INFINITE RECHARGE℠ at Home Guides

FIRST® GAME CHANGERS℠ powered by *Star Wars: Force for Change*

2021 *FIRST*® Robotics Competition

FIRST ROBOTICS COMPETITION

FIRSTINSPIRES.ORG/ROBOTICS/FRC

CONTENTS

Introduction 1
Breaking Down the AutoNav Challenge 2
Tracking Robot Position 2
Choose an Approach 2
Putting it Together 3
Targeting with Vision 5
Step 1: Decide Goals & Plan Approach 5
Step 2: Find the Target 5
Step 3: Vision System Output 5
Step 4: Use the Output to Command the Robot 6
Selecting Drivers 8
Improving Driving Performance 9
How to Practice 9
What to Practice 10
Analyze and Upgrade Cycles 10

INTRODUCTION

Completing Skills Challenges as the only robot on the field is a new challenge for *FIRST* Robotics Competition teams. To help teams consider how they may practice and improve on these skills with their existing robot, *FIRST* has put together guides teams might find helpful. These guides are entirely optional and do not act as a step-by-step process. Use of the guides is not part of the judging process. Though these guides were designed around the specific 2021 Skills Competition challenges, teams may want to think about how to develop and incorporate similar activities into future seasons.

BREAKING DOWN THE AUTONAV CHALLENGE

The AutoNav Challenge encourages all teams to explore complex autonomous navigation. In order to assist teams approaching this for the first time, this guide provides a high level overview of how to get started with autonomous navigation.

Tracking Robot Position

Before determining which approach to take and how to proceed, we need to review the fundamental concept crucial to both approaches: determining and tracking the robot's position.

Some ways to track robot position when navigating around the field include:

1. **Time** - The most simplistic method of measuring robot position is to execute movements for fixed periods of time. Sensors aren't required, so it's applicable to any robot design; however, it doesn't actually measure anything related to the movement occurring. This means the actual amount of movement may vary depending on battery voltage, wheel slippage, or other robot changes that increase or decrease friction (resulting in the robot moving faster or slower than expected).
2. **Sensors -** To get more accurate performance than timed movements, teams can use sensors to make measurements. These sensors fall into two main categories:
    - **Internal sensors** - These sensors measure movement of the robot or robot parts but do not use references from outside the robot. These sensors remove many of the sources of error compared to a time measurement but are still susceptible to things like wheel slip or sensor drift. Two primary types of internal sensors teams use for tracking robot position are:
        - **Encoders** - measure the rotation of shafts (in this case, by extension, wheels). Encoders can track straight-line position, and somewhat track turns (many robot drivetrains experience wheel slip during turns reducing encoder reliability).
        - **Gyroscopes** - track angular position (more specifically, they track angular speed which can be integrated to return angular position). Many gyros also have built-in accelerometers (the combination of one or more gyros and accelerometers in a single unit is commonly called an Inertial Measurement Unit or IMU) which can technically track position through double integration of acceleration measurements, but the double integration amplifies noise, typically making this measurement too noisy for *FIRST* Robotics Competition use.

      Many teams find internal sensors reliable enough for general navigation but may move to external sensors when precision is needed, usually for scoring tasks.
    - **External sensors** - External sensors directly measure the world around the robot. Combined with pre-existing knowledge about the field, they can be used to determine where the robot is located. External sensors include cameras and distance measuring sensors such as Laser rangefinders (LIDAR), IR Rangefinders, and Ultrasonic sensors. Teams often use external sensors when they need to be in a precise location and/or orientation relative to the field, such as for scoring tasks, though they do not have to be limited to this application.

Choose an Approach

Two common approaches to autonomous robot navigation are:

1. **Individual Movement Approach** - break the movement down into smaller individual pieces (generally straight paths and in-place turns, but occasionally single arcs), develop code to complete each individual

piece (ideally with parameters for things like distance or angle for code re-use), then string those pieces together into a complete routine.

