MetroBotics Team #14212

[First Program Meeting] [September 27, 2021]

[Eric Guan, Samir Selmani, Sergio Alegre, Erick Soria, Kareem Khairy, Ashley Scudieri]

Think:

We want to determine the program type that we are going to use (JAVA or BLOCKS), learn more about Autonomous and Op-mode, and construct a strategy for Autonomous and Op-mode

Design:

Tutorial for operating Blocks

Test:

We are conducting tests on blocks by making a code for the robot to move in the lateral direction.

Result:

Signatures:_____Date:_____

Blocks was determined to be the program type to be used based on previous knowledge and experience with the program.

Next Step:

Learn / Teach how to set-up the phones for the robots. Learn / Teach how to set-up the hardware for the robots. Learn / Teach how to start coding with blocks.

Signatures:_____Date:_____

MetroBotics Team #14212

[Setting up] [September 28th, 2021]

[Samir Selmani, Eric Guan, Ashley Scudieri, Sergio Alegre, Kareem Khairy, Erick Soria]

Think:

Learn how to set-up the phones for the robot, how to set-up the hardware for the robot, and how to start coding with blocks.

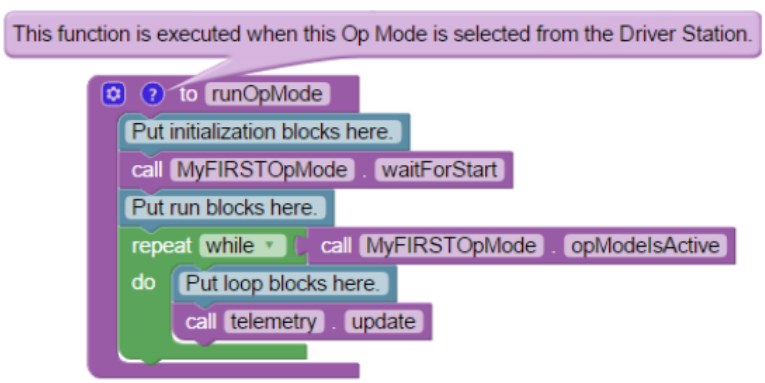
Design:

Telemetry Statements: PDF Manual Page 75
Allows us to see motor power information from the OP-mode.
To connect the remote control to run the op-mode, we need to press Start + A. This links the controller to the phone and allows us to run op-mode.

Test:

6.7 Examining the Structure of Your Op Mode

When you create a new op mode, there should already be a set of programming blocks that are placed on the design canvas for your op mode. These blocks are the default with each new op mode that you create. They create the basic structure for your op mode.



Result:

We summarized the key ideas in the Blocks Programming Manuals.

Next Step:

Practice coding the robot.
Begin research on TensorFlow and other sensors for Autonomous Mode.

Signatures:_____Date:_____

MetroBotics Team #14212

[Programming Practice] [October 1st, 2021]

[Eric Guan, Samir Selmani, Sergio Alegre, Erick Soria, Kareem Khairy, Ashley Scudieri]

