**TEAM 9505 Bot Patrol**

|  |  |
| --- | --- |
| A person standing in front of a whiteboard  Description automatically generated | A child wearing glasses and a black shirt  Description automatically generated |
| **Quinton** – Senior, expert programmer (JAVA), and systems analyst | **Zac** – Senior, expert programmer (JAVA), and systems analyst |
| A person in a black shirt  Description automatically generated | A child in a grey shirt  Description automatically generated |
| **Dominic** – 9th grade, builder, mechanic | **Carter** – 8th grade, programmer-in-training |
| A person standing in front of a whiteboard  Description automatically generated | |
| **Mr. Woock** - Coach | |

**Control Award**

Our control system development followed the engineering design process as we learned from our successes and failures. Having a good plan is one thing. Seeing it through to success is a process we are intimately familiar with. In a humorous way, our engineering design process has been shortened to be Design – Build – Break – Repeat.

All our programming is written in Android Studio using JAVA. Our two programmers have been using JAVA for the last six years and have even taken a college programming class.

**Lessons Learned**: We learned a lot at the Minot qualifier. We had a good robot and programming, but our scoring mechanism when compared to the other robots’ was too slow. Our sample capture mechanism was a soft wheel on a continuous servo. It worked but was slow and somewhat inconsistent.

We decided to switch to a forked grabber design. Mechanically, it includes three servos; one is a “wrist”, another is a “pivot”, and the third one is a “grabber”. Since our plan was to automate as much as possible, the mechanical changes required extensive reprogramming. This clever programming reduces human error. Things like preventing the grabber-forks from hitting the floor using situational step-by-step sequences are essential to longevity and consistency.

We also discovered a few mechanical issues with our pull-up mechanism and had to rebuild it. The rebuild was also an improvement, but once again, it required more reprogramming.

Our two programmers spent an approximate, cumulative total of four or five days reprogramming all the automation. It is amazing how long something that seems simple really takes to fix. The payoff is that we can now consistently deliver all four samples to the top basket during the autonomous period. We can capture samples more quickly in driver mode, and our pull-up for the endgame is very consistent.

We also found more settings and capabilities in the Road Runner software. One of them we are now using is the ability to change the acceleration and deceleration. Using this feature increases accuracy and consistency.

We learned another valuable lesson the hard way when we placed a servo incorrectly and accidentally tried to make it push through the floor. We try hard not to miss things like this, but the lesson is that the more complicated things get, the easier it is to miss an important item. We coined the phrase “murdered another servo.” This year has been a real challenge.

We will certainly learn more at the Bismarck qualifier, but we are hoping not to make major changes. The next thing we need to work on is driver practice.

**Autonomous**: We make use of the Road Runner software to accurately traverse the field during the autonomous mode. Road Runner allows us to program the robot to run to a position on the playing field rather than use a linear sequence where each leg of the journey must be perfect. Adding odometry pods also contributed to increasing the accuracy of the path. In previous years, while using our old method of creating an autonomous, changing the position of one step in the autonomous would throw off every later step in the program. Road Runner eliminates this problem, since we are able to independently change one step without it having an adverse effect on later parts of the program.

Learning how to use the Road Runner program was a long and difficult journey! Throughout the year, we performed the tuning for Road Runner multiple times, each round of tuning was conducted following a robot design modification that could potentially impact the functionality of Road Runner. During our final round of tuning, one of the tests failed to work and we hit a roadblock. We struggled to identify the cause of the issue and sought assistance on the official FTC Discord. This turned out not to be helpful at all. We started from scratch a couple times and even re-downloaded all the software, but nothing worked. We nearly gave up only to discover an obscure setting in the robot configuration that was wrong. What a great feeling when we finally figured it out!

Clearly the new odometry pods programming caused some headaches. In addition, we also had some troubles with the physical pods themselves. We originally mounted them incorrectly, which allowed them to hyperextend, meaning that we had to carefully put them in the correct position every time we set the robot down. In version two, we mounted them too low causing the pods to carry weight, which made calibration impossible because the motors were not correctly propelling the robot. It was only through careful observation that we figured out this problem. We finally got them mounted correctly in odometry pod version three.

