## Contents

## 1.0 Introduction

The FTC Survival Guide is a tool intended to be used for rookie FTC teams. Covering topics from autonomous programs to getting sponsors, the FTC Survival Guide is designed to assist teams in their FIRST Tech Challenge careers.

The FTC Survival Guide was written by student-mentors with years of experience in the FIRST program. Using students with a variety of skills, the FTC Survival Guide was designed to help teams learn engineering concepts, create team sustainability, and to have fun in the FIRST program.

## 2.0 Starting Out

Starting out can be very difficult to organize for new programs, and many teams never achieve enough structure to sustain themselves. This is a quick guide to help create the foundation for a self sustaining FTC program.

### 2.1 Registration

FIRST makes it easy to register on their website, but if teams are unable to locate it, information can be found on the FIRST Robotics website. Follow the instructions on the website to register (link under **16.0 Resources**).

In North America, there is a $275 registration fee each season. Teams must pay this fee in order to be considered a legal team by FIRST. Failure to comply will result in the revocation of competition rights for that year, however, teams still retain the rights to their name and number.

### 2.2 Annual Cost

On average, rookie teams can expect to pay a total of $2,500 in team and event registration, parts, travel, and image. This is by no means a definite amount, so be prepared to spend more if necessary.

After the first year, this cost goes down significantly, as the robot kit of parts is reusable each season. Additionally, teams will most likely have leftover image and spirit materials, assuming they have not made any significant changes.

### 2.3 Fundraising

Teams are encouraged to pursue sponsorships from local, regional, and national companies. Most teams choose to display logos of their sponsors on their robot to show appreciation, as well as send them a “thank you” letter after the season has been completed. This increases the company’s chances of becoming a consistent sponsor, or to increase their contribution.

In addition to classical outreach events (see **13.0 Outreach**), teams will often run fundraisers to cover any costs not paid by sponsorships.

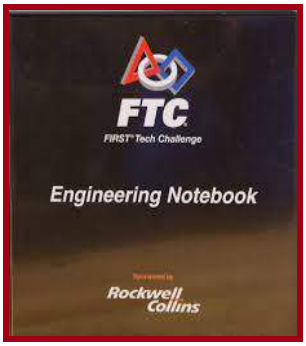
### 2.4 Team Structure

In competition, each team will need to have a set drive team. This drive team consists of two robot drivers and a coach. The drivers will need to be thoroughly trained in robot function and should be able to make quick fixed to the robot should it break before a match. The coach should be the person who knows the team’s strategy best, as well as the rules for the game.

There is no set or recommended structure for a team, but a sustainable team should have the following attributes:

- At least one adult mentor. Additional mentors are highly recommended (especially for larger teams), but not necessary for continuity.
- Many people working on mechanical and design aspects of the robot. In this area, there should be many ideas thrown out initially, so the team can find the best attainable strategy and design for them.
- At least one person who can work on the robot's programming. The more programmers available, the more advanced and stable robot code will be. However, if there are only a handful of people are on the team, only one is necessary.
- At least two people well versed in the electrical components of the robot. More people working on electrical is helpful, so no there are no errors in wiring.
- There should also be a non-STEM group of some kind, which handles the team's social media, public relations, sponsorships, and outreach events. This group should consist of many people, as it covers a vast spectrum of tasks.

**3.0 The Engineering Notebook**




The Engineering Notebook is an important part of FIRST Tech Challenge and the engineering process. It allows for documentation and reflection of a team's journey throughout their build season. Judges look for an Engineering Notebook that looks nice and has detailed content. The better the Engineering Notebook, the better chance that team has for awards. Not only are awards cool to win but they can also qualify teams for the next level of competition.

The FTC Game Manual Part 1 explains the requirements for an Engineering Notebook. Even though the requirements are updated each year with the release of the new game, the general guidelines usually stay very similar.

**3.1 Organization**

Organization is highly important to an Engineering Notebook. Judges look through dozens of notebooks at a single competition. They always appreciate a notebook that looks nice and is easy to follow. When looking through notebooks, judges won't read through every word on every page. Instead, they will look at the sections specific to the awards that team want to win (indicated on the Team Summary page). Teams should make their notebooks as organized as possible and it should be simple for judges to find the pages the team wants them to read.

A table of contents is the easiest way to organize an Engineering Notebook. When writing a table of contents, teams should include page numbers and titles of the sections in the notebook. The Team Summary page, award submission forms, and Business/Strategic Plan should be near the front of the notebook so judges see them right away. The Team Section and Engineering Section should come after that and should take up the majority of the notebook. Page numbers should be listed on a corner of each page.

**3.2 Team Section**

The Team Section is where a team's demographic should be documented. This should include team information, biographies, sponsors, and outreach information. This can be typed or written to the team's preference (ensure written notebooks are legible). The Team Section is used by judges especially for non-STEM

(non-Science Technology Engineering Mathematics) related awards such as the Connect Award and the Motivate Award. It is important to have an aesthetically pleasing and easy-to-follow Team Section.

Break the Team Section into different subsections based on what activities the team participates in (these section could include outreach, biographies, business tours, sponsors, etc.). The table of contents should direct judges to each section easily.

Content should be factual and simply worded. Teams should not brag about their accomplishments but should instead explain their goals, actions, and impact. When writing, teams should refer to their mission statement and explain how their actions and accomplishments reflect their overall ideals.

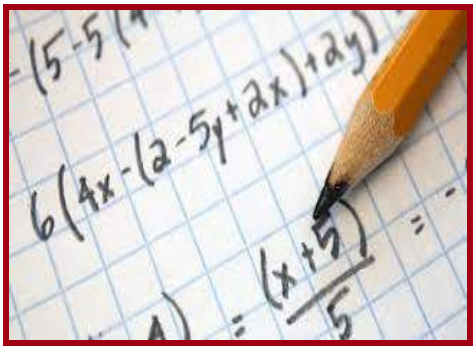
Biographies should give a brief description of each member of the team. An example of a biography of a single member is given below. Pictures of members are also highly recommended. Pictures fill blank space, look nice, and assign a face to a name for judges.

**Name:** Johnny Titus
**Role:** Mechanical and CAD
**Years in Robotics:** 3 years
**Why did you join robotics?** I've always had a love for STEM learning. I loved building with legos and tinkering with various kits at home. When I reached high school, I heard about the FIRST program and thought I would join FTC.

### 3.3 Engineering Section

The Engineering Section is where team's should log what they accomplish each day as well as any notes on STEM related accomplishments the team wants the judges to know. Judges use the Engineering Section as criteria for many awards including PTC Design Award, Rockwell Collins Innovate Award, and the Think Award. The Engineering Section is required for every Engineering Notebook.

A team's daily logs should be written that day when the information is fresh in the team's mind. Teams should document their goals, what they actually accomplished, and then reflect on their successes and failures on each day. Including pictures and drawings of the robot at different stages is a good way to show how the robot is progressing. However, teams should not rely on images alone. The written content should be able to express a team's progress on its own while the images act as supporting evidence.



Apart from the daily logs, teams should also include other STEM based pages in their Engineering Section. This could include CAD documentation, applied math or physics equations, and programming screenshots. These pages should also explain the team's process and reasoning in each category.

Some awards ask for teams to tab pages in the Engineering Section that indicate pages that teams would like judges to especially consider for award winning criteria (see more about tabbing pages in **Awards 11.0**).

### 3.4 Business/Strategic Plan

Ensuring a team is established on a sustainable platform is crucial to the future success of that team. The Business or Strategic Plan (they are the same thing and can be used interchangeably) is a formal document created by a team at the beginning of their season but is expected to change throughout the season based on changing goals and problems that occur. The Business Plan should be professional and easy to read.

The Business Plan documents who a team is, what their goals are, and how they plan to accomplish those goals. Some of this information is financially based so teams should communicate with their coaches, mentors, and sponsors when writing. Go to **Resources 16.0** for FIRST's Business/Strategic Plan template.

A strong mission statement is a very important part of the Business Plan. The mission statement explains what the team's purposes are. A team's mission statement should reflect the goals and achievements of that team. A team's Engineering Notebook should reflect and explain how the actions of that team support their mission.

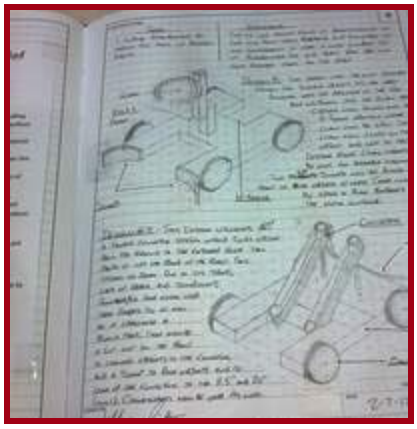
The Sustainability section of the Business Plan explains the goals of the team for that year and how they plan to accomplish them. The Team Action/Implement Plan should include goals, actions, the people responsible for accomplishing the goals, and the projected completion date. The Team Financial Statement should be a working document throughout the season. It should establish a team's fundings at any given time. Teams should include the various ways they plan to spend and raise money, projected and actual prices/funds, and an explanation of what each event is and how it raised funds.

The Outreach and Recognition part of the Business Plan is also important. Here, teams should list all of the outreach they have done throughout the year and how it has impacted their community. In addition, teams should also list the awards the team has ever won in the Recognition section of their notebook.

### 3.5 Standing Out

Judges see dozens of notebooks at each competition they attend. Making a notebook that stands out to judges increases a team's chances for awards.

Decorating the outside of the notebook can create appeal for a notebook. It is advisable to use a general and constant color scheme throughout a notebook. If a team has a "team color" that goes along with their theme/image then that color should be used. Teams can use pictures, bedazzlement, glitter, texturing, and more to decorate the covers of their Engineering Notebooks. However, remember not to over decorate. Find a balance between decoration and professionalism.



The inside of a notebook should keep a similar theme as the outside. If a team has a particular color used in their theme/image, perhaps writing in that color ink could add a unique effect to their notebook. However, legibility should

come before creativity. If a team's color is yellow, writing in that color ink on white paper isn't the best idea.

Drawings and visuals are a great way to catch judges' eyes. Images should be professional and reliable to the content on that page.

Using decorative divider pages for each section could add some life to a notebook between sections. These divider pages could carry a similar theme as the front cover. These could be on a different color of paper as the rest of the notebook.

There are also many Engineering Notebook examples online.

**4.0 Strategy**

Kickoff just happened. The game video and manual was just released. Now what? Strategy is the development of a team's plan over the first few days of build season. This plan should dictate how the rest of the team's season goes including the robot that is built and the alliance partners they choose.

**4.1 Reading the Game Manual**


The game manual is the most important document to a team's strategy. It gives all information about the game. In FTC, there are two ame manuals. For strategy, the FTC Game Manual Part 2 is the required resource. Both game manuals are on the FIRST website.

Understanding the game manual is very important. A good team will evaluate a game manual quickly and effectively. The biggest part of learning how to evaluate a game manual is experience. Looking at previous game manuals can be good training for an upcoming season.

Treat the game manual like a contract that between the team and FIRST. The information given is absolute and exact. Like a contract there are terms, penalties, and sometimes even loopholes. "Breaking the Game" refers to a strategy that uses a loophole in the game manual to tilt the odds heavily in that team's favor. FIRST usually keeps the loopholes to a minimum in their games but if there ever is one, FIRST may alter the game manual. Keep up to date with new versions of the game manual and the FIRST Q/A.