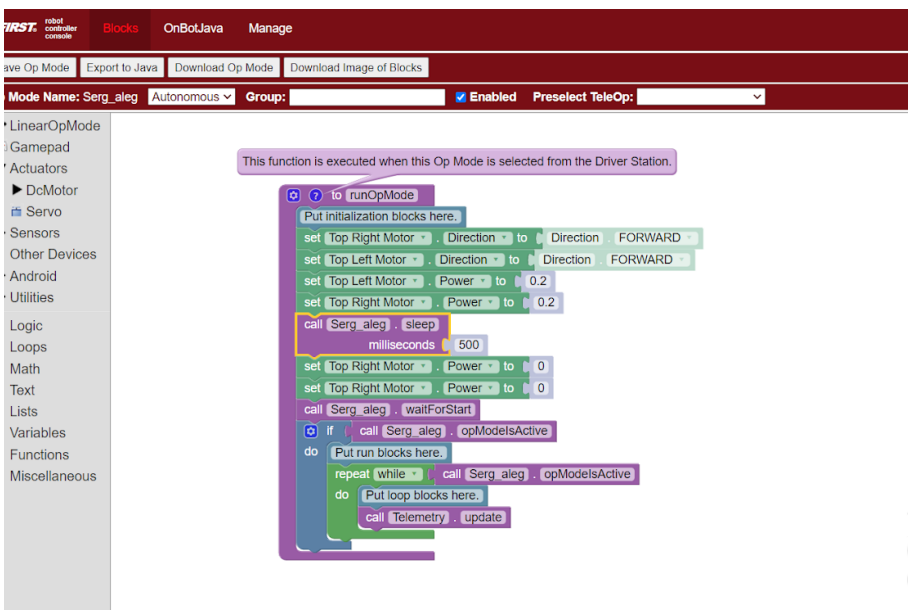
Think:

The basic testing code has been successful and now research is required for the operation of TensorFlow. Practice with coding is in order and the beginning of teaching the basics of coding to new team members is being conducted through the use of a test robot.

Design:

- Code the Drivetrain
- Code a simple autonomous mode

Test:



Testing Autonomous Mode Code

Result:

We managed to map controls to the controller and create a simple autonomous mode.

Next Step:

Continue to discuss plans with the Build Team.
Continue to research TensorFlow.
Continue to improve the code.

Signatures:_____Date:_____

MetroBotics Team #14212

[Autonomous-Mode Programming] [October 5th, 2021]

[Eric Guan, Samir Selmani, Sergio Alegre, Erick Soria, Kareem Khairy, Ashley Scudieri]

Think:

To make a more precise autonomous mode, we need to use the built in encoders on the motor. This allows us to move based on precise ticks per revolution instead of power over an interval of time, which can change depending on the power level.

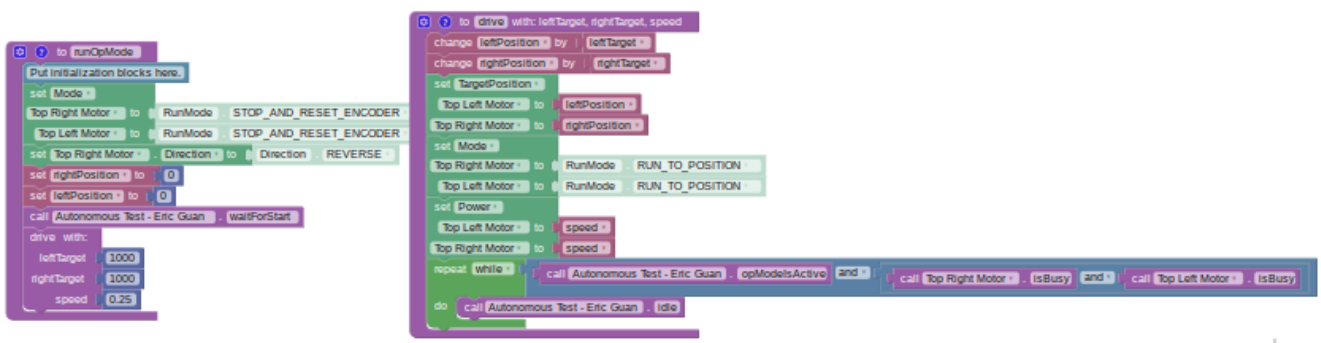
Design:

To do this, we need to follow this format:

- Reset the Encoders
- Turn on the Encoders
- Set a target position
- Set motor to “Run to Position”
- Set a speed for the motor
- Create a while loop to allow time for the robot to move

Test:

Autonomous Mode Test Using Encoders



We wrote a method that follows the design. This makes the code cleaner and easier to use.

Result:

The code we wrote performed as intended. Depending on how we adjust the target and power levels, we can move forward and turn autonomously. This code allows the motors to move more precisely as compared to our tests with power over time.