2. **Path Planning Approach** - use path-planning to generate a smooth path in one or more pieces, each of which may contain multiple arcs (or straight runs).

Individual Movement Approach

The Individual Movement Approach generally starts by developing individual code routines to drive straight for a desired distance (both forwards and backwards) and turn to desired angles (both left and right). Then for each desired path, you can break it down into these three basic building blocks. There are generally two control approaches used when writing those building blocks:

1. **Bang-Bang Control** – Bang-bang control has two states, typically on (though this does not need to be "full speed") and off. Simply turn the motors on at a specified value and periodically check whether an end condition has been met. Once the end condition has been reached, stop the motors. This method typically results in significant overshoot of the target when used for position/angle.

TECH TIP: Remember, the Timed and Command templates, as well as the LabVIEW Teleop VI contain loops around the XXPeriodic functions already. Users should not place long running loops like this inside these functions, instead consider the XXPeriodic method as one iteration of the loop.

---

2. **PID Control** – PID control sets the output dynamically based on the error (and potentially error accumulation and rate of change of error). At a very basic level, you can think of it like a car approaching a stop sign – generally it stops gradually, moving slower as it gets closer to the target.

Path Planning Approach

This approach generally starts by tuning a control loop(s) on the robot (generally velocity control) to enable it to follow an arbitrary path. Then, for each path you want to drive, you break the path down into "waypoints" you want the robot to drive through and use them to generate a full path.

TECH TIP: Detail on the WPILib tools that help with this are in the Trajectory Generation and Following section of the WPILib Documentation and a step-by-step tutorial can be found in the Trajectory Tutorial. LabVIEW users can use this library for similar functionality.

Putting it Together

Once that you've decided on your approach and looked at the path you are trying to drive, those building blocks must be put together. The process for doing so depends on the language and framework you are using for your robot code:

- **LabVIEW or Timed Robot Auto Init** – The most basic method of assembling an auto routine is writing each building block as a method or VI with a loop inside and assembling/calling them sequentially. In LabVIEW this can be done by using a Sequence Structure. In C++ or Java, it can be done by calling your building blocks, in order, from the AutoInit() method. While this approach is the simplest, it can have significant downside (at least in C++\Java). Structuring your code like this in C++\Java makes it more difficult to add other functions to the code while the robot is driving; because your code is running in a

loop trying to complete the current "building block" routine, it can't be doing other things in parallel. LabVIEW is a bit different as the design of LabVIEW code is inherently parallel.

- **LabVIEW or Timed Robot State Machine –** If you want your code to be more flexible about running other behaviors at the same time it is driving, consider a "state machine." You can read up on state machines in plenty of places on the internet. For the purposes of a simple *FIRST* Robotics Competition autonomous they generally consist of the following:
    - **A state variable –** This keeps track of the current state.
    - **Conditional/Branched code –** A "switch" (or case structure in LabVIEW), or series of "if" statements, inside the AutoPeriodic() method based on the state variable that describes what to do in each state.
        - The code inside each branch should generally perform an action (such as setting motors to some speed) and then test whether criteria has been reached to move to a new state. In simple autonomous state machines, flow generally only moves forward (advancing when the target distance or angle has been reached). In more complex state machines flow can jump around and does not necessarily proceed linearly.
        - Make each branch a "building block", setting motor speeds and checking if the target has been reached before advancing to the next state, use the overall loop around the AutoPeriodic() method from the framework instead of writing your own loop.
    - **Command-based framework** – In the Command-based framework the framework has constructed a type of state machine for you. Each building block is a command where the Init() and/or Execute() method(s) command the motors and the IsFinished() method evaluates whether the target has been reached. When using more advanced controls you may end up using things like the PIDCommand or RamseteCommand which may handle some of this logic for you. To assemble the building blocks together into a routine, use CommandGroups. Converting a Simple Autonomous Program to Command-Based uses the "old command based" library, but the principles described should be applicable to either "new" or "old" Command-based frameworks (though some syntax may vary).