As previously stated, everything a team needs is in the game manual. All rules, penalties, ways to score, field dimensions, and field element descriptions are given in the manual (they are usually organized and sorted in a table of contents). All of this information should be used in the strategy process.

Additionally, the game manual uses repeated terms and phrases to keep their intentions consistent. For example, a game piece (such as a block) will always be referred to by the same name. The important "game specific" terms are bolded, capitalized, or otherwise indicated each time each term appears. These terms are defined in the back of the manual. Each time a game specific term is used, that word means that definition verbatim.

### 4.2 Finding a Strategy

Developing a strategy early in the season is incredibly important. Being able to focus on specific goals helps speed along fabrication and assembly of the robot.

A strategy should consist of what the robot will do on the field, how the robot will perform this task, and what robots are optimal for alliance selections for that strategy.

A strategy is NOT a design. These are separate. Strategy has nothing to do with mechanisms or the physical robot. Strategy is simply a list of tasks and priorities to consider when designing.

To start finding a strategy, make a list of all tasks a robot could perform on the field. This includes the scoring objectives and more indirect actions like defense. Once teams have that list, they should rank each task in order of difficulty.

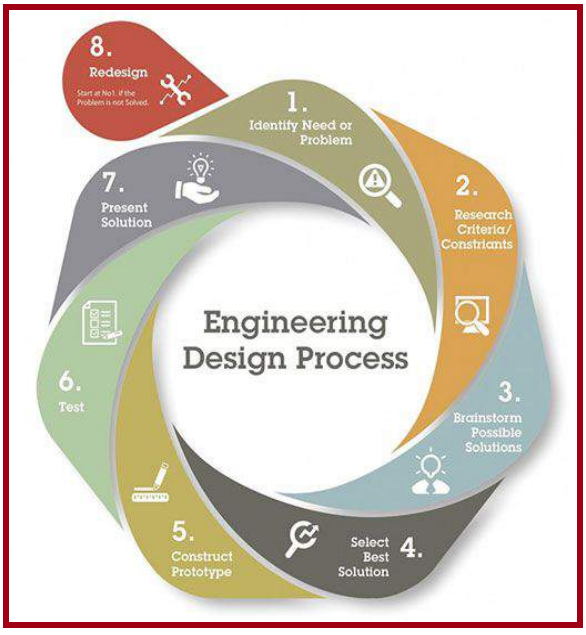
Chances are the more difficult tasks will be worth more points. It is also probable that the most elite teams in the world will be able to perform every task or be able to perform one task extremely well.

Each team's strategy will be different based on their resources and experience. A rookie team may want to focus on a simpler task that is still worth many points to appeal to more experienced potential alliance partners.

When in doubt, a pro-con chart can be a useful tool when deciding between two or more strategies. Combining two strategies isn't a bad idea, but ensuring the two strategies work well together is crucial.

Teams should always strategize based on their resources. It is obvious a robot that does everything perfectly is usually the best strategy. However, most teams don't have the money, time, and experience to make a robot that is good at every task. It is up to each team to evaluate what they can and can't do.

After a strategy is decided on, it is also important to emphasize the strategy to the "design team". This is especially important when discussing different drivetrain options. A six-wheel tank drive is very durable and fast, but may not be as maneuverable as an Omni-holonomic drivetrain. The strategy dictates the design, not the other way around.




### 5.0 Design

When building a robot, it is important to have a plan when going into the season. It may seem easiest to just start building, but usually an unorganized team ends up with an unfinished robot. If a team wants to get the most done in as little time as possible, it is vital to maintain an effective design process.

### 5.1 What is a Design Process?

A design process is simply a plan for how a team will build their robot. In FTC, a team usually gets 2 to 3 months before their first competition to prepare their robot. In order to get there with time to spare, a detailed design process should be followed.

Usually, a design process splits the allotted time into different parts, which will be outlined below. A typical design process will look like this:

1. Strategy Development
2. Prototyping/Proof of Concept
3. Robot Design (CAD)
4. Construction

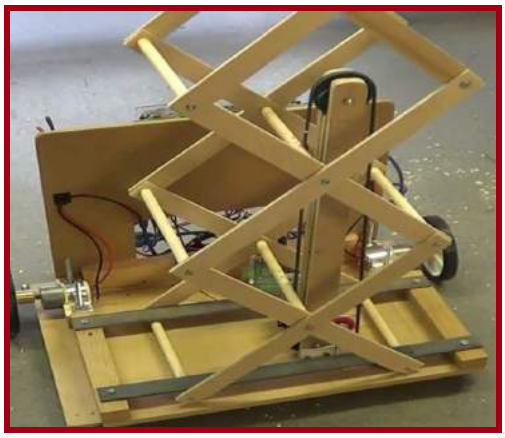
These four steps, each taking a considerable amount of time, should be organized into the timeframe each team is given. For example, many teams attempt to develop their strategy during the first few days of meetings, and jump right to prototyping after that. However, for some teams, they may elect to spend more time in strategy than in prototyping. However, a successful team shouldn't skip a step completely, because each is important in its own way.

### 5.2 Strategy Development

The first thing a team should keep in mind before they build their robot is what it will actually do on the field. In summary, the team should gather together and decide as a group what they think they can accomplish, keeping in mind their capabilities and resources. For more information on strategy development, see **4.2 Finding a Strategy**. Once a strategy is decided, the team should move on to prototyping ideas on how they will carry that strategy out.

### 5.3 Prototyping

Prototyping is an important aspect of any design process, in or outside of robotics. This gives even those who aren't so mechanically inclined to try out an idea. However, before building starts, a team should meet together and share ideas or concepts on how to carry out the strategy. It's beneficial to separate the robot into different mechanisms to prototype.



For example, in Velocity Vortex (2016-17), some mechanisms a team could prototype were a drivetrain (wheels, treads, etc) and a shooting mechanism (flywheel, catapult, etc). These mechanisms could each be prototyped to test out different ideas.

Once the team has decided what mechanisms they want to prototype, they should begin some research. There are tons of resources on the internet to aid teams that might be short on ideas or are looking for some tried-and-true concepts. Chances are, there was some similar game in another year, and it's always good to research some successful robots from that competition! Chief Delphi (found in **16.0 Resources**) is a great resource for sharing ideas! Furthermore, they should document these teams' mechanisms in their Engineering Notebook and give them credit. Believe it or not, judges usually love when teams incorporate others' mechanisms or ideas into their design, as long as they don't flat-out copy them or not give any credit. Plagiarism is still frowned upon, even with robots!

Nonetheless, don't be afraid to incorporate previous designs. Imitation is the most sincere form of flattery, so don't be upset if another team incorporates ideas they see on social media or at competition. It's all in the spirit of gracious professionalism!

When a someone builds a prototype, they should never try to build the final version of the mechanism. That will come with time, but a prototype simply exists to prove that the concept would work. *No prototype will be perfect, and most won't look pretty.* However, that doesn't mean they don't work. Even if a prototype fails, it still served its purpose: to test whether or not a concept would work in real life. Also, keep time in mind when

prototyping. Try building mechanisms out of scrap material or wood as opposed to actual metal that could be used for the real robot. Don't waste material! Once a team has proven how they intend to accomplish the tasks at hand, it is time to move into designing the final geometry of those mechanisms.

### 5.4 Robot Design (CAD)

Once prototypes have served their purpose, a team should think about how they intend to package everything in the volume they're given. An 18" cube may seem like a lot of space, but it soon proves to be quite limiting. This is why CADing the robot beforehand is vital to a team's success. Prototypes are meant to be temporary, but the actual mechanisms need to be robust and accurate. Although they should be based on the concept proven by the prototype, a team should often think about how they can make it stronger, better, or more structurally sound.

Furthermore, a team should consider packaging as a legitimate issue. If a team can't fit all their desired mechanisms in the sizing cube, the point of building them is moot. CAD is best used for this purpose: to visualize how all the mechanisms can fit together so a team doesn't have to worry about sizing after they've built the mechanisms. Once a team has their mechanisms designed in CAD and proven how they can all fit together, the time to build has begun!

For more information on CAD, see **6.0 CAD**.

### 5.5 Construction

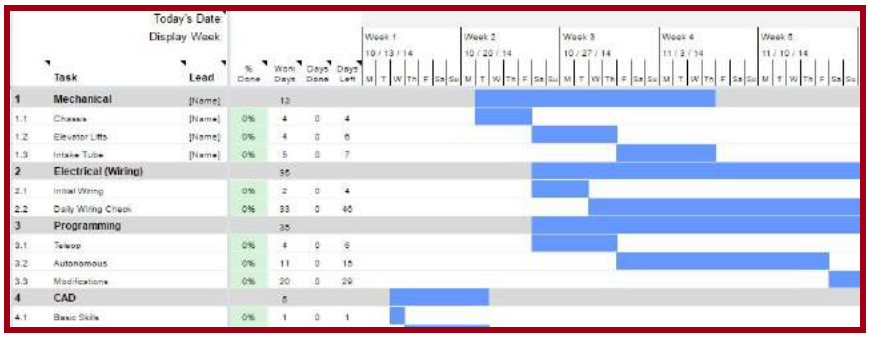
Finally, once a team knows what their robot will look like, they can begin to create it in real life! However, one thing to know is that this step does not necessarily have to follow the design process. Many teams elect to design and build mechanism by mechanism. For example, once the drive train has been finalized in CAD, the fabrication team begins to assemble it and get it driving with code while the design team continues to CAD other mechanisms. They inherent risk of this is that the already-assembled mechanisms have to be changed to make room for others, so a team can decide together if they want to attempt this.

If a team wants to build the robot only after it has been completely designed, it is beneficial to have teams of one or two people assemble each mechanism. This allows each person to continue building their preferred mechanism and also to teach others who may not be as mechanically inclined. Always save ample time in the season for construction, because there is never a lack of mishaps during this process. Whether it be shortage of supplies or limited attendance, a team should always prepare for problems when building the robot.

Once a team has completely built their robot, the last thing left to do is finish up documenting what they haven't already written down in their Engineering Notebook and drive practice. The value of drive practice going into a competition is unrivaled, so make sure to allot some time for this. After all this, a team should be prepared to compete.

### 5.6 Organization

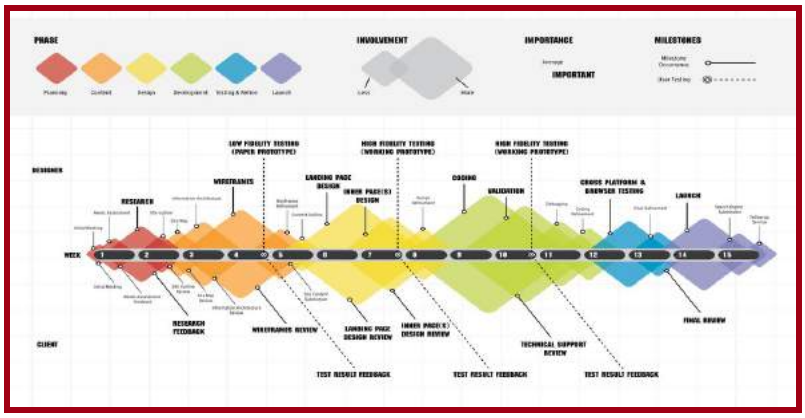
Although this process may seem simple enough, it is always important to organize the design process using some sort of chart or timeline. Here are a couple of ideas to consider, but don't be afraid to be creative.



