the bend lines need to be removed. Click **View Display** under 'Categories', set the 'Display style' to **Hidden**, set the 'Tangent edges display style' to **None**, and click **Apply** and **Ok.**


Drawing View

Categories: View Type, Visible Area, Scale, Sections, View States, View Display, Origin, Alignment

View display options

Use parent view style

Display style: Hidden

Tangent edges display style: None

Hidden line removal for quilts: Yes [✓ No]

Colors come from: [✓ The drawing] The model

Skeleton model display: [✓ Hide] Show

Weldment xsection display: Hide [✓ Show]

Hidden line removal for xhatches: Yes [✓ No]

HLR edge display quality: Follow Environment

8. Right click anywhere on the sheet and select **Sheet Setup** and uncheck the box that says **Show format.**

9. Make sure no bend angles or dimensions are visible. If they *are* visible, select and delete them. If the dimensions or bend angles are left there, the waterjet will cut the dimension labels into your part.

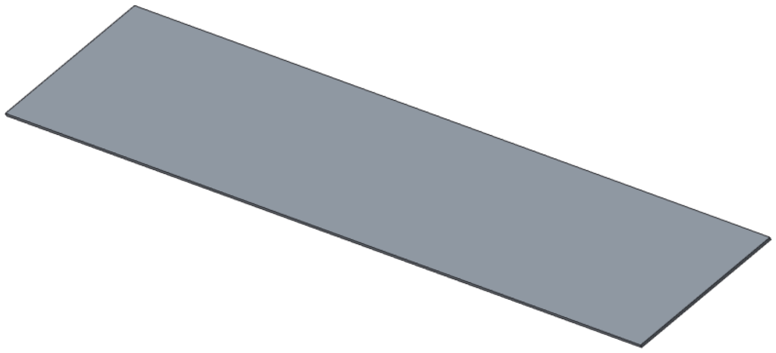
10. Double click on **Scale** SCALE : 1:1 at the bottom left corner and set to **1.**

11. Click **File** → **Save As** → **Export**. Then select **DXF** and check to see that the part looks the way you expect it to, and then click **Export.**

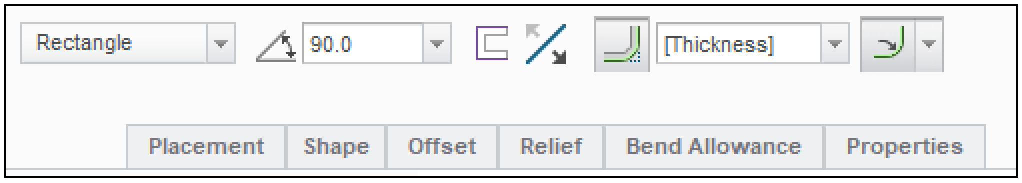
8.5 Designing Sheet Metal Parts (PTC Creo/Pro Engineer)

In this quick example you will see how to create a basic C-Channel in PTC Creo. All the principles can be translated over to more complex sheet metal parts.

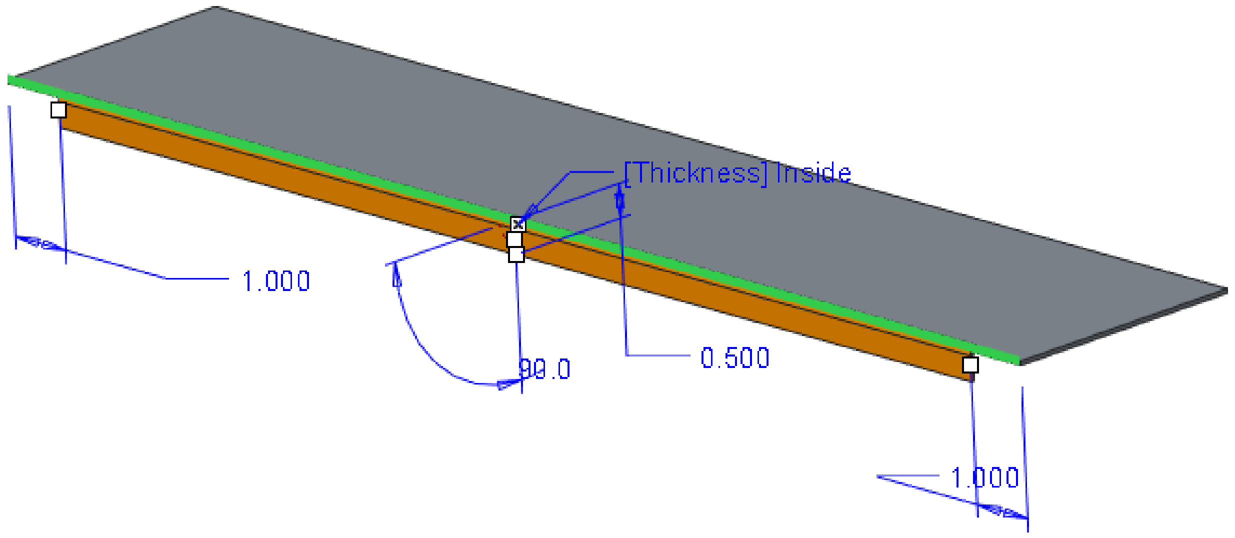
1. Create a **New** → **Part** and be sure to check the **Sheetmetal** Sub-type, then click **OK.**
2. Click **Sketch** then select a plane, in this example we will select the **TOP** TOP plane.
3. Draw a **Center Rectangle** and dimension to **4"X20"** and click **OK.**
4. While your **Sketch 1** is highlighted select **Planar** Planar, this is similar to extrude, but exclusive to sheet metal parts. The first feature of a sheet metal part must be a planar.
5. Input a thickness of **.0625** 0.0625 and click the **green check** ✓.
6. Here's how our part currently looks. As it is, it's pretty flimsy and doesn't help us much. That's where the bends come in.



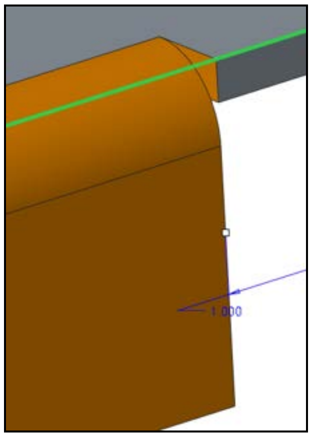
7. Select **Flat** Flat and then select the edge opposite the direction you want the bend to go.
8. At the top, input the bend angle for this example is **90 degrees** and the bend radius defaults to **1 times the part thickness**. Be sure that the far right drop down menu is set to otherwise you will be specifying the outside radius instead of the inside radius.



9. Now we can specify the flange length. Set the flange length to **.500** and set the numbers at the ends of the flange to **-1**. This will make the flange 1 inch shorter on each side. You normally don't make the flange shorter than the overall length of the C-channel because it reduces its strength; however, it's being done in this example to show how to add a bend relief.

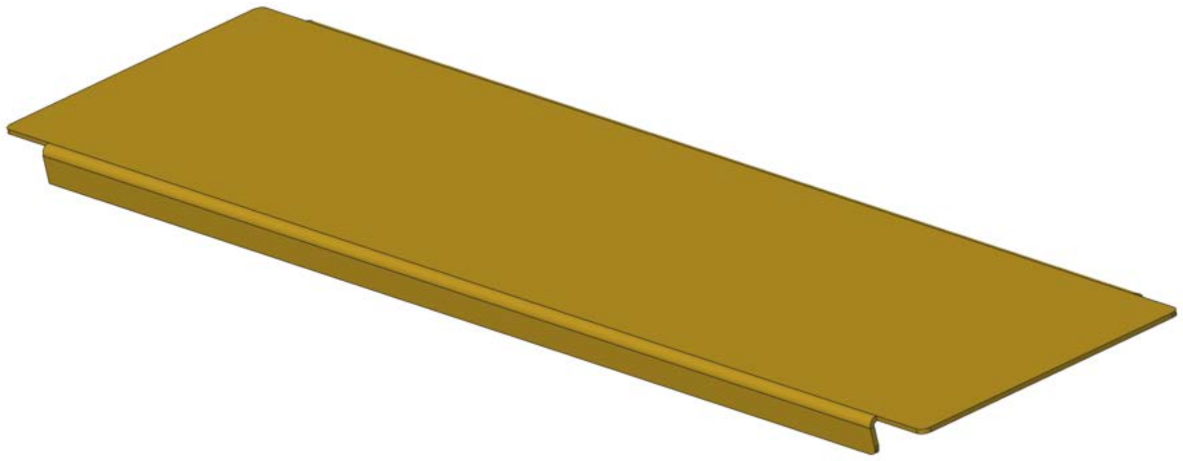


10. Now we need to add bend reliefs. If we don't, the sheet metal will tear. If you zoom in to the edge of the bend you will see that it is tearing. One method of adding a bend relief is by clicking **Relief** and selecting **Obround** from the drop down menu. Obround is better than rectangular because it doesn't produce stress concentrations created by sharp corners. The easier method of preventing tearing is to go into the **Offset** bar and check the box that says **Offset wall with respect to attachment edge** and select **Add to part edge** from the dropdown menu. This adds the bend to the part edge, which eliminates tearing; however, be aware that it changes the outer dimensions of the part so the part is no longer 4 inches in this dimension. It is now 2 part thicknesses longer, one part thickness for the bend radius and one for the material of the flange. This means if you made the bend radius 2 times the part thickness, then the part would be 3 times the part thickness longer.



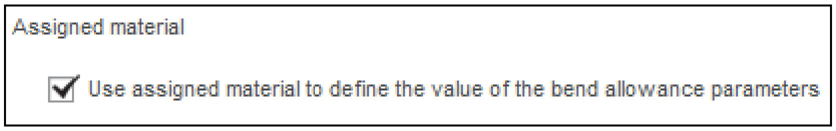
11. Next create another flange opposite the one you just created using the same process.

12. Lastly, as a good design practice, add **Rounds** to all of the sharp corners. A round of **.125”** is what we typically use on outer edges. When you're done your part should look like this.

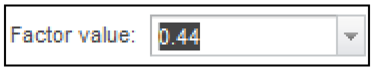


8.6 Setting the K-Factor of a Sheet Metal Part (PTC Creo/Pro Engineer

1. Team 118 uses a K-factor of .44 for all 5052 aluminum bent parts.
2. **Open** your sheet metal part and then select **File**, **Prepare**, then **Model Properties.**
3. Locate **Material** and click **change** change.
4. Double click **metal_aluminum_5052.mtl** and it will be added to the **Materials in the Model** window, then click **OK** OK.
5. Locate **Bend Allowance** and click **change** change.
6. Uncheck the **Use assigned material to define the value of the bend allowance parameters** box.


Assigned material

✓ Use assigned material to define the value of the bend allowance parameters

7. Select the **K factor** radial button ● K factor and then set the **Factor value** to **0.44**

Factor value: 0.44

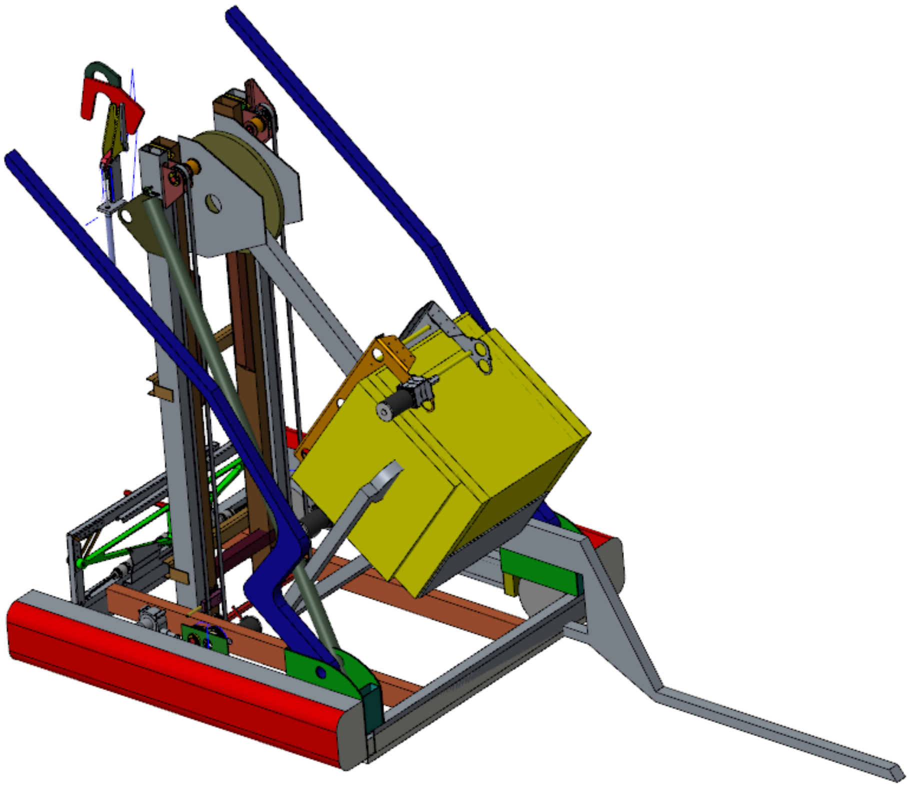
. Then click **OK** OK.
8. Click **Close** Close and you should be set. Remember to do this for every one of your bent sheet metal parts, otherwise the holes will not line up after they are bent.

8.7 The Progression of CAD

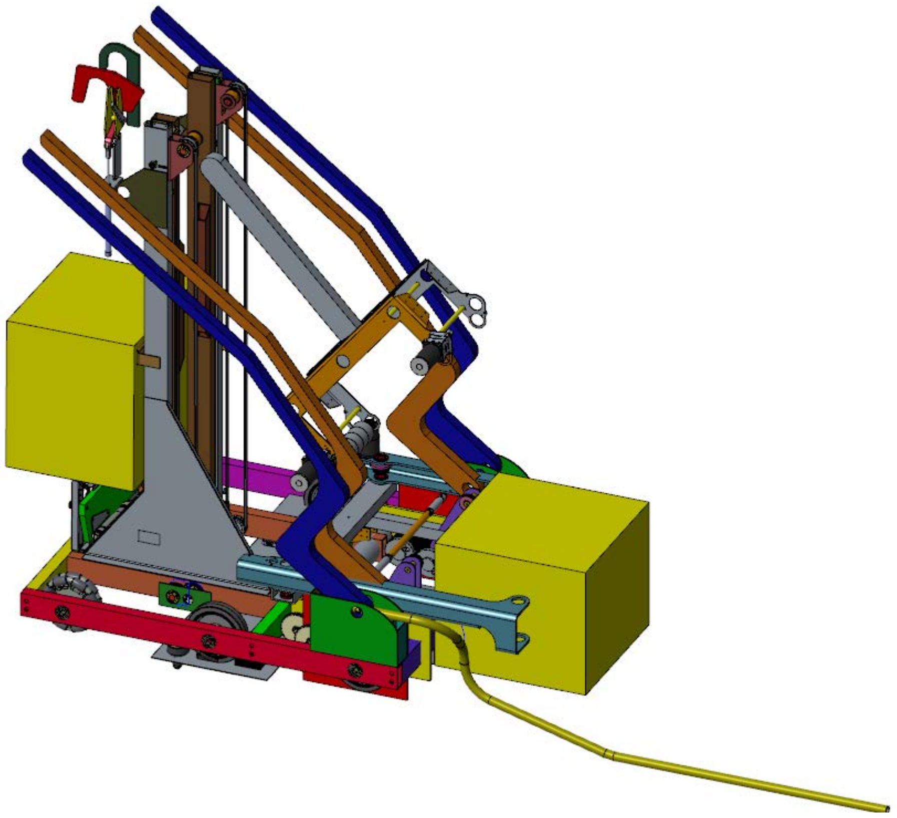
When you look at many FIRST robots, it’s very easy to find yourself thinking “there is no way I could have designed that.” It's important to remember that Rome wasn't built in a day, and the same goes for FIRST robots. Designing robots is an iterative process. It's good practice to build prototypes that perform well and take dimensions from them, and then design the final robot using these dimensions. This does not always work out perfectly, but it is a very good way of doing things.

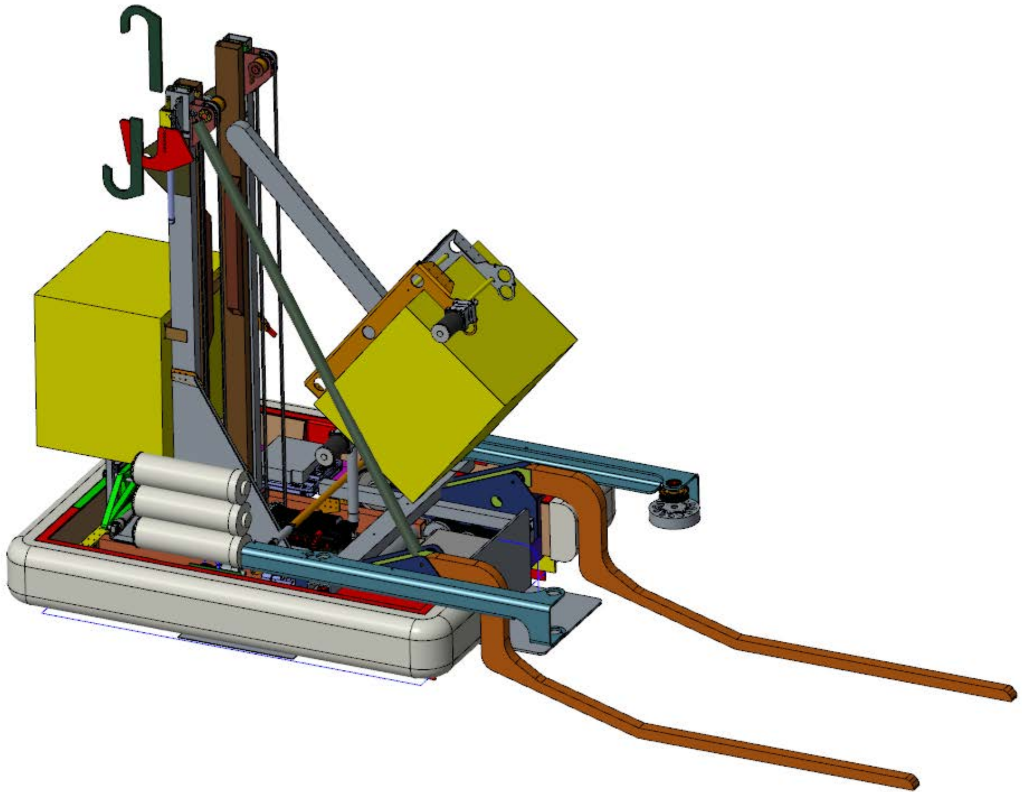
Teams usually start with what is called Crayola CAD, it is called this because it's a very rough drawing of what the robot is going to look like and because all of the parts are usually different colors. This makes design reviews easier. Crayola CAD begins with getting the general layout of the robot and creating parts with the dimensions taken from prototypes. During this phase parts get added, removed, moved around, change shape, and get re-designed. After the mechanisms are assembled, they are frequently re-designed and manufactured again to make them work.

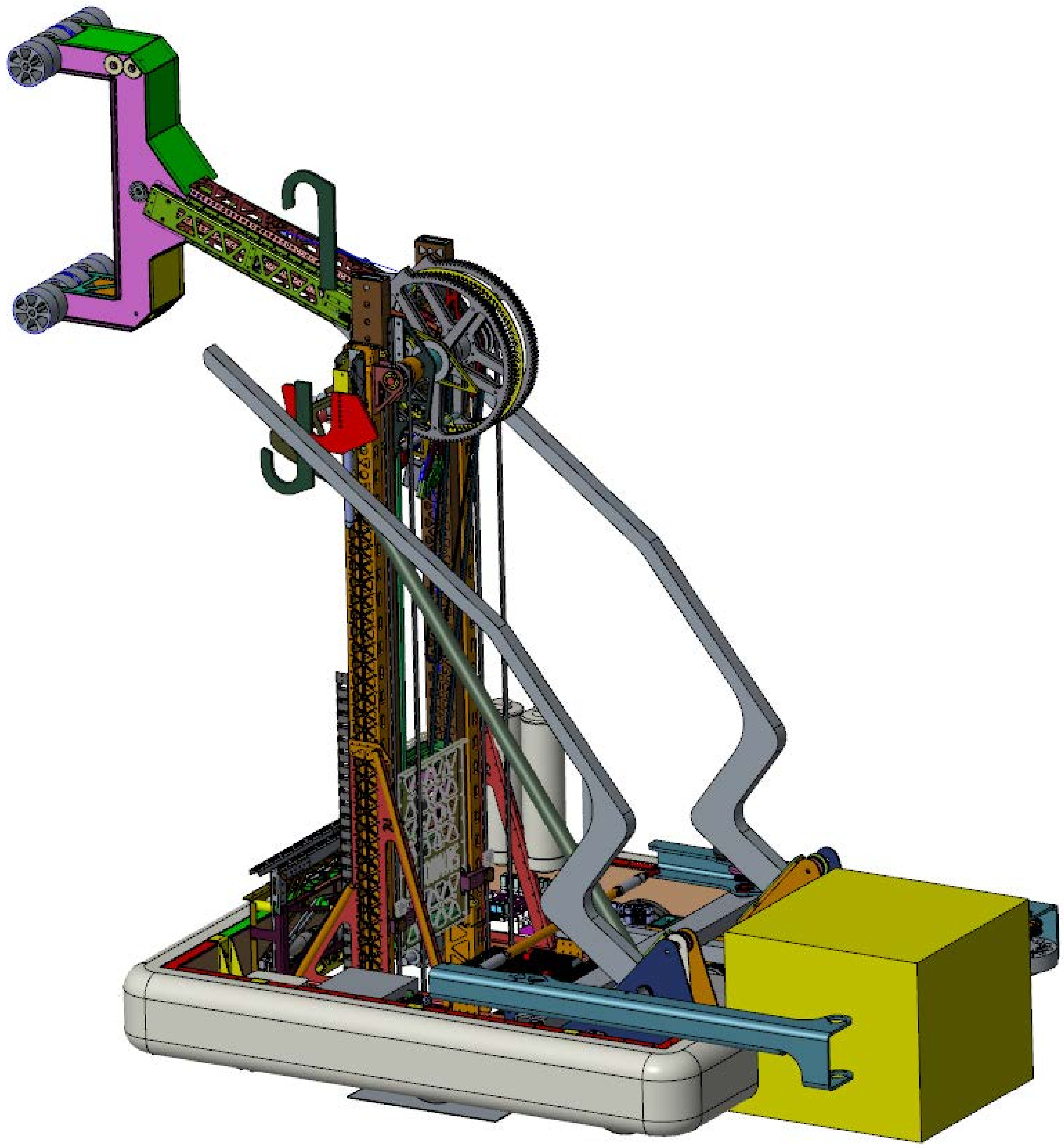
The following pictures show Team 118’s 2018 robot CAD progression through the Crayola CAD phase to the final phase.



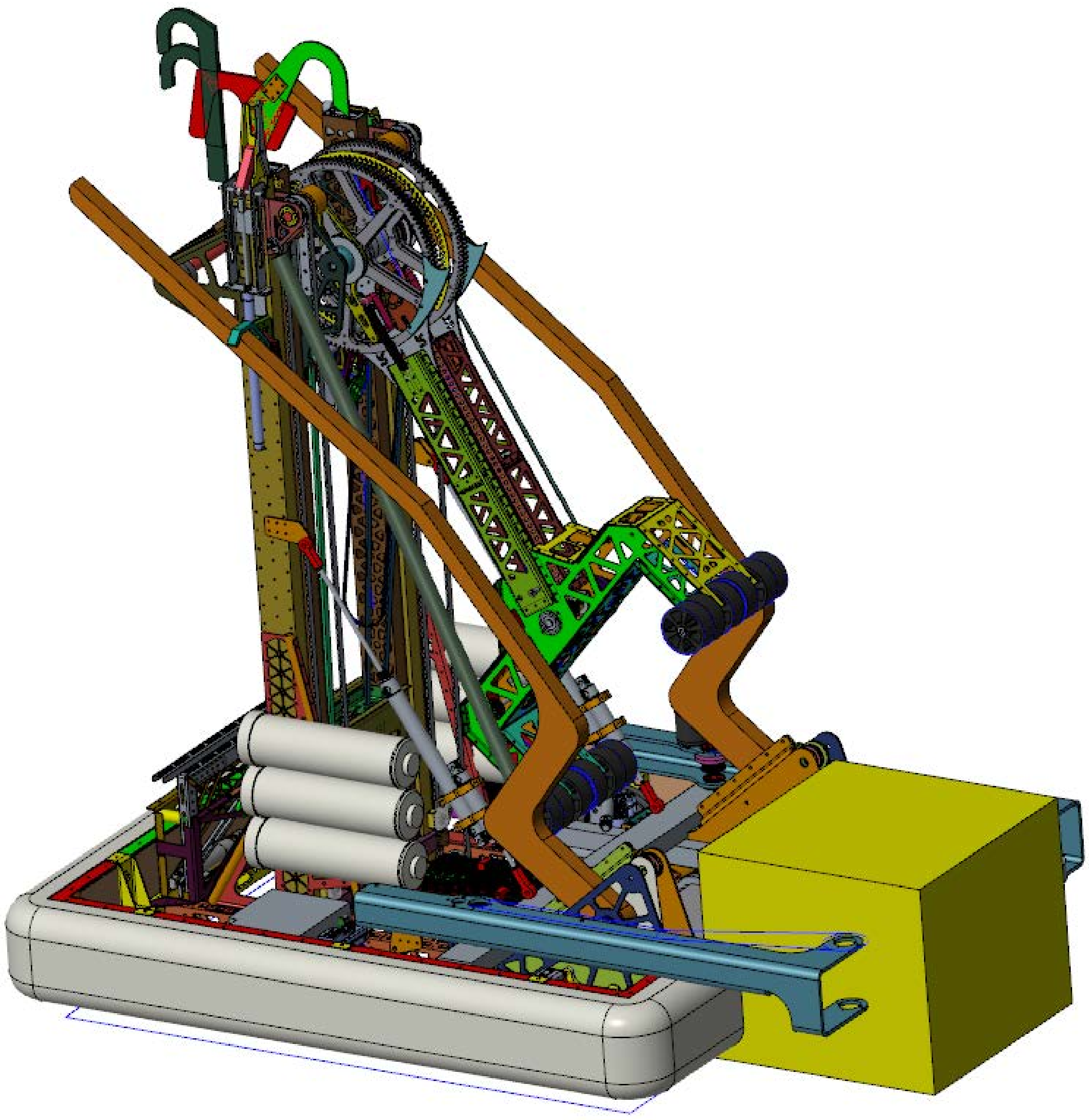
The CAD of Team 118’s 2018 robot on January 20th.


The CAD of Team 118's 2018 robot on January 23rd.

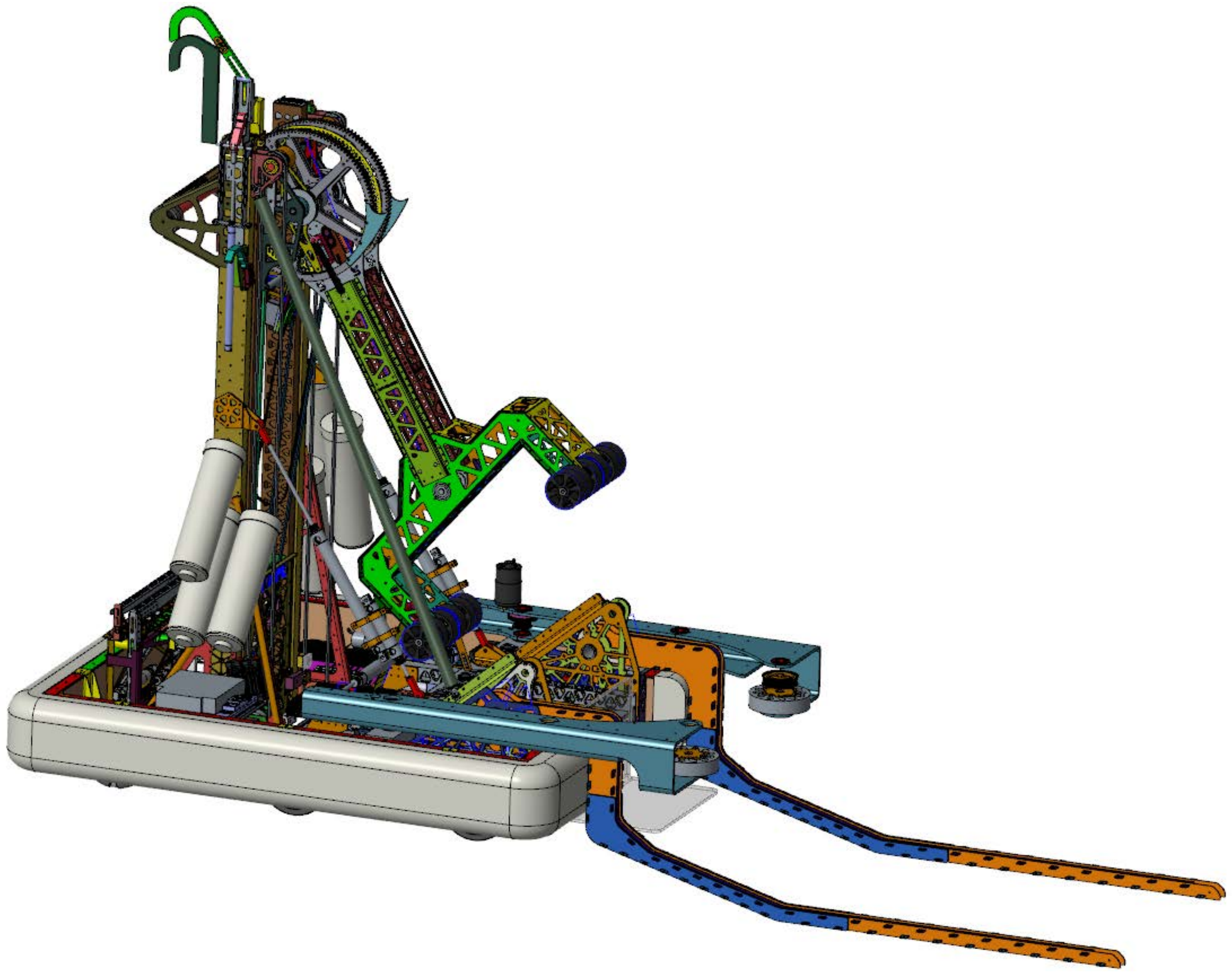

The CAD of Team 118's 2018 robot on January 24th.