TARGETING WITH VISION

There is a wealth of documentation available about finding targets using FIRST Robotics Competition vision systems but not many clear descriptions of how to use the information returned by the vision system. This guide aims to provide an overview of the information you should be trying to get out of your vision solution and how to use it to control your robot.

Step 1: Decide Goals & Plan Approach

Review the problem and determine what you are trying to have the software do. These are some questions you might ask to help you along:

- Is the target always expected to be in frame of the vision system, i.e. will a driver or path based autonomous navigation get you close? Or will the code need to know what to do if no target is found? If the code must find the target, is this done with the drivetrain, or is there a turret or camera servo that will be used?
- Does the robot need to know the distance to the target to perform the desired function? If so, is that coming from the vision system or somewhere else? In some cases, you may wish to know the distance, such as for computing a shooting speed/angle or driving the robot up to the target. In other cases, distance may be handled by other means (e.g. it doesn't matter, the driver will address it, it's handled by other distance sensing).
- Does the robot need to be oriented a specific way to complete its goal (e.g. it may not be able to shoot at a sharp angle, or it may need to be near perpendicular to place a game piece precisely)? If so, capture the details of the limits. How should the code handle situations where the robot orientation is outside those limits?

Step 2: Find the Target

Three common approaches to vision systems are listed below:

1. **"Traditional computer vision**" – This encompasses non-machine learning approaches using libraries like NI Vision or OpenCV. Information on these approaches can be found in the Vision Processing section of the WPILib documentation.
2. **Machine Learning** – This approach teaches the software what the target looks like using labeled example images of the target. More information on this approach can be found in the Machine Learning tutorial in the WPILib documentation.
3. **Off-the-shelf solutions** – This approach uses a packaged off-the-shelf solution to find the target. These solutions generally accommodate some user tunability but are often a quite different experience than attempting to design the code from scratch. Examples of this include Chameleon Vision, Limelight, Opensight, and PhotonVision/Gloworm. Note that some of these solutions require specific hardware and some are designed for teams to assemble their own hardware solution. The WPI developed program, GRIP is a hybrid between this approach and traditional computer vision. GRIP provides an interface similar to some of these solutions, but then generates OpenCV code that can be further tuned by the user.

Step 3: Vision System Output

The answers to the questions in Step 1 and the specifics of the vision system being used dictate the output of the vision system and how it is communicated. The most common communication method is NetworkTables. Typical vision system outputs include:

- **A measurement of horizontal offset** - Occasionally this is an angle, but more often it is a pixel measurement of the offset of the detected target in the frame. This may be used directly or used to calculate an angle to inform robot movement.
- **A measurement of vertical offset** - Occasionally this is an angle, but more often it is a pixel measurement of the offset of the detected target in the frame. It is typically used to calculate a distance to the target using an approach similar to the one described here (note: this particular description describes using the width of the target, but similar math and principles can be used).
- **A measurement of the target width and/or height** - These are almost always pixel measurements and, if used, are typically used to assess distance from the target using methods like the one described here (this description uses target width; the math can be easily converted to use target height instead).
- **Robot pose** - Some vision systems use pose estimation techniques like SolvePNP to provide a complete estimate of robot pose relative to the target (pose consists of both positional and angular measurements).

Step 4: Use the Output to Command the Robot

Once information from the vision system about the target is acquired, it can be used to command the robot. In addition to the info below, you may find the various "Case Study" sections of the Limelight documentation helpful, even if you're not using a Limelight!

Convert Vision Information to Rotational Commands

As described in Step 3, your vision system should provide an output that gives you information about the horizontal position of the target in the image. In order to point your robot at the target, use this information to generate a rotational command for your robot. Typically, a control loop is used. Two ways to do this are as follows (a third, alternative, approach is discussed in "Path-planning" below):

1. **Close the loop with vision** - Repeatedly use this vision measurement directly to generate a steering command. This method is limited by the camera framerate image stability. If turning too fast, you may overshoot the target before getting a new image or you may get bad data as the camera shakes. This method is the simplest and is viable provided the turn rates are limited to accommodate camera capability.
2. **Close the loop with a gyroscope** - Use vision information to generate an angle to steer to, then use a gyroscope as the feedback to steer to that angle. This method takes advantage of the fast response of a gyro to rotate the robot to the vision systems estimate of the target before checking the vision system again to verify that the target is now centered (within some margin of error). This method is slightly more complicated than the method described in #1 and may not be necessary for faster vision systems.