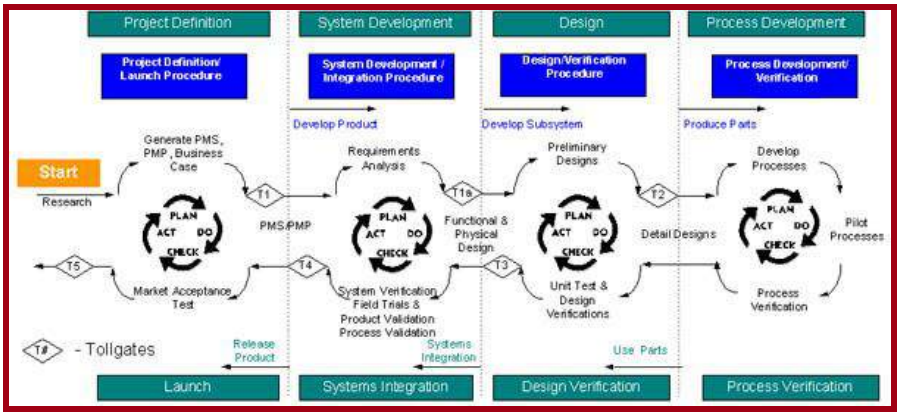

The Gantt Chart is a popular way to organize a design process, and is used by many professionals. It provides a visual representation of the build season and

allows a team to recognize how much time they are given to perform a certain task. By giving each task and subtask a certain allotted time frame and tracking each task by day, a team can organize their design process in a visually appealing way. For a template on how to make a Gantt Chart in Excel or Google Sheets, check out www.ganttexcel.com to download one for free.

Slightly simplified from the Gantt Chart, a project timeline is also very popular in the professional landscape. The example above utilizes different colors to denote different phases of the process, and different sizes of diamonds to represent the workforce needed to accomplish each task. This organizes the process in a more linear format, and is easier to understand. The dates of completion are given places to overlap, and the schedule is represented as one long line.




Finally, the last commonly-used format of process organization is the flowchart. While this format is a bit less specific with due dates or people involved, it provides a clearer idea of what is needed before the team can move on to the next step. The example above shows an example design process for a real-life engineering project, which can easily be adapted into a robotics setting. Even though this requires a bit more explanation to fully flesh out, it is more visually appealing to judges.




No matter what format teams decide to use to illustrate their design process, it's important to have a copy of it in their Engineering Notebook. A process can be beautiful, but if a team doesn't follow it, it doesn't speak for the team as much as a carefully-followed process does.

**6.0 CAD**

Computer Aided Design, or CAD for short, is a branch of computer software used to aid the design, manufacturing, modification, analysis, and optimization of a tangible item.

CAD has been developed for decades and is an industry standard across the world. The invention of CAD has drastically cut down on development time, manufacturing costs, and overhead in general. The ability to track parts and assemblies on the computer with attached documentation has created a streamlined engineering world where it is now easier than ever to create complex ideas and designs.

Here in FTC, CAD software is available to teams worldwide.

### 6.1 Deciding on a CAD Software

CAD software is offered to all FTC teams world wide on the first website or on the PTC website (if a team member registers as a student). The link to this website can be found in **16.0 Resources**.

In FTC, there are three different software programs to model the robot: PTC Creo Parametric, Autodesk Inventor, and Solidworks. All of these softwares will perform the same task. The one a team selects will be largely up to preference.

PTC runs a yearly grant for struggling teams in need of resources, and one of their eligibility requirements is that a team uses their software.

Students in the United States or Canada likely already have Autodesk Inventor installed on school computers. Autodesk is a sponsor for Project Lead the Way, an industrial technology curriculum with some classes offering college credit at the end of the school year, similar to an AP course.

The final option is Solidworks. Solidworks is very popular with industries such as design and machining, and is very common throughout universities in the United States. Since Solidworks is used by many companies worldwide, a team member wanting to use a continuous software throughout life could use Solidworks.

### 6.2 Project Data Management

Project Data Management (PDM) and is used as an industry standard worldwide to keep CAD assemblies up to date and accessible. To keep CAD models up to date, some solutions are built into the software.

Applications such as PTC Windchill and Autodesk 360 are clients that can be applied directly into a CAD software.

However, we recommend a 3rd party solution to PDM: GRABCAD Workbench. GRABCAD Workbench allows online data syncs with a 3D viewer in browser and on any mobile device. Most PDM solutions have a steep learning curve. GRABCAD Workbench is plug and play. It has an intuitive design that allows for ease of access for anyone involved in the project.

### 6.3 Tutorials

There are many general tutorials for CAD softwares. However, for cases like converting files to prepare for 3D printing, it may require a specific tutorial that can be found on the manufacturer's website or on YouTube.

PTC Creo has many tutorials on YouTube and because of the similarity between Creo parametric 2.0 and Creo parametric 3.0 the tutorials for one software can be used for the other. Creo also has a built in tutorial software that will track progress.

Much like Creo parametric, Autodesk has many tutorials available on YouTube that are very helpful for single topics. Because of project lead the way most CAD classes in American and some Canadian middle and high schools use Autodesk inventor as a part of the curriculum.

Solidworks is also most commonly used in real industry, so a mentor will likely have some experience with the software. There are also videos on YouTube for Solidworks.

**7.0 Mechanical**

Mechanical is the physical building of the robot. A team must use the tools and materials at their disposal effectively to have a productive and successful season.

**7.1 Tools**

FTC requires a wide range of simple inexpensive tools, hammers, hex wrench sets, screwdrivers. Tools that fill specific purposes like cutting metal, measuring, drilling, or chain breaking are essential and the quality and efficiency of tools is only limited on what a team's budget is. But here are the basic tools required to build a robot.

Cutting

- (essential) Dremel- a dremel is a small handheld rotary tool with exchangeable heads. A dremel can be used for sanding and cutting metal or other materials. Prices range from $25-$200.
- (essential) Hacksaw- a Hacksaw is a saw that can be used to cut thin pieces of metal. Prices are usually under $20
- (Luxury) Bandsaw- A bandsaw is exactly what is sounds like a band with saw teeth. Band saws can cut almost any material at any thickness. Prices range from $200-over $1000 depending on quality.
- (essential) Files- Cutting and metal will make the edges sharp and rough. Files are inexpensive and can easily get rid of rough pieces.

Drilling

- (essential) Hand drill- a simple hand drill with a drill bit set can suffice the cutting requirements for FTC. Prices for Hand drills can range from $25-$100 depending on quality and kit. A small drill bit kit costs around $20.
- (luxury) Drill Press- Drill Presses have the power and capacity that Hand Drills can lack. Prices range from $100-over$1000 depending on quality

Others

- Clamps- Vice grips can be used to gain leverage in any situation where leverage is needed same with any other type of clamps.
- Chain breaker- tool that breaks chain
- (luxury) 3D printing- 3D printing can print almost any part that is necessary.

**7.2 Materials**

Most materials that are required for FTC are given in the KOP. The most basic forms are listed below:

- Aluminum is the standard metal in FTC, and for good reason. Aluminum is tough, light and inexpensive.
- Steel is only really used in FTC to weigh robots down in specific slots when there centre of mass is of and is causing tipping.
- Plexiglass is great for being the outer shell of robots. Its most common application is forming a barrier between a robots inner workings and the field.
- Velcro is useful for things on the robot that need to be detached often for example the battery or phone.

- Essentially big tube versions of rubberbands, surgical tubes are great for intakes or keeping electrical wires in order

### 7.3 Mechanisms

A mechanism is a system of parts working together in a machine. In FTC many different types of mechanisms are used to accomplish the same tasks. The difference between two mechanisms that achieve the same end is the time, resources, and efficiency of the mechanisms. Choosing to pursue mechanisms that take large amounts of time but are not very efficient are a waste of precious build time. But making a mechanism that takes little time but isn't very efficient can be outdone by other teams. Finding a balance between time and quality is important.

Starting at the front of the robot are intakes. Intakes are used to pull game objects into other mechanisms that can manipulate the objects. By far the most used intake mechanism is a spinning shaft with other protrusions(ex. Pic to the right). This mechanism is easy to build and is very efficient. There are other mechanisms used to control game piece but not necessary “intake” them into the robot. A arm or crane is a great example of a mechanism that can take control of a game piece outside the robot.


Every good strategy has a focal point or goal, the main mechanism is the the mechanism that fulfils this goal. This is the most important singular part of the robot and should require a lot of thought, planning, and effort. Determining what to build in order to accomplish any high priority goals is important, so heres a list of commonly used main mechanisms along with a pro and con list for each.(Not all mechanisms will be viable depending on the game.)

A scissor lift is multiple congruent straight pieces lined together and bolted together through the center. Scissor lifts are most commonly used to lift game elements.

| Pros | Cons |
|---|---|
| • Strength<br>Scissor lifts are able to handle lifting very heavy objects due to the amount of torque they require. | • Torque<br>Scissor lifts require large amounts of torque to get started.<br>• Unreliability<br>Due to the application of large amounts of torque scissor lifts are susceptible to breaking or warping.<br>• Spacing<br>Scissor lifts often require two lifts in order to be effective because the lifts must often require something spanning between them. |

Linear Slides are used mostly for controlling game elements far outside the size constraints of a robot. Most Linear Slide systems are built with pulls systems on both sides so only one motor is needed to power them. Linear Slides by themselves don't require very much torque but when adding mechanisms on top of the system the heavier the mechanism or length the more torque that is required. The ability to use different pulley systems makes Linear Slides slightly more versatile, for example one side could be used to bring the slides up and the other side used to bring the slides back down. This system would make a retractable slide system. But of the slides only purpose is to lift a large game element both sides could be used to lift up giving the system more stability and direct transfer of power.

| Pros | Cons |
|---|---|
| • Easy to build frame<br>• Only uses one motor<br>• Extends to almost any length vertically or horizontally<br>• Only need one in most situations | • Need two to lift big objects<br>• Take up large vertical space in robot<br>• Changes center of mass when extended, increasing risk of tipping |

Flywheels are a mechanism rarely used in FTC but are useful nonetheless. Flywheels accomplish the same things as a Catapult but use a completely different launch premise. Flywheels involve pinning a ball against a fast spinning wheel to launch the ball. Flywheels only work with balls.

**8.0 Control Systems**

While building a robot, all of its motors and sensors require power. The FTC control/electrical systems are dedicated to delivering electrical energy from the 12 volt battery to all of the mechanisms that require power. FIRST has made the electrical systems for FTC robotics simple, but it is important that the electrical tools and components are used properly.

**8.1 Electrical Components**

**Battery:** Either the 12V Tetrix battery, 12V Matrix Battery, or 9.6V Matrix Battery is allowed for FTC control systems. The battery plugs into the power distribution module via a Tamiya connector. Current Tetrix and Matrix batteries include a 20 amp fuse to prevent the battery from short-circuiting and for safety purposes.

**Power Distribution Module:** All power from the battery will flow through the CORE power distribution module to the other modules. It has one Tamiya connector to input power from the battery, one USB Mini port to connect it to the phone, six powerpole connector outputs for motor and servo controllers, and seven USB A ports for the other CORE control modules. Lastly, it has a 20 amp fuse needed to protect itself and the other CORE control modules it is connected to from getting burnt out.

