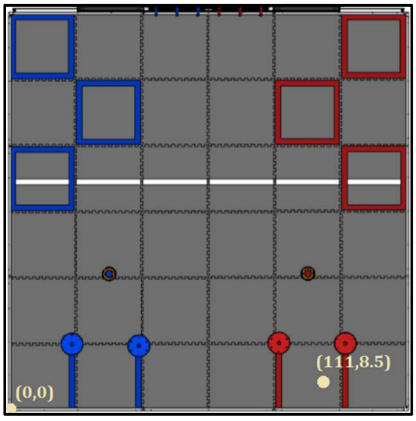
Programming

Once we had decided on the basic design of our robot and what we would be focusing on, we began programming. Using Android Studio Java our programming team wrote functional programs which we integrated into autonomous and Tele-Op.

Our initial take on the game was to navigate the field by counting the number of tape lines of the field and determining what line the robot was over by using a color sensor on the bottom of the robot. Ultimately, autonomous transportation is the goal so we had odometry in the works during the development of the color sensor navigation system.

Odometry

Using three encoders on motorless wheels placed on the underside of the robot's right, left, and middle the program can judge the robot's position on the field as if on a coordinate graph measuring as the robot moves in a different thread than the running program calculating orientation, change in the encoder counts between the left and right wheel, and distance in inches from the user indicated starting point. Since odometry creates a coordinate system, we created a method that calculates and sets the powers for our mecanum drive to navigate the robot to the target position. The robot travels along the hypotenuse of the current and the desired position. In a linear opMode, the movement is updated within a while loop, or an if statement with boolean method in an iterative opMode, determining the powers to set to the motors to get the robot to the correct position where the loop breaks when the robot is within the allowableDistanceError variable, a failsafe that allows for the method to account for an overcompensation of the motors—the robot does a little dance jerking back and forth when trying to get to an exact position because the robot is incapable of going to the exact encoder counts with a powerful mecanum drive that is also the cause of deviation in the accuracy of odometry over time when the robot is driven.



Since odometry measures distance, it can be plotted out as a coordinate grid where the lower corner of the blue side of the field is the coordinate pair (0,0). To use the odometry thread, the OdometryGlobalCoordinatePosition's class constructor is called where the coordinate of the starting position of the robot is indicated so that this position can be utilized to navigate the robot to the right positions. The coordinates of the robot are measured from the point at the center of the robot. During autonomous, when the robot starts on the right red tape line, the point that the robot starts at is 111, 8.5, or 111 inches from the left most wall and 8.5 inches from the wall behind the robot. A typical call to the goToPosition method would contain parameters including desired X and Y coordinates, desired orientation, power, and allowable distance error creating an accurate autonomous transportation system.

Distance and Color Sensors for Collecting/Conveying/Launching System

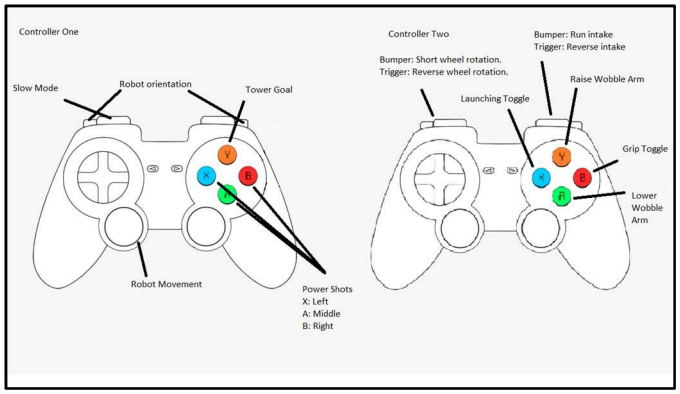
In autonomous to launch power shots, prior to our current method, the plan was to use the conveying wheel's encoders and move the wheel as many inches as the ring's diameter to make sure that the rings

were getting from our collection system to the launchers. However, the method was ineffective because the wheel did not account for the actual position of the ring. We decided to go with a distance sensor

before the ring enters the launching system so that the program could take the distance reading and determine if there was a ring in the vicinity and convey it to the launching wheels when it is appropriate. The first choice was to go with a traditional 2m distance sensor which we then learned could not accurately measure within 5 cm meaning that it was not effective for the desired range.
In the driver-controlled period, the program uses a 2m distance sensor in the collection system which will indicate when there is a ring in the system and power the conveying wheel so that we can collect more rings and store them automatically minimizing the driver's task.

Autonomous

By using a camera we can view how many rings are present on the field within a confined box in the camera's field of view. Once it has been determined how many rings are present, the program diverts into either box A, B, or C and navigates using Odometry. Once the robot has determined the correct target zone, it drives to the given positions (careful not to run over rings) and drops the Wobble Goal. It then continues to use Odometry in order to reach the correct positions to shoot the power shots. After that, Odometry coordinates are used again to grab the second Wobble Goal and bring it to the correct box as well. Once these tasks have been completed the robot drives to the white line. When all goes according to plan, this scores a total of 80 points.



Tele-Op

During Tele-Op drivers focus on ensuring both Wobble Goals are behind the white line and fire rings into the high goal.
During End-Game the drivers place the Wobble Goal over the wall and shoot all three Power Shots.

Controls for driver-operated period

Resources

We used several helpful resources to execute effective programs that taught us what we did not already know and allowed us to learn quite a bit more. Since we appreciate the aid from our resources, we also feel it important to share our knowledge with other teams and otherwise curious people. We upload videos and tutorials on our youtube and website in hopes of passing on our knowledge!

***Learn Java For FTC* by Alan Smith**
For Odometry: *Wizards.exe* on Youtube
Nucor
Mentors
Our resources for others: www.youtube.com/channel/UCV8jIg5kdLCqcKzkJoLUpzA
sites.google.com/view/chs-programming/