Autonomous Critical Diagram

A screenshot of a game

Description automatically generated

In addition to our newly working autonomous, found a neat application called “MeepMeep” that allows us to simulate the path our autonomous program will have the robot follow. This saves us a lot of time. Instead of setting up the robot on a playing field every time we want to see what the change we made will do, we can run a MeepMeep simulation. To be more specific on what MeepMeep does, it opens a tab on our computer and shows a game field map, with a little square representing our robot that traverses the field in the path dictated by our autonomous program. However, MeepMeep simulations are not perfectly 1 to 1. That is, MeepMeep shows the idealized, perfect run of the autonomous program. So, if we actually run a program on the playing field there might be some discrepancies caused by friction or other external force(s) that MeepMeep does not account for.

**Control Award Coding Challenges**:

Making a logical sequence of steps is straight forward and easy to determine. Testing and adjusting the actual values is extremely time consuming. Another challenge is to determine which functions to combine into a single button on the controller and to consider the human factor.

This year we are using even more global variables and methods. In other words, we gave almost every number we use a variable name so that it is easier on our programmers eyes to read and understand. We have also made the physical controls simpler for our drivers by putting the controls onto two separate controllers. We made these controls extremely simple so that one person controls only driving and the other controls the mechanical functions of the robot. These mechanical controls are so simple they just need one press of a button, and the robot will do a sequence of actions so there is no human error. Additionally, our programmers made methods for these sequential actions on the button pressed. This process helped make the program easy to read. Say for example, the driver presses “A” to make the robot go to a scoring position, the code would look something like:

*if (gamepad1.a && !aLastTime) score();*

The “score()” being one of the new methods our programmers made. Isn’t that so simple? Just one line of code for an entire sequence of actions to play. Fascinating. In summary, the controls are single button presses, which trigger sequential events that are pre-programmed; there is no human error.

**Driver Controlled**: We set up our controls in such a way that they are as simple as possible. Our idea was less thinking – less mistakes! We make use of two controllers. One controller is focused on driving, the other is focused on scoring mechanisms.

Ensuring that the robot CANNOT exceed the 42” horizontal limit was a journey in problem solving. To start, we did some complex math (see images below) to visualize the shape of the arc the robot’s arm creates when following the limit. The shape constructed was an ellipse (oval) using inches along the x-axis, y-axes, and angle ɵ (theta). This ellipse included all the values that would restrict the robot from going past 42” while still rotating. We simply had to convert inches into motor tick counts and gathered the values we needed for each arm position the robot will utilize. From there, we created a variable for each value we will utilize so the program is easy to read and understand. The robot will not exceed the 42” limit at any point because the program was mathematically created. Below are images of the math we did to calculate our values for the 42” limit.

However, this method proved very complex when trying to create the code. What we came up with did not really work properly. We had to do some more problem solving! The solution was extremely simple and our original plan, while accurate, was way more complicated than necessary. All we had to do was create fixed, set positions for picking up elements and dropping them that were within the game limitations. By eliminating random, driver-controlled distances, we prevent the possibility of going beyond the limit. The bonus is it also simplifies the driver control systems.

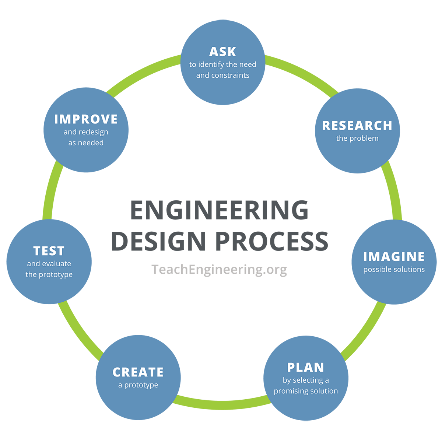
|  |  |
| --- | --- |
| A whiteboard with drawing of a diagram  Description automatically generated | A white board with math equations and graphs  Description automatically generated |

|  |  |
| --- | --- |
| A white board with math equations and numbers  Description automatically generated | A white board with math equations on it  Description automatically generated |

**End Game**: Our plan is to score two more samples in the high basket and do a level 2 hang.

We were originally going to automate the entire hang process with one button press but bent some metal parts getting it programmed. We decided to automate part of it and manually control the rest. More information on this is found in the Think Award section, below.