**Motor Controller:** Used to power any legal FTC motors. Each module can power two separate motors, with an optional encoder

plug-in for both motors. Most robots will utilize at least two of these modules.


**Servo Controller:** Used to power the robot servos. Each module can power up to six servos, so typically only one is necessary.

**Legacy Module:** The legacy module is used to make the CORE power distribution module compatible with older LEGO NXT. This means that the old motor controllers, servos, and sensors can still be used with the newer Android systems.

**Device Interface:** The device interface is used to provide inputs and outputs signals for optional devices such as sensors. It has eight digital I/O ports, eight analog input ports, two analog output ports, two PWM output ports, and six high speed $I^2C$ ports.

### 8.2 Electrical Tools

By attempting to make the control systems for FTC intuitive, FIRST has made it so few to no tools are needed. A majority of the sensors and motors will come with a connector on tem, so they only need to be plugged it. The following tools can be used functional and organizational purposes to make the control systems on a robot more efficient and organized.

**Zip ties:** It is necessary for the wires on a robot to be organized and out of the way so they do not interfere with the robot's mechanisms. One of the easiest ways to fix disorganized wires is by using zip ties. Tying wires together and securing them onto the robot's frame will eliminate most safety issues of getting wires caught on something and disorganization.

**Anderson Powerpoles:** Many of the FTC control systems use the red and black anderson powerpole connectors. To manipulate these connectors, teams have the option of purchasing a wire stripper and crimper to add these connectors onto wires. Teams can use these to better organize their wiring.

**Powerpole Connectors**

**Wire Stripper Crimper**

**Wire**

### 8.3 Miscellaneous Electrical Requirements

While planning out a robot it is important to consider the accessibility of certain components like the power switch on the CORE power distribution module, the battery, and the robot's phone holder. The FTC Game Manual Part One specifies certain requirements for the control systems.



The Main Robot Power sticker should be clearly visible and near the power switch on the power distribution module.

A warning sticker is only necessary if the robot moves during initialization, and should be easy for judges to see.

**9.0 Programming**

Programming is vital to competition and good code can be the difference between 1st place Alliance Captain, and last place in qualifications. This guide will teach teams the basics of FTC Programming in Java.

**Vocabulary**

- **Class:** A group of code
  - **Example:** Tele-Op, Autonomous, and Robot Config are all classes
- **Package:** A folder of classes
  - **Example:** Sample Code, and TeamCode are all packages
- **Method**: A repeatable sequence of code with parameters
  - **Example:** Drive method is an example of a method
- **Variable:** A storage of information under an alias
  - **Int:** A variable with no decimal places (don't use)
  - **Double:** A variable with a few decimal places (use)
  - **Long:** A variable with *a lot* of decimal places (don't use)
  - **Float:** A variable that can technically hold any amount of decimal places but loses accuracy as it gets longer (**don't use)**
  - **String:** An alphanumeric variable. Examples include sentences, names, output, etc (use)
- **Array:** A set of data in a linearly accessible pattern
  - **Example:** Flywheel speed intervals
- **Library:** A premade set of code often for hardware
  - **Example:** The robot library

**Library Terms**

- **DcMotor:** Motor Actuator
- **Servo:** A servo actuator
- **Encoder:** Rotation measurement device
- **Sensor:** A detection device
- **Gamepad:** The controller
- **Driver Station:** The phone by the drivers that connects to the **gamepad**
- **Robot Controller:** The phone that sits on the robot and connects to the **Driver Station**

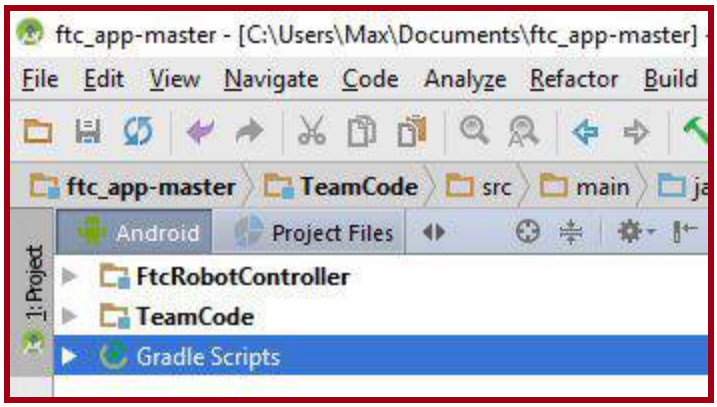
What matters most about the library is the hardware. The library is what allows you to convert inputs from the **gamepad**, and actuate them on the robot. The logic and syntax will be native to Java, meaning it is the same in all Java applications (including many FRC Robots).

**9.1 Setting Up: The Directories and Classes**

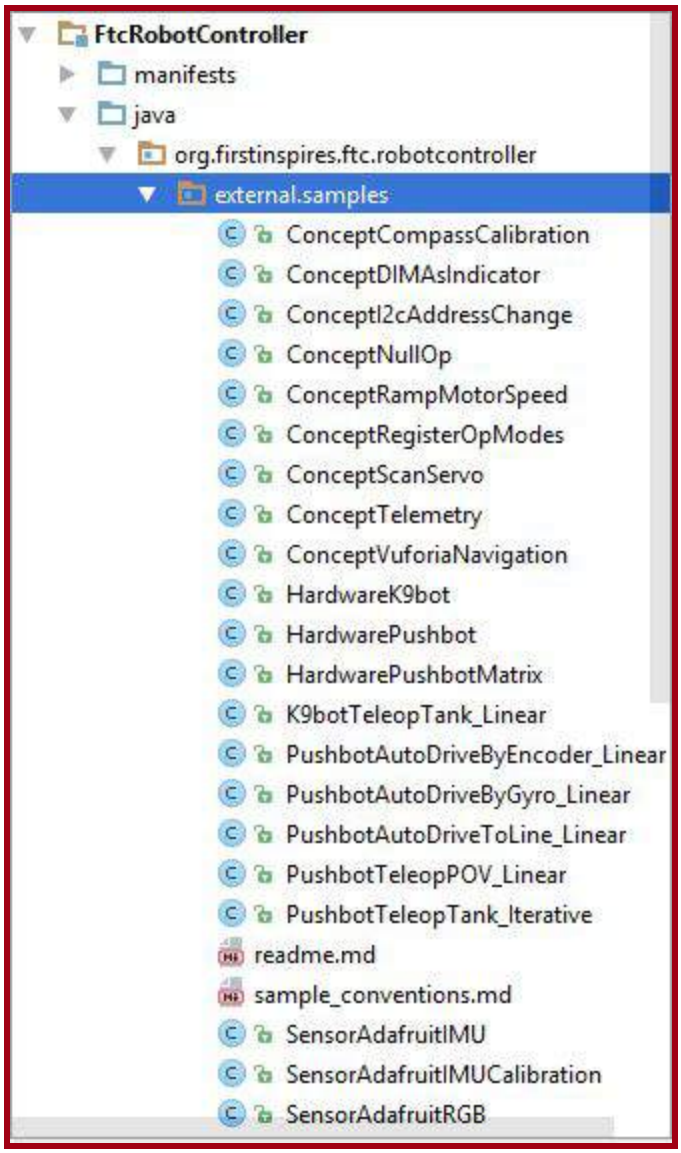
The best way to learn code is to write it, so first you need to set up the workspace. Assuming you already have Android Studio set up based on the instructions in the game manual, the first thing we need to do is import the

library and sample code. At the moment of this being written, the latest version of FTC Java is available for download on GitHub. In order to keep this up-to-date, simply

1. Go to your search engine of preference (Google, Bing, etc) and type in “FTC Java GitHub”. And click on the link.
2. Go to the green button in the top top corner and click it.
3. After that, click “Download as Zip” and allow the file to download.
4. Unzip the file into whatever location you want to be your programming directory
5. Open Android Studio. If you have a project open, go to the top left and click File>Close Project
6. Click Import Project and go to your programming directory. Select your unzipped file.
7. Allow the Gradle to build and look for the “Project Explorer”. It is a layout of the file framework of your project. If it isn't there it will be a folder with branches icon on the left side of the screen. The project explorer looks like this:




8. As of now, there are two useful directories inside the folder. One contains the sample code, and the other is called TeamCode. TeamCode, as you may have guessed, is where you store your Team’s code.
9. Now it's time for you to learn how to find things. Head over to your project explorer and, as it changes every year, go to the directory that *is not* TeamCode. Look for the ExternalSamples folder, or something of the like.




10. There will be a class called “HardwarePushbot”. Copy this into your TeamCode directory and refactor-rename it to the name of your robot. Do this by right clicking the class in the project explorer and hover over “refactor”. Then select rename. Then, delete all parts of the file that include sample info and rename the object so that it corresponds with the class. **Do not remove any imports.** It should look like this, however do not edit anything outside of what is shown in the below image. Those bits of the code are vital for the functioning of your robot.

```
package org.firstinspires.ftc.teamcode;

import ...

public class myRobot {

    /* local OpMode members. */
    HardwareMap hwMap           =  null;
    private ElapsedTime period  = new ElapsedTime();

    /* Constructor */
    public myRobot(){

    }

    /* Initialize standard Hardware interfaces */
    public void init(HardwareMap ahwMap) {
        // Save reference to Hardware map
        hwMap = ahwMap;

    }
```

11. Next find the Iterative Tele-Op template. You may need to add some things to make it look like the code below. That is fine, do that. You can test Linear, but Iterative tends to be more stable. Again, you need to remove all the pre-built info. When you finish this remove the @disabled above the class definition. If you don't, it won't show up. Also reconfigure the robot object so it works talks to *your* robot. It should look like this. Don't remove anything from the imports

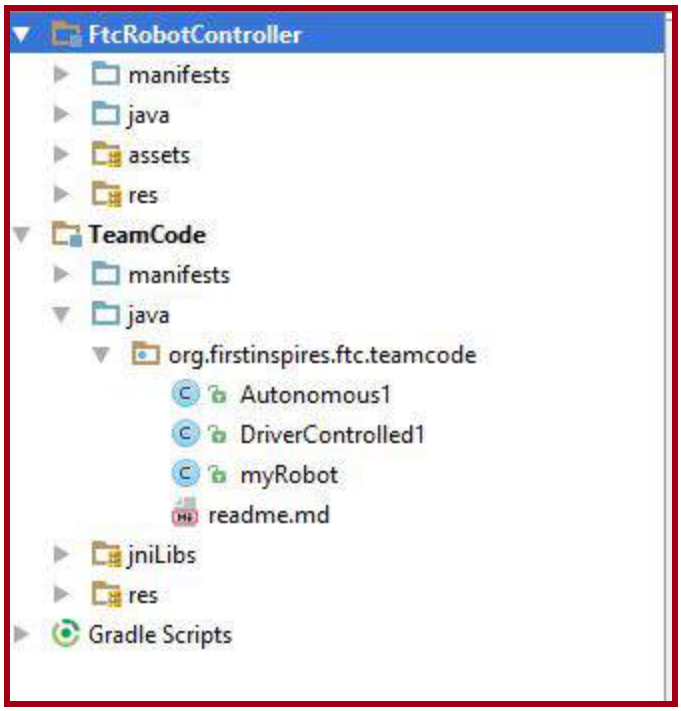
```
/.../
package org.firstinspires.ftc.teamcode;

import ...

/**...*/

@com.qualcomm.robotcore.eventloop.opmode.TeleOp(name="Tele-Op
public class DriverControlled1 extends OpMode
{
    /* Declare OpMode members. */
    myRobot robot = new myRobot();
    private ElapsedTime runtime = new ElapsedTime();

    /*
     * Code to run ONCE when the driver hits INIT
     */
    @Override
    public void init() {
        telemetry.addData("Status", "Initialized");

    }

    /*
     * Code to run REPEATEDLY after the driver hits INIT, but
     */
    @Override
    public void init_loop() {
    }
```

```
    /*
     * Code to run ONCE when the driver hits PLAY
     */
    @Override
    public void start() { runtime.reset(); }

    /*
     * Code to run REPEATEDLY after the driver hits PLAY but before they hit ST
     */
    @Override
    public void loop() {
        telemetry.addData("Status", "Running: " + runtime.toString());

    }

    /*
     * Code to run ONCE after the driver hits STOP
     */
    @Override
    public void stop() {
    }

}
```