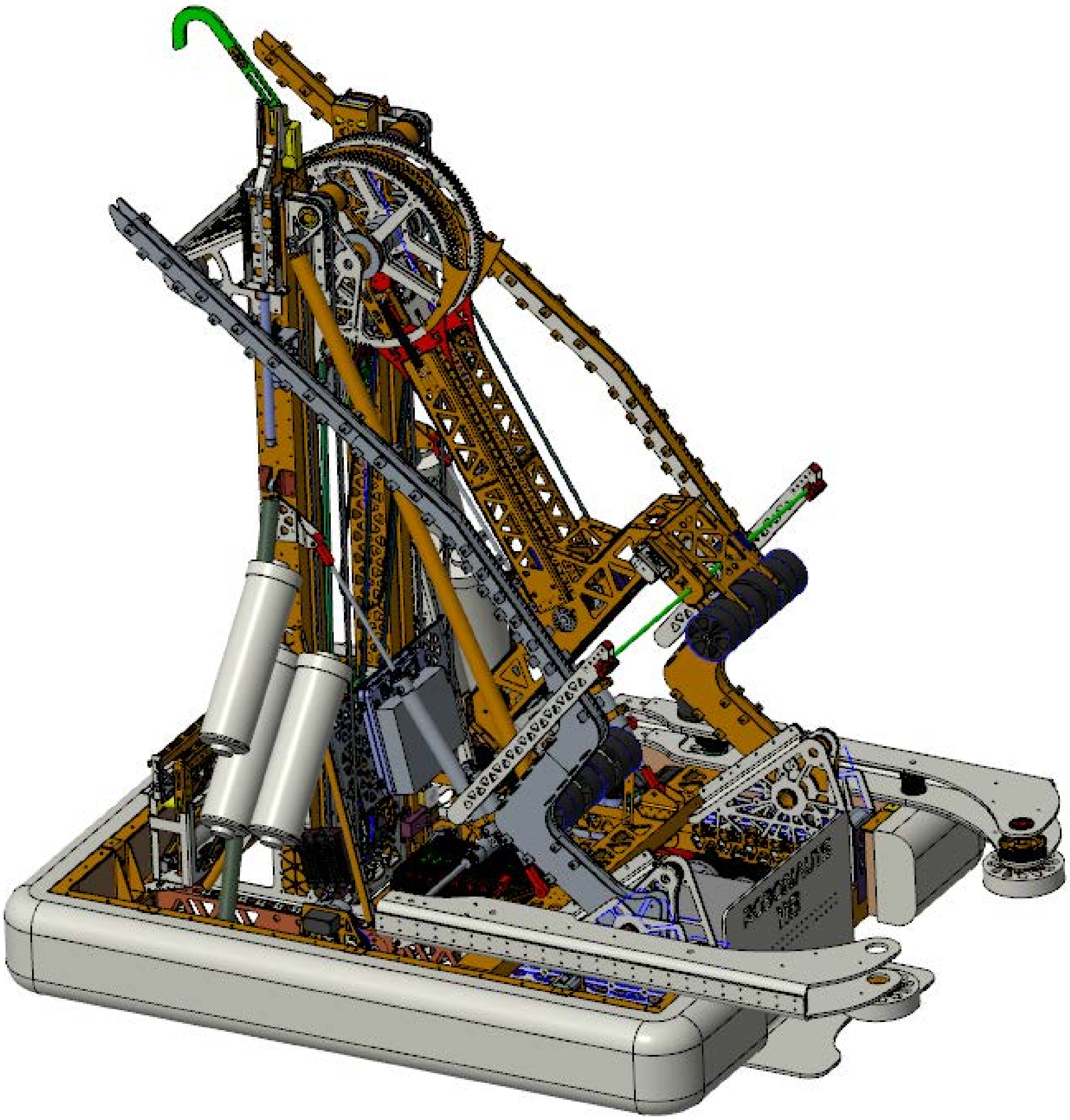
The CAD of Team 118's 2018 robot on January 26th.



The CAD of Team 118’s 2018 robot on January 30th.



The CAD of Team 118's 2018 robot on February 6th.



The CAD of Team 118’s 2018 robot at the end of the season.

9. Software

9.1 Getting Started

When starting with writing software for FRC, if you take some time to prepare yourself and your team with a little background information and some guidelines, you will end up with much more functional and maintainable code.

So before you start working on your code, take a little time to:

1. Select a language
2. Learn about your language and related resources
3. Develop a style guideline
4. Select an FRC Robot Framework
5. Plan the code for your robot

The most important thing about programming, especially in FRC, is for your code to work. However, making sure that your code is easy to update and maintain is a very close second. As the robot design matures, the desired implementation of the code may change. Having the ability to easily update and maintain code will make everything much easier. This is especially important when working as a team; Maybe you'll remember how all the code you wrote works, but if it isn't written in a way your teammates can understand, they won't be able to update it, if it needs to be changed. Perhaps the programming lead is sick or unavailable when you need to make an important change. Things like comments, clear implementation, and variable names allow more people to be able to efficiently make necessary code changes in a time crunch.

Over the years, FRC has developed a set of tools, libraries, and guidelines for writing easily maintainable and adaptable code. Used properly, these can make it far easier to understand code that you or someone else wrote and keep up to date with changes to robot mechanisms and functionality.

9.1.1 Selecting a Language

The first step for any team when starting to program is to select a programming language. The most common way for teams to select a language is by gauging the expertise of the teams mentors and students. However, with some time and effort, you can change to another language if you weigh the pros and cons and deem it to likely lead to an improved performance or a worthwhile learning exercise The 3 officially supported languages used within FRC are:

Java: Java is a text-based programming language, and is the most widely used language with FRC teams. Some benefits to Java is that it is often taught in high school computer science classes, is the universal language for First Tech Challenge (FTC) teams, and you are likely able to find a team at a competition that can help you, as nearly 70% of all teams use Java according to the 2019 usage report. Java is also a great choice for beginners because it has many high level features that protect against common errors. However, these features come with the drawback that it makes the code non-deterministic, meaning that the order of execution of the code can sometimes be impacted by timing of other events. Because of these shortcomings, Java is generally not used in commercial and industrial robotics. Java is used in many other industries that don't depend on performance and predictable timing of code execution. In FRC, Java may be developed in Windows, MacOS and Linux.

C/C++: C/C++ is a text-based language that allows for more flexibility and lower-level coding than other languages used in FRC. If programmed properly, C++ is more powerful and efficient than the other languages. It can provide extremely precise control over execution timing which results in better operation of control loops. It is often used for robotic applications in industry. C++ is more complex, harder to teach, and harder to learn. With increased capabilities, comes an increased chance to introduce errors. And with C++, errors can often be difficult to track down and correct. In FRC, C/C++ may be developed in Windows, MacOS and Linux.

LabVIEW: LabVIEW is unique from all the other languages available for use on FRC robots in that it is a graphical language (i.e. not text-based). LabVIEW can be good for new programmers who don't have any experience with text-based code. LabVIEW also has good debugging tools built in. However, it also has slower compile times and is difficult to do version control, as merging is impossible with graphical languages. LabView has a relatively small following in industry and commercial use. Where it is used, it is a very powerful tool and has incredible support by those who use it. LabVIEW for FRC only runs on the Windows operating system.

There are some teams using unsupported languages successfully. These include, but are not limited to Python, Rust, Kotlin, Fortran 77, C, Malbolge, and Javascript. Python is probably the most popular of the unsupported languages with around 50-100 teams as of 2020. If your team has a strong desire to use another language besides the 3 officially supported languages, be sure to spend time in the offseason familiarizing yourself with the ins and outs as you are unlikely to be able to find support at a competition if something goes wrong.

9.1.2 Learn About Your Language and Related Resources

After you select a language, take a little time to learn more about the language you selected and the resources that are available to help FRC teams with that language. Hopefully you selected a language that some members (students and/or mentors) have some experience with that language. If not there are many online tutorials for all FRC supported languages. Tutorials that will help you learn the basics of the language and tutorials and examples that will help you program your FRC robot. This is just a small list of some of these resources, search the internet to find hundreds of more freely available resources.

Also remember, FIRST rules allow you to use almost any code you find freely available on the internet on your robot -- as long as all teams have access to that resource. You don't always need to start from nothing.

Resources for Every Language:

- **WPI Library:** Includes resources for all of the basics including installation and wiring for every language. WPI Library also contains different commonly used tools such as Shuffleboard as well as more advanced resources such as vision.
- **FRC Programming Done Right:** A comprehensive overview of all things FRC programming. Includes resources for Java, C++, and python.
- **Github and GitLab:** Searching for FIRST or FRC in the search bar will pull up repository after repository of other teams' code, which can be used as reference.
- **Commonly Used Libraries:** When programming a robot, you will need 3rd party libraries to be able to talk to certain motor controllers and update them to the correct version. Commonly used libraries include the REV Robotics: Spark Max, Cross The Road Electronics: Talons, and Cross The Road Electronics: Falcons. In conjunction with the libraries, these motor controllers run firmware developed by the vendors. The team loads the firmware onto their devices, following instructions provided by the vendors. Typically, one can find needed firmware and software resources pretty easily by doing a simple web search.
- **Vendor's Documentation:** Many vendors of motors and sensors will have detailed documentation regarding the specs of their product. The documentation also includes how to load firmware and interface to their products from the team's software. Some examples of common vendors in FRC include Cross The Road Electronics, Analog Devices, National Instruments, and REV Robotics.
- **Chief Delphi:** Similar to the Stack Overflow for FRC. Chief Delphi is a forum for all things FRC and many times, there will be helpful software threads. Additionally, when teams do code releases, they often release it onto Chief Delphi.

- **The Blue Alliance: **The Blue Alliance is a website with data on every FRC match in history and has YouTube videos for the more recent matches. The Blue Alliance is a great place to learn about other team's by watching their matches and learning new auto/software ideas. Additionally, The Blue Alliance allows teams to look up other successful teams in their area and reach out if necessary.
- **FRC Discord:** There are channels in the FRC Discord server dedicated to programming where you may be able to get help from mentors and students from other teams.

Java Resources:

- **WPILib Example Projects - FRC documentation:** Contains links to basic, control, sensing and other getting started examples for Java/C++ on the WPILib githib account.
- **Free Java Textbook:** A great place to start learning the java language.
- **Oracle Documentation:** Can be used for grasping the basics of the java language (no FRC knowledge included)
- **Java Robotics Course by CJ Weir of FRC 5472 Stallion Robotics:** A fun and broad course on all things FRC in a textbook format with questions to test one's knowledge.
- **Java Resources by FRC 2102 Paradox:** Split into FRC and basic java, these resources step you through the basic syntax and FRC applications of java.

C++ Resources:

- **WPILib Example Projects - FRC documentation:** Contains links to basic, control, sensing and other getting started examples for Java/C++ on the WPILib github account.
- **Official C++ Tutorial:** Takes a step by step approach to explaining C++. A good place to start as a beginner.
- **971 C Programming:** Basics and a getting started guide for using C/C++ for FRC.
- **FRC C Programming:** Step by step guide to getting started with C/C++ in FRC. Includes install instructions!

LabVIEW Resources:

- **FRC LabVIEW Programming - FRC documentation:** The WPILib supported labview quick start guide and tutorial.
- **The Secret Book of FRC LabVIEW:** This book is an extensive overview of almost everything one would need when it comes to programming a robot in LabVIEW. This book is a great way to get started learning LabVIEW.
- **Eagle Robotics (FRC 358) Website:** This website has extensive resources for more detailed use cases for LabVIEW as well as the more commonly used methods. For example, it has code for using I2C, POV Hat Switch, and State Machines.

- **National Instruments Official Support:** Great for using to debug code and look up error messages.
- **National Instruments Tutorial Guide:** A walk through for basic roboRIO and FRC LabVIEW programming using a series of video guides.

9.1.3 Develop a Style Guideline

If you create a "Style Guideline" for your software team, you will end up with a more functional and more maintainable set of software. When most people think of a Style Guideline they think about the formatting of the code, but a good style guideline will include much more. A good style guideline will include best practices that you learn over the years. Start with a simple guideline that talks about code formatting and let it grow into a complete guide for your programmers.

Because style guidelines tend to contain specifics related to the language you are using, it is difficult to give you a comprehensive document that will work for your team. A basic search using your favorite search engine will provide examples of style guides for the C/C++, Java and LabVIEW languages (for example, google "labview style guide". There are also many different opinions on programming styles from how to name a variable to what programming paradigm you should use. The following sections cover some things that you may want to include in your style guideline. The bullets are examples of what you could put in your style guideline. Of course you should make them fit your team's styles and needs.

It is also important to remember that you will be using code written by other organizations (that may or may not have style guidelines, or may be different from your team's style guide). You cannot change the style they decided to use, so there will be some bits of your code that do not adhere to your guidelines -- it's just something you have to live with.

9.1.3.1 File Management

- One file should contain one class. The name of the file should match the class name. For example the class CompetitionRobot should be in CompetitionRobot.java (Java), CompetitionRobot.h/CompetitionRobot.cpp (C++), or CompetitionRobot.vi (LabView).
- You should never have two files in the same directory/folder that differ only in the case of the letters. This will cause problems if you ever put those files on a Windows based computer.
- We will use git for software version control.

- All modified code will be committed to the version control system (git repository) feature branch at the end of each meeting. After unit testing the feature branch will be merged into the development branch. After system testing, the development branch will be merged into the master branch.
- C++ source files will be placed in a directory called “src”. C++ header files will be placed in a directory called “include”. Both directories will be at the same level within the file system.

9.1.3.2 Code Formatting

A critical part of writing understandable code is making sure the words in the code make sense to humans as well as computers. The compiler doesn’t care what you call your classes or variables, but giving them descriptive names means that you and your teammates can more easily tell what it’s for without having to read the code to find out how it’s used. A method called `scaledArcadeDrive(double xSpeed, double zRotation, double scaleFactor)` is a lot easier to work with than one called `foo(double arg0, double arg1, double arg2)`, even if they do the same thing.

Fair warning, people can be passionate about code formatting guidelines. Some people will disagree with these examples. Please remember they are just examples to help you get started. Formatting guidelines will almost never impact the compiling or execution of your code but they will make it easier for everyone to read your code and thus will make it easier for everyone to maintain your code. Most Integrated Development Environments (IDEs) have some way to help enforce many of your formatting rules, especially those related to spaces and indentation.

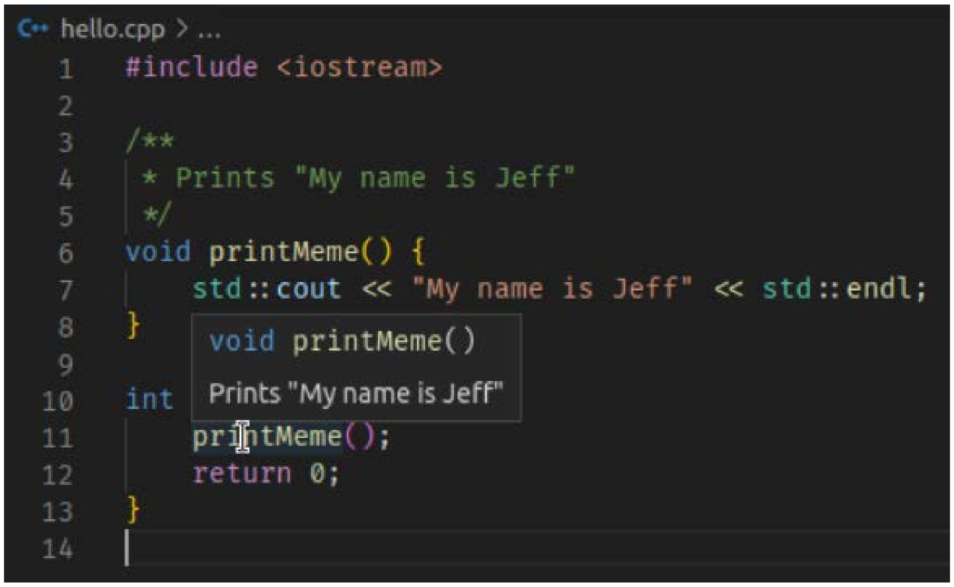
- Every variable type (including every class) should be named in camel case with an initial uppercase letter. That is the first letter of the name and the first letter of subsequent words in the name should be capital, all other letters should be lowercase (ThisIsCamelCase). Underscores and other special symbols should be avoided in type names.
- Every function, method, or procedure should be named in camel case with an initial lowercase letter and should contain a verb (get, put, set, read, initialize, update, handle, etc.) (updateTheRobotLocation).
- Every variable, including instances of classes and function pointers, should be all lowercase with words separated by underscores.
- Every constant should be all uppercase with words separated by underscores.
- Every opening brace should be on its own line.

- Every closing brace should be in the same column as the matching opening brace and all of the code in the block contained by the braces should be indented more than the enclosing braces.
- There should be spaces before and after mathematical symbols.
- [Never|Always] use hungarian notation.

Hungarian notation is a method of constructing names that carry additional information about what they name. It is sometimes used in WPILib; for example, many instance variables in WPILib have names like `m_channel` or `m_axes` (the “m” stands for “member”), while constants and enum variants are typically given names like `kLeft` or `kForward` (the “k” stands for “constant”; “c” was already taken by “character”). Hungarian notation was conceived in the days when understanding the properties of a variable would otherwise require hunting through every source file in the project to find its declaration; with the rise of IDEs and advanced text editors that can find declarations for you and provide compiler hints in real time, it has dropped off in usage significantly. You can use it in your code if you want, but plain-language variable names are generally easier to understand, especially for new programmers.

9.1.3.3 Commenting the Code

When writing a descriptive name isn’t possible, or when fully describing the purpose of the variable would make the name far too long to comfortably type, you can include additional notes in comments. Classes and functions in particular often require these “documentation comments,” and it is considered good practice to include them. If your documentation comment is a block comment that starts with `/**` instead of `/*`, Visual Studio Code and most other editors will display the documentation comment if you hover your cursor over the name anywhere that it’s used.



```
hello.cpp > ...
#include <iostream>

/**
 * Prints "My name is Jeff"
 */
void printMeme() {
    std::cout << "My name is Jeff" << std::endl;
}
          void printMeme()
int       Prints "My name is Jeff"
    printMeme();
    return 0;
}
```

Example of hovering over the function "printMeme" to find out from the documentation comment that it prints "My name is Jeff".

A common example of information that is important to include in a variable's name or documentation comment is units. Is `double driveDistance;` in inches, centimeters, meters, or something weird like "encoder ticks?" Naming and documentation can help make sure that everyone working on the robot program understands the units involved.

Comments in code also have no impact on the compiling or operation of the code, but they definitely make it easier to maintain the code. There are also tools that can extract your comments from the code to make a stand alone document for your code.

- All files should start with a comment block that includes the file name, the purpose of the code in the file. This comment block should also include any copyright and license information or information about where the copyright and license information can be found.
- All class definitions should have a comment block that at least explains the purpose of the class. It is often appropriate to give an example of how the class is used in this comment block.
- All methods, functions, and procedures should have a comment block that includes a brief description of what the method does, what arguments it takes including units of those arguments, default values for any optional arguments, and what is returned by the method.
- Additional comments should be included to explain complex blocks of code or any code that is done in a "clever" way.

9.1.3.4 Managing Variables

- All variables should have the narrowest scope possible. This means you should almost never have global variables, you should minimize the number of class variables and you should use member (instance) variables only when the information they hold is used by more than one method.
    - An exception to this rule would be a variable that is only used inside of a loop. If that loop is called many times and with high frequency you may improve performance by defining the variable outside of the loop.
- All variables should be explicitly initialized.
- Constants should be used instead of magic numbers.

A "magic number" (or, more generally, "magic value") is a constant value used in a program with little or no context. Some common examples of numbers that often end up being magic numbers in FRC code include:

- Port numbers or CAN IDs for sensors and actuators
- Sensor detection thresholds
- Control loop gains (e.g. PID)
- Scaling factors applied to motor output

While constant values are necessary in many cases using them directly in the logic should be avoided as much as possible. The following code is relatively easy to read but it's hard to know what the numbers 80 and 70 mean.

```
if (distanceToTarget() > 80)
{
    driveForward();
}
else if (distanceToTarget() < 70)
{
    driveBackward();
}
else
{
    stopDriving();
    startShooting();
}
```

You could probably figure out that the robot is trying to get itself to some range of distances before it starts shooting. The following code does the same thing but is easier to read and maintain when the desired distance and tolerance are specified as constants.

```
private static final int SHOOTING_DISTANCE = 75;
private static final int SHOOTING_TOLERANCE = 5;

if (distanceToTarget() > (SHOOTING_DISTANCE + SHOOTING_TOLERANCE))
{
    driveForward();
}
else if (distanceToTarget() < (SHOOTING_DISTANCE - SHOOTING_TOLERANCE))
{
    driveBackward();
}
else
{
    stopDriving();
    startShooting();
}
```

This is much easier to read, the programmer can simply look at the names given to the numbers and understand their purpose without needing to see how they are used. This also means that if either value is updated (or made into a variable rather than a constant), the new value is automatically used everywhere the old value was.

9.1.3.5 Class Management

- All classes should have constructors.
- All classes should have destructors.
- Consider copy constructors for all classes.

9.1.4 Select a Robot Framework

WPI provides four base frameworks as starting points for your robot code. Simple, Timed, Iterative, and CommandBased. You can also find some frameworks that were developed from these on-line.

The **SimpleRobot** framework provides the most basic structure that is needed to function in an FRC competition. Because the framework is as simple as possible, it is easiest to understand what the framework is doing but it requires more work to write the software for a fully functioning robot. To use this framework you would create a class that extends SimpleRobot and write your code in the operatorControl() and autonomous() methods. More information about this framework can be found here. Using this base class is not recommended for new teams, it involves writing a lot of code to manage different phases of the game.

The **TimedRobot** framework provides more structure and takes care of switching between game modes. Using the TimedRobot as a base class requires you to write code for several phases of robot operation. They are Disabled, Autonomous, Teleoperated, and Test. Each phase has two functions that get called. The initialization method gets called when your robot enters the phase and the periodic method gets called at a regular interval (by default 20 times a second) while you are in that phase. This can be a little tricky because you often need to remember the state of things between calls to the periodic methods. More information about TimedRobot can be found here. This is a good starting point for many teams; it is easy to understand why and how your code is being executed. As your robot becomes more complex, your code needs to expand to handle interactions between different subsystems. Your team could quickly grow beyond what this framework provides, which means your team could be building on top of this framework, or you may switch to CommandBasedRobot.

The **IterativeRobot** framework is almost identical to the TimedRobot framework. The only difference is in how the timing of calls to the periodic methods is handled. With IterativeRobot the periodic methods get called each time a message is received from the Driver's Station. TimedRobot uses a very consistent trigger to ensure the period is consistent. The messages are normally received at a fairly regular interval but network delay's and loading on the Driver's Station laptop can result in some jitter in this interval. Unless you have a specific reason for using IterativeRobot you should use TimedRobot instead.

The **CommandBasedRobot** framework moves most of your code into small classes that define behaviours for your robot. These behaviours then get executed based on changes in the phases of the game and/or as a result of user input and sensor changes. This framework allows you to group behaviours together and to specify how the behaviours work together. Although there is more to learn related to this framework, once you understand how the pieces fit together you can write relatively easy bits of code to support a complex and extremely functional robot. By having several small classes, it can also be easier to separate your tasks for multiple people to work on the code. You can find more information about CommandBased programming here.

9.1.5 Plan Your Code

Your robot is not built by dumping a bunch of parts in a pile then adding bolts and zip ties to hold it all together. Don't let your code be built that way either. Your robot normally has some subsystems or groups of parts that are designed to work together to perform a specific function. Your code should be designed to mimic this organization.

If your robot has a base that drives around, create a class for the base. WPILib almost does all of this work for you, just create a simple wrapper for what they have to make things your own.

If your robot has an intake mechanism your code should have an intake class to hold all of the code related to that mechanism. The class should have methods to perform the different functions associated with the mechanism like stowIntake(), deployIntake(), intakeForward(), intakeReverse(), etc.

Although using the CommandBased robot framework pushes you towards having organized code, using it does not guarantee good code and it is not the only way to have well organized code. You can create classes and tie them together using any of the provided frameworks.

If you get to writing your code and you catch yourself copying a section of code from one place to another, stop and think: should this be a function that just gets called from both places? Chances are high that it should be. It may even mean creating a new class to hold that function. Then when you have to change or fix that section of code, you only need to do it in one place. In the short term, creating classes and functions is a little more work, but having code that is organized will help you in the future.

9.2 Configuring Hardware

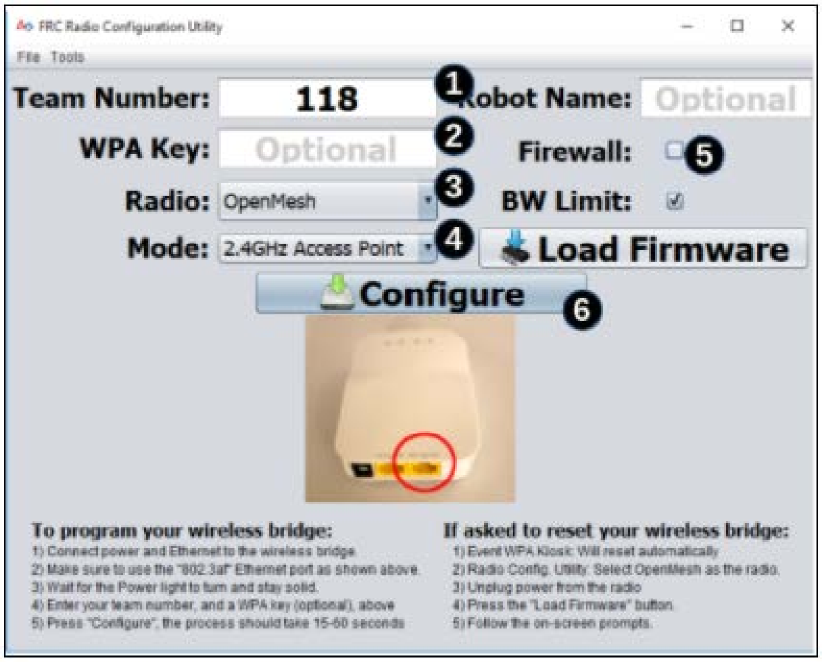
Before your robot can operate properly, teams must configure several pieces of hardware including the robot radio, roboRIO, and motor controllers. It is highly recommended that most of these configurations get done pre-season as they can be tedious and time consuming.

9.2.1 Robot Radio

When using the FRC competition radios, it is best to configure them with the FRC Radio Configuration Utility. Be patient when configuring the Open Mesh radios, it takes about 30 seconds for them to boot. Watch the lights, they will come on, flash for a little while, then they all go off and it starts over. Don't start configuring until the network light starts blinking the second time.

The utility provides several options to support different use cases. Here are three we use.

For Most Team Workspaces and Demonstrations
In most cases, you will want your laptop to connect directly to your robot so the radio on your robot will need to be configured as an access point, depending on your laptop you may want to configure it as a 2.4GHz Access Point or as a 5GHz Access Point, most laptops will support 2.4GHz, if your laptop supports 5GHz you will probably have less interference from other equipment using the 5GHz option.