**Innovate Award Design Process:**

At the end of the year, last year, we decided to buy a new robot kit for two reasons. Our old robot was near the end of its life cycle with worn out gear boxes, shafts, and tired motors, some of which no longer worked. The other reason is we wanted to use the odometry pods to improve our autonomous mode. We went with the goBILDA kit because we liked the way it performed in competition. The new kit turned out to be quite a challenge. We had to come up with different kinds of solutions than we were used to because of the new parts.

We start each year with a clean slate and do not plan on using the robot from last year unless it meets our new criteria. We start by watching the challenge video a couple times, then digging into the game manual points section to determine the best way to achieve the most points. We also balance robot complexity and the parts we have with what we want to achieve. We weigh the complexity of our designs to the possible points achievable and go from there. We set our points goals for each of the three sections (autonomous, driver, end game) separately. These goals drive the design process.

The next phase is to brainstorm individual robot components. This often requires a quick build, prototype, as a proof-of-concept. This phase can take quite a while because we often design, build, and test several different versions to find the best one. Sometimes we work on components together and sometimes we assign a part to a specific person.

This year, our initial plan was to use one arm to do it all; capture elements, deliver them to the top basket and do the pull-up during end game. We discovered that to have enough torque to do the pull-up and to stay up for 3 seconds, the arm was too slow for reaching the top basket. We knew the pull-up arm slider needed a worm gear. This is because a worm gear can only be driven from the input side, and this would prevent gravity from pulling the robot back down. The 28:1 worm gear was the root cause of the speed issue. We ordered a faster motor, but that one did not have enough torque to lift our robot. Finally, we went with two arms instead of one, one to deliver elements to the top basket and one for doing the pull-up. We were trying to avoid this because the second arm added weight to the robot, complexity to the build, and complexity for the drivers. However, in the end having two arms proved more efficient so we can accomplish all our tasks faster.

**Transition Here**. Our original grabber mechanism was a single rubber wheel with a plate on one side. To operate the wheel, we would position the wheel up against a sample, then start the wheel spinning to capture the sample. However, this was very inefficient since it took an eternity to line up with a sample correctly. Additionally, we would have to make the wheel’s plate flush with the ground before we could capture a sample, meaning we would have to capture horizontally as opposed to over the top. This was just the beginning of the problems; we also ran into issues when scoring a captured sample into a basket. Sometimes, when our scoring arm was up, the wheel would clip the basket as it would dispense the sample into the basket. Clearly this was a bad thing, we did not want it to occur. So, after our first competition in Minot, we did some research and decided to design a pincher mechanism for grabbing samples. This pincher design was composed of two fixed metal rods on one side, and two other rods on the opposite side attached to a servo. The functionality of this pincher was very much akin to a person’s index finger and thumb pinching. To make this design much more versatile in its application of grabbing, we added another servo to the mechanism to allow for another axis of rotation. So, this design was able to rotate up and down vertically, and twist in the same way someone turns their wrist when rotating a doorknob. Although this design was very proficient, it required quite a bit of precision to line up with a sample from above. Due to this, we felt that there was room for improvement.

At the Bismarck Qualifier, we were inspired by other teams’ grabbing mechanism. We ditched our pincher design and replaced it with a dual-wheel design. Now, I know what you’re thinking: *Why would you go back to a wheel design, when your very first one didn’t work so well?* Great question and we would love to answer that. Our original wheel design had just one wheel and a plate on the other side to hold a sample in place. This new design has two wheels, one replacing the plate from our original design. This allows us to grab samples from many orientations like our pincher design and doesn’t require as much precision. To date, this is our best design.