12. Finally find the Auto template. Remember to add things if they aren't there. This should be Linear. Clear it out and remove @disabled. Remember to reconfigure the robot object. It should look like this:

```
/.../
package org.firstinspires.ftc.teamcode;

import ...

/**...*/

@com.qualcomm.robotcore.eventloop.opmode.Autonomous(name="Pushbot: Auto Drive By Time", group="Pushbot")
public class Autonomous1 extends LinearOpMode {

    /* Declare OpMode members. */
    myRobot robot   = new myRobot();   // Use a Pushbot's hardware
    private ElapsedTime     runtime = new ElapsedTime();


    @Override
    public void runOpMode() {

        robot.init(hardwareMap);

        // Send telemetry message to signify robot waiting;
        telemetry.addData("Status", "Ready to run");    //
        telemetry.update();

        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        // Step through each leg of the path, ensuring that the Auto mode has not been stopped along the way

        runtime.reset();
    }
}
```

13. You're done setting up your project. If you did everything right, you should have a Tele-Op, Auto, and Robot-Name class. Your project explorer should now look like this:




\

**Setting Up: The Robot Class**

The robot class is what defines all of the motors, sensors, servos, encoders, etc. You have to initialize all of these objects as null, and then you have to define them to an ID that is the same on the phone. The robot class is also where you will store all of your methods, so that they can be called from every other class and don't need to be redefined.

The best way to demonstrate this is through example.

**To define a motor:**

```
package org.firstinspires.ftc.teamcode;

import ...

public class myRobot {

    /* local OpMode members. */
    HardwareMap hwMap           =  null;
    private ElapsedTime period  = new ElapsedTime();

    DcMotor myMotor;

    /* Constructor */
    public myRobot(){
        myMotor = hwMap.dcMotor.get("motor");
    }

    /* Initialize standard Hardware interfaces */
    public void init(HardwareMap ahwMap) {
        // Save reference to Hardware map
        hwMap = ahwMap;

    }
```

**To define a sensor:**

```
package org.firstinspires.ftc.teamcode;

import ...

public class myRobot {

    /* local OpMode members. */
    HardwareMap hwMap           =  null;
    private ElapsedTime period  = new ElapsedTime();

    ColorSensor sensor;

    /* Constructor */
    public myRobot(){
        sensor = hwMap.colorSensor.get("colorSensor");
    }

    /* Initialize standard Hardware interfaces */
    public void init(HardwareMap ahwMap) {
        // Save reference to Hardware map
        hwMap = ahwMap;

    }
```

**To define a servo:**

```
import ...

public class myRobot {

    /* local OpMode members. */
    HardwareMap hwMap           =  null;
    private ElapsedTime period  = new ElapsedTime();

    Servo myServo;

    /* Constructor */
    public myRobot(){
        myServo = hwMap.servo.get("servo");
    }

    /* Initialize standard Hardware interfaces */
    public void init(HardwareMap ahwMap) {
        // Save reference to Hardware map
        hwMap = ahwMap;

    }
```

**An example Robot class:**

```
package org.firstinspires.ftc.teamcode;

import ...

public class myRobot {

    /* local OpMode members. */
    HardwareMap hwMap           =  null;
    private ElapsedTime period  = new ElapsedTime();

    DcMotor leftFront;
    DcMotor rightFront;
    DcMotor leftBack;
    DcMotor rightBack;
    Servo buttonPusher;
    ColorSensor sensor;

    /* Constructor */
    public myRobot(){
        leftFront = hwMap.dcMotor.get("leftFront");
        rightFront = hwMap.dcMotor.get("rightFront");
        leftBack = hwMap.dcMotor.get("leftBack");
        rightBack = hwMap.dcMotor.get("rightBack");
        buttonPusher = hwMap.servo.get("buttonPusher");
        sensor = hwMap.colorSensor.get("colorSensor");
    }

    /* Initialize standard Hardware interfaces */
    public void init(HardwareMap ahwMap) {
        // Save reference to Hardware map
        hwMap = ahwMap;

    }
```

## 9.2 Programming Basics

Syntax is important. The only way that the robot can understand what you are trying to do. If you have improper syntax, it will not allow the code to build, and the robot will not receive the code, however modern IDE software, such as Android Studio, have built in debugging software that tells you what your error is, and often how to fix it. In Android Studio, the best way to do this is to put your cursor on the line with the error and press Alt-Enter. A dialog box will appear with options on how to resolve the error. If the error is logical, you are going to have to do some thinking for yourself. A technique that works very well for solving logic problems is whiteboarding. Draw out the plan of your code on a whiteboard with details of what sets what variables, how variables are modified, how Arrays are called, ect. Then talk out the process and see if you notice the error. If you can't figure it out, ask a mentor or a fellow teammate, even if they aren't a programmer. Anybody can understand variables and if statements, so walk your teammate through the code and see if they have any ideas.

Now, when it comes to teaching syntax, the best method continues to be examples.

Personally, I recommend taking the CodeCademy course on Java before continuing past this point. It teaches you the basics of Java through interactive examples, and has you writing code with guidance. It will provide a much better method of learning the basics of Java than a read-only guide can. Check **16.0 Resources** for the website.

If you choose to take the alternative path, this guide will take you through an *extremely basic* version of Java Syntax. If you are reading this and have decided to ignore the previous recommendation, please reconsider. It's 4 hours of your life that you will make up in *not* tearing your hair out trying to understand what you're copying and pasting from the FTC Forums.

### Variables:

Variables carry information. You have already read about the types of variables in the **Vocabulary** section, but what was never covered is defining and using those variables.

To make a variable exist, but not have a value, simply go to the spot where the class starts, before the first method, and type the name of the variable with a semicolon ';' at the end. It looks like this:

```
double power;
double negativePower;
```

To give it a value, add an equal sign with a value in between the name and the semicolon. That would look like this.

```
double power = 1;
double negativePower = -1;
```

**Methods:**
Before you start with methods, allow one thing to be pounded into your head. **All methods should be defined in the robot class. This way they can be called from every OpMode and do not need to be redefined, along with being good practice for future programming.**

In most programming languages, methods can be used to call a task repeatedly.

The cleanest and most efficient way to program a robot is by using **methods.** Every function can be turned into a method with **parameters**.
A **parameter** is something that can be modified to change a repeating task. For example, you could create a method for *drive,* with parameters left and right. This way, instead of writing

```
public void loop() {
    telemetry.addData("Status", "Running: " + runtime.toString());

    robot.leftFront.setPower(gamepad1.left_stick_y);
    robot.leftBack.setPower(gamepad1.left_stick_y);
    robot.rightFront.setPower(gamepad1.right_stick_y);
    robot.rightBack.setPower(gamepad1.right_stick_y);
```

Every time you want to drive, you could make a method that calls it in one line, like so:

```
public void loop() {
    telemetry.addData("Status", "Running: " + runtime.toString());

    robot.drive(gamepad1.left_stick_y, gamepad1.right_stick_y);

}
```

Methods need to have the **void** parameter , and if you want the method to be accessible everywhere, you need the **public** parameter.

The method definition looks like this

```
public void drive() {
```

**Method Variables** can also be used. These are what allow methods to change their actions based on the situation they are needed for. For example, for the drive method, you might want leftPower and rightPower **Method Variables** which can be used for input as well. This way the method can work for Auto and Tele-Op. **Method Variables** are defined inside of the () after the method is defined. To define one, put it's type (in this case a double) and it's name. If you want two, put a comma and do it again with a different name. You can have as many as you want.

Here is what that should look like:

```
public void drive(double leftPower, double rightPower) {
```

What might not make sense is how this can be used for driving, but remember that the sticks return values between 1 and -1 in the form of doubles, so when we call the method, we can directly call the joystick values. By calling the method above, the robot's drive train is controlled by the Y axis of the sticks of Gamepad1 in a tank drive pattern. Calling the method needs to happen inside of loop() in drive and would look like this:

```
public void drive(double leftPower, double rightPower) {

    leftFront.setPower(leftPower);
    leftBack.setPower(leftPower);
    rightFront.setPower(rightPower);
    rightBack.setPower(rightPower);
```

(Sorry for the sudden new info, but it is needed for this example, .setPower(); sets motor rotation speed. You will learn more about this very soon.

And then could be called in the OpMode like this:

```
public void loop() {
    telemetry.addData("Status", "Running: " + runtime.toString());

    robot.drive(gamepad1.left_stick_y, gamepad1.right_stick_y);
```

This code alone would power a drive train.

Methods should be used for *everything.* In good code, almost nothing stands alone in the **main method**, which is the method that runs continuously. Driving, Intakes, Auto, etc should all live inside of methods. Advanced Tele-Op code should have at least one method for each mechanism and most likely more.

**Variable and method properties:**
Variables carry information. Methods carry tasks. However, they share many properties. If you look at common Java code, you will see the words **public, private, void, final,** and **static,** often. These all have different meanings that are vital to working code.

**If statements:**

As Tele-Op code is looping, meaning it repeats constantly, if statements, or conditionals, are vital. For example, if you want a servo to be in position A when resting, and position B while a button is pressed, you would use a conditional. It must be said, that in Java and most programming languages, the "!" means not, and the name of a boolean with nothing else means true. If you had

By the way, buttons are booleans.

```
double posA = 0;
double posB = 90;

//gamepad1.a is "A" button

if (gamepad1.a){
    robot.buttonPusher.setPosition(posA);
}
```

It would mean that buttonPusher is set to posA when button A is pressed. However, if it looked like this:

```
double posA = 0;
double posB = 90;

//gamepad1.a is "A" button

if (!gamepad1.a){
    robot.buttonPusher.setPosition(posA);
}
```

The opposite would be true. It would set to posA if the A button *wasn't* pressed.
It would only run if boolean was false. Else if and else are also important. Else if allows for another check for something else if the initial if statement failed. You could test for boolean being true, and if it was false, you could have an else if to check if else for button B being false. That would look like this:

```
double posA = 0;
double posB = 90;
double posRest = 180;

//gamepad1.a is "A" button

if (!gamepad1.a){
    robot.buttonPusher.setPosition(posA);
}
else if (gamepad1.b){
    robot.buttonPusher.setPosition(posB);
}
```

Note: You can have as many else if statements as you like

An else statement can also be added to run if all else if statements failed. For example, if you have three else ifs, and they all fail, else would run. That would look like this:

```
double posA = 0;
double posB = 90;
double posRest = 180;

//gamepad1.a is "A" button

if (!gamepad1.a){
    robot.buttonPusher.setPosition(posA);
}
else if (gamepad1.b){
    robot.buttonPusher.setPosition(posB);
}
else {
    robot.buttonPusher.setPosition(posRest);
}
```

Note: You **will not** want to use while loops, because if a while loop is activated nothing else can be activated. The best way to do this is to create an if statement like as show above.