Team Number: 118
Robot Name: Optional
WPA Key: Optional
Firewall: ☐
Radio: OpenMesh
BW Limit: ☑
Mode: 2.4GHz Access Point

To program your wireless bridge:
1) Connect power and Ethernet to the wireless bridge
2) Make sure to use the "802.3af" Ethernet port as shown above.
3) Wait for the Power light to turn and stay solid.
4) Enter your team number, and a WPA key (optional), above
5) Press "Configure", the process should take 15-60 seconds

If asked to reset your wireless bridge:
1) Event WPA Kiosk: Will reset automatically
2) Radio Config. Utility: Select OpenMesh as the radio.
3) Unplug power from the radio
4) Press the "Load Firmware" button
5) Follow the on-screen prompts.

Figure 1 - Radio Configuration.

After connecting directly to the radio via ethernet and waiting for the radio to power on,

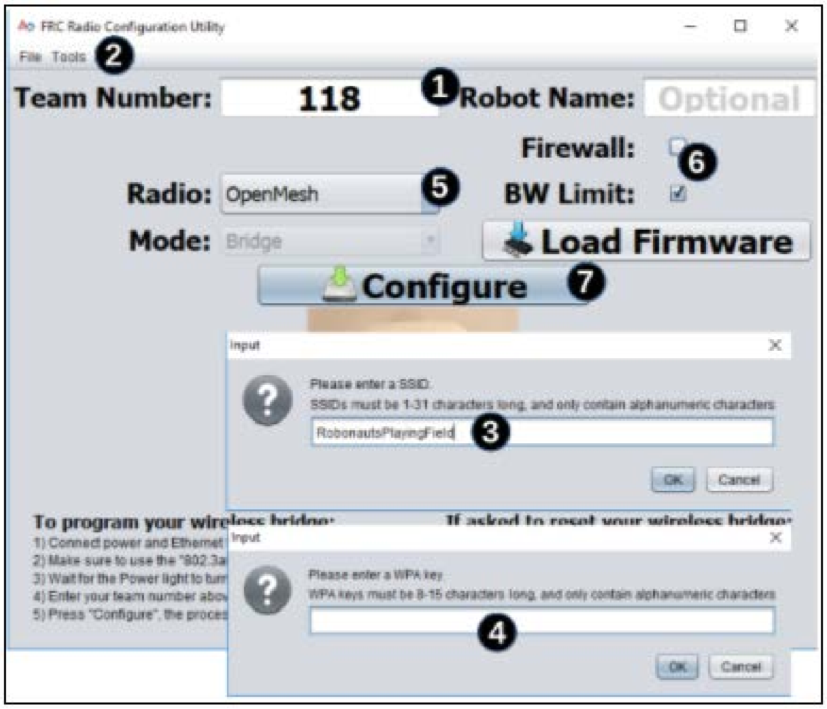
1. Enter your team number
2. You can enter an encryption key but it's not needed in most cases, if you do, remember it.
3. Select the type of radio
4. Select the Access Point frequency -- it depends on your laptop
5. Make sure the Firewall is not checked, make sure BW Limit is not checked
6. Click on the Configure button and follow the on screen instructions. (If the radio configuration fails, a common fix is to temporarily disable any other network interfaces, such as wifi, that you might have running.)

At Competitions

At each competition, you will need to use the Radio Configuration Kiosk set-up by the event. When running in this mode, the Radio Configuration Utility will configure each team's radio to act as a bridge with a unique encryption key for that team and competition. The tool is very similar but many of the options will be disabled.

In Team Workspaces with Dedicated Field Networks

Many shops have an extremely large number of wireless networks in the area, so teams set up a field network similar to those at competitions. The access point used provides a more consistent connection and data throughput than connecting directly to the robot. Using a field network requires a slightly different radio configuration.



Team Number: 118
Robot Name: Optional
Firewall: ☐
BW Limit: ✓
Radio: OpenMesh
Mode: Bridge

Input
Please enter a SSID
SSIDs must be 1-31 characters long, and only contain alphanumeric characters
RobonautsPlayingField

To program your wireless bridge:
1) Connect power and Ethernet
2) Make sure to use the "802.3a
3) Wait for the Power light to turn
4) Enter your team number abov
5) Press "Configure", the proces

If asked to reset your wireless bridge:

Input
Please enter a WPA key
WPA keys must be 8-15 characters long, and only contain alphanumeric characters

After connecting your laptop directly to the radio and waiting for the radio to power on,

1. Enter your team number
2. From the Tools Menu, select FMS Offseason Mode
3. Enter the fields SSID (for example "118HomeNetwork") and click OK
4. You can enter an encryption key but it's not needed, click OK
5. Select the type of radio
6. Make sure the Firewall is not checked
   Make sure BW Limit is not checked
7. Click on the Configure button and follow the on screen instructions.

9.2.2 RoboRIO

The roboRIO is a computer designed for use on robots. It has several Reconfigurable Input and Output (RIO) ports and several communication ports. The roboRIO runs a customized real-time Linux operating system. Updates to the operating system are released as image files that can be installed using the roboRIO Imaging tool (or NI MAX). The imaging tool can also be used to set a team number on the roboRIO.

Image the roboRIO
To image a roboRIO, you should connect to the roboRIO with a USB cable plugged into the roboRIO USB device (USB-B) port. It is safest to disconnect your laptop from all other networks (including wireless networks) when imaging a roboRIO. After connected, make sure the roboRIO is powered on and start the roboRIO Imaging tool. It should automatically scan and locate your roboRIO, if not you can click on the rescan button. When your roboRIO is found:

1. Make sure the roboRIO is selected
2. Enter your team number
3. Check the Console Out box
4. Uncheck the Disable RT Startup App box
5. Check the Format Target box
6. Select the desired (latest) image
7. Click Reformat to begin the imaging process

More detailed instructions can be found here.

Update the roboRIO Firmware
Although it is rare, it is sometimes necessary to update the firmware on your roboRIO. To do so, connect your laptop to the roboRIO USB device (USB-B) port and turn the roboRIO on.

1. Using a web browser, go to http://172.22.11.2.
2. Click Login. Enter "admin" in the Username field and leave the Password field blank.
3. Click Update Firmware
4. Browse to locate and select the appropriate (latest) firmware, it is normally at c:/Program Files (x86)/National Instruments/Shared/Firmware/cRIO/76F2/
5. Click Begin Update
6. Wait for the firmware installation to complete

More detailed instructions can be found here.

9.2.3 Other Hardware

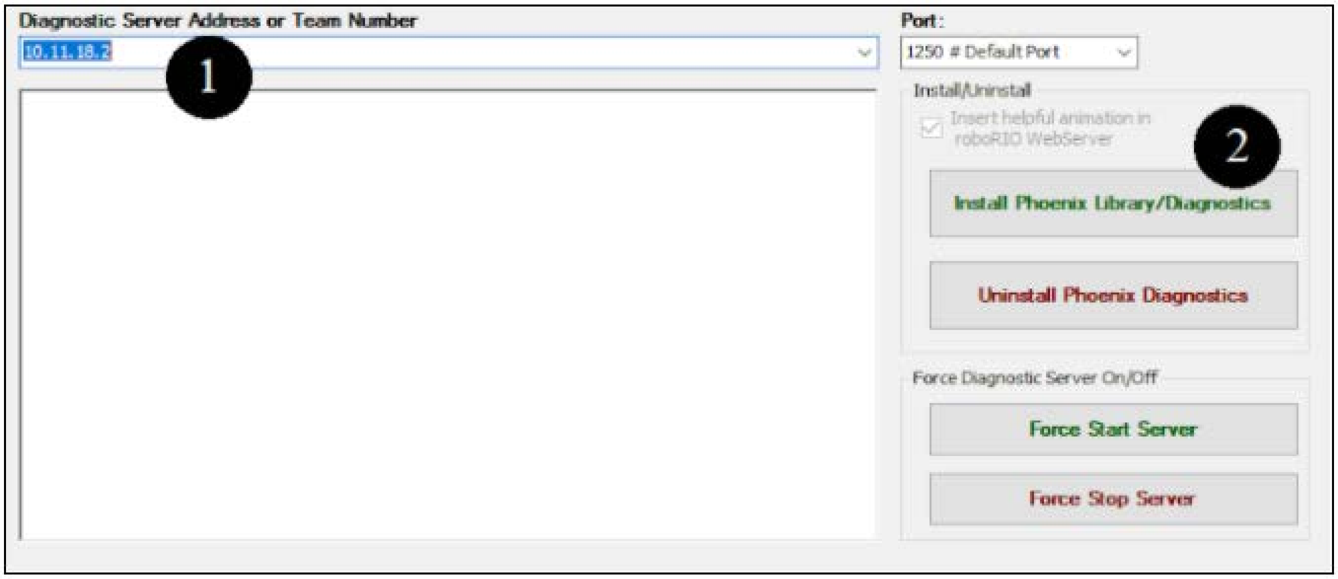
Configure CAN Devices

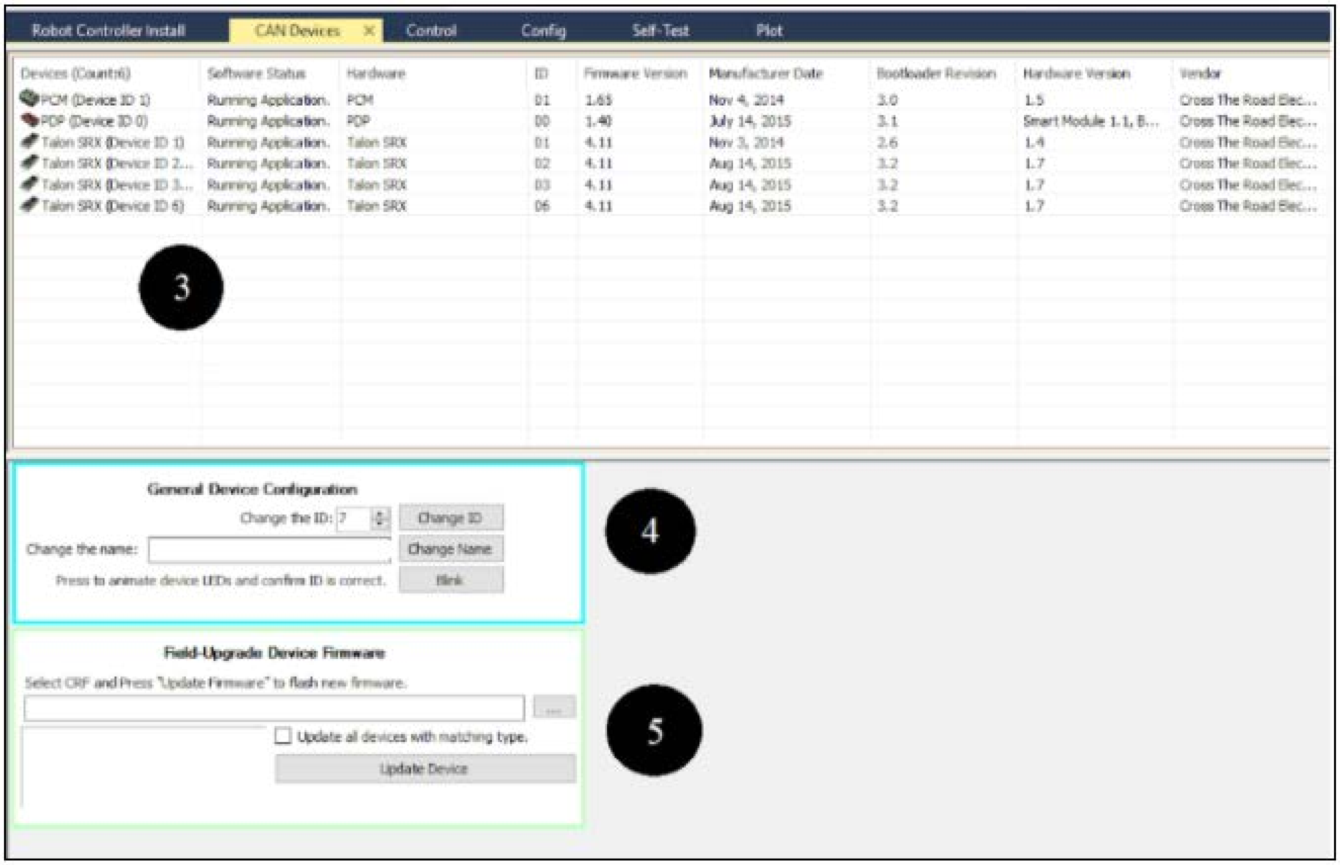
CAN (Controller Area Network) is a commonly-used communication standard in FRC. CAN devices are all connected along a “bus” made of (typically) green and yellow wires, and communicate with each other based on individually-assigned IDs. There are several CAN devices used on FRC robots, including the Power Distribution Panel (PDP), the Pneumatic Control Module (PCM) and the TalonSRX speed controller. In the past, these devices were configured using the web-based roboRIO dashboard; however, as of 2019, separate programs must be downloaded in order to configure CAN devices. Generally, any device manufactured by Cross The Road Electronics (CTRE) will be configured using Phoenix Lifeboat, while the Spark MAX from REV Robotics uses a separate program called the REV Spark Max Client.

Phoenix Tuner

Configuration tools for CAN devices manufactured by CTRE must be installed using the Phoenix Tuner software. This link provides detailed instructions on how to use the Phoenix Tuner to install the web based configuration tools.

When updating firmware, CTRE has all CAN devices in Phoenix Tuner**.** When configuring CAN devices, open Phoenix Tuner, type in 172.22.11.2 if connected through USB or your Static IP if through ethernet (static ips are explained further down in the guide).


Diagnostic Server Address or Team Number
10.11.18.2
1
Port:
1250 # Default Port
Install/Uninstall
Insert helpful animation in roboRIO WebServer
2
Install Phoenix Library/Diagnostics
Uninstall Phoenix Diagnostics
Force Diagnostic Server On/Off
Force Start Server
Force Stop Server

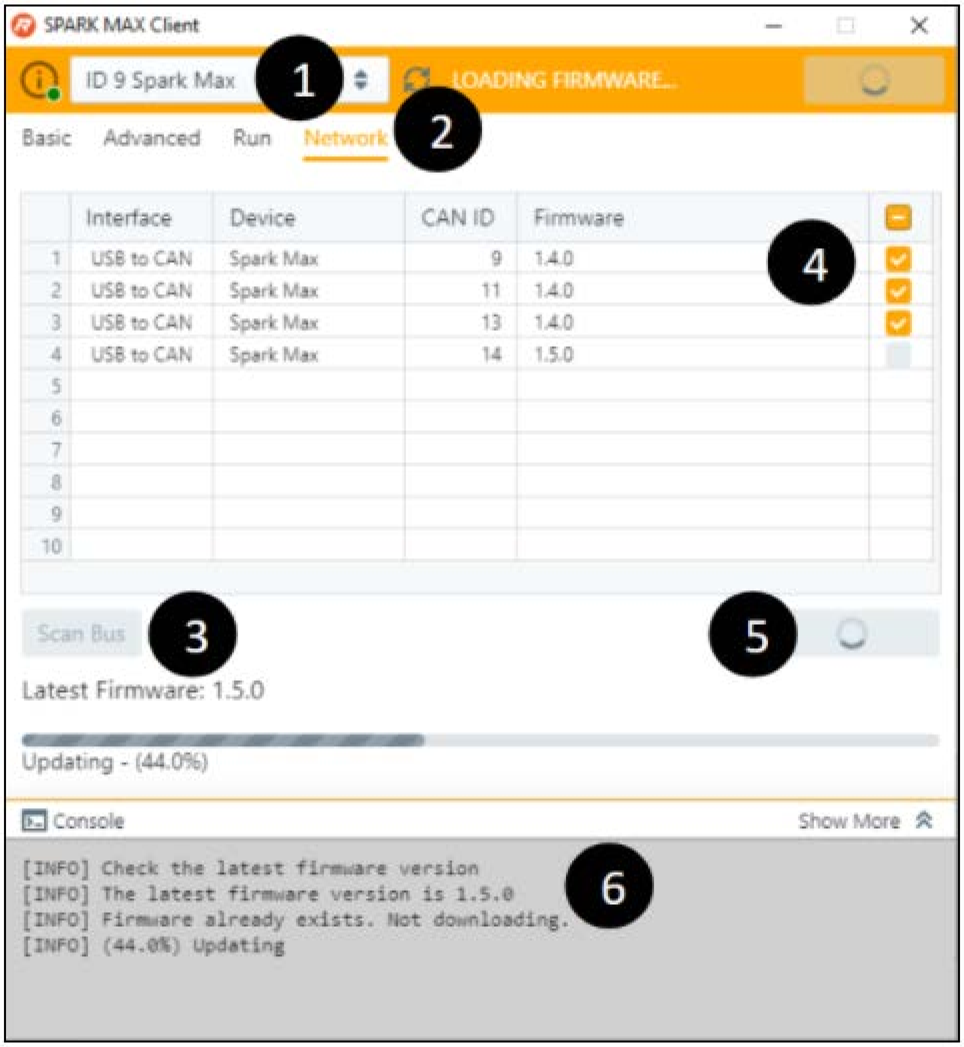

Robot Controller Install
CAN Devices
Control
Config
Self-Test
Plot
3
4
5
General Device Configuration
Change the ID: 7
Change ID
Change the name:
Change Name
Press to animate device LEDs and confirm ID is correct.
Blink
Field-Upgrade Device Firmware
Select CRF and Press "Update Firmware" to flash new firmware.
...
Update all devices with matching type.
Update Device

Devices (Count:6)	Software Status	Hardware	ID	Firmware Version	Manufacturer Date	Bootloader Revision	Hardware Version	Vendor
PCM (Device ID 1)	Running Application.	PCM	01	1.65	Nov 4, 2014	3.0	1.5	Cross The Road Elec...
PDP (Device ID 0)	Running Application.	PDP	00	1.40	July 14, 2015	3.1	Smart Module 1.1, B...	Cross The Road Elec...
Talon SRX (Device ID 1)	Running Application.	Talon SRX	01	4.11	Nov 3, 2014	2.6	1.4	Cross The Road Elec...
Talon SRX (Device ID 2...	Running Application.	Talon SRX	02	4.11	Aug 14, 2015	3.2	1.7	Cross The Road Elec...
Talon SRX (Device ID 3...	Running Application.	Talon SRX	03	4.11	Aug 14, 2015	3.2	1.7	Cross The Road Elec...
Talon SRX (Device ID 6)	Running Application.	Talon SRX	06	4.11	Aug 14, 2015	3.2	1.7	Cross The Road Elec...

1. Open Phoenix Tuner and type in either 172.22.11.2 if connected via USB or your Static IP 10.TE.AM.2 if connected through ethernet or wireless. Once you have connection, the bottom of Phoenix Tuner should say you are connected.
2. Install Current Libraries to your CAN devices by clicking Install Phoenix Library/ Diagnostics.
3. By clicking on the CAN Devices page, this is where you can see all your current CAN Devices, their Device ID, and their Firmware Version.
4. If you ever need to change a device's ID or name, you can do these in the General Device Configuration.
5. To upgrade firmware, add a device to the Field-Upgrade Device Firmware section.

REV Spark Max Client

The Spark Max Client can be installed from https://www.revrobotics.com/sparkmax-software/.



	Interface	Device	CAN ID	Firmware	
1	USB to CAN	Spark Max	9	1.4.0	✓
2	USB to CAN	Spark Max	11	1.4.0	✓
3	USB to CAN	Spark Max	13	1.4.0	✓
4	USB to CAN	Spark Max	14	1.5.0	
5					
6					
7					
8					
9					
10					

```
[INFO] Check the latest firmware version
[INFO] The latest firmware version is 1.5.0
[INFO] Firmware already exists. Not downloading.
[INFO] (44.0%) Updating
```

1. Select a specific Spark Max for more detailed information about the firmware version.
2. Switch to the networking tab for firmware options.
3. Scan the CAN Bus for all the available Spark Maxes.
4. Select all of the Spark Maxes needed to be updated.
5. Press the Load Firmware Button (greyed out when currently uploading firmware).
6. Monitor current status of firmware update.

The software will complain if any of the devices have a CAN ID set to 0. If you are getting errors, make sure all your IDs are 1 or higher.

9.3 Development Tools

9.3.1 Git and Version Management

Version management is an integral part of software development done in any field. There are any of a variety of services that enable version management and can bring this capability to a team. However, the most popular version management system within FRC is Git. As a result, this section will primarily discuss version management within the scope of Git, although some of the same concepts will carry over to alternate version management systems. This section will not describe how to use Git explicitly but focus on high-level ideas for the version management infrastructure of a team. The tutorial section will have links to guides that can help if a team has never used Git before and needs to learn how to make use of it.

9.3.1.1 Setting up a Repository

The first step to making use of version management is to set up a repository in a way that allows for efficient use throughout the season and afterward as a resource to consult during future seasons. The team should be an organization within whatever version management software your team has decided to use. (GitHub is one of the most popular although many others are available, as well.) After this, the team can create a new repository each season so there will be a record of the software used by your team to look back to and reference. All the members of your software development team can then be added as contributors and you can even define different users' permissions within the repository to ensure that destructive actions are restricted to student leaders and mentors for example, while still allowing everyone to contribute. The process of correctly setting up a repository will ensure that all the software developed throughout a season is backed-up in a safe location in case of an issue with a team laptop or other development device which would otherwise cause the loss of the work you had done up to that point in a season.

If you lack internet connection, or the internet connection available blocks github, it is possible to have a local git repository. You would execute git commands via a command line interface, and just not do the git push/git pull commands to the remote repository (i.e. github).

9.3.1.2 Branches for Version Control

One of the most important tools for version management is the ability to create separate development areas within one repository to allow for rapid development while not risking the stability of the current state of the software. A team should have a stable main branch that contains the software used the last time everything was working as intended. This main branch should then be accompanied by several development branches that are used to develop new or improved functionality to the robot. These development branches, once proved stable and reliable, can then be remerged into the main branch and continue the development cycle. While there are many potential branches a team could make use of, there are several which are generally very helpful to FRC teams. One of these is a general practice branch which would be used for the general changes to be made during a meeting of the team and then merged if proven stable throughout the meeting. Assuming your team has a practice robot or mechanism, this example can be expanded to different branches for different programming subteams so that multiple parts of the robot can be developed simultaneously without interfering with the others. Another useful category of branches is competition branches, which are made as copies of the main branch just before the competition. Although software changes are generally something many teams avoid at competitions, there are rare circumstances where changes are needed for whatever reason. These branches can be used to make such small changes as well, and then ultimately be merged back into the main branch. Additionally, these branches will save your code to the way it was at the last competition, allowing a major point where you can reference if something were to go wrong with the software, or simply as a way to analyze the progress you made between competitions in the offseason to see how you could improve the team in upcoming years.

9.3.1.3 Planning with Issues and Milestones

Another feature that helps with organization and distribution of tasks is milestones and issues. Git allows for the creation of milestones which are a defined point that your team is aiming to meet by a certain deadline. These can be created per competition or more often if there are major deadlines that have been defined within your team. Each milestone includes several issues that would define different desired improvements to your software before that point. These issues can then be closed as the features are added, which allows for a great system to monitor the progress of your software development. This can also help develop better planning and ideas of what can be accomplished before the next competition by seeing how long similar tasks have taken you in the past.

9.3.2 Putty

One of the more advanced tools available to you is Putty. Putty has a multitude of tools and protocols built into it, such as FTP, SCP, SSH, Telnet, rlogin, and raw socket connection. You can use it to directly manipulate files on the RoboRIO, which opens up a world of possibility around programming your robot. WPILib has a great resource on using Putty to SSH into your RoboRIO.

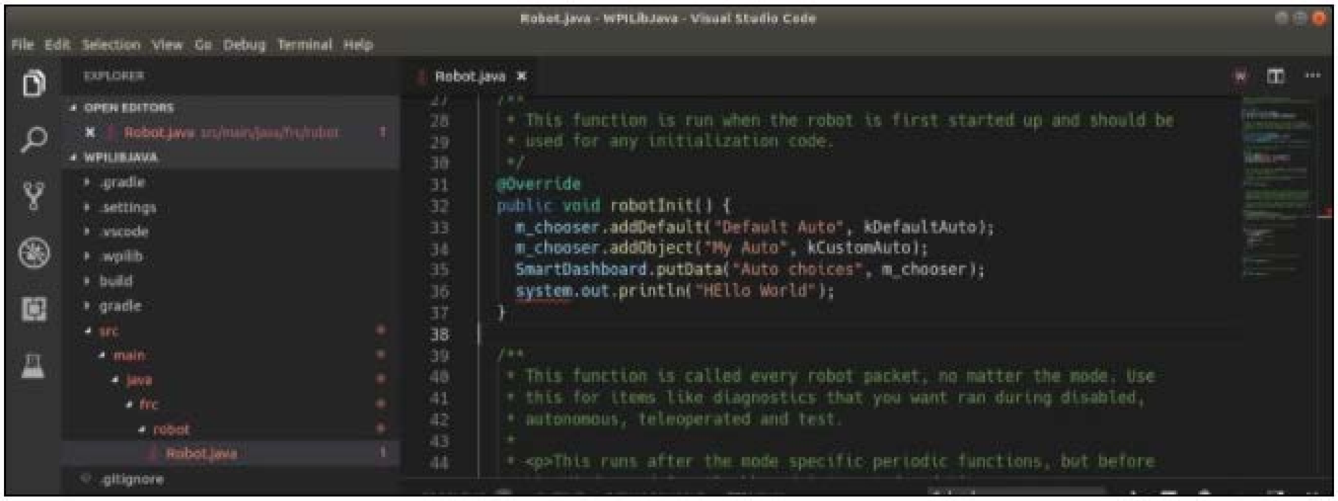
Putty can be used to solve a multitude of issues! It can be used to check if certain files failed to download to your robot (an uncommon, but hard to debug, issue), to check the integrity of the RoboRIOs filesystem (sometimes necessary to determine if a RoboRIO is bricked), or to aid in the process of recovering a damaged RoboRIO. Putty is not something you will always use, but it can be a lifesaver in certain circumstances, and it's a great tool to have! Putty variants are available on Windows, Mac, and Linux.

9.3.3 Winscp

Winscp is another file transfer and file manipulation application, like Putty. Unlike Putty, Winscp is only natively supported on Windows, but is a great alternative to Putty, with a similar repertoire of tools! Learn more about Winscp here.

9.3.4 Integrated Development Environment

An Integrated Development Environment, or IDE, is the software which programmers use to write their code. Most IDEs have basic features such as a source code editor, build automation/build tools, and a console. Some IDEs have additional features such as interfacing with git, collaboration tools, etc. IDEs can boost programmers' productivity with real-time feedback, as most programmer's wouldn't want to wait until compile time to notice they are missing a necessary semicolon.


Robot.java - WPILibJava - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS
Robot.java src/main/java/frc/robot
WPILIBJAVA
.gradle
.settings
.vscode
.wpilib
build
gradle
src
main
java
frc
robot
Robot.java
.gitignore
Robot.java

```java
  /**
   * This function is run when the robot is first started up and should be
   * used for any initialization code.
   */
  @Override
  public void robotInit() {
    m_chooser.addDefault("Default Auto", kDefaultAuto);
    m_chooser.addObject("My Auto", kCustomAuto);
    SmartDashboard.putData("Auto choices", m_chooser);
    system.out.println("HEllo World");
  }

  /**
   * This function is called every robot packet, no matter the mode. Use
   * this for items like diagnostics that you want ran during disabled,
   * autonomous, teleoperated and test.
   *
   * <p>This runs after the mode specific periodic functions, but before
```

Example of an IDE (Visual Studio Code).

The selection of an IDE is based on team/personal choice preferences, history, technical depth, etc. Within FRC, the officially supported IDE is Visual Studio Code for C++/Java development. For teams just getting started or without significant corporate knowledge in software development, this is the recommended approach as there is support available online during the build season and in person at events. If a team wants to explore IDE options, below is a list to get started.

Java, C++, and Python IDEs:

- Eclipse: Was the FRC officially supported IDE prior to 2019. Is still a great option for teams who are familiar with the tools and features of Eclipse.
- IntelliJ: Debatably has more refined and easy-to-use features than some of the other options; However, it isn't as simple as Visual Studio Code.
- Pycharm (python only): A specialized python IDE that isn't as heavy as Eclipse and other IDEs.

LabVIEW IDE:

- NI LabVIEW: Different from other languages in that it only lives within the NI LabVIEW IDE, rather than having several IDEs to choose from.

C++/Java development does not require an IDE. Files may be edited using a preferred text editor. The source may then be compiled and deployed using command prompt or terminals.

9.4 Safety

Safety while programming is very important. A runaway robot or a mechanism that is accidentally commanded can cause serious injuries. Please be safe, rather than sorry! Put the robot on blocks, even if you do not think the robot is going to move. Clear people away from the robot, even if you think nothing is going to move. Wear safety glasses - your eyes are precious. Be safe!