TECH TIP: While this section is written in terms of controlling the robot drivetrain, the same principles apply to rotating a portion of the robot, such as a turret, instead. To use approach #2 in this scenario, the gyroscope must be mounted to the moving robot component.

Convert Vision Information to Driving Commands

If the goal is to navigate the robot to a specific distance from the target, distance measurements can be used to generate driving commands for your robot. The approaches are the same as those described above apart from replacing "angle" with "distance" and "gyro" with "encoders".

Combine Rotation and Driving Commands

To drive towards a target, you'll likely want to simultaneously drive and maintain the correct angle. To do this, you add the results of your angular and drive outputs together.

TECH TIP: If the combination of the max output from both loops can exceed the max output of your system (e.g. exceeds 1.0 if using PWM controllers or % VBus mode), check for this condition and divide the output for both sides accordingly so the greater of the two commands does not exceed the system maximum. Otherwise the desired turning output will not be maintained (e.g. the combination of the two outputs yields final outputs of 1.0 for the left side and 2.0 for the right side; as the system can only output 1.0, both sides get the same command and go the same speed instead of one side going twice as fast.)

Convert Vision Information to Shooting Commands

If game pieces are shot or thrown into a goal, distance measurements may be converted into commands related to shooting the game piece instead of driving the robot. While it depends on the design of the robot, these commands are often speeds, angles, or both. While there are physics approaches that can determine a good starting point for these measurements, capturing 100% of the applicable physics in equations can be difficult. Instead, empirical data is often collected and either fit to an equation or entered into a look-up table (with or without interpolation) in order to convert distance measurements (or other outputs of the vision system) to shooting commands.

Alternative approach – Path-planning

An alternative approach is to use path-planning tools to generate a smooth robot path between the current position and the desired goal. This method requires the ability to determine the robot pose relative to the target. The code then uses the existing robot location as the starting point and the desired location (usually a small offset from the actual goal/target) and angle as the final waypoint to generate a smooth path to the target. The robot can either follow this path completely or begin following this path and then compute new paths by reading updated pose from the vision system along the way.

SELECTING DRIVERS

One important skill in FIRST Robotics Competition gameplay is driving and operating the robot. Like in sports, art, music, and education (just to name a few), skills are a product of the application of talent and practice. Often overlooked due to the demands of constructing the robot, thoughtful driver selection is regularly a key differentiator for competitive teams. This guide lays out some helpful considerations when selecting drivers.

Drivers must have good communication skills, demonstrate talent for operating the robot, and a commit to practice and gaining experience throughout the competition season. Some techniques and considerations for selecting drivers include:

- **Talent vs Experience** – When evaluating drivers, consider a candidate's raw talent and their accrued experience. Talent, or the candidate's natural innate ability to operate the robot, is a powerful differentiator that sets candidates apart. A candidate with high talent but little experience can often perform at a level equal to or greater than candidates with experience but little talent. Analogous with sports, arts, music, and education, an individual with high talent cannot rely on talent alone and must practice and reflect to improve. The driver selection process should look for talented, dedicated candidates willing to put in the time and effort to grow their skills. Consider how much time you will have available to practice with the robot. If you have little time available for practice, a candidate with high talent may be the right choice. If you have more time available for practice, a candidate with lower initial talent that excels in communication and composure may become the better choice over time. You don't have to wait for your current robot to be complete in order to assess driver talent, a previous robot, test drivetrain, or the bare drivetrain of the current robot can all be used to assess driver talent.
- **Communication** – The ability to effectively and respectfully communicate new ideas, strategies, and opportunities for improvement is critical for drivers. Effective drivers are also able to communicate robot improvements and issues with the rest of the team. Drivers often find it easier to communicate and share ideas if their drive team has "chemistry," often a reflection of good rapport and developed through shared experiences like team-building exercises. Drive team members should feel safe to communicate frustrations and successes with each other and work as a team to grow and learn together. A post-match "debrief" is a great way to highlight areas of improvement and celebrate successes.
- **Composure** – A *FIRST* Robotics Competition tournament is a complex environment with immense competitive and time pressures. Effective drivers will be able to remain calm under pressure and coolly evaluate the situation before choosing the best actions. When evaluating drivers, look for individuals that are able to adapt well when something goes wrong and quickly move to an alternate plan. While careful planning can reduce unexpected scenarios and the panic that may come with them, over the course of a tournament it's likely that unexpected things will still happen, and the driver will be forced to react immediately.
- **Rules** – While it's not the sole responsibility of the drive team to read and understand the rules, thorough knowledge of the rules by every member of the drive team is certainly a must. This especially applies to rules specific to each drive team member's role on the drive team. Successful drive teams observe constant vigilance with the rules by keeping up with all Team Updates and the Q&A and making sure to have a reference copy of the updated Game Manual with them at all events. Some matches are decided by a single penalty, so understanding the rules is key!

IMPROVING DRIVING PERFORMANCE

A standard part of almost every FIRST Robotics Competition game is the gameplay loop of acquiring and scoring game pieces. This loop, going from having just scored, to acquiring game pieces, to scoring again, is often referred to as a "cycle". For most games, the majority of a team's match time is spent cycling so optimizing cycles is crucial. This guide provides some ideas about how to practice, what to practice, and how to further analyze and improve your cycle times.

In addition to this guide, you may wish to search out some of the many community resources that have been written on similar topics. Two resources that were particular inspirations for this guide are Cycling Optimization by Team 610 and Team 2168's Drive Team Manual.

How to Practice

Identifying and implementing processes and techniques to improve driver skills is important for optimizing the robot's performance. One of the most impactful ways to improve driving skills is practice. Practice doesn't have to wait for the robot to be completed, a previous robot, test drivetrain, or the bare drivetrain of the current robot can all be used to start practicing before the robot it complete. While simple practice alone (either physically or virtually through simulators or cooperative video games) is effective, how you practice is also very important. There are several techniques that can help make driving practice more effective – a few of these include:

- **Start Slowly and Work Up** – When learning a new skill, it's best to begin slowly. Successful drivers often start by driving and completing tasks slowly. Once familiarity with the controls, the task, and the sequence of events is established, speed up the completion of each task. If practice at a specific speed or level needs to be extended to build confidence or proficiency, that's okay. Eventually the driver develops muscle memory and actions become "second nature," allowing the driver to begin to "practice like you play" and further refine their abilities.
- **Don't Be Afraid of Crutches** – One useful method for learning new skills is to use assistive elements. Some examples include the following:
    - arrows on the floor to help learn how to navigate a turn sequence
    - lines taped on the floor or onto field elements to assist with location or accuracy, or
    - making field elements different colors to improve contrast while learning.
- Over time, the driver picks up cues from places other than the assistive elements, such as how the robot looks when it's a specific distance from the driver or the alignment of the robot's bumpers with a field element. At that point, the assistive add-ons can slowly be removed.
- **Drive with Finesse** – Teams have different ways of saying this, including "Slow is Smooth, Smooth is Fast" and "Time is Priceless," but they all have the same underlying goal: approach driving with finesse, not brute force and raw speed. Overtly ramming game pieces into field elements might feel warranted in the moment, but it's not always productive and can instead cause avoidable robot damage, wasted time, or penalties. Competitive drivers know that five well-played seconds can change the outcome of the match.
- **Practice with Purpose** – The driver will often feel a sense of ownership and responsibility to the team to "not break the robot," causing them to not push the robot to its full potential and miss possible weak points in the robot early in the design and build phase. In contrast, the "Drive it like you stole it" technique helps identify and address issues before an event. Rely on the team's technical skills to keep the robot in tip-top shape; don't be afraid to push the robot to its limits while still driving with finesse.