**Telemetry:**

Telemetry can be used to return data from the robot to the Driver Station for in-match info, and more importantly, debugging. To add a line of telemetry, simply use addData(). The information will need a key, and actual values to return. These values can be strings. Let's return what posA is.

```
double posA = 0;
double posB = 90;
double posRest = 180;

telemetry.addData("posA", posA);
```

Next let's return the position of the servo as a string for easy understanding for the driver. Judges also like this type of stuff. The easiest way to do this is to bind it to the already made, however there are many ways to do it. You could use (servo).getPosition, or you could bind it to the already made code.

```
double posA = 0;
double posB = 90;
double posRest = 180;

//gamepad1.a is "A" button

if (!gamepad1.a){
    robot.buttonPusher.setPosition(posA);
    telemetry.addData("servoPos", "Servo is left");
}
else if (gamepad1.b){
    robot.buttonPusher.setPosition(posB);
    telemetry.addData("servoPos", "Servo is right");
}
else {
    robot.buttonPusher.setPosition(posRest);
    telemetry.addData("servoPos", "Servo is resting");
}
```

**Keep in mind all of this should exist inside methods in the robot class. It is in the OpMode for example sake *only*.**

### 9.3 Programming Tele-Op

Tele-Op is the easiest to program. In reality, a basic Tele-Op could be programmed in 3 minutes. The easiest way to program Tele-Op is done by using methods.

### Using Motors, Servos, Sensors, and Encoders

To call something from the Robot class, where all the actuators and sensors are defined, you need to call the Robot class for the actuator. You will use the term you defined as the key for your Robot class, which is usually just Robot. You can find that "key" on the line that you create an instance of your Robot. That is found at the top of any OpMode code and looks like this:

Note: If it isn't there create it. It is **vital.**

```
/* Declare OpMode members. */
myRobot robot = new myRobot();
```

The string after the name of the class before the equal sign is your "key". In this picture it is purple

Remember when we set up those motors, encoders, sensors, and servos in the robot class? Well now we're going to use them to make the robot move.

The basic command for setting motor power is .setPower(); In order to set a motor, the syntax is (robotClass (as long as it isn't in the robot class, which is should be)).(motorName).setPower(); For example, if we set rightBack to full clockwise speed, it would look like:

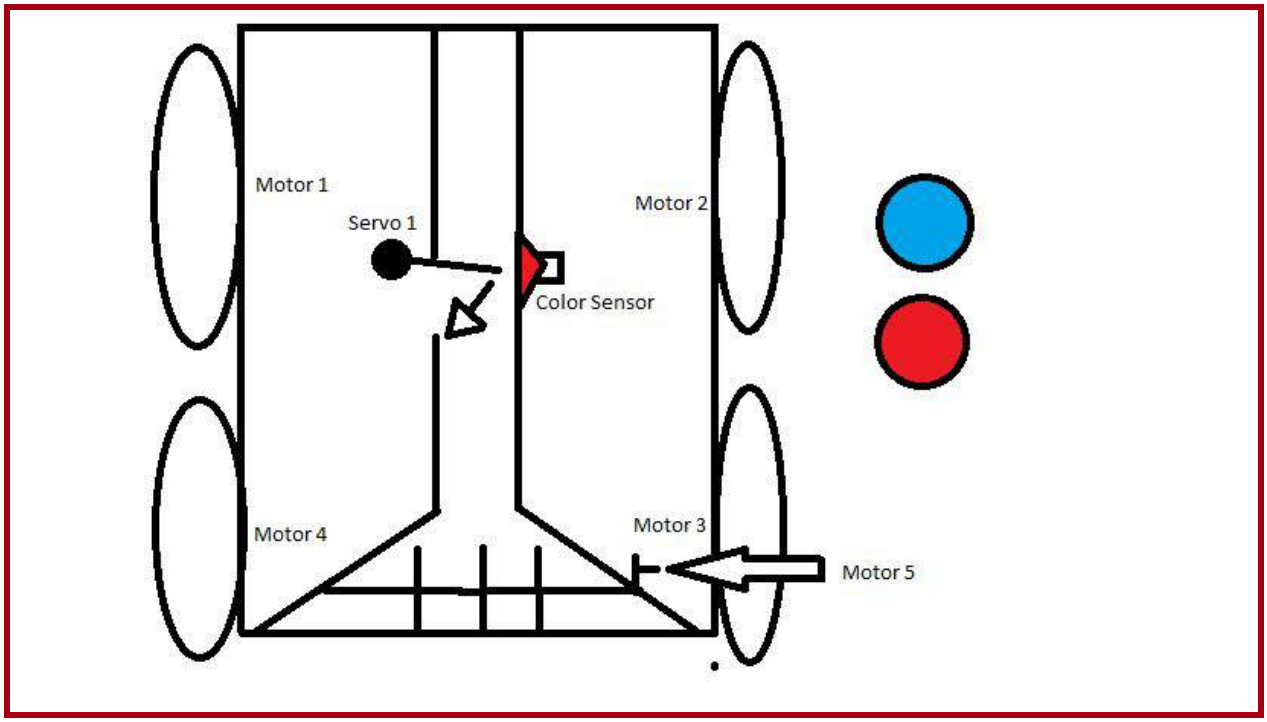
```
rightBack.setPower(1);
```

Servos work on position. All servos act slightly different, so you're going to have to use trial and error, but the setPower equivalent is setPosition(); The formatting is otherwise the same. (robotClass).(servoName).setPosition(); To set the ButtonPresser servo to 90 degrees, it would look like this:

```
robot.buttonPusher.setPosition(90);
```

Sensors are different, and each one of them is different, so you are going to have to do some research on your specific sensor, as there are plenty of resources online, but for now we'll sample using a color sensor. Color sensors return (sensor).red(), (sensor).green(), and (sensor).blue() as baseline. You can use telemetry to test them out and determine their return ranges.

**That's the basics, now let's program a robot.**




This robot does three things.

1. It picks up balls in the intake (Motor 5)
2. Checks if they are the right color using the Color Sensor
3. Drops balls out using a servo

Assuming you've already set up your project and have your robot class, your Tele-Op class, and your Auto class, the first thing to do is set up the robot IO.

Go to your empty robot class, which should be empty, and add all of your devices under the keys on the phone. When you are finished it should look like this:

```
public class myRobot {

    /* local OpMode members. */
    HardwareMap hwMap           =  null;
    private ElapsedTime period  = new ElapsedTime();

    DcMotor leftFront;
    DcMotor rightFront;
    DcMotor leftBack;
    DcMotor rightBack;
    DcMotor intake;
    Servo ballPusher;
    ColorSensor colorSensor;

    /* Constructor */
    public myRobot(){
        leftFront = hwMap.dcMotor.get("motor1");
        rightFront = hwMap.dcMotor.get("motor2");
        leftBack = hwMap.dcMotor.get("motor4");
        rightBack = hwMap.dcMotor.get("motor3");
        intake = hwMap.dcMotor.get("motor5");
        ballPusher = hwMap.servo.get("ballPusher");
        colorSensor = hwMap.colorSensor.get("colorSensor");
    }

    /* Initialize standard Hardware interfaces */
    public void init(HardwareMap ahwMap) {
        // Save reference to Hardware map
        hwMap = ahwMap;

    }
```

Remember not to change anything outside of the picture, as it could break everything. Now we need to make it drive. Scroll down in the robot class and we'll create our methods.

First let's make a list of the methods we need to make

1. Drive
2. Intake
3. Detect Color
4. Drop ball

## The Drive Method

```
Servo ballPusher;
ColorSensor colorSensor;

/* Constructor */
public myRobot(){
    leftFront = hwMap.dcMotor.get("motor1");
    rightFront = hwMap.dcMotor.get("motor2");
    leftBack = hwMap.dcMotor.get("motor4");
    rightBack = hwMap.dcMotor.get("motor3");
    intake = hwMap.dcMotor.get("motor5");
    ballPusher = hwMap.servo.get("ballPusher");
    colorSensor = hwMap.colorSensor.get("colorSensor");
}

/* Initialize standard Hardware interfaces */
public void init(HardwareMap ahwMap) {
    // Save reference to Hardware map
    hwMap = ahwMap;

}

public void drive(double leftPower, double rightPower) {

    leftFront.setPower(leftPower);
    leftBack.setPower(leftPower);
    rightFront.setPower(rightPower);
    rightBack.setPower(rightPower);

}
```

## The Intake Method

```
public void intake(double power, boolean reverseButton){
    if (reverseButton){
        power = -power;
    }
    intake.setPower(power);
}
```

### The Drop Method

```
public void drop(boolean open){
    double openPos = 90;
    double closePos = 0;
    if (open){
        ballPusher.setPosition(openPos);
    }
    else {
        ballPusher.setPosition(closePos);
    }
}
```

***Note:** The Drop method uses a button as a parameter, and will react based on the buttons position.

### The Detect Color Method

```
public int ballIsRed(){

    //0 is unable to tell (probably no ball there)
    //1 is red
    //2 is blue

    if (colorSensor.red() > .8){
        return 1;
    }
    else if (colorSensor.blue() > .8){
        return 2;
    }
    else{
        return 0;
    }
}
```

***Note:** This looks different, because it is similar to a method, but returns a number that is accessible throughout the project. It is a more complex type but the best for this type of problem.

### Calling the Methods in Tele-Op

Go over to your Tele-Op and call the three actuating methods with their parameters. This excludes the Color Detection method.

Those will live in the loop. It will look like this:

```
public void loop() {
    telemetry.addData("Status", "Running: " + runtime.toString());

    //Drives the robot
    robot.drive(gamepad1.left_stick_y, gamepad1.right_stick_y);

    //Checks if A is pressed to drop the ball
    robot.drop(gamepad1.a);

    //Controls speed of the intake by the trigger, and checks if button B is down,
    // which if it is it reverses the intake
    robot.intake(gamepad1.right_trigger, gamepad1.b);


}

/*
 * Code to run ONCE after the driver hits STOP
 */
@Override
public void stop() {
```

Now we need to send the color of the ball back to the driver. That will be done using an if, else if, else statement. We will make that into a local method. Scroll down below the stop method and make a new method after it. Make if else statements based off of the value of robot.ballIsRed(). It should look like this:

```
@Override
public void stop() {
}

public void sendBallColor(){
    if (robot.ballIsRed() == 1){
        telemetry.addData("ballColor", "The Ball Is: Red");
    }
    else if (robot.ballIsRed() == 2){
        telemetry.addData("ballColor", "The Ball Is: Blue");
    }
    else{
        telemetry.addData("ballColor", "There is no ball");
    }
}
```

Add that method in the loop. It doesn't need the robot. because it is defined in the same class.

```
@Override
public void loop() {
    telemetry.addData("Status", "Running: " + runtime.toString());

    //Drives the robot
    robot.drive(gamepad1.left_stick_y, gamepad1.right_stick_y);

    //Checks if A is pressed to drop the ball
    robot.drop(gamepad1.a);

    //Controls speed of the intake by the trigger, and checks if button B is down,
    // which if it is it reverses the intake
    robot.intake(gamepad1.right_trigger, gamepad1.b);

    //Send ball color
    sendBallColor();

}
```

### 9.4 Programming Autonomous

Time will be controlled by runtime.

