

Coding & Autonomous

We program our robot using Java in Android Studio. The three main modules in our robot's code are the **Hardware Map** which connects software to hardware based on the Control Hub's wiring, the **OpMode** which assigns inputs from the drivers' gamepads to actions on the robot, and the **Autonomous** segment which navigates the field to complete tasks without human input. Below is a table summarizing our main modules and the methods comprising each one.

Hardware6417.java - initialize hardware variables, store basic navigation methods	
init()	Assign all motors, servos, and sensors to a variable to access and control throughout the robot's code
Driver-Controlled methods	
shoot()	Spin front motor on launching mechanism to shoot rings
intake()	Spin motor controlling intake system and back motor on launching mechanism to intake and carry wheels
arm()	Raise and lower servo arm to grab Wobble Goal
grab()	Open and close grabber servo to grip Wobble Goal
setDriveSpeeds()	Supply power to drivetrain motors to drive, strafe, and rotate
Autonomous methods	
drivetoPosition()	Drive forward and backward using encoders
strafetoPosition()	Strafe left and right using encoders
stop()	Set all drivetrain motors to 0 power
resetAngle()	Set current detected angle to 0
getAngle()	Get current IMU reading
rotate()	Rotate robot using IMU

MecanumDriveOpMode.java - control robot during TeleOp using gamepads	
runOpMode()	Continuously read values from gamepads to control robot motors and servos <ul style="list-style-type: none"> - Gamepad 1: Control robot's physical position using joysticks to drive, strafe, and rotate - Gamepad 2: Control robot's systems to complete tasks on the field (picking up wobble goal, taking in and shooting rings)
nudgeRobot()	Power the robot's drivetrain motors to drive, strafe, and rotate in tiny amounts—good for navigating tight corners and completing tasks that require a lot of precision (i.e. picking up Wobble Goal)

Auto6417.java - control robot during TeleOp using gamepads	
initTfod()	Initialize TensorFlow object detection engine <ul style="list-style-type: none"> - This library provides us with an image recognition model that allows us to identify objects on the field
initVuforia()	Initialize Vuforia localization engine <ul style="list-style-type: none"> - This library allows us to determine the location of different objects on the field using the webcam
runOpMode()	Run 30-second Autonomous code <ol style="list-style-type: none"> 1. Move forward and rotate slightly to detect objects 2. Determine whether there is a Quad, Single, or no rings on field 3. Rotate back to face white line 4. Move forward 5. Shoot pre-loaded rings at Power Shot targets 6. Based on the object detected in Step #2, deposit pre-loaded Wobble Goal in correct target zone

Autonomous Program Diagrams

Scenario 1: 4 rings detected

1. Move forward ~12 in.; rotate ~5 degrees to detect rings and rotate back
2. Move forward ~60 in.
3. Shoot rings towards Power Shot targets
4. Drive forward into wall
5. Move back ~6 in.; rotate 90 degrees; drive forward ~30 in.
6. Deposit Wobble Goal in Target Zone
7. Move back ~6 in.; strafe for 4 seconds until parked on white line

Scenario 2: 1 ring detected

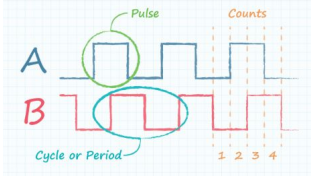
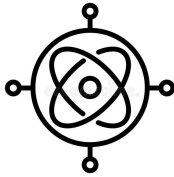

1. Move forward ~12 in.; rotate ~5 degrees to detect rings and rotate back
2. Move forward ~60 in.
3. Shoot rings towards Power Shot targets
4. Drive forward ~60 in.
5. Rotate 90 degrees; drive forward ~30 in.; deposit Wobble Goal in Target Zone
6. Move back ~6 in.; strafe for 2 seconds until parked on white line

Scenario 3: 0 rings detected

1. Move forward ~12 in.; rotate ~5 degrees to detect rings and rotate back
2. Move forward ~60 in.
3. Shoot rings towards Power Shot targets
4. Drive forward ~12 in.; rotate 90 degrees; drive forward ~60 in.
5. Deposit Wobble Goal in Target Zone

Sensing:

To aid us during autonomous navigation, we use the following sensing techniques:

<p>Encoders</p> 	<p>Encoders are sensing devices that detect the status of a motor based on electrical pulses. Our robot's motors run on 753.2 pulses per rotation - each time the motor completes one rotation, the encoder counts 753.2 electrical pulses.</p> <p>We can tell the robot to drive until it counts a specific number of pulses - if we want our wheels to complete one half rotation, we will program the motor to drive until the encoder counts $\frac{753.2}{2}$ pulses, or about 377 pulses. Using the circumference of the wheel, we can convert pulses to inches to drive a specific distance on the field.</p>
<p>Inertial Measurement Unit (IMU)</p> 	<p>The IMU is an internal gyroscope built into the robot's Control Hubs. We use the IMU to detect the current orientation of the robot and rotate specific distances.</p> <p>In the future, we plan to use the IMU to program self-correcting driving and strafing - the IMU will detect small changes in the robot's orientation while the robot drives, and we can use this value to supply or subtract power from the motors to correct the robot's orientation.</p>
<p>TensorFlow Object Detection</p> 	<p>TensorFlow Object Detection uses a pre-trained machine learning model to recognize objects through the webcam. We use this library to detect the number of rings on the field to determine which target zone to drop the wobble goal.</p>

Aside from those listed above, we've experimented with other sensing techniques throughout the season that didn't make the final robot, including:

- **Vuforia Localization:**

- Vuforia localization is used to determine the relative location of objects on the field using image recognition. Originally we tried using Vuforia Localization when our camera started detecting random "objects" (i.e. it thought the red wall was a Quad object)
- We used localization to record objects only within a certain distance and at a certain angle - if the object was too far away, or if it is not directly in front of the robot as the object on the field should be, we would ignore the recognition
- When we repositioned the camera, the image recognition became much more reliable so we no longer needed localization - we removed localization from our code for greater efficiency

- **Color Sensor:**

- We tried implementing a REV Color Sensor to detect luminosity and recognize white line on the field
- Even after we added the color sensor to our software and hardware map, the Driver Station couldn't recognize the color sensor even after we plugged it in and configured it. We spent some time trying to debug (swapping out wires, re-configuring, restarting robot) and research and realized that other teams have encountered this issue as well
- It is suspected to be a glitch in the RobotController Software Development Kit - there isn't much we can do about that so we decided not to spend too much time to add it
- In the future, we hope to add the color sensor back (perhaps using different hardware, i.e. Modern Robotics color sensor) for an additional reference point during Autonomous

Development Log:

Below is a table documenting our progress on the robot's code throughout the season. We store our code in a GitHub repository which helps us track our development. Anyone on the team can access the GitHub repository and download the code onto their own computer. We've used GitHub in previous years as well, meaning that we can go back and reference old code should we decide to implement a similar feature on our current robot.

Thanks to GitHub version control, we can easily review and restore previous versions of our robot's code.

Date	Notes
Oct 24, 2020	<ul style="list-style-type: none">- Set up Control Hub wi-fi connection- Connect webcam to control hub- Upload and test sample webcam program
Nov 14, 2020	<ul style="list-style-type: none">- Add drivetrain code for driver control during TeleOp- Challenges: adjust weights of robot to ensure that driving and strafing are straight
Nov 28, 2020	<ul style="list-style-type: none">- Add code for shooter- Test to see which motor strength is optimal for shooting different goals on the playing field
Dec 5, 2020	<ul style="list-style-type: none">- Continued to test shooter code after replacing broken motor- Finalize optimal powers to launch at specific targets<ul style="list-style-type: none">- robot.shoot(0.85) = best for shooting top slot- robot.shoot(0.75) = best for shooting Power Shot pegs- robot.shoot(0.65) = best for shooting middle slot
Dec 19, 2020	<ul style="list-style-type: none">- Add servo code to drop wobble goal
Jan 9, 2021	<ul style="list-style-type: none">- Refine TeleOp code for driver convenience as we start to test more:- Adjust controls for armServo and grabServo to better grip and deposit Wobble Goal- Change setDriveSpeeds so that the left and right joysticks control the left and right motors of the robot, respectively- Distribute roles between Gamepad 1 and Gamepad 2:<ul style="list-style-type: none">- Gamepad 1 controls driving and nudging- Gamepad 2 controls grabber arm, intake, and shooting

<p>Jan 14, 2021</p>	<ul style="list-style-type: none"> - Add encoders to drivetrain <ul style="list-style-type: none"> - For now, only one encoder - last year we had problems making sure all of the encoders were working at once, so this year we're starting with only one - Add methods to drive and strafe with encoders during autonomous - Challenges: modify motor speeds in order to drive and strafe straight (weight distribution is still not perfect but at this point we can't change the robot structure too much)
<p>Jan 21, 2021</p>	<ul style="list-style-type: none"> - Add method to rotate using IMU - Start programming autonomous - Determine where the robot must drive in order to accurately identify number of rings <ul style="list-style-type: none"> - Too far = cannot recognize - Too close = rings may not be in the camera frame - Challenges: adding color sensor to detect white line <ul style="list-style-type: none"> - RobotController didn't register color sensor despite being configured correctly on the hardware map - Researched problem and eventually realized that many teams were running into this same issue - must be a glitch in SDK which we can't do anything about - Decided to stop using ColorSensor for now to save time
<p>Jan 25, 2021</p>	<ul style="list-style-type: none"> - Test autonomous and complete scored practice rounds - Max score autonomous: 35 - Challenges: trouble recognizing Quad objects (TensorFlow model will think that a stack of 4 rings is a single ring) <ul style="list-style-type: none"> - Tried modifying position of robot but Quad recognition is still unreliable—TensorFlow recognizes single rings and no rings consistently and sometimes recognizes Quads but usually labels it as a single
<p>Jan 26, 2021</p>	<ul style="list-style-type: none"> - Continue testing autonomous - the code itself is mostly finished but we are having problems with consistency of the image recognition - Challenges: sometimes the bot will detect a random object on the field that isn't there - detects "Single" when it's really none; detects three objects when there's really one object, etc. <ul style="list-style-type: none"> - Figured out a more reliable position to mount the webcam - we raised it so that it was looking down at

	<p>the rings rather than looking up at them</p> <ul style="list-style-type: none"> - Less interference from background - higher contrast when it's only looking at grey field + bright yellow rings
Jan 27, 2021	<ul style="list-style-type: none"> - Finalize tweaks to autonomous <ul style="list-style-type: none"> - Mainly just adjusting distances so that the robot more reliably parks + deposits + shoots - Challenges: Varying battery levels changes the robot's behavior slightly (i.e. supplies less power to motors while shooting = rings are shot a little low if the battery is low); solution was to create multiple versions of autonomous with tiny changes to take into account state of robot's battery
Present	<ul style="list-style-type: none"> - Continuing to test and refine autonomous