Make sure people know when the robot is enabled. Of course, the RSL flashes when the robot is enabled, but not everyone knows that, and you have to be looking to see it, and the people building the robot sometimes don't even consider it a priority to put on. Many teams give a shout when enabling the robot so that anyone not looking knows that they should *start* looking in case the robot does something dangerous (and/or cool). It should also go without saying that anyone in the vicinity of an enabled robot should absolutely be wearing safety goggles. Even in years where the robots *aren't* explicitly designed to launch projectiles, tiny robot bits can break off and fly an impressive distance.

Don't leave the robot enabled when not in use. If someone incidentally nudges the controller (or if it falls off the table), it can lead to disastrous consequences for anyone working on or near the robot. In fact, just turn it off if you can. It's even safer, and it saves the battery anyway. *Definitely* never knowingly work on an enabled robot.

Check what mode the robot is in before enabling it. It's a lot easier than you might think to turn the robot on for some teleoperated testing and watch it autonomously drive itself off the table. Have fun explaining *that* one to mechanical.

Have someone ready to disable or E-Stop the robot. If the robot is enabled, pressing Enter on the computer running the Driver Station will immediately disable it, and pressing Space will trigger an emergency stop, regardless of if the Driver Station window is focused. Usually the disable is sufficient; a full emergency stop requires the roboRIO to be physically reset or power cycled before it can be reenabled. Having someone ready to press Enter key if something goes wrong is always a good idea.

Always test the robot "on blocks" before putting it on the ground. Make sure the drivetrain works properly by raising up the chassis such that the wheels aren't contacting anything, preventing the robot from moving around, before enabling the robot. It's a lot safer to test the drivetrain when the robot isn't going to careen off in some random direction (or off a table) if there are any problems. Some teams have dedicated testing jigs for this, but a couple of pieces of aluminum tubing or wood blocks work just as well.

Be mindful of the full range of motion possible by the robot, particularly on the first startup in a while (for example, after another group was working with the robot). While the new code you are testing might be for a flywheel, it is possible someone accidentally changed a physical configuration (for example, plugging a motor in to the wrong port), a software configuration of the robot (for example, a CAN ID changed), or a fault in the software, resulting in a climber or arm unexpectedly extending.

9.5 Driving and Troubleshooting Tools

An important milestone to programming a robot is to learn how to set it up and begin commanding and controlling it.

9.5.1 Drivers' Station

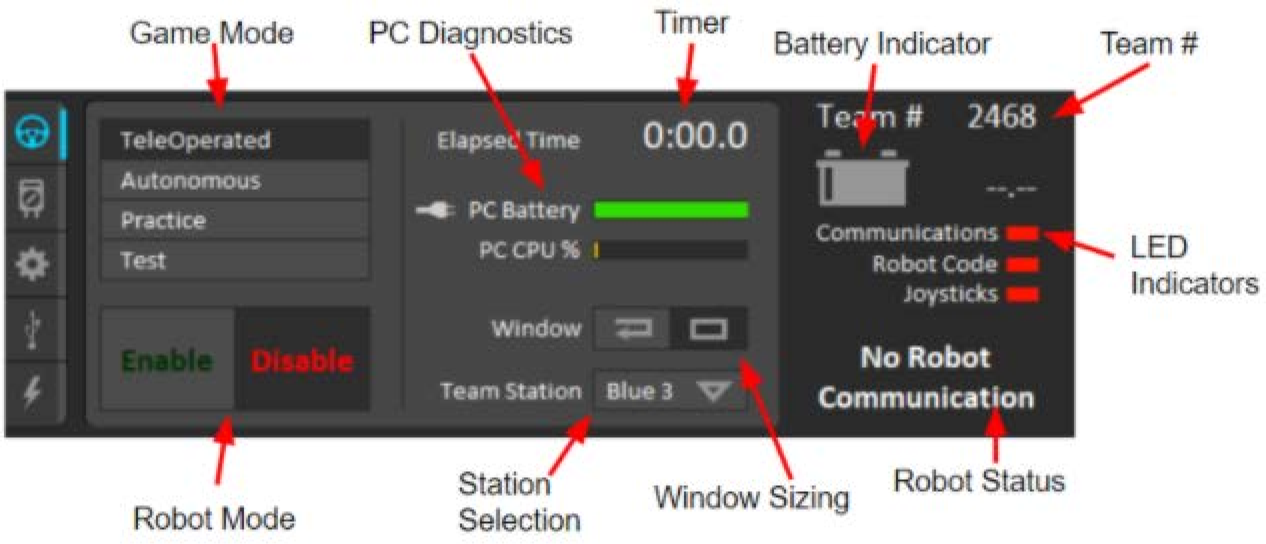
The Driver's Station is the primary interface which you control your robot through. In this section, we will be going over how to use the Driver Station as well as it's features. To download the driver's station, follow the instructions here.

Driver Station Best Practices: (Inspired by a CD thread by MrRoboSteve)

1. Pick out one or two laptops to be the official driver stations. By designating one or two laptops as the driver stations, you make it easier to maintain software versioning of 3rd party libraries, and if you have two, you have a backup in case something unfortunate happens to the first driver station.
   Here are the ideal specs for the driver stations:
   - 2G of RAM
   - 13in+ screen: since your driver station will also be home to your dashboard, it is helpful for the drivers to be able to see the screen clearly from a slight distance. An alternative to this is to have an external monitor, however, that may be unwieldy.
   - Built in ethernet port and at least two USB ports: the driver station will need to be connected to the FMS (Field Management System) through ethernet every match and it's not convenient to have a converter (as it also takes up an USB port). Most teams use at least two controllers/button pads when driving a robot.
   - Windows 10 Operating System - has less bugs than Windows 7.
   - Make sure your Windows is up to date; Be careful to only run updates on the driver station when you have time to fully test the robot after the updates are complete. It is **highly recommended** that at competitions teams turn off Windows auto-updates, if possible. If turned off, Windows updates should be turned back on after the competition; otherwise, it is recommended to use the "pause updates for 7 days" option. While unlikely, it is possible (and has occurred in the past) for Windows updates to run while setting up for or during a match.
   - Remove as many unneeded programs as possible, including games, 3rd party antivirus softwares, etc.

- Ensure the laptop can hold an hour or more of charge while in use. At the competition, always monitor the laptop battery and make sure to charge it whenever the opportunity arises.
- Turn off all firewalls.
- Ensure that everyone knows the password to the driver station.
- Avoid straining the ports on the driver station, whether that is the ethernet tugging on the port or USBs yanking.
- Before a match, check that your controller inputs are in the correct order.
- Have hook and loop tape on the bottom of the driver station laptop so you can fasten it to your cart as well as the driver's box.

Tab 1: Driver's Tab



Tab 1: Operation.

Game Mode: The four game modes that the driver station allows you to select are TeleOperated, Autonomous, Practice, and Test. The TeleOperated allows you to run the TeleOp code exclusively and is useful for testing and doing driver practice. The Autonomous option allows you to run your auto code, and will not allow your controllers to control the robot (unless you specifically have code to change that circa 2019). The Practice mode allows you to run an entire match (i.e. autonomous, teleOp, and endgame), with all of the sound effects and modes. The practice mode allows you to practice an entire match and is really helpful for practicing the transitions between modes. You can configure the time that each mode lasts in the third tab (settings tab). Lastly, the test mode is a really flexible mode allowing teams to write diagnostics whether autonomous or controller based.

PC Diagnostics: Two bar graphs displaying the host computer's battery level as well as CPU%. These graphs can be really helpful for reminding users of when to charge the PC and a good debugging tool for when things are lagging (CPU% too high).

Timer: When running the teleop, auto, and test modes, the timer will display the time elapsed. When running the practice mode, the timer will indicate the amount of time left in the match, similar to a real match.

Battery Indicator: One of the most used and crucial indicators on the driver station, the battery indicator gives users the robot's current battery level.

Team #: The team number is important to make sure the connection with the robot is successful. You can change the team number on the third tab of the driver station (the settings tab).

LED Indicators: The LED indicators are very useful in the setup and debugging processes. The first LED shows whether you have robot comms. The second light indicates whether or not there is active robot code on the RoboRIO. In order to enable the robot, the first two lights have to be green. The last light shows the user whether a controller is connected to the PC. More detailed configurations for controllers can be found on tab four.

Robot Mode: The robot mode allows you to enable and disable your robot. Some helpful keyboard shortcuts for changing the robot mode:

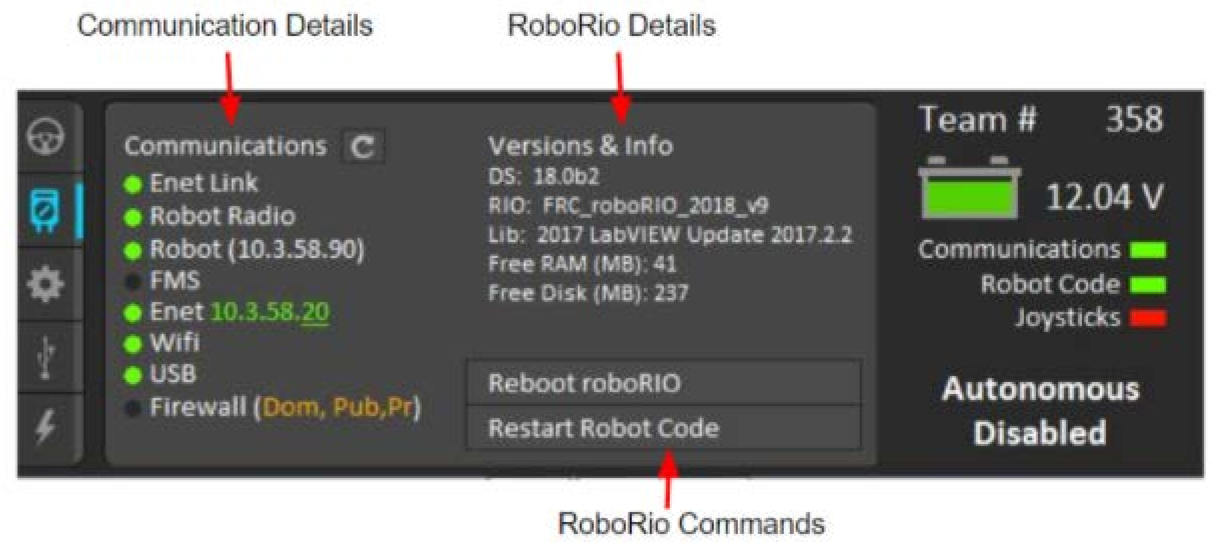
- "[ ] \" - Enabling the robot. The three keys above the enter key.
- *Enter Key* - Disabling the robot.
- *Space Bar* - E-stopping the robot. Requires a full restart in order to re-enable the robot.

Station Selection: Allows you to tell your code which FRC field driver station your team is located in for that match. In an official FRC competition match, this is handled by the FMS and won't be viewable.

Window Sizing: When running the driver station, you might notice that it stretches the entire length of the screen. You can change this by using the window sizing setting to minimize the window. While this option isn't extremely helpful for minimizing the driver's station (since there's a button for that on the DS), it can be useful for minimizing dashboards (that don't have a button built in).

Robot Status: Shows the current robot status such as No Robot Communication, Robot Enabled, Robot Disabled, and Watchdog Not Fed.

Tab 2: Diagnostics



Communications
● Enet Link
● Robot Radio
● Robot (10.3.58.90)
● FMS
● Enet 10.3.58.20
● Wifi
● USB
● Firewall (Dom, Pub,Pr)

Versions & Info
DS: 18.0b2
RIO: FRC_roboRIO_2018_v9
Lib: 2017 LabVIEW Update 2017.2.2
Free RAM (MB): 41
Free Disk (MB): 237

Reboot roboRIO
Restart Robot Code

Team # 358
12.04 V
Communications 🟩
Robot Code 🟩
Joysticks 🟥
Autonomous
Disabled

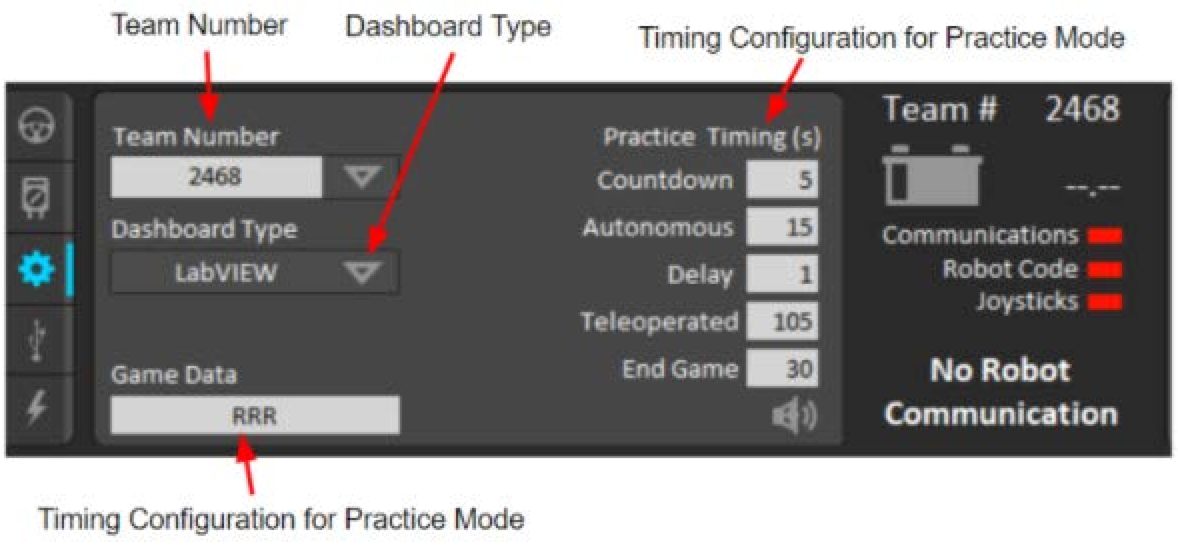
Tab 2: Diagnostics.
(Photo from Team 358's website)

Communication Details: This section helps teams debug and track their communication status with the roborio. The IP address 10.Te.am.2 means that the robot is connected over wifi/ethernet while the IP address 172.22.11.2 means that the robot is connected over USB. Additionally, the Firewall indicator lets you know if your host computer has firewalls enabled. If you have your firewalls on, it can hinder your ability to connect to your robot. Lastly, the refresh button next to Communications can help you connect to your robot if you're having issues.

RoboRIO Details: This section includes roboRIO details such as driver station version, firmware version, etc. During inspection, the inspectors will ask to see this section so make sure to have it handy.

RoboRIO Commands: The two roboRIO commands that are available in the driver's station are Reboot RoboRIO and Restart Robot Code. The Reboot RoboRIO command is equivalent to programmatically power cycling the robot and takes about a minute and a half to complete. Rebooting the RoboRIO is a convenient way to "power cycle" the robot without having to stand up and flip the switch. The restart robot code command reboots your code and takes anywhere from 20-40 seconds to complete depending on your code. Restarting the code can be super helpful when running autos (one-time executions) and is faster than redeploying the code.

Tab 3: Setup


Team Number
Dashboard Type
Timing Configuration for Practice Mode
Timing Configuration for Practice Mode

Team Number: 2468
Dashboard Type: LabVIEW
Game Data: RRR

Practice Timing (s)
Countdown 5
Autonomous 15
Delay 1
Teleoperated 105
End Game 30

Team # 2468
--.--
Communications
Robot Code
Joysticks
No Robot Communication

Tab 3: Setup.

Team Number: This box is for configuring your team number so that the driver station knows how to connect to your robot (assuming your radio/roboRIO are configured correctly).

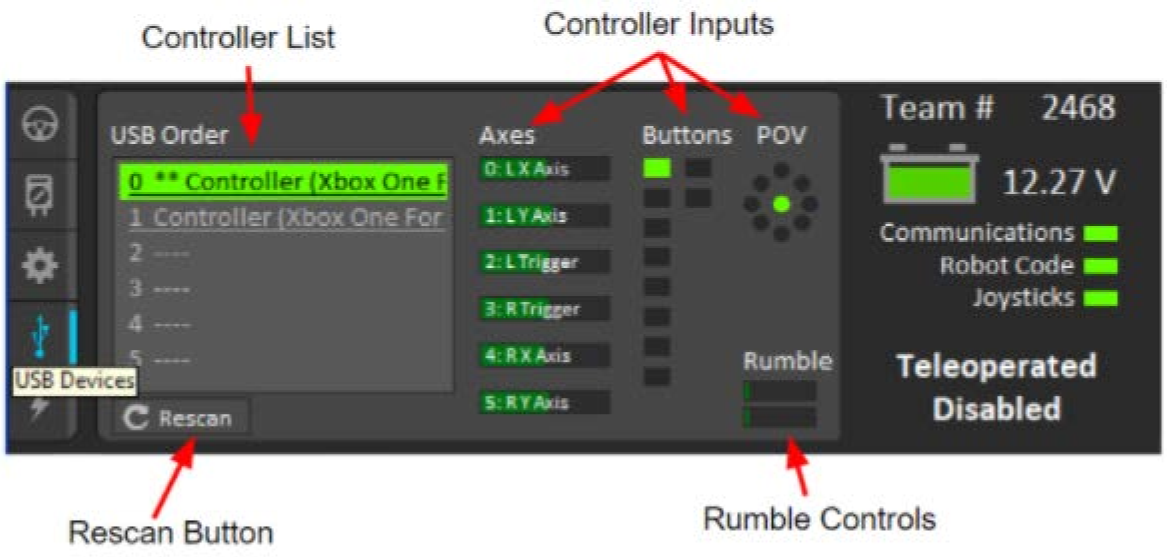
Dashboard Type:

- *Default:* This pulls up the default dashboard and doesn't require running any code. This is helpful for pulling up camera feeds quickly and seeing joystick movement.
- *LabVIEW:* Allows you to run your own custom LabVIEW dashboard.
- *SmartDashboard:* SmartDashboard is a java program that has widgets that you can use to customize your dashboard. Easier than making a custom dashboard using LabVIEW or ShuffleBoard.
- *ShuffleBoard:* Designed for C++ and Java. Fundamentally very similar to SmartDashboard but includes a range of features that SmartDashboard doesn't have.
- *Remote:* The remote option allows you to send the Dashboard data to another remote machine. When selecting this option, it will ask you for an IP address.

Timing Configuration for Practice Mode: This section allows you to change the timing for the practice mode. Perhaps you want the delay between auto and teleOp to be slightly bigger at first to help your drivers practice picking up their controllers. This is the place for that. Additionally, you can choose to either have sound or no sound.

Game Data: For games like Power Up in 2018, the switch and scales changed colors randomly every match. Due to this, robots had to be sent game data in match as to which color the scale and switches changed to. When not at a tournament, the game data box is how to tell your robot those configurations.

Tab 4: USB Devices



Tab 4: USB Devices.

Controller List: The controller list displays all of the controllers that are currently plugged in. In order for your controls to work correctly, the controllers must be in the correct order. In order to change the order of the controllers, simply drag the controller to the right place. A quick way to tell if your controller is in the right place is, after ensuring your robot is disabled or off, to push a button which will make the position light up green as depicted in the picture above.

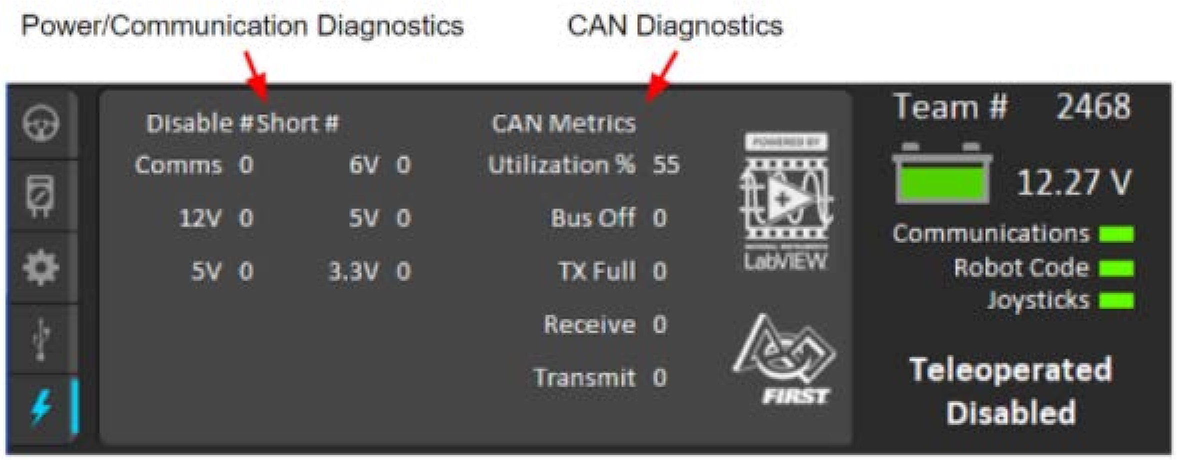
Controller Inputs: To check the controller inputs, click on the controller and that will pull up a section displaying all of the controller inputs. When writing code, the driver station is a quick and easy way to map buttons to value. For example, when mapping the “X button” on the xbox controller, one can connect it to the driver station and see the 0th value of the buttons section light up, meaning that the button X maps to 0.

- *Axes:* The value of axes are doubles ranging between -1 and 1, or 0 and 1. On a standard xbox controller the axis are the x and y values on the joysticks and the triggers.
- *Buttons:* The value of buttons are booleans (true or false).
- *POV:* The POV is unique in that it maps to a 360 scale. Clockwise, the values on the POV starting from the top are 0, 45, 90, 135, 180, 225, 270, and 315.

Rescan Button: The rescan button can be very helpful when your controllers are not connecting to the driver station immediately. Rescan can greatly speed up the controller connection process.

Rumble: Rumble is a feature that exists in some xbox controllers to provide driver feedback. Inside the controller, there is a motor attached to an unbalanced weight that spins when rumble is activated, causing the controller to vibrate. Unlike the inputs, rumble is actually an output. By dragging around the sliders, you can command the left and right sides of your controllers to vibrate.

Tab 5: Power and CAN



Disable # Short #

Comms 0 6V 0

12V 0 5V 0

5V 0 3.3V 0

CAN Metrics

Utilization % 55

Bus Off 0

TX Full 0

Receive 0

Transmit 0

Team # 2468

12.27 V

Communications 🟩

Robot Code 🟩

Joysticks 🟩

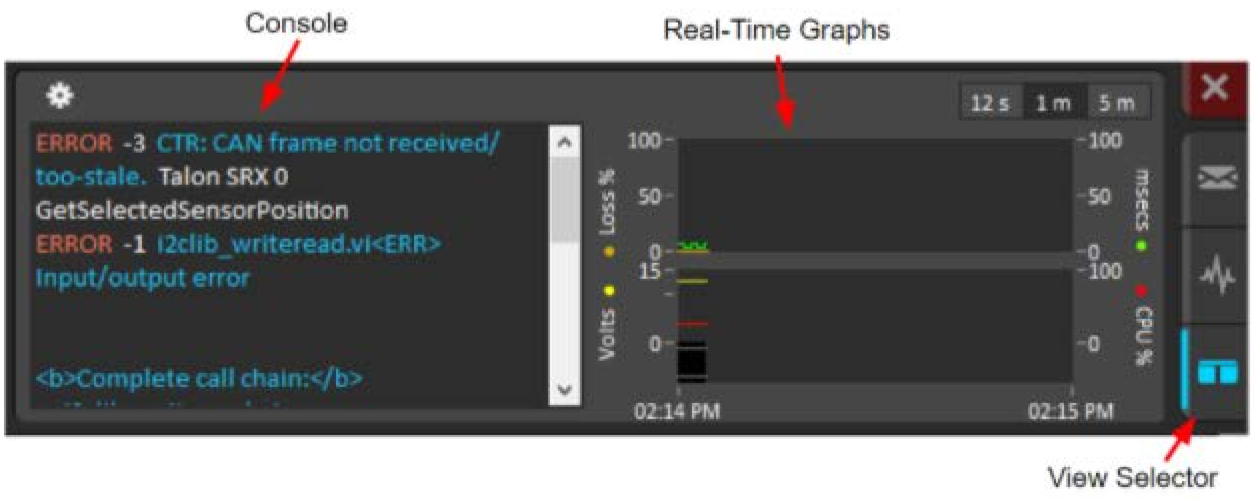
Teleoperated

Disabled

Tab 5: Power and CAN.

Power/Communication Diagnostics: The power/communication diagnostics are located on the right side of this tab and display the number of different faults that have occurred since the start of your code. They include communication faults, 12V faults (usually brownouts, or when your robot's battery dips below the acceptable voltage range), and 3.3V/5V/6V faults (commonly caused by a short circuit).

CAN Diagnostics: The CAN diagnostics display shows the can bus utilization percentage (how much of the total bandwidth is being currently used). This section also counts how many times various of faults have happened since the start of the code including Bus Off Faults (when the controller stops transmitting CAN temporarily), TX Full (lost transmissions from the roboRIO to CAN devices), and Receive/Transmit (short for receive error counter and transmit error counter).



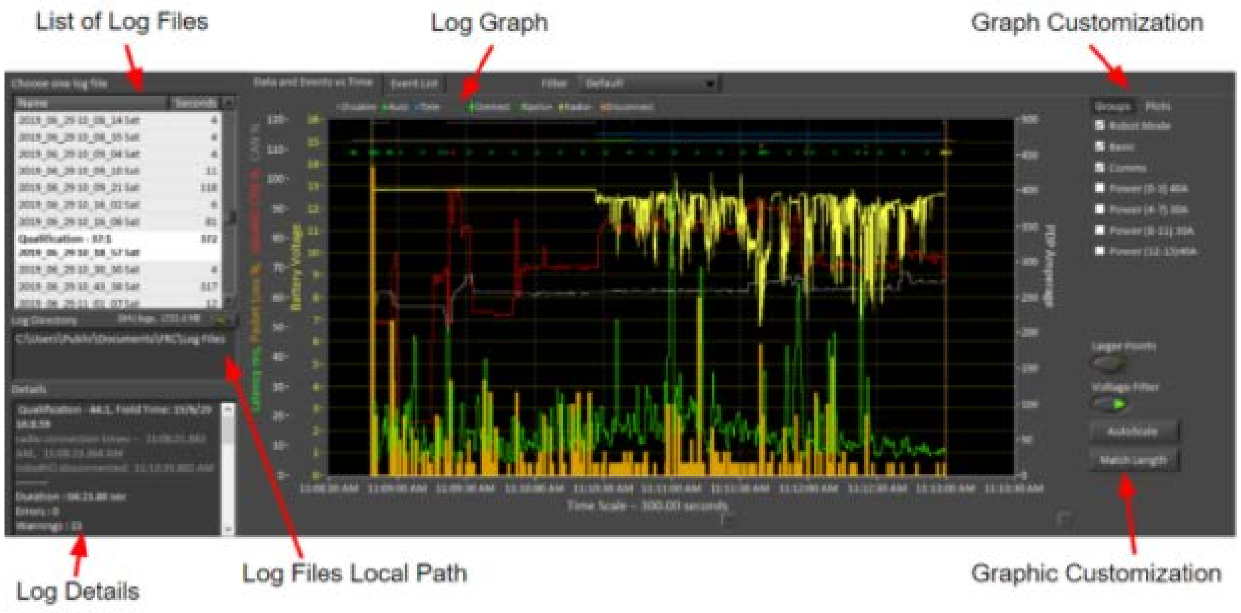
```
ERROR -3 CTR: CAN frame not received/
too-stale. Talon SRX 0
GetSelectedSensorPosition
ERROR -1 i2clib_writeread.vi<ERR>
Input/output error


<b>Complete call chain:</b>
```

Diagnostic Tools: Console and Real-Time Graphs.

Console: Displays current errors and states. The information displayed on the console can be configured by the settings button located on the top right. Additionally, in the settings button, there's an option to pull up a full screen console as well as clear the current console.

Real-Time Graphs: Shows the current state of robot connection, roboRIO CPU%, and robot power. This graph can be configured to be longer/shorter using the selector in the top right hand corner. Monitoring the CPU% can be very helpful for knowing when your robot code is going to boot up.

View Selector: Allows one to fill the entire space with the console, the real time graphs, or split screen the console and real-time graphs.



Diagnostic Tool: View Log Files.

The log files can be accessed from the dropdown menu from the settings button in the driver station.

List of Log Files: Log files are recording data every time the driver station is open and code is running. This list of log files allows you to select log files to look closer at. They are characterized by the date and the length of activity. Additionally, to make diagnostics easier during tournaments, matches are characterized by the Qualification number of the match.

Log Graph: The log graphs contain important details regarding robot performance such as brownouts, robot state (i.e. disabled, teleop, autonomous), cpu %, CAN %, robot voltage, robot connection latency, and packet loss %. The key for the values on this graph can be found surrounding the graph, color-coded. The x-axis of the graph can be zoomed in on by dragging the mouse over the graph and highlighting a section. The y axis can be modified by editing the numbers on the y-axis. Once zoomed in, a scroll bar will appear allowing full views of the graph. The values appearing on the graph can also be modified to include more or less information, which will be covered more in depth in the Graph Customization section.