Let's decide what the auto is going to do. Let's have the robot drive forward with the intake running for 5 seconds, check the color of the ball while turning to the right for 3 seconds. If the ball is red, drive forward for 2 seconds and drop the ball. If not, drop the ball where you are.

The easiest way to show this is showing it in code with *lots* of comments, so here is that:

```
/.../
package org.firstinspires.ftc.teamcode;

import ...

/**...*/

@Autonomous(name="Autonomous")
public class Autonomous1 extends LinearOpMode {

    /* Declare OpMode members. */
    myRobot robot   = new myRobot();   // Robot hardware
    private ElapsedTime     runtime = new ElapsedTime();


    @Override
    public void runOpMode() {

        robot.init(hardwareMap);

        // Send telemetry message to signify robot waiting;
        telemetry.addData("Status", "Ready to run");    //
        telemetry.update();

        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        while (opModeIsActive()){

            //If the time is below 3 seconds, drive forward and run the intake
            if (runtime.milliseconds() > 0 && runtime.milliseconds() < 5000){
                //Drive forward
                robot.drive(1,1);
```

```
            //Run Intake
            robot.intake(1,false);
        }

        //If time is between 3 and 7 seconds
        else if (runtime.milliseconds() > 5000 && runtime.milliseconds() < 7000){
            //Turn hard left
            robot.drive(1,-1);
            //Stop Intake
            robot.intake(0,false);

        }

        //If time is after 7 seconds and you have a red ball
        else if (runtime.milliseconds() > 7000 && robot.ballIsRed() == 1){
            //If time is after 10 seconds
            if (runtime.milliseconds() < 10000)
                //Drive forward
                robot.drive(1,1);
            //If the time is before 10 seconds
            else{
                //Stop the robot
                robot.drive(0,0);
                //Drive the ball
                robot.drop(true);
            }

        }

        //If time is after 7 seconds and you don't have a red ball
        else if (runtime.milliseconds() > 7000 && robot.ballIsRed() != 1){
            //Drop the ball
```

```
                robot.drop(true);
            }
        }
    }
}
```

## 10.0 Competition

### 10.1 Qualification Matches

Qualification Matches are the matches that take up the majority of a competition. In these matches, teams can be allied or opposing any other team at the tournament at any time. The Qualification Matches occur after the inspection process and the judges.

Teams compete for Ranking Points and Qualifying Points during Qualification Matches. These points create a ranked order of every team at the competition for Alliance Selections.

During Qualification Rounds, teams should communicate with their alliance partners before each match begins. Teams should discuss strategy for the upcoming match by evaluating the strengths and weaknesses of each robot and then designating roles during the match. If a team's robot is struggling, they should tell their alliance partner so they know ahead of time. Teams should be honest about the condition of their robot.

If an alliance knows they are facing a difficult opponent, defense may seem like a viable option depending on the game. In this case, alliances should either assign the team capable of scoring less points to play defense or

base their decisions off of the drivetrain and weight of the robots, as a heavier robot with a powerful drivetrain will usually play more effective defense. This also applies in Elimination Matches.

**10.2 Alliance Selections**

The Alliance Selection process is rather simple but incredibly important. This process sets up the Elimination Matches of each tournament which determine the winners.

Throughout the day, teams compete with randomly generated alliance partners and against random opponents. However, Alliance Selections give the top ranked teams an opportunity to choose what teams are on their alliance.

The Alliance Selection process is simple. The top four ranked teams from Qualification Rounds are the alliance captains. The first ranked team will invite first. They (being the team making the first choice) may invite any team from the competition to join their alliance including the other three alliance captains.

An invited team may choose to accept or decline the offer. If accepted, that team joins that alliance, the order updates (explained below), and the next alliance captain invites a new team. If declined, the invited team can NOT be invited to another alliance and the alliance captain chooses a new team to invite to their alliance.

If a team that is currently an alliance captain is invited to an alliance, they may choose to decline and will still be an alliance captain of their own alliance. They can not be invited to another alliance but they hold the right to invite other teams to their alliance. If a current alliance captain accepts another alliance captain's invitation, they join that alliance and the order rotates from the pool of teams based on rank.

For example if the number one seeded captain invites the number three seeded captain and they accept, the number three seeded captain joins the number one seeded captain's alliance. Then the order "rotates" forward. The number four seeded alliance captain becomes the number three seeded alliance captain, the number five seeded team becomes the number four seeded alliance captain, and then the number two seeded alliance captain invites a new team to join their alliance.

During Alliance Selections, a team may not be invited to an alliance if they already declined another invitation, they are the captain of a higher seeded alliance, or are a member of a higher seeded alliance.

The Alliance Selection process continues until each alliance has three robots (or two in a competition with twenty robots or less). Unlike FRC, the draft is not a "snake draft".

When teams are choosing their alliance partners, they should consult their scouting data to form a picklist BEFORE Alliance Selections start. Just because a team is high ranked doesn't necessarily mean they are the correct pick. A team could be in the top four by being carried by their alliance partners or maybe they aren't the best pick for the strategy of another team. For more information on scouting, see **10.6 Scouting**.

### 10.3 Elimination Matches

Using the four alliances of three from Alliance Selections, the Elimination Matches determine the winning alliance of that competition. The number four seeded alliance plays against the number one seeded alliance and two plays three. Each match is best two out of three, so to win the competition an alliance must win at least four matches. Additionally, each robot must play at least once in each matchup. This means that in the first two matches of semifinals and finals, the same combination of robots on an alliance can't occur. If more than two matches are played in one matchup any combination of robots can be used in the third match.

The winner of each semifinal matchup moves on to the final match up with the same rules.

At some competitions, each alliance is given one "timeout" after Alliance Selections. The intent is to give an alliance more time to prepare between each match.

### 10.4 Judging Process

There are three parts of the judging process: the judges' interview, evaluation of the Engineering Notebook, and evaluation of match performance. To read more about Engineering Notebook, see **3.0 The Engineering Notebook**.

The judges' interview occurs before opening ceremonies so every member of a team can participate in the interview. Mentors and coaches can not participate in the interview but they can sit in on the process (max of two mentors).

Each team is given 15 minutes or less to present their robot, journey, outreach, etc. The remaining time is given to the judges to ask any questions not previously answered. This can be compared to the FRC Chairman's Presentation.

Rehearsing the judges' interview before competition is a great way to ensure a smooth and impressive performance.

### 10.5 The Pits

At FTC Competitions, teams are given space to work on their robot between matches. This is also where judges will come to question teams regarding specific awards.

The pits should remain organized and clean at all times. Work on the robot occurs here as well as judges and other teams will come to a pit to strategize, question, or otherwise communicate before matches.

It is important to wear safety glasses in the pits. Teams should bring their own safety glasses. The ones provided are intended for visitors, judges, etc. See **15.0 Safety** for more about safety.

The pits is also where pit scouting should take place. See more under **10.6 Scouting**.

Nothing is guaranteed to be in the pits. Teams should bring everything they need to compete. This could also include extension cables for charging.

Decorating the pit area is done by many high level teams to help identify them. This is not optional. There are rules on what is allowed in a pit area. See the FTC Game Manual for more information.

### 10.6 Scouting

Scouting is the part of strategy that occurs at competition. Scouting is simply taking notes on other teams to assist in the Alliance Selection process.

There are three main types of scouting; quantitative, qualitative, and pit scouting. Quantitative scouting is pure data. Numbers. How many of each objective did each team score each match?

Qualitative data is less statistical and more analytical. This form of scouting is based on written notes. These notes could be based on driver skill, maneuverability of the drivetrain, driver's ability to play defense, scoring patterns, etc. This information can be used for Alliance Selections AND for competing against robots in matches. For example if scouting shows a team can only score in a certain position on the field, playing defense in that position could be advantageous.

The third form of scouting is pit scouting. This is based off of the physical robot alone. One of the most important parts of this process is a team's drivetrain. A drivetrain can alter whether a team can push or be pushed. This information, if used properly, can help determine the outcome of a match. Noting what motors a team uses is also important. Gear ratios and motor quality can also make a difference in a pushing match. Additionally, the quality of a mechanism can matter too. If a robot looks like it will fall apart, it probably will.

Using scouting data, a picklist can now be created. This picklist should be made before the Alliance Selection process. This picklist should reflect a team's individual interests in an alliance partner. This list will not likely be the exact ranking order. This list should be extensive in length and be sorted based on preference (from most wanted to least wanted alliance partner). In addition, teams should decide what invitations they will accept if any.

Scouting can be done by hand or digitally. It is up to each team what they wish to do based on that team's resources.

## 11.0 Awards

Awards are given to teams that best exemplify aspects of FIRST and go above and beyond to demonstrate the importance of Gracious Professionalism, teamwork, creativity, innovation, and the engineering process. Award guidelines serve as a roadmap to success for each and every FIRST team.

Criteria for each award can be found in the Game Manual Part 1. This guide is intended to advise teams to win awards and should NOT be used as a substitute for the Game Manual.

### 11.1 Inspire Award

The Inspire Award is given to the team that best embodies the ideals of FIRST. This team is a role model for other teams in FTC, and inspires other teams in the FIRST program. They act with Gracious Professionalism in and out of competition by teaching others and helping other teams. This team will also demonstrate what they know by building a well functioning robot.

If a team wants to ensure an Inspire Award win, they should be solid contenders for every award. Having a strong Engineering Notebook, doing lots of outreach, and performing well on the field are three very important aspects of the Inspire Award as well as many other awards.

Teams aiming for an Inspire Award win should also give a strong presentation during their judges' interview. While judges don't base any awards off of just the interview alone, this is one of the first opportunities teams will receive to show judges what they know.

### 11.2 Think Award

The Think Award is awarded to the team that best reflects the journey and engineering process they experienced. The Engineering Section of a team's Engineering Notebook is the key reference for judges for this reward. This section must focus on the process of the building of a team's robot, and must include the science and mathematics used in the process.

It is required to have an Engineering Notebook entry for each day of robotics build throughout the season. The entire engineering process must be documented in a logical and legible manner for a team to win this award. Furthermore, judges appreciate when teams document the applied maths they use throughout their season. Teams that include pages dedicated to the math they used throughout the season could increase their chances at the Think Award.

### 11.3 Connect Award

The Connect Award is awarded to the team that best connects to the science, technology, engineering, and math community around them. This team searches for future engineers and engineering opportunities. The team has a Business or Strategic Plan that outlines the future plans of the team.

To win the Connect Award, teams should have a strong and obvious impact on their community. They should also have documented and realistic goals for their team's future success.

### 11.4 Rockwell Collins Innovate Award

The Rockwell Collins Innovate Award is given to the team that implement new and innovative ideas in their robot design. The judges reward the team with the most creative design solution that works well. The creative aspect of the winning robot must work consistently and well, and the team's engineering notebook should highlight the process of designing the creative component.

To win the Rockwell Collins Innovate Award, it is advisable for teams to pick one or two main design features of their robot and emphasize them thoroughly to the judges. While all portions of the robot should be covered in the Engineering Notebook, teams striving for the Rockwell Collins Innovate Award should focus in on an aspect of their robot that is truly unique and innovative. This emphasis should be primarily mechanical based since the Control Award is focused on the programming aspects of the robot.