Combining all the components onto the chassis was also a challenge. This process required individual components to go back to the drawing board. Several times this year, we built out two different arm apparatuses and put them on the chassis for testing and realized a design problem. This required us to disassemble the arms and redesign them separately. Similarly, there were times when we needed access to parts of the robot that were obscured by other apparatuses. In those cases, we had to remove several parts of the robot in order to gain access to the originally obscured parts and then put everything back together.

We often ran into the problem of overall size. Keeping the robot within an 18-inch cube is tough, especially this year. Additionally, there is the limit of 42 inches in the horizontal plane. To account for this limit, our team did complex math to figure out the perfect path for our arm to follow. For reference, the math has already been explained previously with images in the Control Award section. In the end we found a much more simple solution to the 42-inch limitation.

Our original plan was to do a level 3 pull-up on the top bar, but we discovered it was too time consuming, and we could gain one extra point simply by adding two more elements to the top basket and doing a level 2 pull-up instead. During practice and coding refinement for the pull-up, we actually bent one of our slider bars. We were going to automate the entire pull-up process in code with a single button press but bent a slider because we set a value too high. We learned it is better to walk our way up to values and to think more about what will happen if we are off. What is the danger if we get it wrong? Our final solution was to automate part of the process and manually do the rest.

The new kit was a challenge. Although we like it and it is more refined with things like bearings and larger shafts rather than bushings, we frequently missed some details that we previously did not need to consider. We also ended up rebuilding things to move them just a little bit to make clearance for other parts. This is a learning curve and we will be much better next year.

Another thing that required a major redesign was underestimating the weight and center of gravity of our finished robot. This became an issue the first time we tested doing the pull-up. We originally wanted to mount the pivot point of the arm at the end of the chassis to aid in delivering elements to the top basket and to prevent tipping the robot over with a fully extended arm. The problem was this caused the robot to lean at a severe angle, which prevented it from hanging on the lower bar and prevented the arm from reaching the top bar. Moving the arms toward the middle of the robot was a major redesign and time consuming.

We made several revisions to our two slider arms and had to order a few different motors to try to get the right balance between speed and torque. This is, once again, a learning curve with the new kit.

**Team Development**: Our programmers, Zac and Quinton are Seniors, and have been teaching an 8th grader, Carter, to become their replacement. Carter started by building his own robot and programming it himself. Then, he added more and more features to the robot, including an arm, wrist, and a fully functioning autonomous program. Carter has truly learned a lot from our Seniors and will be ready to take on the challenge of programming on his own next year.

**Goals for INTO THE DEEP**: (current goals)

*Autonomous mode*:

1. Score pre-loaded sample into the high basket for 8 points.
2. Pick up and score 3 more samples from the tiles into the high basket.
3. Park in ascend zone in preparation for driver mode.

*Driver mode*:

1. Grab as many samples as possible and score in the high basket.
2. Help our partner.

*End game*:

1. Put two more samples in the high basket.
2. Make a level 2 ascend.

**Outreach**: We focus our outreach efforts on our elementary and middle school by promoting STEM activities and robotics demonstrations. We were also instrumental in helping to create an FLL team. We assist the FLL team at times with helpful hints and tricks, but we try to help them learn rather than doing things for them.

This year, our school hosted a family coding night using code.org. This was targeted towards younger students and their parents to raise interest in programming and robotics. On this coding night our team introduced the students to block programming and helped them throughout the code.org puzzles.

  
Carter – Explaining how last year’s robot chassis works. (family code night)

**Fundraising & budget**: We operate the school concession stand three times a year and earn a portion of the proceeds. We also have some great sponsors such as The Union Bank and Hefty Seed.

We spent over $3600 on our new goBILDA robot kit and extra parts this year. We were able to do this because we did not spend much last year and have been saving for a couple years.

**Lessons learned and plans for next year**: We are already talking about how we can be more efficient and get a working robot more quickly for the next year. Also, we are preparing for the loss of our instrumental Seniors. Our 8th grader, Carter will have to fill their shoes all by himself.