Graph Customization: In the graph customization section, you can change the values that appear in the graph. This is very useful when looking for some values but not others. Another game-changer is that you can plot specific PDP slots, determining how much amperage specific motors are drawing when in use.

Log Details: The log details contain a summary of the entire log. The details contain the duration, number of errors, warnings, messages, brownouts, and the amount of energy each pdp slot takes up. This section can be used to gauge the overall performance of a robot during a time.

Log Files Local Path: This box shows you where the actual log files are located on the local driver station's disk. You can directly open the folder using the small open icon on the top right of this section. By accessing this folder, you can share your log files with others, and view other's log files.

Graphics Customization: The graphics customization allows you to read the graph better and includes options such as larger points, voltage filter, autoscale, and match length.

Event List: This tab in the log file shows all of the events that have occurred during the duration of code. For example, if a controller disconnects unexpectedly, the event list will display the exact time the controller was unplugged.

Event Filter: The event filter includes three options: default, all events and time, and all events and all info. This allows more or less information to be shown in the event list.

DSLOG Reader 2 is an unofficial driver station log viewer that has some useful capabilities not included in the default log viewer that's a part of the driver station. Details can be found here.

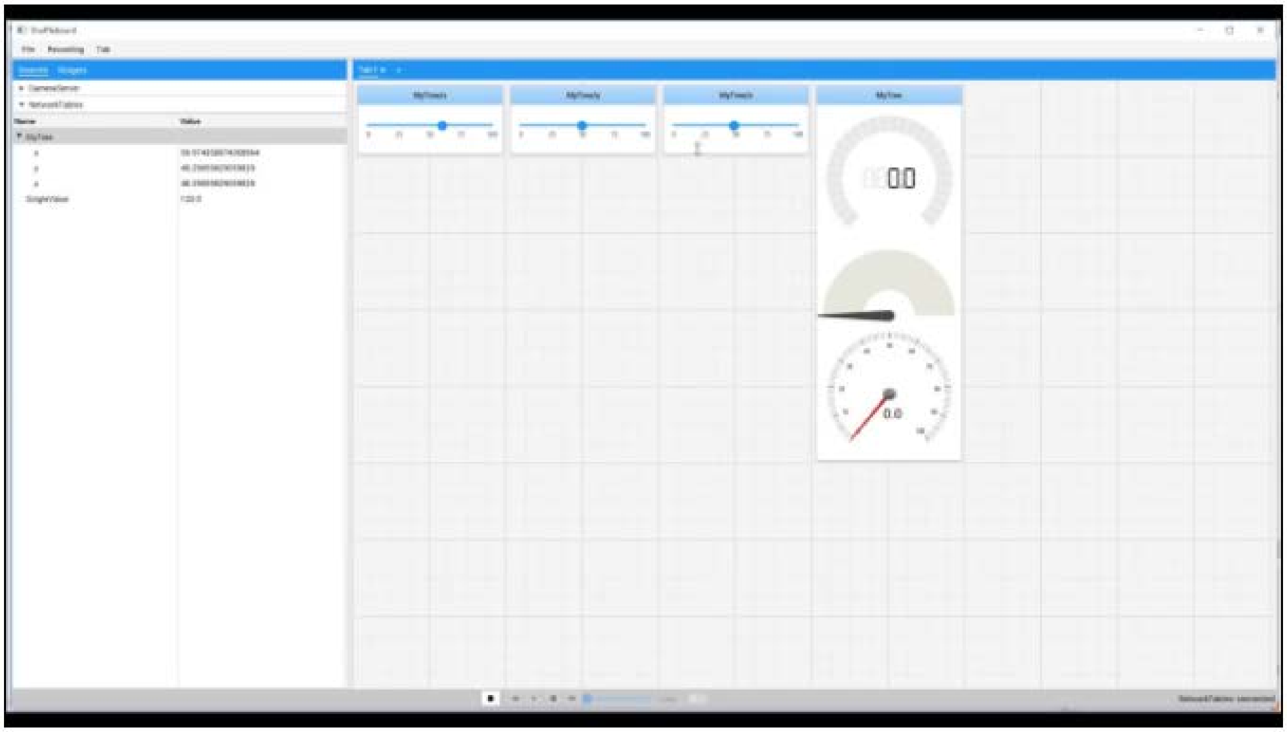
9.5.2 Telemetry and Driver Feedback

It's often said that the three main parts of a computer program are input, processing, and output. In FRC, the input is mostly from the joystick and sensors and the output is mostly from actuators. However, other forms of input and output are often useful as well. Many teams have a "robot dashboard" that can display information from and send configuration to the robot while it's running. This is useful both for driving and testing/debugging; for example, the driver might want to choose an autonomous before the match and see a camera's view, while PID tuning (see section 9.7.1) is much easier when you can monitor the sensor readings and update the constants on the fly.

Depending on your language, different options are available for creating a dashboard.

In LabVIEW, this can be accomplished via the LabVIEW dashboard, which is discussed in further detail in section 9.5.3.

For other languages, WPILib provides a program called Shuffleboard that serves the same purpose. Shuffleboard is sometimes buggy, but it allows the robot and operator to read and write values to a graphical interface without writing any networking or rendering code. Simple calls from the robot code - for example, `SmartDashboard.putNumber("Encoder Speed", encoder.get());` - create configurable "widgets" on the dashboard that get updated every time new data is pushed. The legacy `SmartDashboard` API works well enough for simple readouts, but the newer `Shuffleboard` API allows for direct, programmatic control over widget layout and configuration. The WPILib documentation has more information about both. Also see section 9.5.3.



Shuffleboard.

While information can be useful, not all of it is always necessary. For example, the driver isn't going to be tuning a PID loop during a tense match. In order to get the most use out of your dashboard, it's important to ensure that the most vital information is presented in a way that the driver can quickly perceive. Removing stuff like PID tuning information can make space on the screen for a larger camera view, which is far more useful. Shuffleboard allows you to create different tabs with different widgets, so you could have a tab for the driver and then various tabs for debugging information. It's also a lot easier to quickly make sense of graphics than numbers; try replacing a numerical readout with a gauge or meter, especially one with important values marked on it, or even a simple on/off display if the value has a single critical threshold that the driver needs to be aware of.

It’s a different story when debugging or testing at the shop: in that case, you do often want to see the raw numbers, in order to create the nice visuals for the driver in the first place. However, it can also be useful to create graphs that track values over time, something that both LabVIEW and Shuffleboard allow you to do, in order to visualize how quickly the values change, how much overshoot there is, etc.

9.5.3 Dashboard Guide

The dashboard is a method by which teams provide feedback to their drivers and programmers on the robot’s status. Things commonly found on dashboards include camera streams, autonomous selections, status of different areas on the robot, sensor readings, amount of time left in a match, etc. For example, in 2019, it might’ve been helpful for the drivers to know whether or not the robot was holding a piece of cargo or a hatch panel.

It is important that the dashboard is on a large display and is not cluttered so that drivers can get the information that they need in a glance. Before making the dashboard, it is important to discuss with your drivers to understand what features and information they would be interested in receiving during a match. An example of a custom feature that 2468 drivers requested in 2020 was that when it was time to climb (<20 seconds left in the match) the dashboard borders would turn red and flash at them. The other, less important information can be placed in other tabs (i.e. autonomous selection can happen in another tab).

The dashboard receives data from the robot through a method called network tables. If you want to hand information to the dashboard, you need to do it in your robot code by giving the network table a name and a value. Then, in your dashboard code, you can retrieve that information using the name that you put in the robot code. Section 9.9.1 discusses network tables more in depth.

There are several options when selecting a dashboard:

- **Default Dashboard:** Can be pulled up from the driver station. The pros to the default dashboard is that teams don’t have to create their own dashboard which can be very helpful during the hectic season. The cons are that it’s unmodifiable, which means that you can add more features or take away unnecessary information. The default dashboard can also be used as a quick way to access the camera stream.
- **LabVIEW Dashboard:** The LabVIEW Dashboard provides the default dashboard as a starting point but allows users to modify it to their team's necessities. The dashboard has a slightly different learning curve than learning how to program the robot however,

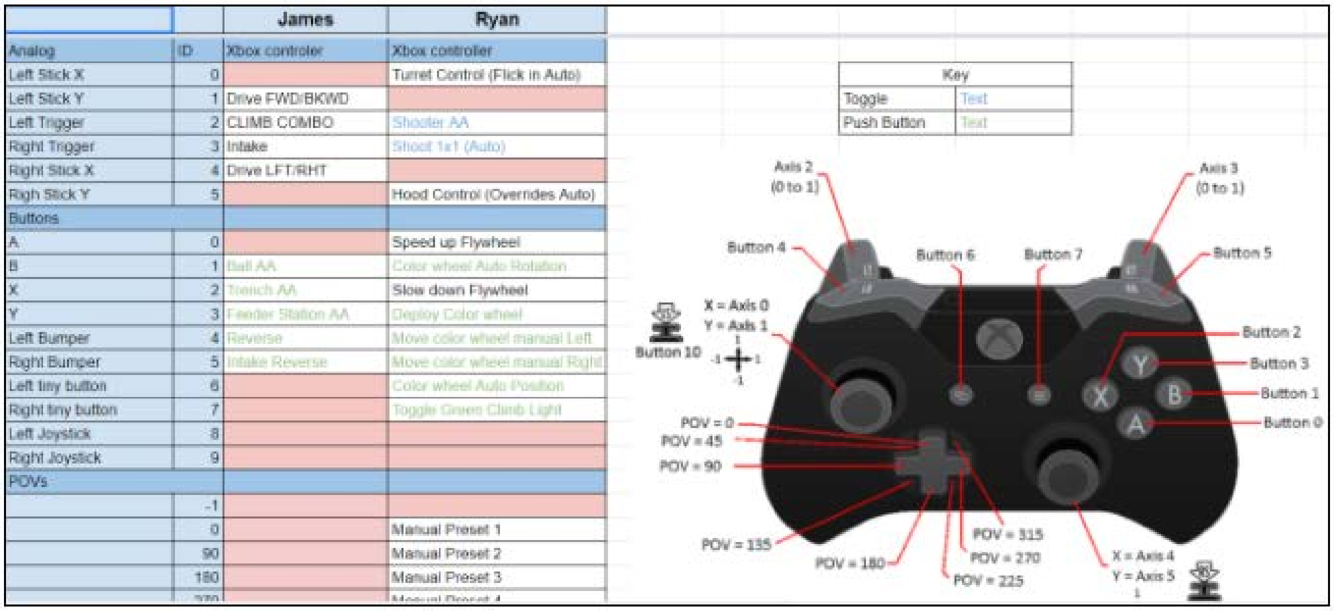
which teams need to keep in mind. Instead of launching the dashboard from the driver station (like the default dashboard), users must run the dashboard from the project.

- **SmartDashboard:** The SmartDashboard is a java program that can display different information through various widgets. The SmartDashboard however, is a bit outdated as most FRC teams have now transferred over to ShuffleBoard. A detailed tutorial for the SmartDashbaord can be found here.
- **ShuffleBoard:** ShuffleBoard is a newer addition to the FRC world and is already widely adopted by teams. Shuffleboard can be programmed in both Java and C++.
  A quick overview of Shuffleboard can be found here: Tour of Shuffleboard - FRC documentation.

9.5.4 Controller Mapping

Before writing the teleop code to command the robot, it is always good practice to do a controller map with your drivers. This allows for a reduced amount of controls changes and confusion. Then, store the agreed upon values in some sort of document that everyone has access to. Some best tips and best practices for choosing controls include:

- Keep it simple! The simpler it is, the faster your drivers will get comfortable with it, which is crucial.
- Always talk through decisions with the drivers. Let them know what it would entail from a programming perspective and what the alternative would be if the code wasn't ready by competition.
- Remember that you can always do combinations with the button (i.e. if the driver pushes the trigger first before pushing A, the robot will strafe right). This allows for more commands on a limited scope of buttons. However, too many combinations can get overwhelming for the driver.
- A controller isn't always going to be the best option. Many teams end up using touch pads for their operators when the number of commands needed exceed the number of buttons on a standard controller.



		James	Ryan
Analog	ID	Xbox controler	Xbox controller
Left Stick X	0		Turret Control (Flick in Auto)
Left Stick Y	1	Drive FWD/BKWD	
Left Trigger	2	CLIMB COMBO	Shooter AA
Right Trigger	3	Intake	Shoot 1x1 (Auto)
Right Stick X	4	Drive LFT/RHT	
Righ Stick Y	5		Hood Control (Overrides Auto)
Buttons			
A	0		Speed up Flywheel
B	1	Ball AA	Color wheel Auto Rotation
X	2	Trench AA	Slow down Flywheel
Y	3	Feeder Station AA	Deploy Color wheel
Left Bumper	4	Reverse	Move color wheel manual Left
Right Bumper	5	Intake Reverse	Move color wheel manual Right
Left tiny button	6		Color wheel Auto Position
Right tiny button	7		Toggle Green Climb Light
Left Joystick	8		
Right Joystick	9		
POVs			
	-1		
	0		Manual Preset 1
	90		Manual Preset 2
	180		Manual Preset 3
	270		Manual Preset 4

Key	
Toggle	Text
Push Button	Text

FRC 2468's Controller Map Document for 2020.

9.6 Robot Commanding Schemes

A crucial part of competing with a robot is being able to command it. There are various techniques typically used in FRC for commanding various parts of the robot.

9.6.1 Tank Drive vs Arcade Drive

Two of the most common and basic drive schemes in FRC are "tank" and "arcade" drive. Both are designed for tank-style drivetrains, such as the one on the Kit of Parts chassis. With this kind of chassis, the robot moves and turns by rotating the wheels on either side with different amounts of power and potentially in different directions. Each control scheme requires a different approach from the driver in controlling this movement.

Tank controls are the more direct approach to controlling a tank drive. Inputs from the driver are used directly to set the power of each side of the drivetrain. Typically, on an Xbox-style gamepad, the left stick's Y axis controls the left side of the drivetrain and the right stick's Y axis controls the right side. This means that going straight forwards or backwards requires pushing both sticks up or down, while turning requires pushing the sticks with different magnitudes or directions, depending on how quickly you want to turn. This can be hard to get accustomed to if you are new to it, but some people prefer it.

Arcade controls are generally more intuitive and easier for new drivers. Instead of using the inputs directly, a single “movement” input and a single “turning” input are used to calculate the speed to run each side of the drivetrain at. On an Xbox-style gamepad, the “movement” input is usually the Y axis of the stick for the driver’s dominant hand, and the “turning” input is usually the X axis of the other stick. (If you have drivers of different handedness, you might want to add a handedness toggle to your dashboard.) This is sometimes described as being similar to first-person shooter games, where you also move with one stick and turn with the other. You don’t have to think about how to achieve the desired movement; you simply indicate it with the sticks and the robot does the math.

WPILib provides implementations of both of these control styles through the `DifferentialDrive` class. Simply pass the motor controllers for each side of the drivetrain to the constructor - if you have multiple motor controllers per side, you can use the `SpeedControllerGroup` class to group them together - and use the `tankDrive` or `arcadeDrive` method, as appropriate. One note about WPILib’s implementation of both is that, by default, they apply a “signed square” function to their parameters (squaring them while retaining the sign), which generally makes the robot easier to handle with joysticks. You can turn off this behavior if you want, but it’s recommended to leave on for human control.

9.6.2 Swerve Drive

Of the various drivetrains used in FRC, swerve is by and large the most complex. This is true on both the mechanical and software side. Consider that to even do something as basic as “drive forward,” the robot has to be able to make sure all of its wheels are facing the right way! To do a swerve drive and do it well means putting a lot of effort into programming, tuning, and testing the mechanism, quite apart from all of the design, fabrication, and electrical work that needs to happen first. In addition to getting the swerve modules to behave as desired, your drivers need a considerable amount of drive time to utilize the full benefit of swerve drive. Resources for swerve can be found in section 9.8.1.

9.6.3 Presets vs Manual Control

There are some things that humans do better than computers. Precise, consistent operation is not one of them. When it comes to working a mechanism that needs to be in a particular place at a particular time - especially if that time is “as soon as possible without damaging anything” - automating the process is a good way of improving reliability and efficiency while reducing the amount of work the drivers need to do. Consider a mechanism like an elevator: you want to get it to the correct height as quickly as you can without overshooting, then back down again without breaking it. A skilled co-driver can get pretty good at manually operating the elevator with a joystick, but they can easily be outdone by a robot that’s been set up with a few buttons for preset elevator heights and a sensor-driven PID loop that optimizes its movement. The electrical chapters of this guide have more information about various sensors you can use, and section 9.5 goes into more detail about how to use these sensor readings to operate a mechanism.

9.7 Autonomous and Preset Pathing

When starting out many teams rely on open loops which lead to many more inaccuracies within a system and little to no automation. An example of this would be driving a base manually throughout the teleoperated period of a match. While this is much simpler to both build and program, as it does not require sensors, being able to automate a sequence will reduce the number of things a team’s drivers will need to remember to do during the high tension atmosphere of a match. It’s also required to successfully complete the autonomous period of the game. However, simply running a motor at a set power for a predetermined amount of time will prove inconsistent for a variety of reasons - such as varied battery levels and external influences from other robots/game elements. These issues can be overcome using a variety of methods each with their own advantages and disadvantages which will be discussed in the subsequent sections.

9.7.1 PID Control System

Control systems regulate the output of a system using control loops. Control engineering is frequently considered a stand-alone discipline A basic introduction is included here as within FRC, control systems are (almost exclusively) implemented in software.

The most basic control system is open loop, where sensor feedback from the system is not used to influence the input to the system. FRC examples for open loop control might be mapping a button to an intake roller command or using joysticks to drive a base. Open loop control can be more than simple mapping of inputs directly to outputs. System inputs may be shaped to influence system response, such as ramping the input to a motor on a button press, limiting the maximum value to a motor or shaping a joystick input to provide more finer control at partial stick deflection.

Feedback or closed loop control uses sensing of the system's state to ensure stability of the controlled system, control the dynamics and regulate the output of the system. The proportional/integral/derivative (PID) approach for feedback control is commonly used within FRC and presented here. There are many other approaches to control a system, including bang-bang, compensators, state-space, optimal, fuzzy logic and nonlinear control.

Bang-bang control is a type of control system that mechanically or electronically turns something on or off when a desired target (setpoint) has been reached. Bang-bang control loops are easy to implement and can be a great first step towards closed loop control. While you may not have as precise of control over a mechanism, there are times when it may be "good enough" and reduce software complications. Bang-bang works by running the motor at a set power (traditionally 100% power) until some limit is reached and then setting it to a different power (traditionally 0% power). For FRC, it's generally wise to consider set powers besides just 100% on or off. For example, a bang-bang control on a flywheel may spin the flywheel at 100% power until the desired RPMs are reached, then set the flywheel to a voltage of 8.0V to maintain a speed. If the RPMs get below the limit, then it is powered at 100% power again. Similarly for an arm or elevator, you might power it at 10V until it reaches a position based on sensor feedback, then power it only enough to offset the effects of gravity.

PID is an approach for performing feedback control to regulate the output of a system, with three parameters influencing the dynamics of the system: proportional, integral, and derivative. While it is helpful to understand these mathematical concepts, it is not required so long as the student understands how adjusting each of these parameters will impact the overall behavior of the machine - which will be covered in the following section. A benefit of PID is that it only needs to be set up once for each mechanism, even if the setpoint changes; as long as it's tuned properly, the target distance or velocity can be freely adjusted and still work. However, proper tuning can be difficult to achieve for new students until they gain some experience with this process.

Overview of PID Feedback Loops: The basic form for a PID control loop is the equation:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

where:

K_p is the proportional gain (a tuning parameter)

K_i is the integral gain (a tuning parameter)

K_d is the derivative gain (a tuning parameter)

$e(t)$ is the error. $e(t) = SP - PV(t)$

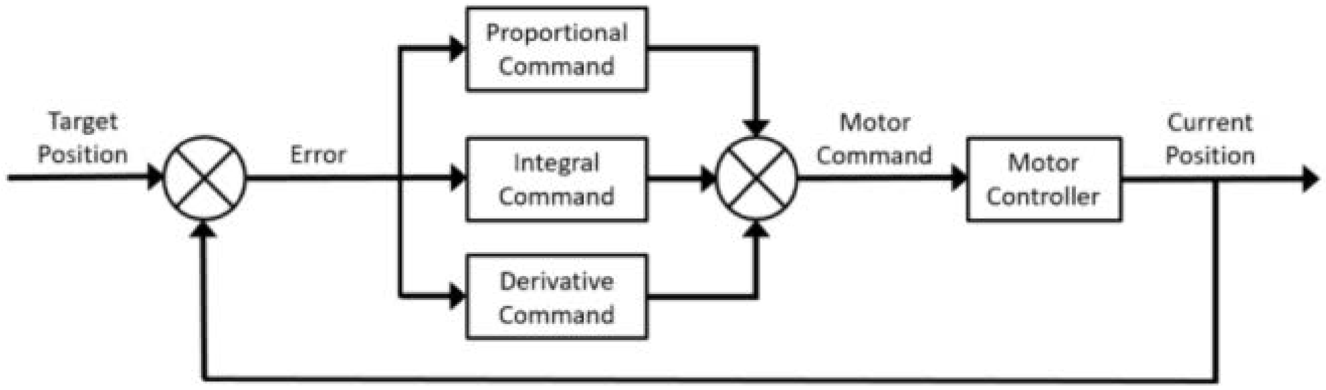
where SP is the setpoint and $PV(t)$ is the process variable.

t is the time (the present)

$\int_0^t e(t)dt$ is the accumulation of error over time

$\frac{de(t)}{dt}$ is the rate of change of error

In this equation, $e(t)$ is the current error of the system at time t, and $u(t)$ is the motor command to be output to the system. Throughout the process of adjusting or tuning the PID to a particular system the constant gains represented by K_p, K_i, and K_d are changed to adjust the impact these will each have on the motor output. It is also important to note that for some use cases setting one or multiple of these gains to 0 to disregard that segment of the feedback loop could in fact be beneficial by saving the team time (that would have been spent tuning) to work on other aspects. It is suggested to use the minimum set of parameters for the required application, generally starting with only K_p then adding in K_d, if needed. There are other elements to a full control system, such as feed-forward and integral zone values, but these are not covered in-depth here as they are more complicated and not always required to have a well running system.



PID Control Loop Flow Diagram.

Below is a pseudocode implementation of the equation and figure above.

The setpoint is the desired value of the controlled variable. Example of this might be arm height in centimeters, wheeled shooter speed in RPM or left side of drive train in ticks/second.

The process variable is the current measured value of the controlled variable. The setpoint and process variable must have the same units.

The output of the code is the value written to the motor controller, in units native to the motor controllers interface. In FRC, PWM motor controllers are encoded -1.0 - +1.0, but could be 0-255 (where 127 is neutral) or ticks or RPM or depending on the motor controller function.

The error is the difference of the setpoint and process variable.

The error delta is the difference between the error and previous error. Note that this is not a strict derivative of the error. A strict derivative would have units of process variable units / time (deg/second for example).

The error sum is the sum of all previous errors. Just like the delta error, this is not a strict integration of the error. A strict integration would would units of process variable * time (degree*seconds for example).

K_p is the proportional gain. Its units are output units per input unit. (For example: motor counts/degree of error).

K_i is the integral gain. Its units (in this example) are output units per input unit (motor counts/summed degrees of error. See comment about not using strict integration above.

K_d is the derivative gain. Its units (in this example) are output units per input unit (motor counts/delta degrees of error. See comment above about not using strict derivatives/integrals.

```
// calculate error related values
error = reference - measured
error_delta = error - error_prev
error_sum = error + error_sum

// determine out to motor
output = kp * error + ki * error_sum + kd * error_delta

// save error for next time through the loop
error_prev = error
```

Tuning: This can be one of the most difficult parts of using a PID control loop but as a team works with PIDs over the course of many years they will begin to become more efficient in this, such as a reused or similar drivetrain having similar values the next year. There are a variety of established tuning methods which can be used as a base to build off of, such as Ziegler-Nichols Tuning, however it is also helpful to understand the impact changing each gain will have so a team can easily adjust these values to achieve the desired behavior. It is important to make sure to start with all of the gains at 0 and then slowly tune them one at a time to ensure the safety of everyone working on the system and the robot itself.

Proportional gain (K_p) is the most basic of the three components to understand. It adjusts at what point the system begins to reduce output of the control system (which is also the input to the motor controller). The higher the K_p, the nearer to the target position the motor command begins to go below maximum command. Higher K_p values will eventually lead to control system oscillation or instability. One way of getting a starting point for this is to determine how many counts of the encoder, gyroscope, etc. away from the setpoint the control system should begin reducing command to the motor controller (and be conservative). The initial estimate of K_p might be the maximum motor controller output divided by the output reduction threshold. A simple tuning approach is to double the value of K_p until the mechanism oscillates, then start fine tuning.

The integral gain (K_i) is used to increase the motor output to “nudge” the system towards the target position. This capability is not always needed in a system. Because the integral gain multiplies by an integration (or summation) of errors, the integral contribution can grow very rapidly and should generally be much lower in magnitude compared to the other gain values.

A good starting point for an integral gain would be $K_i = \frac{K_p}{100}$. Tuning should be done for the integral gain by increasing it very slightly if the system is never reaching the target position completely and holds just a few counts off position - if it is more than this you most likely need a higher proportional gain. While the integral gain can drive out steady state errors in a system, it also can lead to oscillation.

The final component of this style of control loop is the derivative gain (K_d), which dampens the total response to prevent overshoot of the target position. If the derivative gain is higher, more dampening will be placed on the motor values, and the lower the derivative gain, the less dampening will be placed on the motors. This value should start at 0 and be raised if you see consistent overshoot in the system until it is no longer overshooting the target position. Cross the Road Electronics recommends using 10x the K_p gain as a starting point, should the derivative gain be needed. In applications where the measurement of the process variable is noisy (for example, when using analog sensors such as potentiometers), taking the derivative of a noisy signal can amplify the noise, leading to further oscillation.

Tuning is a complex part of software design and this guide contains only a simple description of one way of tuning a system. There are many resources which can provide more guidance on this matter including CTRE's motor controller closed loop docs which, although only provide code samples for CTRE products, explain universal tuning methods very well. For code samples of REV SPARK MAX motor controllers see the SPARK-MAX-Examples GitHub page.

Feedforward can be used to also help have a smoother operating mechanism. Feedforward is basically a "best guess" of what the output should be at a given set point. User JesseK has an example use case for feedforward here. The Phoenix documentation has information about determining arbitrary feedforward and velocity feedforward here.

9.7.2 Motion Profiling

Motion profiles are essentially the next step in a team's progression in software complexity following mastery of PID usage. Please note that this does not mean a team should use exclusively motion profiles over PID control; for many applications PIDs are still plenty adequate for the task. Motion profiles are a tool to use for precise tasks that require more fine control over the motion compared to PID sequences, but they require much more setup and tuning as well as requiring a separate set of defined variables for each path that will need to be executed. The primary producers of motor controllers for FRC have developed motion profile software for their controllers - Motion Magic for CTRE and Smart Motion for REV motor controllers - which is especially useful to teams when starting out with motion profiling.

When to Use Motion Profiles: Motion profiles, while being more precise than PIDs, are generally over complicated for most simple tasks and can often be skipped to save time while tuning and allow for more development of other systems and driver practice. However, there are some times when the increased control provided by the motion profile allows for significant increases in the reliability and speed of a task. As a general rule of thumb, if inaccuracies in the execution of a task would cause major loss of effectiveness - such as a precise climbing operation at the end of the game which could swing the score of a match - then a motion profile should be used. However, if such inaccuracies would not matter, such as for crossing a line during the autonomous, then a PID would likely be more than adequate. So, the natural next question is which tasks become too complex for a PID to execute effectively and warrant the use of a motion profile? The simple answer to this question is whenever you have a use case where you need to define target velocity and acceleration throughout the process in addition to a target position. Some examples of when you may want to add these elements and establish a motion profile are when doing any action which is integral to your success and has high variability with a simple PID (shooting mechanisms). Another good time to use a motion profile is if there is a particularly fragile sequence your robot must complete in which rapid acceleration or deceleration could damage the related mechanisms (climbing sequences). Of course there are other times where motion profiles are useful but they primarily fall into these two cases for getting started. If there is extra development time then a fully motion profiled machine would be great to develop but may be out of reach for many developing teams and even established ones. FRC Team 6377 has a presentation on how they utilized and configured their CTRE's motion profiling (motion magic) to control their arm in 2019.