### 11.5 PTC Design Award

The PTC Design Award is awarded to the team with a robot that has both functional and aesthetic components. These components could simply bring together the appearance of the robot, or complement its purpose.

To win the PTC Design Award, teams should build an industrially sound robot. Strong contenders for the Rockwell Collins Innovate Award could also be strong contenders for the PTC Design Award. The primary difference is that the PTC Design Award is more focused around the entire robot while Rockwell Collins Innovate Award focuses around one specific mechanism.

### 11.6 Motivate Award

The Motivate Award is given to the team that embraces the culture of FIRST and displays what it means to be a team. This team demonstrates their teamwork by showing spirit and enthusiasm throughout competition. This team also makes an effort to make FIRST known throughout their school and community, and sparks an interest in FIRST as a whole.

All Team members should participate in their presentation, and actively engage with the judges in order to win the Motivate Award. The team should also show a creative approach marketing the Team and FIRST.

### 11.7 Control Award

The Control Award is awarded to a team that uses sensors and software to enhance their robot's functionality. The team must have an understanding of control systems to solve challenges such as autonomous operation, having a better understanding of the field with sensors, or enhancing their existing mechanical. The control component must work consistently, and the team's engineering notebook must contain details of the implementation of the control system.

There is a Control Award submission sheet that must be submitted to apply for this award. This sheet and the Engineering Notebook should be able to explain in detail the process of a team's programming journey. Advanced software techniques and algorithms are encouraged.

### 11.8 Promote Award and Compass Award

The Promote Award and the Compass Award are both optional submission awards based on a video the team creates. There are more rules and regulations for both of these videos in the FTC Game Manual Part 1.

The Promote Award is given to the team with the most compelling video message directed towards the public, to celebrate the spread of science, technology, engineering, and mathematics in culture. The video must be a one minute long PSA based on the PSA subject for the season.

To win, teams should create a unique video concept and make the video professional. It is advisable to view other team's videos and appropriate commercials for inspiration.

The Compass Award recognizes an adult coach or mentor who guides their team through the engineering process and demonstrates what it means to be a gracious professional. The winner will be selected based off of a 40-60 second long video submission from their team highlighting how their mentor or coach inspired them.

This award is rather straightforward. Using a quality video with unique content and delivery styles can greatly impact a team's success for this award.

### 11.9 Judges' Award

The Judges' Award is given to a team that catches the judges' attention with their unique efforts, dynamics, or merits, yet does not fit into any existing award categories. The Judges' Award does not factor into advancement criteria.

## 12.0 Team Image

The way a team is perceived changes the awards they win, what teams select them during alliance selection, and more. Team Image is comprised of three aspects: spirit, social media, and theme. Spirit is cheering at competitions, social media is how a team connects with the public, and theme is a team's unique design aspects and chosen colors that represent them.

### 12.1 Spirit

Spirit is a large part of how teams are judged, both by other teams and judges, for alliance selection and awards. A loud team that cheers at appropriate times is much more likely to be noticed by other teams, and is much more likely to win various awards. Cheers should be done before or after matches, or right away in the morning. A team should not have to cheer during a match, as it is disruptive.

Spirit gear includes special costumes, pom poms, flags, and more. Teams can either buy or make their spirit gear. Some teams designate specific people to wear costumes during competition and lead cheers. Spirit also serves to boost team morale, of both the team cheering and others. Dancing and singing along to songs is a large part of spirit, as it both attracts attention and boosts the mood of the team.

Teams are encouraged to create their own cheers, but here are some to get started:

Red Hot
[all together - when on Red Alliance]
Our team is RED hot! X2
Our team is R-E-D, RED!  H-O-T, HOT!
Once we start we can't be stopped,
red hot!  Yeahhh yeahhh, red hot!

Good Morning
[all together - only in morning]
G-O-O-D!  M-O-R-N-I-N-G!
Good Morning! Yeah yeah! Good morning!
[three claps]
(x3)

### 12.2 Social Media

Public Relations (PR) is an integral aspect of non-STEM robotics. It makes it easy to energize a support base, keep sponsors updated, and have a recognizable presence in FTC robotics by communicating with other teams.

For an FTC team, the main goal of PR is to promote community outreach. Teams should also share about any events they host, give updates on the progress of the robot, and competition updates as well. Teams that are successful at PR update social media regularly and use a variety of platforms to promote themselves and the principles of FIRST robotics.

The trick to an engaged audience is strategically using various social medias to target different demographics.

For example, Twitter is very popular amongst teachers and higher level school district administrators, as well as FTC teams. Therefore, Twitter is a good platform to share community outreach projects and progress on the robot. Instagram and Snapchat are more popular amongst the members of different teams, so they are best used to share photos of robot construction. Facebook is most commonly used amongst sponsors and parents of the team, so it's best used to share community outreach, occasional progress updates on the robot, and how the robot does at competitions.

**12.3 Theme**

Theme is how a team chooses to represent itself through unique design aspects, colors, and a motto or central message. This theme can impact team costumes, team presentations, and team cheers. Good team themes are interesting to the judges and other teams, as they attract attention. Team names can be selected with the team theme in mind, and vice versa. In order for a team to become more recognizable, they should keep a theme from year to year, and should have a common theme between costumes, cheers, colors, mottos, and all aspects of their team.

It can be difficult to decide on a team theme, as everyone’s opinions should be considered in the process. An ideal theme provides opportunities to be remembered and recognized, can be used to provide a special spin during presentations, and is appropriate.

In order to best present their theme during competition, a team should decide on a theme as soon as possible. They should then decide on how they will be presenting their theme, whether they decide to present it through a shirt, cheers, or any other aspects of their team.

A team name is usually selected based on common interests, jokes, or plays on words. So long as the subject matter is appropriate, FIRST will approve any team name. Try to be original as possible, as team branding is normally based off of team name.

Most teams will design a tee-shirt, which they all wear to competition. This ensures that they look like a unique team. When wearing their shirts, team members may be judged based on their actions by coaches, other teams, and event judges at any time during competition, so make sure to display gracious professionalism at all times.

Other teams may get more creative, and design a unique outfit for their team. This is entirely open ended, so it can mean a uniform outfit, or mascots, or a similar but different outfit for each member. Keep in mind that if any inappropriate or unprofessional outfits are displayed, the team or team member risks being ejected from competition.

Finally, teams will often create a pit display, showcasing their members, mentors, outreach events, mission, timeline, etc. This is a great idea for all teams, as it is a visually attractive way to show what a team has done over the season without making visitors read through the Engineering Notebook. An effective pit display will correspond closely to the team's theme and positively reflect their experiences throughout their season.

**13.0 Outreach**

Outreach, while not required, is one of the most important aspects of non-STEM awards at competitions. Outreach is giving back to the community. Lots of outreach in FTC is reaching out to the community to inform about FIRST Robotics.

**13.1 Purpose**

The purpose of outreach is to give back to your community and inform them about the importance of STEM. It's a chance to show your community how much their support means to you. Whether it's doing community service or touring businesses, outreach is a vital part of an FTC team.

There are many awards that require and are focused around outreach at an FTC competition. Some of these awards include the connect award and the most prestigious FTC award, the Inspire Award.

**13.2 Common Ideas**

Teams are encouraged to use outreach as their own outlet to expand the messages of FIRST. It is encouraged for team's to do outreach in their own way. Some ideas commonly used by other teams are:

- Community Service
- Robotics Demonstrations
- Tours of Businesses
- Mentoring Other Teams
- FIRST Summer Camps

**14.0 Mentorship**

FTC mentors are the backbone of every team, and are by FIRST's definition the most important part of the program. Committing to be an FTC mentor is committing to an equal partnership. FTC mentors do not require any special skills, just patients, and the willingness for their students to succeed. Mentors work side by side with students to help them learn and grow in all areas of life. FIRST is more than just robots.

**14.1 How to Get Mentors**

Getting mentors can be a challenge to new teams. Many students have parents or other adults in their lives that work in engineering fields. These are potential mentors. In addition, a teacher at a team's school can work as a mentor. Many schools have engineering based classes that are taught by a teacher with knowledge in engineering.

A mentor should know what they are committing to. Teams should be straightforward about their needs when approaching a potential mentor.

**14.2 Gracious Professionalism**

Gracious professionalism is one of the greatest pillars in the FIRST program and thus the mentors should know and practice gracious professionalism in all things. Students will look to the mentor as a person to emulate. If mentors practice their own gracious professionalism they will see their students follow suit.

### 14.3 Goals as a Mentor

A mentor's goal, above anything else, is to have fun and make the process fun. Mentors and the competitions are the source of that fun and mentors will decide on who mentors and what competition students go to. Whenever a team goes somewhere, or works together there will be team bonding. This is a mentor's best tool, keep the team having fun together. If students have fun outside of competitions, inside competition stress will decrease, while the fun of a competition will increase.

Mentors need to push the students to their greatest potential. This starts at the beginning of the year when the team is created.

Assign roles for each student, give them a purpose, a job, something they are able to invest into. Use what the students are already used to and push them past that. If they are used to mechanical, push them to learn programming.
See that students are moved to different departments based on needs as well. In the case of a team with one programmer and five mechanical, a good mentor will split the mechanical department to help the workload of the programming team, as well has the non-STEM departments such as business and marketing. This isn't keeping students from doing what they want, it's dual purposing the students, and setting them up for a brighter future.

### 14.4 What Should a Mentor Know?

A good mentor will know enough. They do not need to be any sort of expert on FTC building and programming. They do not need to know how to fabricate complicated parts, or even how to drive a robot. All a good mentor needs is the willingness to learn a little bit of everything. Having a base knowledge in all aspects of the team creates a stronger more personal bond with students.

## 15.0 Safety

FIRST promotes safe behavior at all times. Being safe is very important to every team and every member. Teams should always be working towards being more safe.

The most basic form of being safe is being knowledgeable. New members of the team should be trained in on various equipment (such as the drill press and bandsaw) before they are allowed to use it. Furthermore, when using heavy machinery all students should be supervised by a coach, mentor, or otherwise knowledgeable and legal adult.

The enforcement of wearing safety glasses is another important part of FIRST's efforts to keep their program safe. Team members will not be allowed into the pits at a competition without wearing safety glasses.

Robots that move when initiated should have a sticker or otherwise warning marker to indicate to judges and field technicians that the robot moves when initiated. See **8.3 Miscellaneous Electrical Requirements** for an example sticker.

**16.0 Resources**

FIRST Registration Information - http://www.firstinspires.org/robotics/ftc/cost-and-registration

FIRST Engineering Notebook Guidelines and Business Plan Outline - http://www.firstinspires.org/sites/default/files/uploads/resource_library/ftc/engineering-notebook-guidelines.pdf

Creo Parametic free trial - http://www.ptc.com/academic-program/products/free-software

Tetrix and Matrix part files - http://www.catalogds.com/db/service?d=first&c=browse

Autodesk Inventor free trial and Kit of parts - http://www.autodesk.com/education/competitions-and-events/first/all-products

Solidworks free trial - http://www.solidworks.com/sw/education/student-software-3d-mcad.htm

Solidworks Kit of parts - http://www.solidworks.com/sw/education/robot-student-design-contest.htm

Codeacademy - https://www.codecademy.com/learn/learn-java