What to Practice

To the extent that you are able, set up a practice environment as representative of the real field as possible. The Layout and Marking diagram and Team Element build guides on the Playing Field page can be a helpful reference for this:

- **Start Simple** – When starting practice with a new robot, start simple. Focus on the basic fundamentals of your robot. Test out the various functionalities you expect to use in a match to see how they work. Drivers need to become familiar with the robot and its controls. Look for things that feel awkward or difficult as they could be opportunities for iteration.
- **Practice Drills** – Musicians know an impactful way of learning to play an instrument is learning how to play scales, or specific ordered sequences of notes. Scales helps the musician develop speed, dexterity, and muscle memory. For drivers, drills are to the robot as scales are to the instrument. Drills can include driving slalom or other obstacle/patterned courses, picking up and releasing game pieces, and repeatedly scoring game pieces into/onto a practice apparatus. Some of these drills can even be used during driver selection, as talented drivers can be identified through drills and then their skills enhanced through practice.
- **Cycle Tests** – Practice the full loop of game piece acquisition, travel, and scoring. It will feel repetitive, but the more familiar you are with this process, the easier it will be to replicate it under pressure. Consider timing cycles or recording practice so you have objective data to review and understand how different techniques impact results. Make sure to practice the different paths, actions and situations you expect to encounter in a match.
- **Add Precision** – Try drills that test and improve your driver precision. For use objects to enforce a narrow and optimal drive path or repeat an object pickup or delivery drill until you consistently achieve 100% accuracy.
- **Add Variables** – A driver needs to be prepared for the various things that can happen in a match. Think about all of the “what ifs” that could happen during a match and start planning and practicing for the most likely ones:
    - If there’s an optimal path or scoring location, what if it’s obstructed?
    - What if certain functionalities of your robot break down during a match?
    - How do you respond to defense?
    - How can you share the space you are using with your partners?
    - Can you drive portions of the route backwards?

Analyze and Upgrade Cycles

While practice alone does help with improving cycles, analyzing what’s is and isn’t happening during those cycles can inform iteration and result in further improvements. Specific ways to do that include:

- **Think about the controls** – Think about the way the robot is controlled. WPILib comes with default implementations for arcade, curvature and tank drive. You can learn more about the differences between these in the WPILib documentation. Other permutations have been released within the community that may or may not feel more intuitive to your driver. Another aspect to consider is robot-oriented vs field-oriented drive. In field-oriented driving, the driver no longer considers the orientation of the robot when moving the joystick, pushing the joystick away from the driver moves the robot away from the driver, regardless of robot orientation. Field-oriented driving is most often used with robots with omni-directional movement such as mecanum or swerve drive robots, but is possible to implement using differential drive as well. Drive controls should be figured out early, as it will take time for a driver to adapt to any changes. Make sure the controls are intuitive. It’s okay if the incorrect buttons are being pressed at times while first learning but monitor this closely as it could be a sign that the layout doesn’t feel right to the driver.

- **Think about the controller** – Another thing to look at and test early can be the actual controller the driver is using. Different teams have found success with joysticks, gaming controllers, and event steering wheel & throttle combinations. Find an option that is intuitive and has the right range of motion for the drivers.
- **Analyze the cycles –** Analyze the cycles as they're performed. Break down the timing into three buckets: game piece acquisition, field traversal, and game piece delivery or scoring. Which of these three components uses the largest proportion of your cycle? How can you improve each component? Are there design or software tweaks that could improve your speed? Is there wasted time and movement? How does accuracy effect cycle efficiency?
- **Add automation** – One way to decrease cycle time is by adding automation. Examples include:
    - use computer vision to aid lineup with certain targets and destinations, as discussed in the Targeting with Vision section.
    - automatically position robot mechanisms, or
    - automate movement of game pieces within the robot.