Tuning: All the elements mentioned in PID tuning are still present in motion profiles with a few additional elements. These include integral zone, feed-forward, output range, velocity range, max rpm, and max acceleration. Most of these can be determined based on what those in charge of designing the mechanism determine the system can handle and setting the values respectively. However, the integral zone and feed-forward will have to be tuned similarly to the PID elements. Integral zone is essentially determining at what point the integral command becomes impactful on the behavior of the system. This allows you to have a more aggressive integral gain as it doesn't impact the behavior until the system is close to the target position. The integral zone should be slightly larger than the typical settling error of the system to ensure that the integral will impact the behavior as it is intended to once the system approaches the target position. The feed-forward is used to reach the target position faster by having a known approximate power which the system will run to until the system approaches the target position. This can be set by simply having the drivers complete an action and recording the power level reported by the driver station to get an idea of the range and then tuning slightly afterwards - increasing if there is significant undershoot and decreasing if there is overshoot.

9.8 Common Subsystems

Year after year, FRC games have shown to require a common set of subsystems required for teams to excel on the field. In order to expedite the process of programming these systems during the season, it's important to become familiar with how these systems work and how to approach programming them. In this section, we will discuss common subsystems, how to program them, and include resources for the programming process.

9.8.1 Drivetrain

For the drivebase, it is important to start moving quickly so that the design team can continue adding their mechanisms. Every language has example code to run a basic tank drive or arcade drive. We recommend using the example as a starting block, and adding more advanced features (such as odometry or autonomous modes) on top of the basic framework. If you are stuck, there are many resources available for you.

Arcade Drive Resources:

- Chief Delphi Thread (Java)
- Another CD Tread (Java)
- CD Thread (C++)
- WPILibC++ Arcade Drive (C++)
- CD Thread (LavVIEW)
- NI Forum Help (LabVIEW)

Tank Drive Resources:

- Tank Drive Tutorial (Java)
- Tank Drive Example and Walkthrough (C++)
- Tank Drive Tutorial (LabVIEW)

Swerve Drive Resources (challenging - for highly experienced teams):

- Intro to Swerve
- Swerve Drive Code Walkthrough (Java)
- CD Thread about Swerve (Java)
- Swerve Tutorial (C++)
- Swerve Drive Example (LabVIEW)
- Swerve Drive Specialities Example Swerve Project

9.8.2 Elevator or Lift

An elevator will be the most complicated device many teams programm, so we advise teams to start writing skeleton code for a lift as soon as they can. To program any system well, it is important to consider the driver. This applies doubly so to an elevator, because it can be extremely difficult to control if the software does not do enough to help the driver.
We recommend three steps to program an elevator.
1) Use an encoder to measure the position of the elevator.
2) We recommend creating a PID(F) loop. The F term (called the feed forward term) can be used to counteract gravity. See the PID section of this document for more information.
3) Finally, be sure to have safety interlocks (such as limit switches) so your robot is not a danger to itself and others if things go awry. Again, if you are stuck, there are many resources available for you. To find the latest, just search “frc elevator code <your_language_here>”.

9.8.3 Collector/Intake

The collector is often the easiest system to program, so we advise this be given to new members of the programming team as a way to build their skill and involve them in the process. Often, this will just be a simple motor, or perhaps a state machine if there are different positions for the armature. For most teams, this will be as simple as hooking a trigger input into a motor output, with perhaps a button for reverse. Again, if you are stuck, there are many resources available for you. To find the latest, just search “frc collector code <your_language_here>”.

9.8.4 Shooter

Aside from an elevator, a shooter will be one of the most complicated devices teams will program. A simple shooter may just set a power to the drive motor, but we recommend using a PID velocity control loop, if your team has the software manpower to develop one. Doing so will drastically increase your shooting performance. See the PID section (9.7.1) of this guide for more info on that. We also recommend that teams keep records of shot data and PID setpoints, which will allow you to learn and develop your system, as well as garner the benefits of aggregate data.

9.8.5 Climber

Every climber is different. Some have many actuators working together, others are just a single piston. Whatever system you use, be wary of it! If it can lift a 120lb robot, it can hurt people. ALWAYS include safety interlocks wherever possible. As always, sanitize any data input (i.e. don't let the climber try to divide by 0), and test your code in a safe and controlled manner. NEVER trust your code alone to keep people safe.

9.9 Networking

In FRC, networking tends to be a bit of an advanced concept. For simple cases, like getting input from the Driver Station or using the various dashboards on offer, all of the networking is done by the libraries, and you don't even need to consider it. However, as your application grows more complex and new problems present themselves, you might end up needing to do some networking yourself. Networking is the process of getting devices to communicate effectively over a network. This section will focus specifically on the robot's Ethernet network using IP (Internet Protocol) and its applications. The robot has other networks as well, most notably CAN, but IP is more complex and is the only network that includes the Driver Station.

9.9.1 NetworkTables

The main way of communicating over a network in FRC, aside from the Driver Station, is through NetworkTables. It's the underlying mechanism that Shuffleboard, etc. use to communicate with the robot, and it is also useful on coprocessors and other peripheral devices. NetworkTables is included with WPILib and can also be used separately in programs on other devices. The WPILib documentation explains how to use it. There are additionally several unofficial implementations of NetworkTables supporting languages like Python, TypeScript (via Node.js), and Rust. Because NetworkTables is implemented the same way across languages, you could even use different languages for programs on your robot, computer, and coprocessor.

NetworkTables acts as a store of keys and values. If you associate a value with a key in one program, then you can read that value back out from another. Values can be strings, numbers, booleans, or arrays of any of those types. A common example, apart from Shuffleboard, is for vision coprocessing: a device separate from the roboRIO takes camera input and computes how far off-center the robot is from a target, then pushes this number to NetworkTables. The roboRIO can read the number from NetworkTables and the robot can adjust itself accordingly.

A nice feature of NetworkTables is that you can create a "callback function" to run if the value associated with a key changes. Normally, if you wanted to run code whenever a value changed, you would have to continuously check that value and compare it to its value from last time you checked, which is extremely inefficient. Thankfully, you can instead tell NetworkTables to run a piece of code whenever it receives an updated value from another device, which is easier to maintain and far more efficient.

It should be noted that, while NetworkTables is incredibly convenient for use within FRC, it is heavily recommended not to use it anywhere else. The biggest reason for this is that it has no security whatsoever; in FRC, cybersecurity isn't an issue because of Gracious Professionalism™ (and also because the FMS puts each team on a separate VLAN), but in any real-world situation, you probably want to have some sort of security for your application data. NetworkTables is also somewhat simplified compared to the solutions used in the industry, and doesn't have some of the nicer features that software developers have come up with. It's optimized mostly for real-time execution speed, which is very important on a robot, but means that other concerns must be sacrificed. If you're looking for something to use as a local data store outside of FRC, try Redis, which is similar to NetworkTables, but much more powerful.

9.9.2 IP Configuration

This section should be considered as even more advanced reading; many teams will do perfectly fine without having to worry about IP configuration. However, configuring static IP addresses can help prevent or troubleshoot connection problems that might occur, and can generally improve the reliability of connections.

An IP address is a list of numbers that identify a device on a network. FRC, as well as most other networks, use IPv4 addresses, which consist of 4 numbers between 0 and 255, separated by periods in the traditional representation (e.g. 123.45.67.89). In FRC, your IP address will typically fall in the range from 10.TE.AM.1 to 10.TE.AM.200, where TE.AM is replaced with your team number. For example, team 118's roboRIO might have the IP address 10.1.18.2, while team 2468's Driver Station might be 10.24.68.5. However, by default, there's no guarantee as to what the IP address will actually be. (The major exception to both of these rules is when connecting to the roboRIO over USB, in which case its IP address will always be 172.22.11.2.) The radio configuration will, by default, assign the roboRIO the 10.TE.AM.2, address. FRC devices typically use a protocol called mDNS to "discover" each other's IP addresses by name. (This is a simplified explanation; the WPILib docs go into more detail.) mDNS works most of the time - if you've ever successfully used the Driver Station or deployed code, that's mDNS at work - but it can sometimes fail, especially on networks with additional devices. If you have problems establishing a connection between the Driver Station, radio, roboRIO, and/or any extra doodads on the robot, you might want to try static IP configuration.

In a normal, "dynamic" IP configuration, devices are assigned temporary IP addresses by a "DHCP server." For a home network, this is typically the router; on an FRC robot, this is the radio (which is how the radio gives the roboRIO 10.TE.AM.2). However, devices can be set to forego DHCP and instead give themselves IP addresses. This has some disadvantages; In addition to assigning an IP address, DHCP is used to communicate various details about the network to devices trying to join it, so these details must be set manually in order for a static IP configuration to work. Also, a static configuration can cause problems if multiple devices are set to the same static IP; DHCP normally makes sure every device gets a unique IP address, but it can't do that if it's being bypassed. However, a consistent IP address means that you don't have to go through mDNS every time you connect to the device, which can be very helpful if mDNS is causing problems. It also makes connecting to the network noticeably faster, since you don't have to wait for DHCP, which isn't super important but is a nice side effect. If you are planning to implement static IP configuration, refer to the WPILib docs for information on what IP addresses and network details to use for various devices. Below are step-by-step instructions for configuring a static IP address for the roboRIO and a computer running Windows 10. Other devices/operating systems have their own processes; dig around in the settings or load up your favorite search engine.

RoboRIO Static IP Configuration

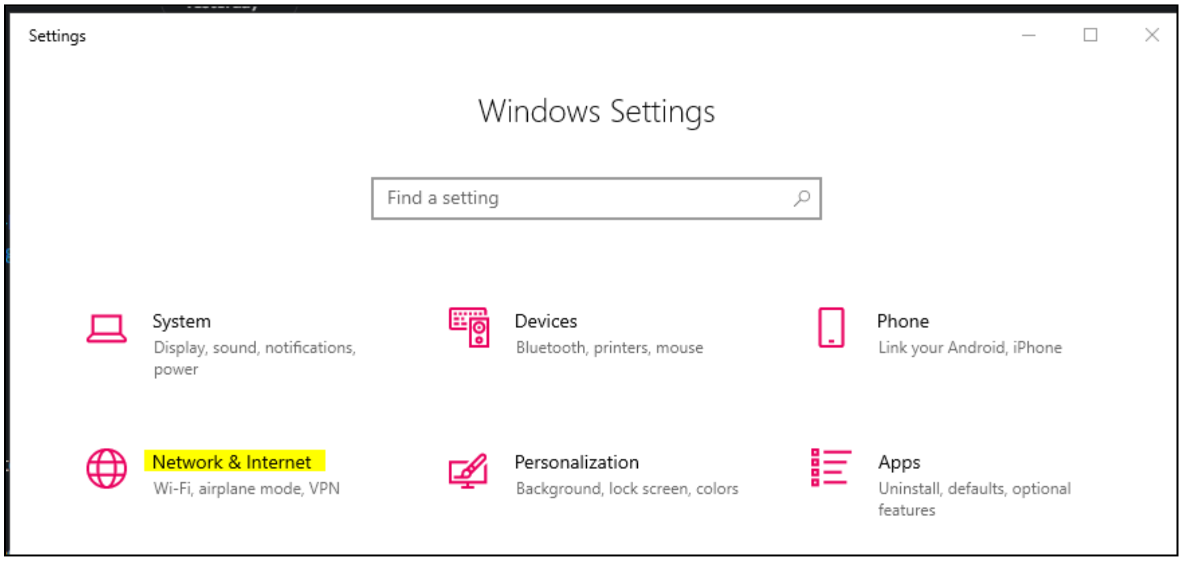
1. Connect a computer to the roboRIO via USB, Ethernet, or the wireless radio.
2. Open a web browser and navigate the roboRIO web dashboard.
    - If you think mDNS will work, type "http://roborio-TEAM-frc.local" (replacing TEAM with your team number) into the address bar.
    - If you are connected over USB (which you should do if mDNS fails you), type "http://172.22.11.2" into the address bar.
    - For more details, check the WPILib docs.
3. Log in using the "log in" link on the top-right (?) of the page. The username is "admin" and the password is blank.
4. Navigate to the "networking" tab with the Ethernet-cable button on the left side of the page.
5. Open the dropdown labeled "Configure IPv4 address" and select "Static."

6. Fill in the fields appropriately:
   - IPv4 Address: “10.TE.AM.2,” replacing TE.AM with your team number as described earlier
   - Subnet Mask: “255.255.255.0”
   - Gateway: “10.TE.AM.1”
   - DNS Server: leave blank
7. Reboot or power cycle the roboRIO.
8. Connect via Ethernet or wireless radio and test the configuration by accessing the web dashboard at “http://10.TE.AM.2” from your browser.

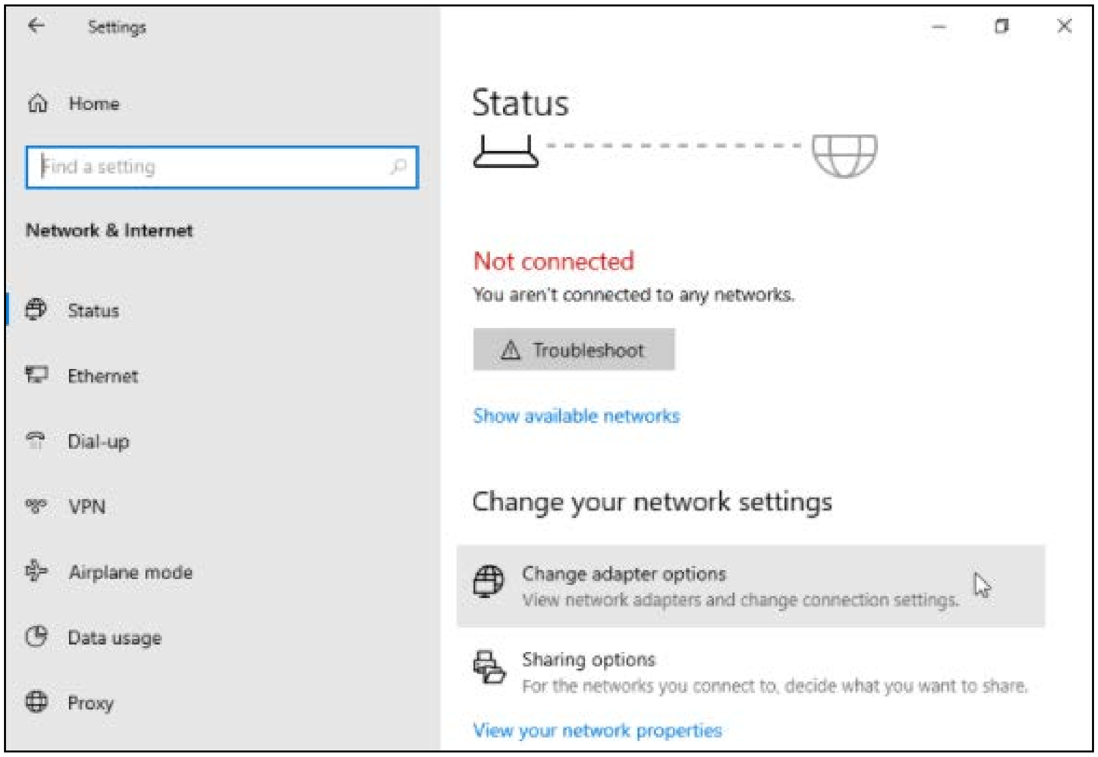
Windows 10 Static IP Configuration

The steps described in this section will leave your computer unable to connect to the Internet on the network interface you configure. It is recommended that you only use a static IP address on your Driver Station for wired connections, and leave DHCP on for wireless connections, since the latter is how you probably mostly connect to the Internet, unless you can’t connect over the wireless radio without a static IP (unlikely). Additionally, the FRC Radio Configuration Utility resets all of these settings to their defaults. If you run the Utility from your Driver Station and want to keep the static IP address, you have to follow all of these steps again.

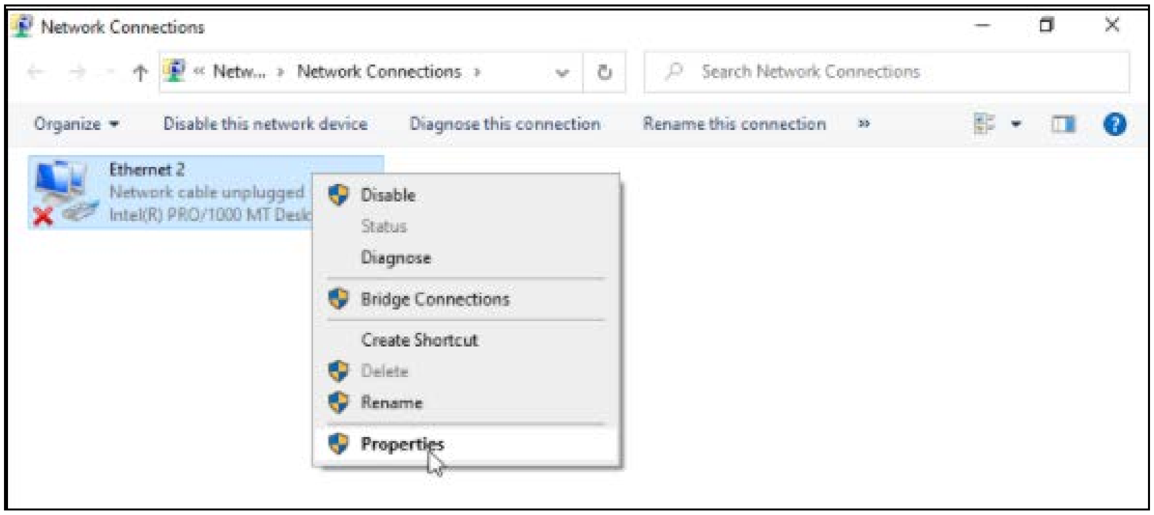
1. Open Settings from the Start menu.
2. Go to Network and Internet.


Settings
Windows Settings
Find a setting
System
Display, sound, notifications, power
Devices
Bluetooth, printers, mouse
Phone
Link your Android, iPhone
Network & Internet
Wi-Fi, airplane mode, VPN
Personalization
Background, lock screen, colors
Apps
Uninstall, defaults, optional features

3. Find and select “Change adapter options.”

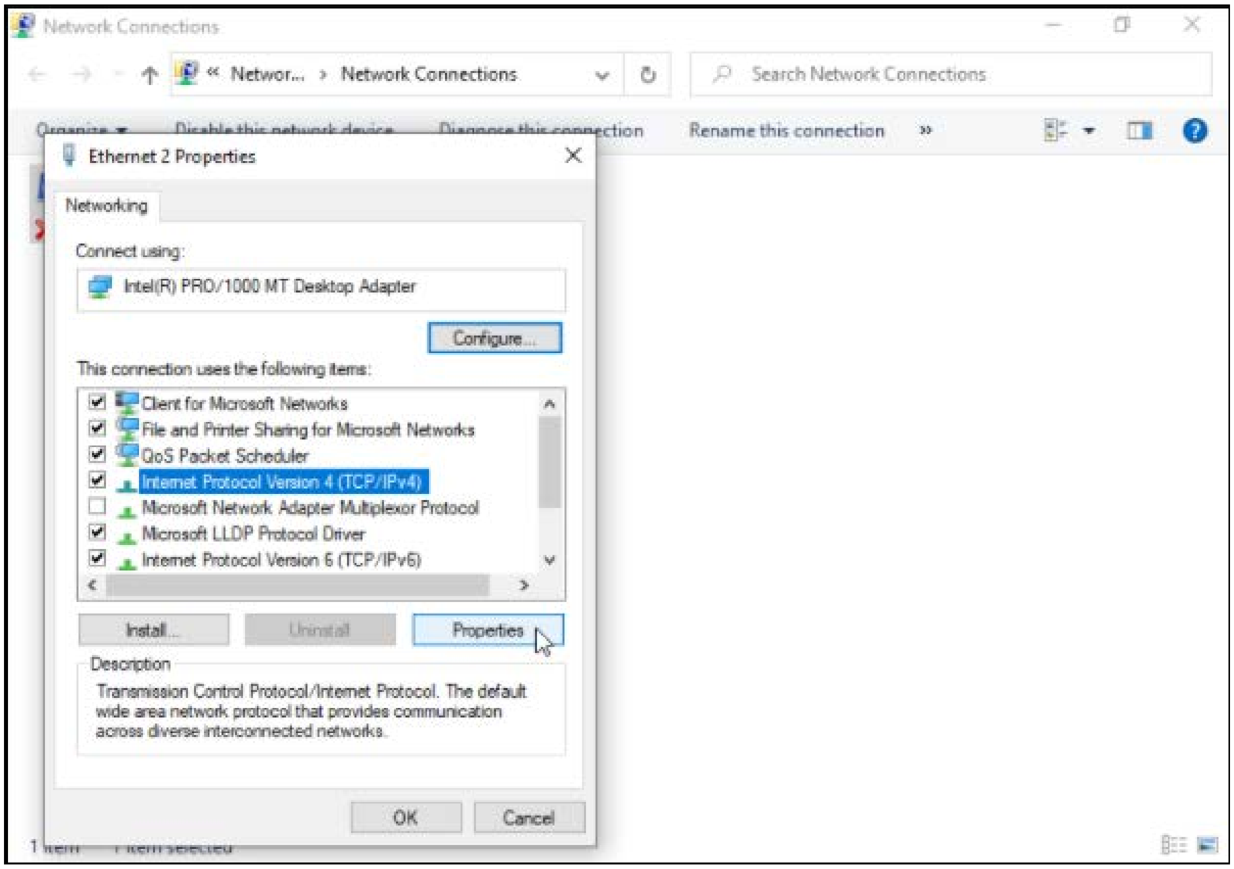

Settings
Home
Find a setting
Network & Internet
Status
Ethernet
Dial-up
VPN
Airplane mode
Data usage
Proxy
Status
Not connected
You aren't connected to any networks.
Troubleshoot
Show available networks
Change your network settings
Change adapter options
View network adapters and change connection settings.
Sharing options
For the networks you connect to, decide what you want to share.
View your network properties

4. Find your Ethernet adapter in the menu that opens. Right-click it and choose "Properties." You may need to enter a password, depending on how UAC is configured on your machine.



There will almost certainly be multiple network adapters in your menu, and probably multiple Ethernet adapters at that. Choose the one that is not labeled "National Instruments something or other," as that one is used to connect to the roboRIO over USB and should not be messed with. If you aren't sure which adapter is correct, plug the Ethernet port you want to use into a network and see which adapter reacts in the menu.

5. In the dialog that opens, select “Internet Protocol Version 4 (TCP/IPv4)” (which is a stupid name because it also controls UDP/IPv4, which is mostly what we care about) and click the “Properties” button.


Network Connections
Networ... › Network Connections
Search Network Connections
Organize ▾ Disable this network device Diagnose this connection Rename this connection »
1 item 1 item selected

Ethernet 2 Properties

Networking

Connect using:

Intel(R) PRO/1000 MT Desktop Adapter

Configure...

This connection uses the following items:

- [✓] Client for Microsoft Networks
- [✓] File and Printer Sharing for Microsoft Networks
- [✓] QoS Packet Scheduler
- [✓] Internet Protocol Version 4 (TCP/IPv4)
- [ ] Microsoft Network Adapter Multiplexor Protocol
- [✓] Microsoft LLDP Protocol Driver
- [✓] Internet Protocol Version 6 (TCP/IPv6)

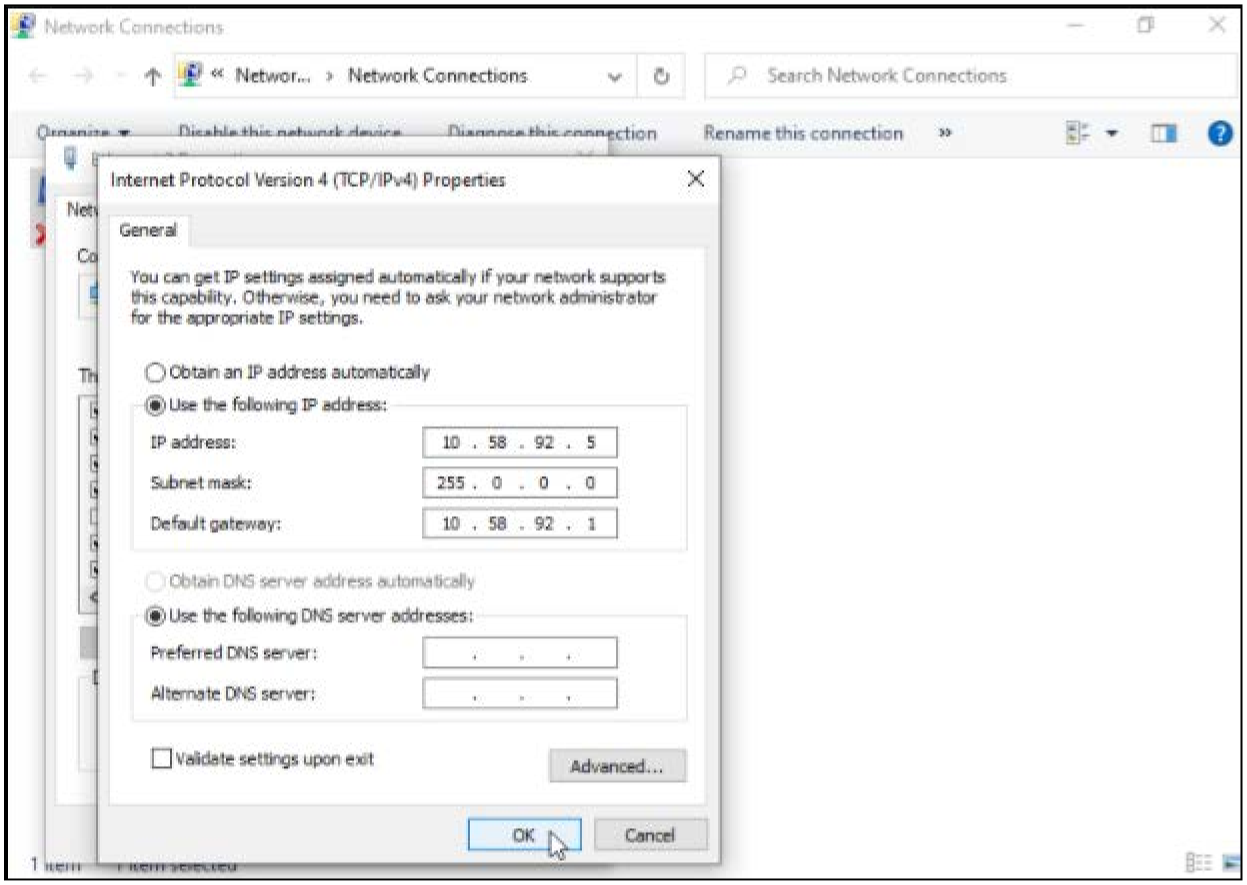
Install... Uninstall Properties

Description

Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

OK Cancel

6. Choose “Use the following IP address” and fill in the fields appropriately:
   - IP address: “10.TE.AM.5,” replacing TE.AM with your team number as described earlier
   - Subnet mask: “255.0.0.0” (which should autofill, but make sure it is correct)
   - Default gateway: “10.TE.AM.1”
   - Leave the DNS server addresses blank
   - Make sure “Validate settings upon exit” is **not** checked


Internet Protocol Version 4 (TCP/IPv4) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

( ) Obtain an IP address automatically
[✓ Use the following IP address:]

IP address: 10 . 58 . 92 . 5
Subnet mask: 255 . 0 . 0 . 0
Default gateway: 10 . 58 . 92 . 1

( ) Obtain DNS server address automatically
[✓ Use the following DNS server addresses:]

Preferred DNS server: . . .
Alternate DNS server: . . .

☐ Validate settings upon exit

7. Press the “OK” and “Close” buttons on the dialogs to save the new settings.
8. Connect to the robot via Ethernet and test the configuration by making sure the Driver Station can find the robot or opening the roboRIO web dashboard.

9.9.3 Port Forwarding

This, like IP configuration, is a more advanced section. Port forwarding is only useful if there are devices on the robot network other than the roboRIO and the Driver Station - for example, a coprocessor. If your robot doesn’t have any Ethernet-connected devices besides the roboRIO, you can skip this section. If you *do* have another Ethernet-connected device, it’s probably plugged into the other Ethernet port on the radio so that the device, radio, and roboRIO can all communicate over Ethernet and the Driver Station can communicate with all of them wirelessly. This all works pretty well, until you have multiple coprocessors (in which case you need a network switch) or until you are in the pits at competition, where you can’t use the radio wirelessly. Then you have to disconnect a device from the rest of the network to connect it to the Driver Station instead, or connect the Driver Station to the roboRIO via USB so that the Driver Station can communicate with the roboRIO and the roboRIO can communicate with everything else, and just accept the fact that the Driver Station won't be able to communicate with anything other than the roboRIO.

However, port forwarding solves this issue. It allows the Driver Station to communicate with other Ethernet devices *through* the roboRIO USB connection. Port forwarding allows you to access the web-based configuration for another device over a USB connection to the roboRIO, such as a limelight or Raspberry Pi running the WPILib vision processing image. A "port" in this context is just a number used by a network connection to know what service it is directed to. You can choose an arbitrary number on the roboRIO (common choices are 8000, 8080, or 8888) and tell the robot program to "forward" connections to that port to port 80 (standard port for HTTP) on your other device. Then, on your USB-connected Driver Station, you can navigate in your web browser to "http://172.22.11.2:8000/" (the ":8000" tells the browser to override the default port, replace with whichever port you chose) and access the configuration dashboard! The WPILib docs have more information on how to use this in your robot program. The limelight uses ports 5800 and 5801.

9.9.4 SSH

SSH (stands for Secure SHell) is a protocol for remotely logging into another computer. You may have used it before if you've ever used the Raspberry Pi: once you have it set up, you can run a terminal session on your Pi without having to connect inputs and a display, which is all kinds of convenient, especially if the Pi is somewhere that would make it difficult to connect to a monitor - for example, on a robot. It even allows you to transfer files between your Pi and another computer thanks to SFTP (stands for SSH File Transfer Protocol) (not to be confused with FTPS, which is a *different*, less common protocol to accomplish the same thing). The roboRIO also runs an SSH/SFTP server, which is mainly used by the code-deployment scripts, but you can also access it for whatever purpose you need. Maybe your robot code creates a log file that you want to download onto a computer? Maybe you want to make use of the HTTP server on the roboRIO, normally used to serve the configuration dashboard? (Files are served from `/var/local/natinst/www`, if that interests you. Just don't mess with anything already there.) Most teams won't ever need to SSH into their roboRIO, and the WPILib Raspberry Pi image is designed to mainly be configured from the browser, but it can still be helpful to know about SSH for those situations where it is needed.

In order to use SSH or SFTP, you'll need to install a "client program" that understands the protocol. For SSH, a commonly-used client on Windows is PuTTY, which includes a graphical interface to set up connections. Another option is OpenSSH; you can install OpenSSH on Windows 10, and it ships with macOS and many Linux distributions, but it can only be used via the command line, making it a bit harder to use. For SFTP, a useful program on all platforms is FileZilla, which supports several other file transfer protocols in addition to SFTP (including FTPS), though SFTP tends to be the most useful in FRC. OpenSSH and PuTTY also both have command-line SFTP clients.

In order to connect to a remote host using SSH or SFTP, you need a username and password. For the roboRIO, you can use either `admin` with a blank password or `lvuser`, which won't even prompt for a password, depending on what you're trying to do. For the Raspberry Pi, the default user on most Raspbian-based images (including WPILib's) is named `pi` and has the password `raspberry`. (The Pi will recommend that you change its default password, and you're free to do so, but don't change/set passwords on the roboRIO, or you won't be able to deploy or run code. If you're looking for a place to practice cybersecurity, FRC may not be the program for you.)

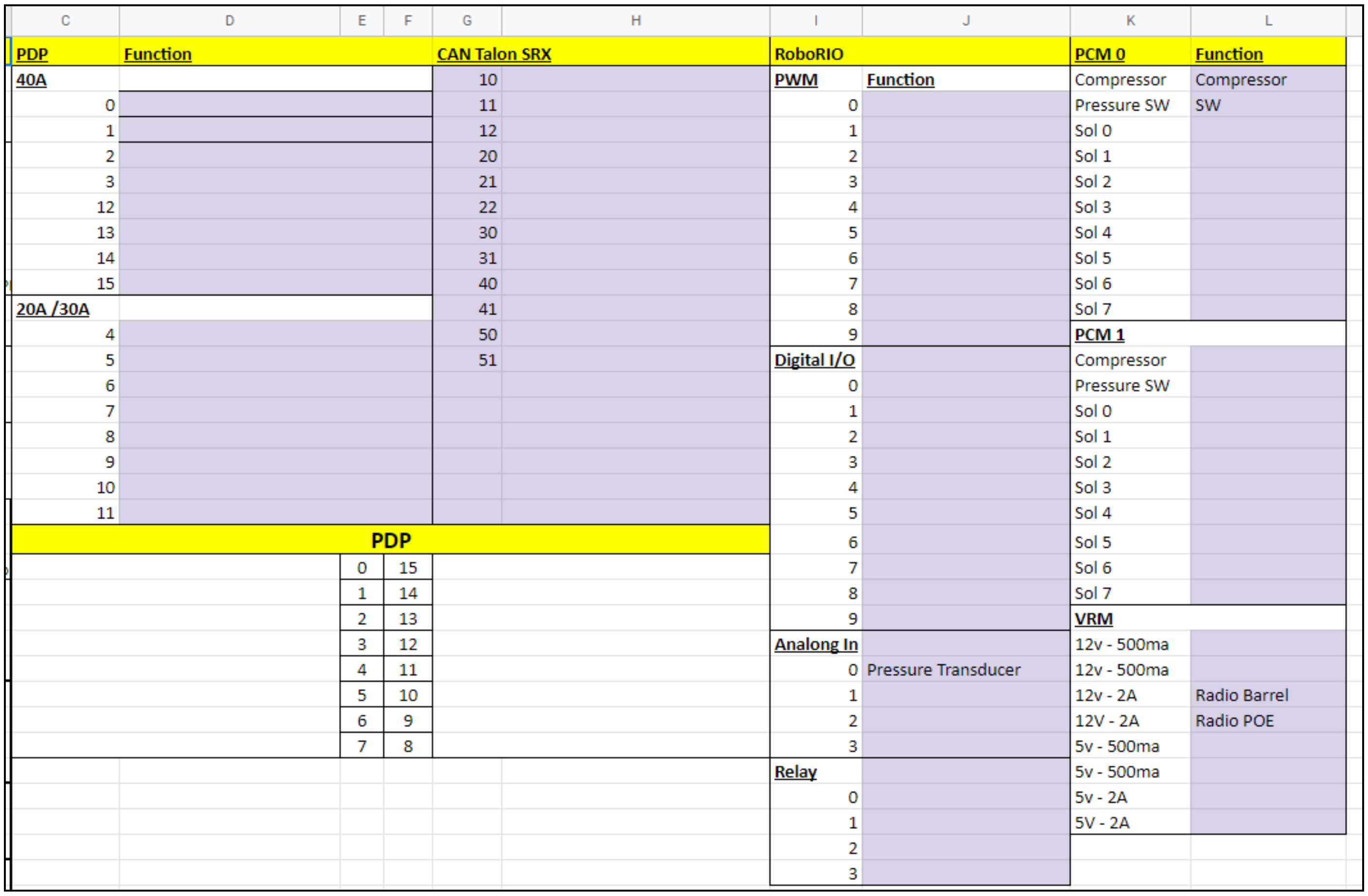
9.10 Common Problems and Troubleshooting

It would be impossible to note and address all the different issues you may come across in software. Oftentimes, the issue ends up being something that can be relatively simple to fix (or even better - prevent!), but panic-inducing in the quick tempo while at a competition. Below are some common problems you might come across, especially while trying to make quick changes and repairs at a competition. In almost all cases, it is best to become familiar with what LEDs for the electrical components mean by looking at the user guides. It's also important to learn how to utilize the tools provided in the Driver Station to monitor things such as CPU usage, CAN utilization, voltages, currents, event logger, and messages tab. A video going into some examples of the driversation log viewer can be found here.

- A mechanism starts behaving differently without changing code. Or a mechanism isn't moving when I expect it to
    - Check all sensor lines and confirm the sensor is reading correctly. It is useful to display these sensor values in shuffleboard so you can see if, for instance, a potentiometer or encoder isn't increasing/decreasing as expected while moving a mechanism. Or maybe in a repair limit switches got plugged in backwards or swapped.
    - Did a motor controller need to get replaced? Newer motor controllers have settings that are saved in flash memory on the controller, so the motor controller may need to be reconfigured. Similarly, it's CAN ID may need to be reset
- The robot doesn't boot. Driver station reports a HAL error

    - HAL, in this context, stands for Hardware Abstraction Layer. This is what allows the roborio to take the software identifications and map them to the physical hardware pins. This type of error is commonly caused when you have two devices trying to use the same pins/ID. Examples can be multiple devices with the same CAN ID or two sensors configured for the same pins. Note, that this is especially easy to happen with dead code, i.e. code that is no longer used. If you have dead code declaring an encoder on DIO 1 and 2 that is no longer used, you will get this error when you decide to use DIO 1 and 2 for something else.
- My mechanism's motors output is stuttering
    - This could be caused by something as simple as a loose electrical connection.
    - This happens if in software you have output being written to the motor controller in multiple places. This can occur easily in a SimpleRobot or IterativeRobot framework, or if commands are not properly configured in the CommandBasedRobot to require subsystems and multiple commands are trying to control the same subsystem at once. In the Command-Based framework, make sure to declare all subsystem dependencies to help identify this issue before it occurs. In other frameworks, it's good to use a variable to hold the value you want to assign to the motor controller, and make that assignment only once at the end of the function.
- The mechanism sometimes seems to work, but then stops. It'll work again later.
    - If you hear any popping sounds, you might be popping a breaker in the PDP. The breakers in the PDP are self-resetting. However, once they reset they are likely to trip more quickly in the future and should be replaced if possible. Make sure you have your motor on a proper breaker amperage. It is generally good practice to have current limits on your mechanism anyway that should help prevent this from occurring.
- Roborio seems to randomly disconnect while playing
    - There are a number of things that can cause this. Perhaps the most common are brownouts to the roborio/radio or an insecure connection to radio. However, the root cause can be in software; if the roborio CPU usage is too high or a control loop is taking too long, you might see messages in the driver station to the effect that the watchdog was not fed.
- Having an issue with a spark max/Neo or Falcon 500
    - Check all electrical connections/LEDs
    - Check the troubleshooting guides/sections for your specific motor controller/motor:
        - Spark Max/ Neo - flow charts available in the Troubleshooting guide
        - Falcon 500 - troubleshooting section in the user guide

        - Talon SRX - troubleshooting section in the user guide
        - Victor SPX - troubleshooting section in the user guide
- My motor doesn't seem to be doing anything
    - Check to make sure the motor controller firmware is up to date
    - Ensure that the physical hardware mapping matches the software mapping. Do the LEDs indicate the motor is trying to spin, or not getting any signal? It is useful to use a design sheet to make sure everyone is on the same page. Below is an example from designsheet.spectrum3847.org for recording all of the electrical and software information regarding motors



C	D	E	F	G	H	I	J	K	L
PDP	Function			CAN Talon SRX		RoboRIO		PCM 0	Function
40A				10		PWM	Function	Compressor	Compressor
0				11		0		Pressure SW	SW
1				12		1		Sol 0	
2				20		2		Sol 1	
3				21		3		Sol 2	
12				22		4		Sol 3	
13				30		5		Sol 4	
14				31		6		Sol 5	
15				40		7		Sol 6	
20A /30A				41		8		Sol 7	
4				50		9		PCM 1	
5				51		Digital I/O		Compressor	
6						0		Pressure SW	
7						1		Sol 0	
8						2		Sol 1	
9						3		Sol 2	
10						4		Sol 3	
11						5		Sol 4	
		PDP				6		Sol 5	
		0	15			7		Sol 6	
		1	14			8		Sol 7	
		2	13			9		VRM	
		3	12			Analong In		12v - 500ma	
		4	11			0	Pressure Transducer	12v - 500ma	
		5	10			1		12v - 2A	Radio Barrel
		6	9			2		12V - 2A	Radio POE
		7	8			3		5v - 500ma	
						Relay		5v - 500ma	
						0		5v - 2A	
						1		5V - 2A	
						2			
						3			

Appendix A - Acronyms and Abbreviations

3-D	3 Dimensional
ABS	Acrylonitrile Butadiene Styrene'
ANSI	American National Standards Institute
API	Application Programming Interface
APP	Anderson Power Poles
BLDC	BrushLess Direct Current
BOM	Bill of Materials
BW	Bandwidth
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CAN	Controlled Area Network
CD	Chief Delphi
CG	Center of Gravity
CNC	Computer Numerical Control
COTS	Commercial Off-The-Shelf
CTRE	Cross The Road Electronics
DC	Direct Current
DIO	Digital Input/Output
DMLS	Direct Metal Laser Sintering
DP	Diametral Pitch
DS	Dual Speed
DXF	Drawing Exchange Format
FDM	Fused Deposition Modeling
FIRST	For Inspiration and Recognition of Science and Technology
FMS	Field Management System
FRC	FIRST Robotics Competition
FTC	FIRST Tech Challenge

FTPS	File Transfer Protocol Secure
GT2	Gates 2mm PowerGrip Belt Drive System
HDPE	High-Density Polyethylene
I/O	Input/Output
ID	Identification
ID	Inner Diameter
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
IP	Internet Protocol
JVN	John V-Neun (148 mentor)
KOP	Kit Of Parts
LED	Light Emitting Diode
LIDAR	Laser Imaging Detection and Ranging
MXL	Mini XtraLight Belts
NASA	National Aeronautics and Space Administration
NI	National Instruments
NyLock	Nylon Locknut
OD	Outer Diameter
Omni	Omnidirectional
OS	Operating System
PCM	Pneumatics Control Module
PDM	Product Data Management
PDP	Power Distribution Panel
PEI	Polyetherimide
PG	Planetary Gearbox
PID	Proportional Integral Derivative
PLA	Polylactic Acid
POM	Polyoxymethylene
POT	Potentiometer

PTC	Parametric Technology Corporation
PTFE	Polytetrafluoroethylene / Teflon
PTO	Power Take-Off
PVC	Polyvinyl chloride
PWM	Pulse Width Modulation
RAMP	Remote Assistance and Mentorship Program
RAP	Robotics Alliance Project
RC	Radio Controlled
RIO	Reconfigurable Input/Output
ROM	Read Only Memory
RPM	Revolutions Per Minute
RSL	Robot Signal Light
SCP	Secure Copy Protocol
SDP/SDP-SI	Stock Drive Products - Sterling Instrument
SFTP	SSH File Transfer Protocol
SLA	Stereolithography
SLS	Selective Laser Sintering
SS	Single Speed
SSH	Secure SHell
SSID	Service Set Identifier
UHMW	Ultra-High-Molecular-Weight Polyethylene
USB	Universal Serial Bus
VDC	Volts Direct Current
VRM	Voltage Regulator Module
WCD	West Coast Drive
WCP	West Coast Products
WPILib	Worcester Polytechnic Institute Library
XL	XtraLight Belts

Appendix B - Materials

B1 - Material Profiles

Sheet: typically come in 48”x96” sheets or smaller.

Honeycomb: low density material with high compressive strength, typically two thin sheets of material with hexagon shaped cells between them. Can transmit high compression loads along one axis.

Bars/blocks: solid blocks of material typically used when milling complex parts.

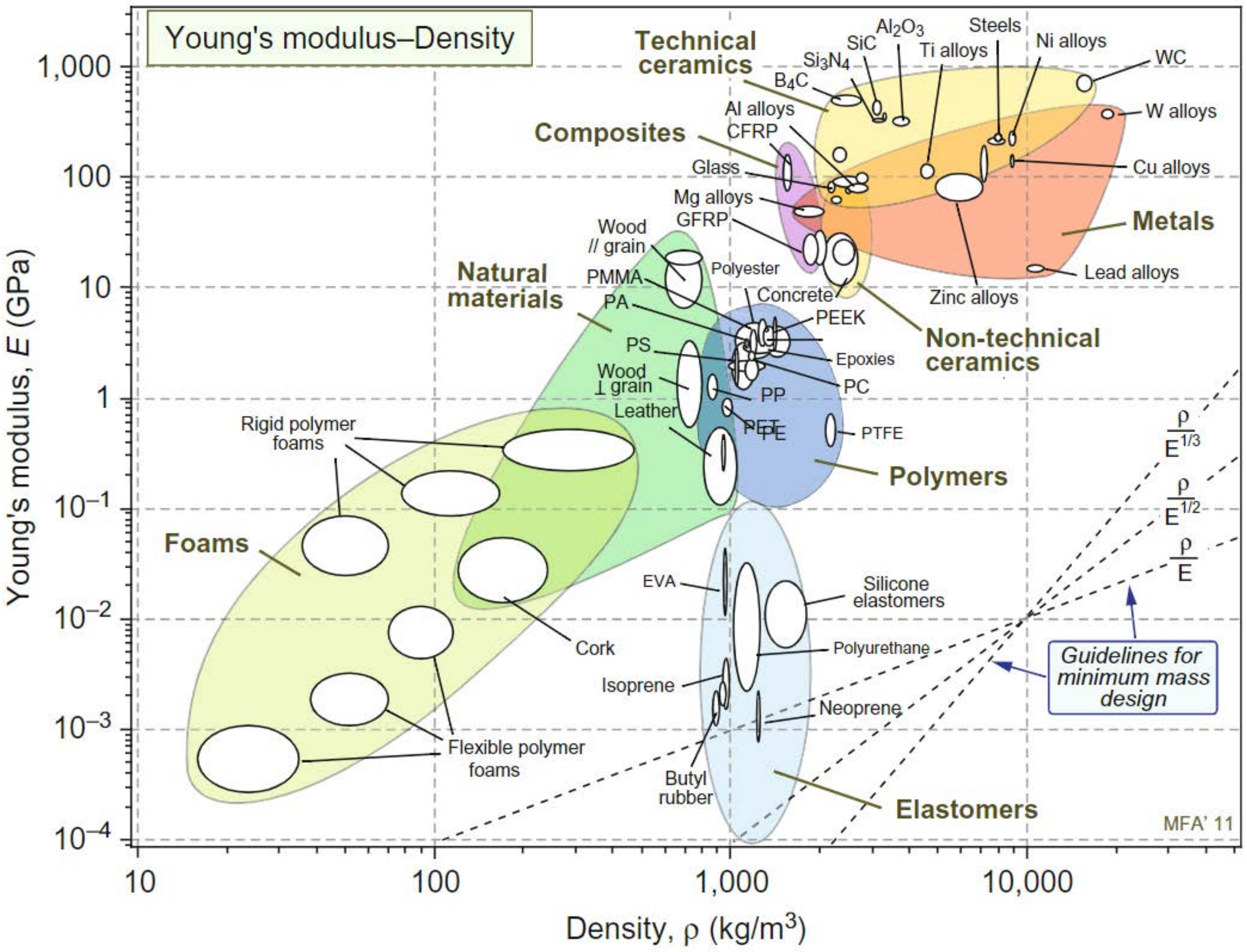
Rods: solid round material, typically used for custom round parts such as wheel hubs or roller plugs.

Extrusion: extruded material has a specific cross section such as a hex profile, a box tube, etc.

- **Tubes:** an extruded section of material that forms a hollow shape. Available in many shapes and sizes including **hex, square, rectangular, and round**.
- **Shaft:** stock extruded for use in rotary motion transfer, available in **hex, thunderhex, square, round, and round keyed.**
- **Angle:** 90 degree extrusion in the shape of an “L” Available in a variety of sizes from Vexpro and McMaster-Carr.
- **C-Channel/U-Channel/I-Beams-H-Bars/T-Bars/Z-Bars:** these are all extrusions that are in the shape of the letter in their name.

What do you build a robot out of?

The answer: Which parts? What do these parts do? Material science is a vast study because there are so many different properties of materials. We'll focus on mechanical properties such as strength and density.

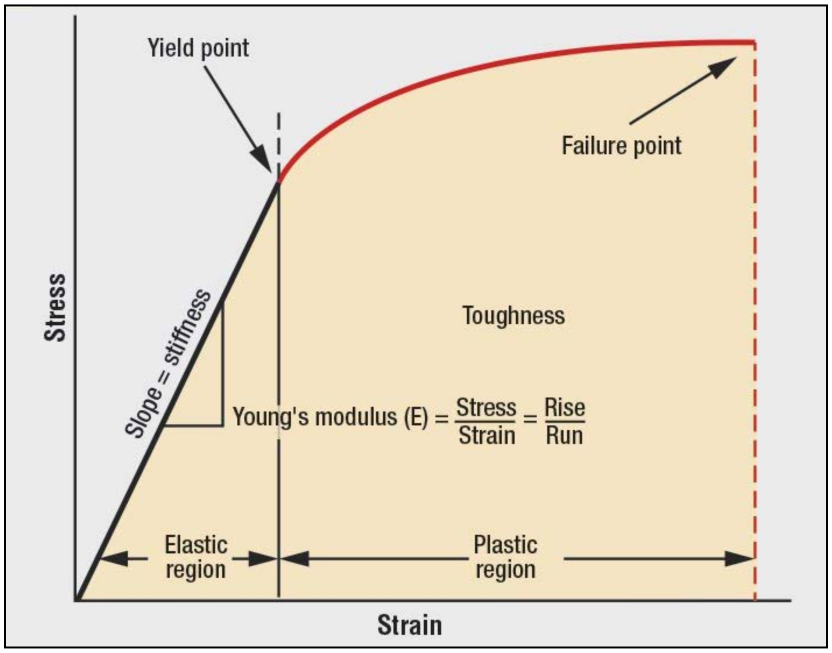


(Photo from www.sciencedirect.com/topics/engineering/e-modulus - Figure 9.12.The Modulus-Density chart: the one for stiffness at minimum weight)

While this graph looks complicated, it's actually surprisingly simple, with useful implications. The y-axis, **Young's modulus** (aka modulus of elasticity), is your intuitive idea of "stiffness". The x-axis, **density**, is the weight per unit volume of the material. By reading this graph, you can see that polymers (plastics) are lighter than, but also less stiff than metals. Also, you'll see that elastomers (rubber-like materials) are as heavy as polymers, but generally easier to deform.

Young's modulus is a measure of stiffness. To understand Young's modulus, and define strength, we'll look at one more graph, relating **stress** and **strain.**

Stress-Strain Curve



(Photo from https://www.industrialheating.com/articles/93624-an-overview-of-fundamental-material-properties)

Each material has a unique shape to its graph of stress and strain. Strain is deformation; how much the material has changed from its original shape, often length (expressed as a percentage). Stress is proportional to the force exerted on a material (expressed as a pressure, such as psi). The left part of this graph, where the relationship between stress and strain is linear, is used to calculate Young's modulus. The steeper the graph, the higher the Young's modulus, and the **stiffer** the material is. The **ultimate strength** of a material is the stress at the peak in the curved section of this graph. This curved section is known as the plastic region, as the material will not return to its original shape once the stress is removed.

B2 - Metals

Types of Metals

Aluminum: aluminum is the go-to material in FIRST; it is non ferrous, reasonably strong, and lighter than most other reasonably priced materials. There are many different types of aluminum alloys all with their own 4 digit number that sometimes has a dash number that denotes the temper. Below is a list of what is typically used; the list of all the alloys out there is endless.

Aluminum Alloy	Common Temper	Characteristics
6061	T6	the standard alloy, typically used on non bent parts. 6061 isn't the best at anything but it's pretty good at everything. It's good for welding, easy to machine, and resistant to corrosion.
5052	H32	the alloy usually used for bent sheet metal parts.
7075	T6	7000 series aluminum is some of the strongest aluminum available. 7075 can be used in applications requiring strength higher than 6061, but without the higher weight of steel. 7075 is more brittle than other alloys.
2024	T3	stronger than 6061, but not as strong as 7075. Typically sold as round tube.
6063	T5	Similar to, but half the tensile strength of 6061, as well as more difficult to cleanly machine. Try to avoid using 6063.
3003	Not Rated	McMaster-Carr sells honeycomb aluminum in 3003 alloy.

Steel: This is not usually the go-to material for FIRST robots, but it does have its uses. Like aluminum, it comes in many different alloys. Although steel is heavier, it has a strength-to-weight ratio similar to aluminum.

4100 Series Steel (Chromoly steel): Often used in automotive applications, chromoly steel has a very good strength-to-weight ratio. The reason they are referred to as chromoly steel is due to the fact that they have chromium and molybdenum as alloying elements in them. If you ever see a welded round tube frame on a vehicle such as a race car, it is most likely made from chromoly steel tubing.

1075/1095 Spring Steel: Spring steels usually need to be heat treated to give them their spring like properties. 1075 is less stiff and has less bounce back after heat treating.

B3 - Plastics

Plastics: frequently used on FIRST robots due to their unique properties and lower weight than aluminum. There are thousands of plastics out there, each with their own unique properties, but the ones most commonly used in FRC are listed below.

Polycarbonate (Lexan): polycarb is one of many non-shattering engineering plastics. It has a high impact resistance, therefore resists cracking during hard collisions. In collisions, polycarb will bend and bounce back, unlike aluminum which will bend permanently and not bounce back. Polycarb is non-conductive of heat and electricity, and it is clear. It does, however scratch very easily. It can be bent like sheet metal, with larger bend radii and/or heat. It weighs about half as much as aluminum. If Loctite touches polycarb it will begin to crack; it is imperative to not let Loctite touch polycarb. Polycarb's resistance to impacts makes it great for mechanisms that extend outside of the frame, such as intakes.

Delrin/Acetal/Polyoxymethylene/POM: Delrin is known for its low friction properties along with being a stiff plastic. Delrin can be used as a bearing surface for low speed applications, such as on turrets. It typically comes in black or white, but is available in other colors. It can be laser cut.

Teflon/PTFE/Polytetrafluoroethylene: Teflon has one of the lowest coefficients of friction of any solid; because of this unique property it is fairly expensive. It can be used as a bearing surface for low-to-medium speed applications. It is available with an adhesive backing to stick it under moving components.

HDPE/High-Density Polyethylene: HDPE is a thermoplastic that has a reasonably low coefficient of friction; not used that often on robots, but is frequently used on field elements.

UHMW/Ultra-High-Molecular-Weight Polyethylene: UHMW is a very tough plastic that has the highest impact strength of any thermoplastic currently produced. It has high wear resistance, a low coefficient of friction (Comparable to PTFE/Teflon), and is self-lubricating. These properties make it a frequently used material in lightweight combat robots.

B4 - 3D Printer Filaments

3D Printer Filament: FDM 3D printers use material called filament. It's basically a thin plastic wire that comes on a spool and is fed into the printer. Most filament is plastic, however there are also metal, composite and rubber filaments.

PLA/Polylactic Acid: The most common hobby-grade 3D printed material. PLA has a low melting temperature that may cause it to fail if near hot components of a robot. More brittle than ABS. Much lower strength than a high end material like Ultem or Onyx.

ABS/Acrylonitrile Butadiene Styrene: ABS is one of the most basic 3D printer filaments. Most consumer printers can print in ABS and it comes in many colors. It’s pretty strong and is great for low load applications, such as camera and sensor mounts.

Ultem/PEI/Polyetherimide: Ultem is a high strength, rigid 3D printer filament we use for parts that need to take any load. It is the strongest FDM thermoplastic available. Usually it comes in black or ivory. Can only be printed on select Fortus printers. Has been used frequently by Team 118 and Team 330.

Nylon: a very flexible material for 3D printed components. It can bounce back to its original shape; it has similar properties to polycarbonate. Nylon is hygroscopic, meaning it will absorb moisture; it should be kept in a dry box or sealed bag with desiccant packets.

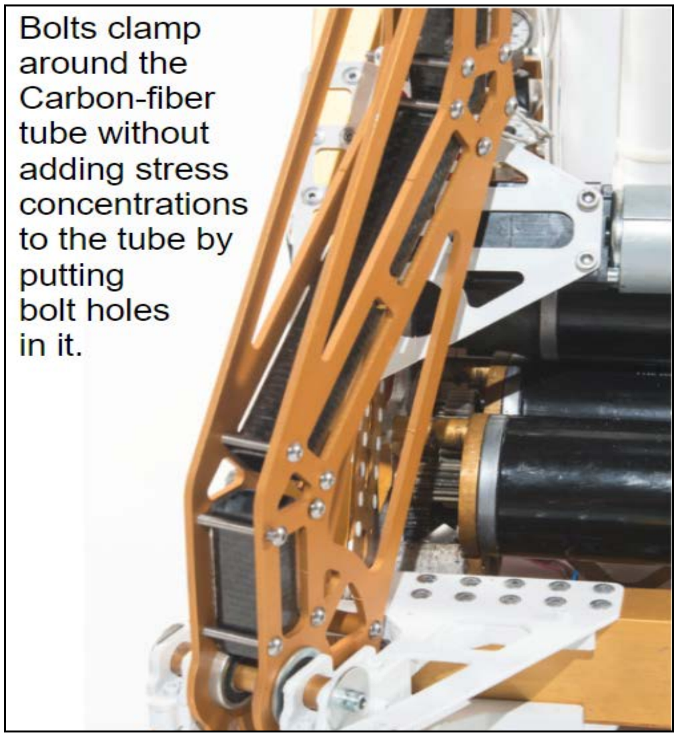
Onyx: a nylon-based filament with chopped carbon fiber in it. It is made for Markforged printers and is about 3 times stronger than ABS. Good enough for most printed parts on FIRST robots. One downside is it’s expensive at $190 for less than a kilogram of filament. On Markforged Mark Two printers it can be reinforced with fibers, such as carbon fiber, Kevlar, and fiberglass. Onyx parts reinforced with carbon fiber can rival the strength of aluminum parts. NylonX is a very similar filament and is a much cheaper alternative to Onyx.

B5 - Composites

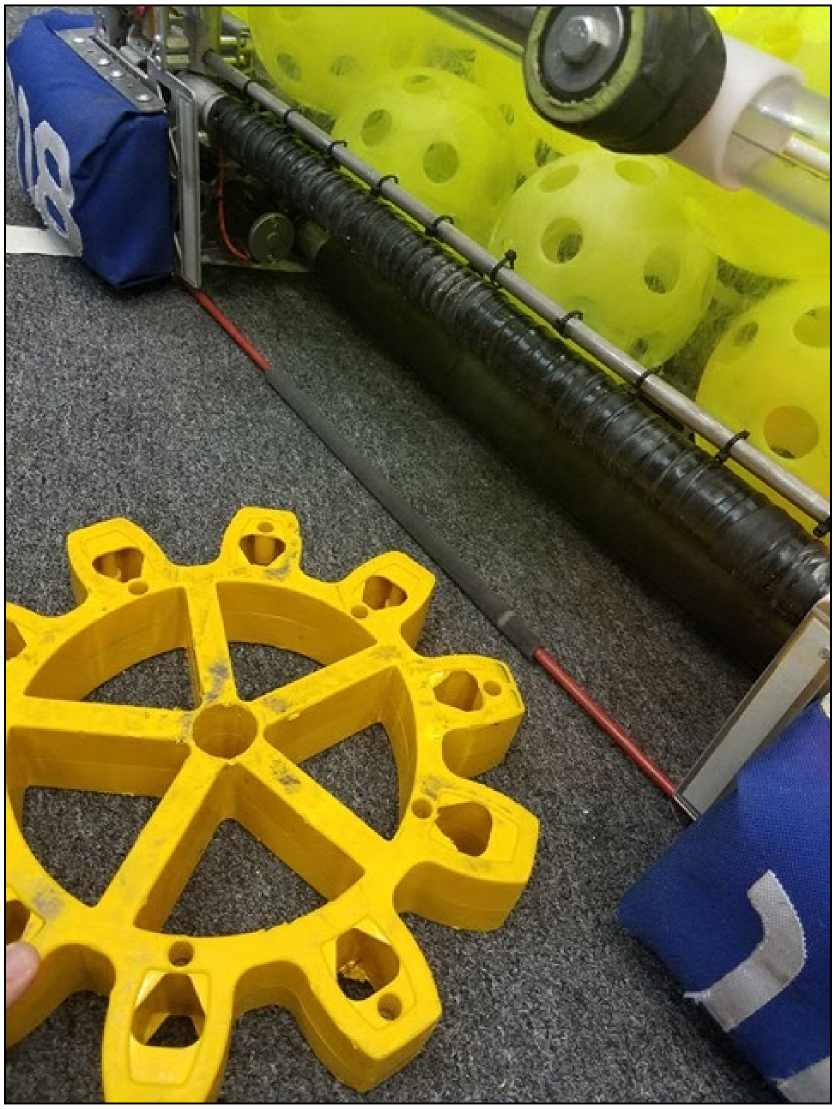
Composites: typically mixtures of non-metals that give the material unique properties that are sometimes desirable in FRC.

Plywood: a great material for prototyping and has its uses on competition robots. It is cheap, great in compression and lightweight. Some teams have designed their entire robots out of laser cut wood. Russian (Baltic) Birch as a very high quality, high ply plywood that is great for robot components. Russian Birch is more expensive than most plywoods but is cheaper than aluminum.

Carbon Fiber: a type of exotic composite comprised of carbon fibers mixed with plastic resin. Carbon fiber is extremely stiff, but is brittle. It is stronger than aluminum and typically weighs around 50% less. However, it is very expensive, especially in box tube form. Carbon fiber is hard to machine properly and has undesired stress concentrations around holes. Carbon fiber, like fiberglass also produces a fine debris when cutting so proper PPE such as a mask should be worn when working with carbon fiber. Certain methods of machining, such as cutting on a waterjet, can cause the material to delaminate. Carbon fiber box tube is insanely strong in compression. To avoid stress concentrations from holes in carbon fiber, it is best to clamp around the tube. The following is a picture of aluminum plates being used to clamp carbon fiber box tubes on Team 118's 2015 robot.

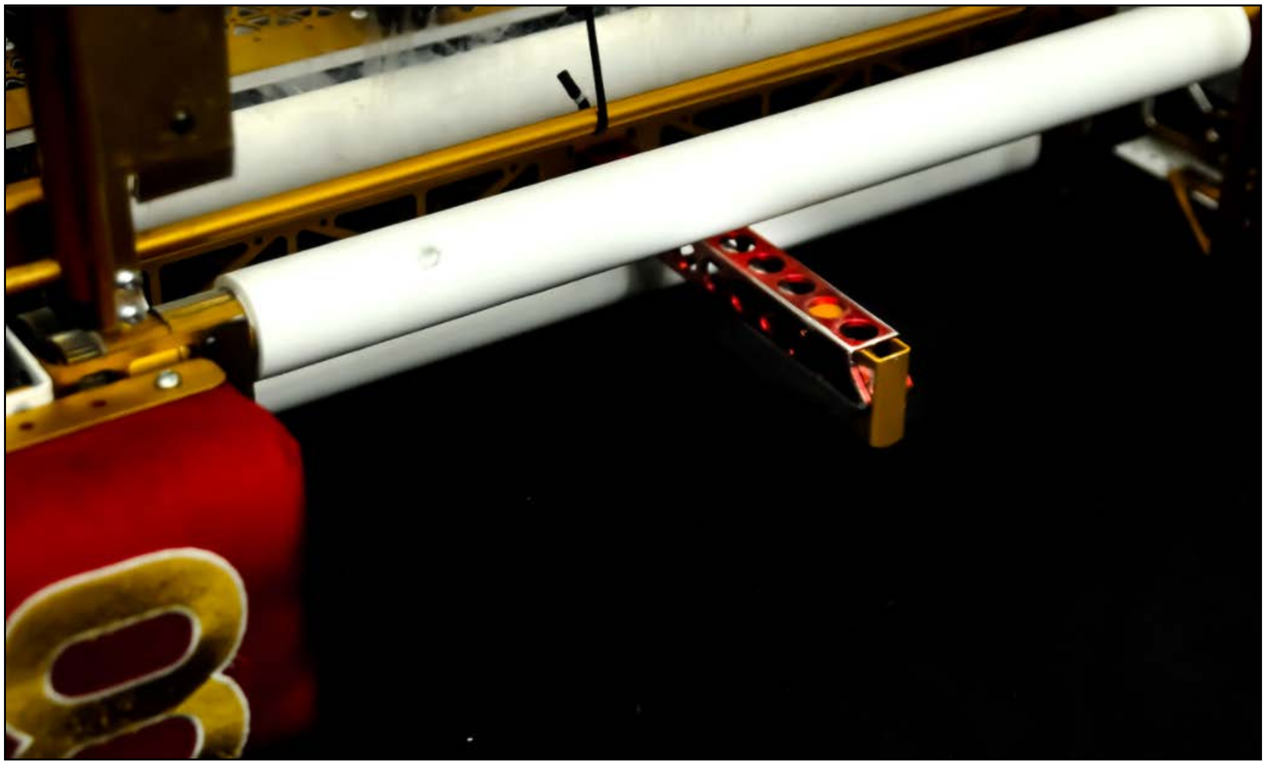

Bolts clamp around the Carbon-fiber tube without adding stress concentrations to the tube by putting bolt holes in it.

Fiberglass: a type of fiber-reinforced plastic (FRP). Fiberglass can be formed in very complex shapes, however it is a messy process that is typically unnecessary for FRC robots. Fiberglass rods however are very flexible and have their uses in FRC. Fiberglass produces fine debris when cut so proper PPE such as a mask should be worn. It also produces splinters that are very painful so you should avoid touching the cut edge of a fiberglass part. The extreme flexibility is ideal for shafts that need to bend without breaking. Fiberglass can also be printed into parts on a Markforged Mark Two printer. The following is a picture of a fiberglass shaft (red) being used as an intake roller for gears on Team 118's 2017 practice robot.



Latex rubber stretched over a fiberglass shaft on Team 118's 2017 gear intake.

High Temperature Silicone: a type of rubber material that is useful on intakes. Team 118 used this material in 2017 because it matched their color scheme. It turned out to be a very sticky material that works great on intakes. Like silicone tubing, it can be forced onto tubes using air to make rollers. Team 1678 has a tutorial on how to get the rubber tubing onto the rollers here.

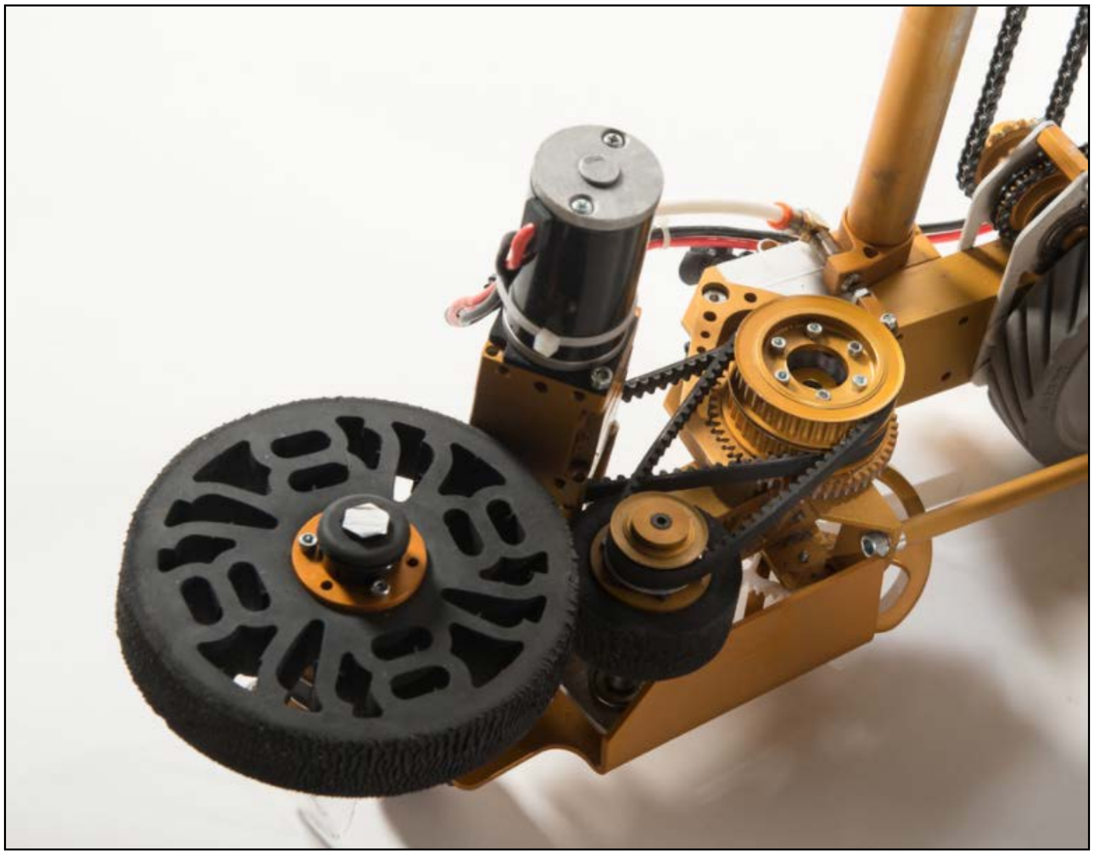


High temperature silicone rubber intake rollers on Team 118's 2017 robot.

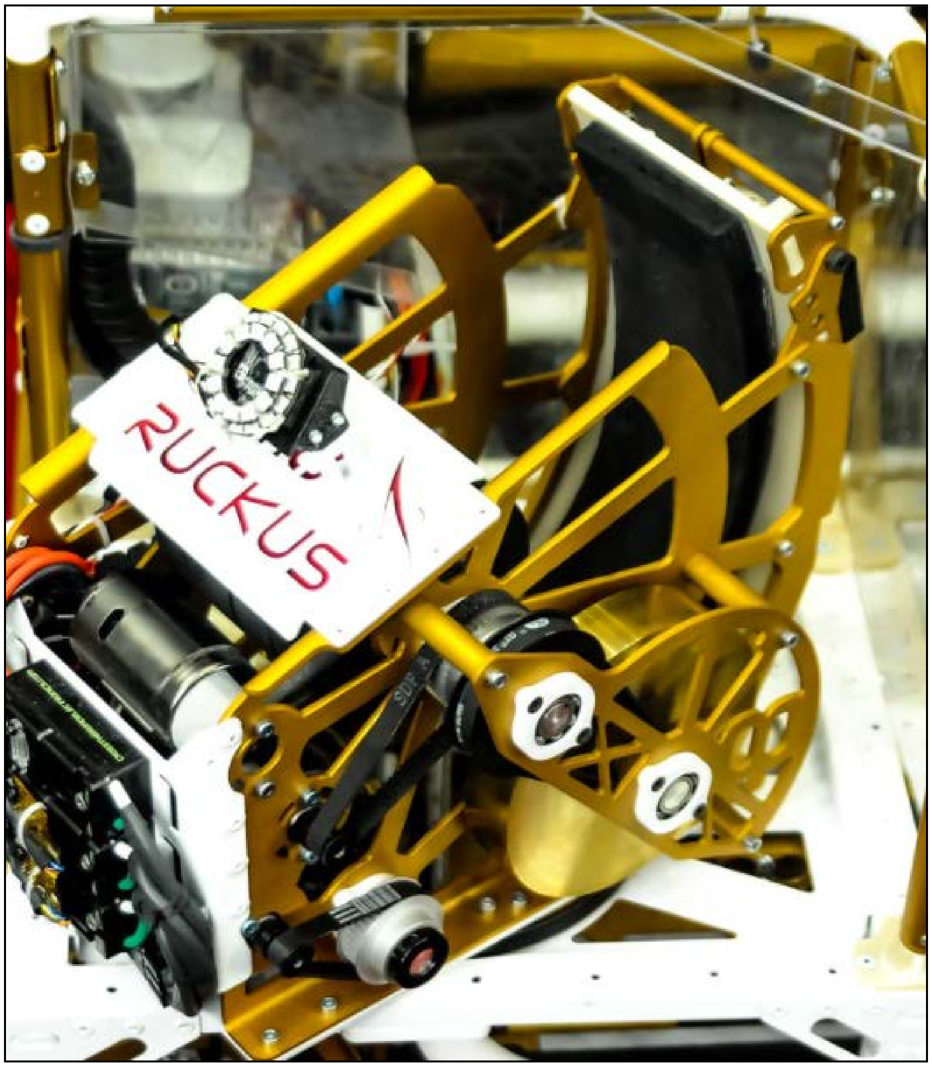
Latex (surgical tubing): stretchy and can be used as a rubber band, but is also sticky and can be used on intakes. Latex can also be used as a spring for catapults or linear punches. The ability to configure it to each specific need makes it a very versatile material.

Polyurethane: a rubber material that can be purchased and used to make custom intake wheels.

Neoprene: a rubber material that is used to make custom Space Wheels. Team 118 and Team 148 have used this material for custom intake rollers in 2015 and 2018. Wheels made of this material are also sold by West Coast Products.

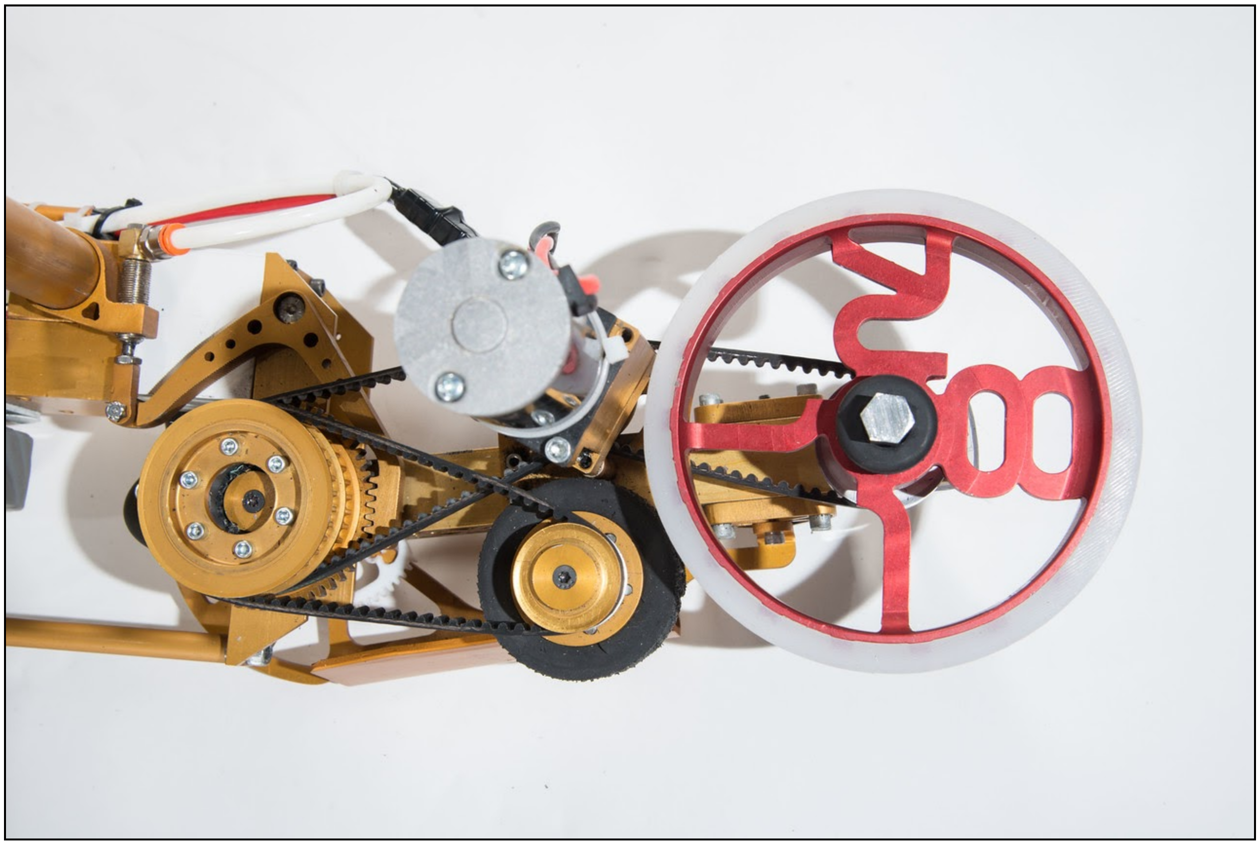

Custom neoprene intake wheels on Team 118's 2015 robot.

Gum Rubber: a sticky rubber that is sometimes used on the backboard of shooters. It is available from McMaster-Carr.



Gum rubber being used as a backer on team 118's 2017 shooter.

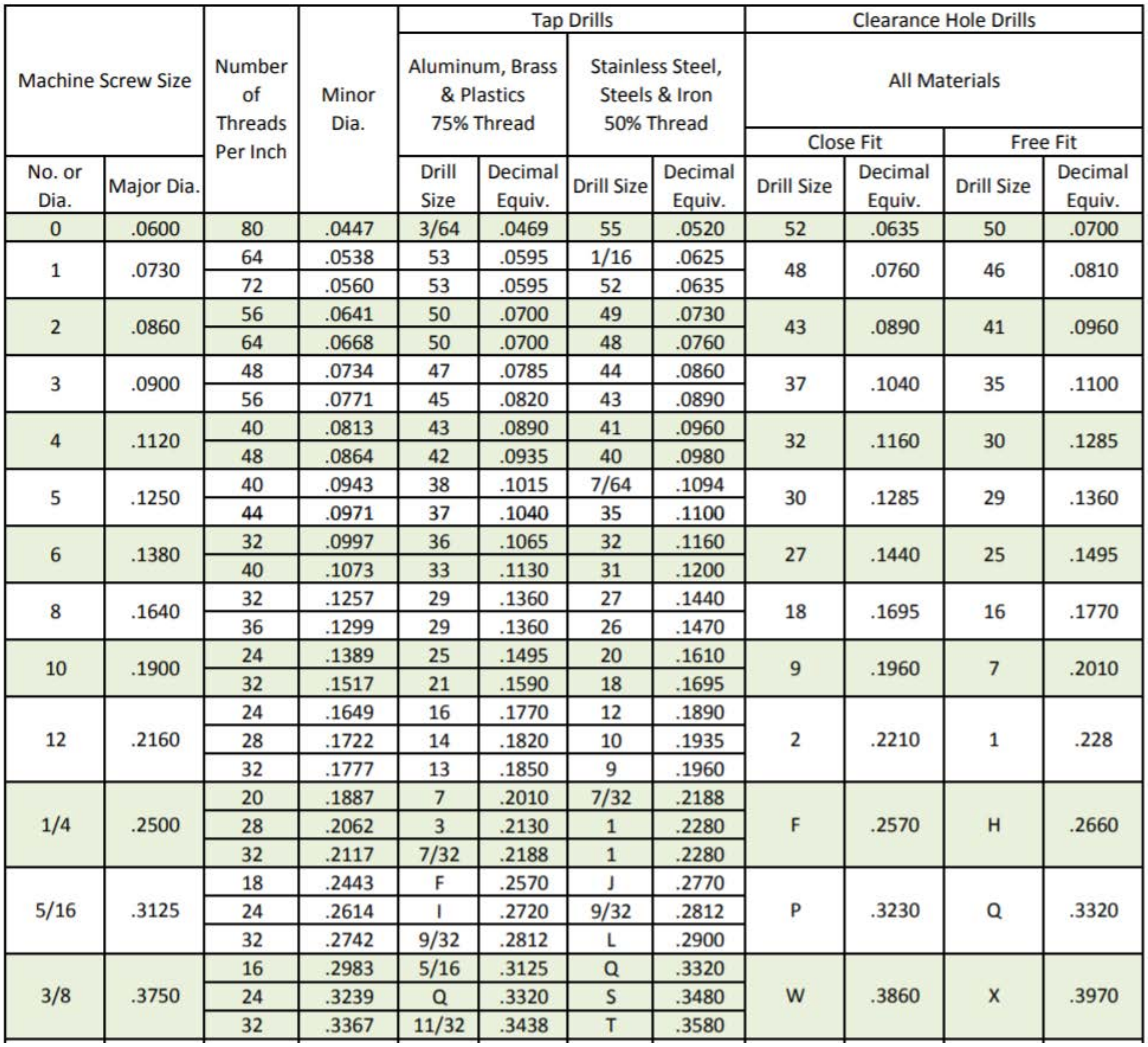
Molded Materials: teams have molded their own materials for intakes and drivetrain wheels using a variety of materials. This is usually done with over molding and almost never with injection molding due to costs. Team 118 has molded their own intake and drivetrain wheels in the past. With the large variety of COTS wheels available it is usually better to buy COTS wheels and focus resources elsewhere instead of making custom molded wheels.


Team 118's custom wheels with custom molded material on their 2015 robot.

Appendix C - Hole Size Charts

C1 - Bolt Tap and Drill Chart

Tap and Drill Chart



Machine Screw Size		Number of Threads Per Inch	Minor Dia.	Tap Drills				Clearance Hole Drills			
				Aluminum, Brass & Plastics 75% Thread		Stainless Steel, Steels & Iron 50% Thread		All Materials			
								Close Fit		Free Fit	
No. or Dia.	Major Dia.			Drill Size	Decimal Equiv.	Drill Size	Decimal Equiv.	Drill Size	Decimal Equiv.	Drill Size	Decimal Equiv.
0	.0600	80	.0447	3/64	.0469	55	.0520	52	.0635	50	.0700
1	.0730	64	.0538	53	.0595	1/16	.0625	48	.0760	46	.0810
		72	.0560	53	.0595	52	.0635				
2	.0860	56	.0641	50	.0700	49	.0730	43	.0890	41	.0960
		64	.0668	50	.0700	48	.0760				
3	.0900	48	.0734	47	.0785	44	.0860	37	.1040	35	.1100
		56	.0771	45	.0820	43	.0890				
4	.1120	40	.0813	43	.0890	41	.0960	32	.1160	30	.1285
		48	.0864	42	.0935	40	.0980				
5	.1250	40	.0943	38	.1015	7/64	.1094	30	.1285	29	.1360
		44	.0971	37	.1040	35	.1100				
6	.1380	32	.0997	36	.1065	32	.1160	27	.1440	25	.1495
		40	.1073	33	.1130	31	.1200				
8	.1640	32	.1257	29	.1360	27	.1440	18	.1695	16	.1770
		36	.1299	29	.1360	26	.1470				
10	.1900	24	.1389	25	.1495	20	.1610	9	.1960	7	.2010
		32	.1517	21	.1590	18	.1695				
12	.2160	24	.1649	16	.1770	12	.1890	2	.2210	1	.228
		28	.1722	14	.1820	10	.1935				
		32	.1777	13	.1850	9	.1960				
1/4	.2500	20	.1887	7	.2010	7/32	.2188	F	.2570	H	.2660
		28	.2062	3	.2130	1	.2280				
		32	.2117	7/32	.2188	1	.2280				
5/16	.3125	18	.2443	F	.2570	J	.2770	P	.3230	Q	.3320
		24	.2614	I	.2720	9/32	.2812				
		32	.2742	9/32	.2812	L	.2900				
3/8	.3750	16	.2983	5/16	.3125	Q	.3320	W	.3860	X	.3970
		24	.3239	Q	.3320	S	.3480				
		32	.3367	11/32	.3438	T	.3580				

C2 - Rivet Size Chart

Rivet Size	Drill Bit
1/8”	30
5/32”	20
3/16”	11

Appendix D - Additional Design Resources

How to Build Your Everything Really Really Fast
-Has a lot of different design and build methods relevant to FRC.

Spectrum Design Presentation
-A Google Slides presentation with information similar to what is in this guide made by Allen Gregory from Team 3847, Spectrum.

JVN Design Calculator
-Spreadsheet with lots of helpful calculators for FRC. Most useful for gearbox design. Created by John V-Neun from Team 148, Robowranglers.

610 Design Tutorials
-A design tutorial similar to this one written by Ryan Tam from FRC team 610 Crescent Coyotes. Has good design exercises.

Do’s and Don'ts for Parametric Modeling
-Good guidelines to follow when using parametric CAD software such as PTC Creo.

Using the Engineering Design Process for Design of a Competition Robot
-A great document by John V-Neun, focusing on the engineering design process and how it applies to building a competition robot.

973’s RAMP series
-Lots of info on various steps of the design process with good info on CAD (Solidworks and some OnShape, but contains info applicable to any parametric modeling software).

1678 Fall Workshop Videos
-A great set of videos covering a wide range of FRC topics, including mechanical and strategic design.

GameSense Behind the Design- Mechanism Design
-An episode of Behind the Design with Adam Heard from 973 discussing manipulators, Allen Gregory from 3847 discussing shooters, and Lucien Junkin from 118 discussing prototyping and intakes.

RAPID Sheet Metal Design Guide

-Has great general guidelines to follow when designing sheet metal parts. If you are serious about designing sheet metal, it's a must read.

Engineer's Edge

-Detailed charts for various things you ideally should take into account when designing. Lots of information: often more advanced (and dull) than you need for FRC.

MIT FUNdaMENTALS of Design

-A fantastic collection of MIT lectures and spreadsheets that focuses on the fundamentals of engineering design covering everything from linkages to power transmission to ethics.

Simbotics Solidworks Tutorials

-A series of YouTube videos with Solidworks CAD tutorials specific to FRC. Great if you are just learning Solidworks.

Triple Helix (2363) Builds a Drivetrain

-A series of YouTube videos with Team 2363, Triple Helix, of NASA Langley, demonstrating how to build a drivetrain step by step.

Appendix E - Additional General FRC Resources

Effective FRC Strategy Seminar Video (June 2012) by Karthik Kanagasabapathy
-Focusing on strategic design, match planning/execution, and scouting.

Recordings from other Strategy Seminars led by Karthik over the years.

-FRC Ask An Expert (November 2013)
-@2016 World Champs
-@2019 Houston World Champs
-Effective FIRST Strategies 2019, pt.1
-Effective FIRST Strategies 2019, pt.2
-Effective FIRST Strategies 2019, BONUS - Q&A

https://www.youtube.com/watch?v=SVacrE4sKig
-Strategic Design Seminar Videos by Team 1678 Citrus Circuits

http://frc971.org/workshops
-Workshops by Team 971 Spartan Robotics on CAD, Scouting & Strategy, Design & Prototyping, Controls, Storytelling through Multimedia, and much much more!

National Aeronautics and Space Administration

Johnson Space Center
2101 E NASA Pkwy
Houston, TX 77058
www.nasa.gov/centers/johnson

Robotics Alliance Project
www.robotics.nasa.gov

Version 1.01