Next Step:

Further improve our autonomous mode. Try to implement TensorFlow and set up an external camera.

Signatures:_____Date:_____

MetroBotics Team #14212

[Testing Robot on Mats] [October 8th, 2021]

[Eric Guan, Samir Selmani, Sergio Alegre, Erick Soria, Kareem Khairy, Ashley Scudieri]

Think:

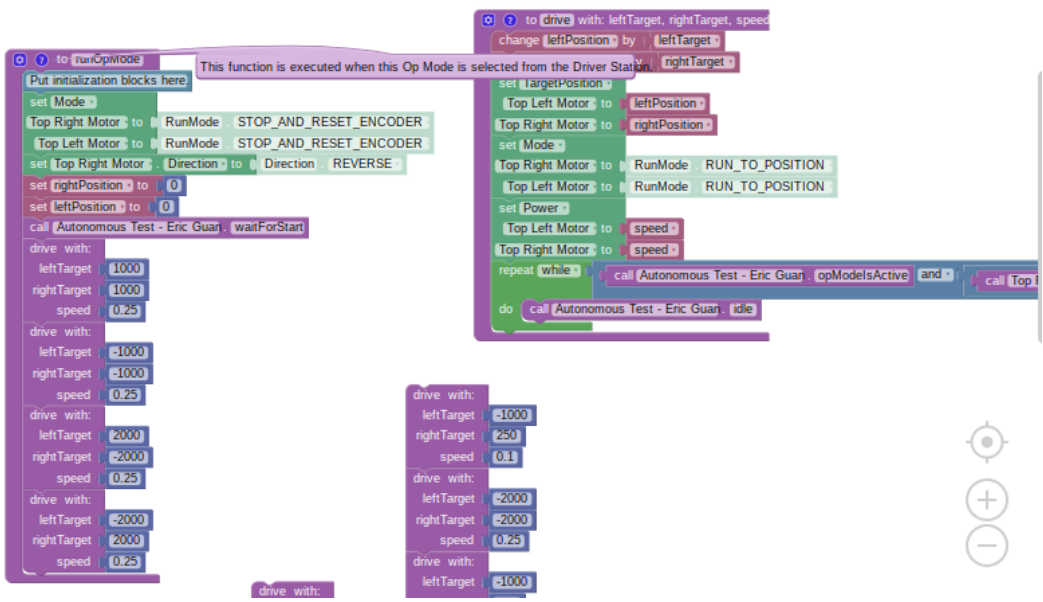
To test and experiment with our new autonomous code, we put our robot on mats to simulate the playing field. We wanted to see how the robot performs on mats as opposed to hardfloor.

Design:

We cleaned up the playing field and placed robots against the wall.
We then used the encoder method to perform various types of movements.

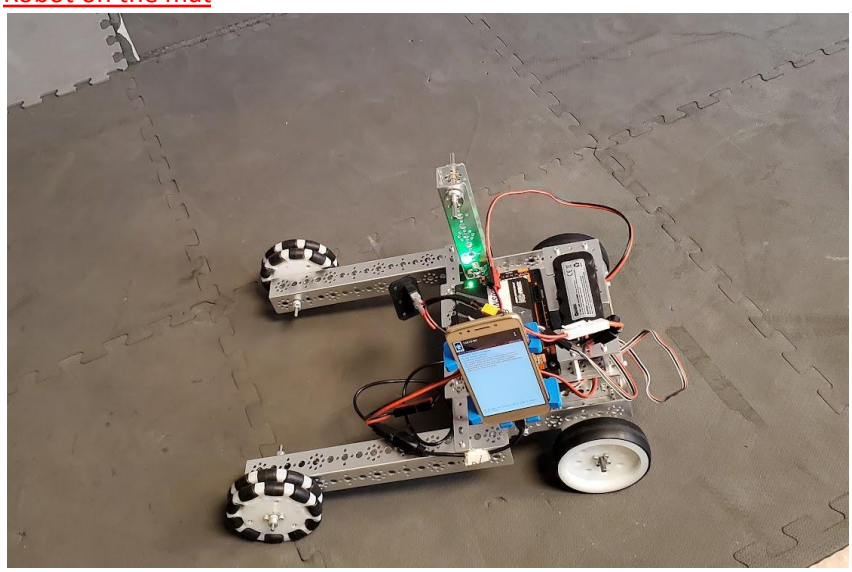
Test:

Testing how multiple commands work on the playing field:



Signatures:_____Date:_____

Robot on the mat



Result:

The test robot runs fine on the playing field mats. It is still very fast and reliable. At the moment, there is no misdirection occurring on the mats.

Next Step:

Try to implement TensorFlow and try to set up the camera.

Signatures:_____Date:_____

MetroBotics Team #14212

[Integrating Sensing Capabilities] [Week of October 12th, 2021]

[Eric Guan, Sergio Alegre]

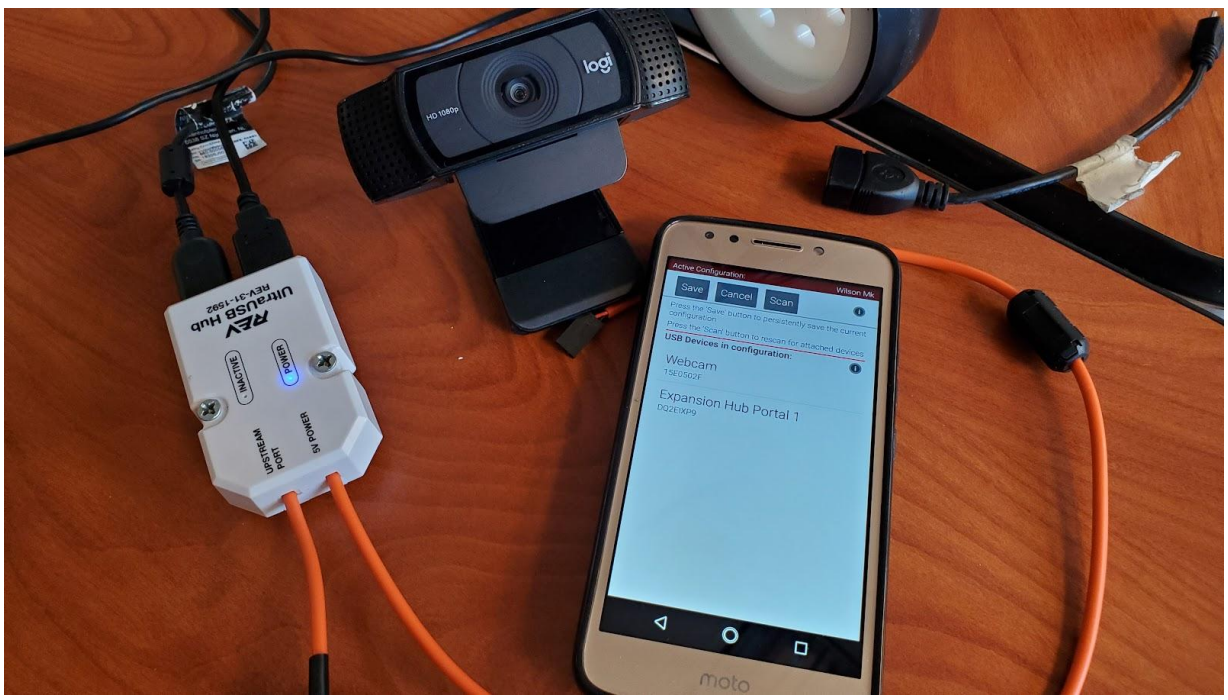
Think:

Attach the Webcam to the robot and connect the camera to the Hub,Test the robot on the playing field. Organize wiring. Continue research on Tensorflow

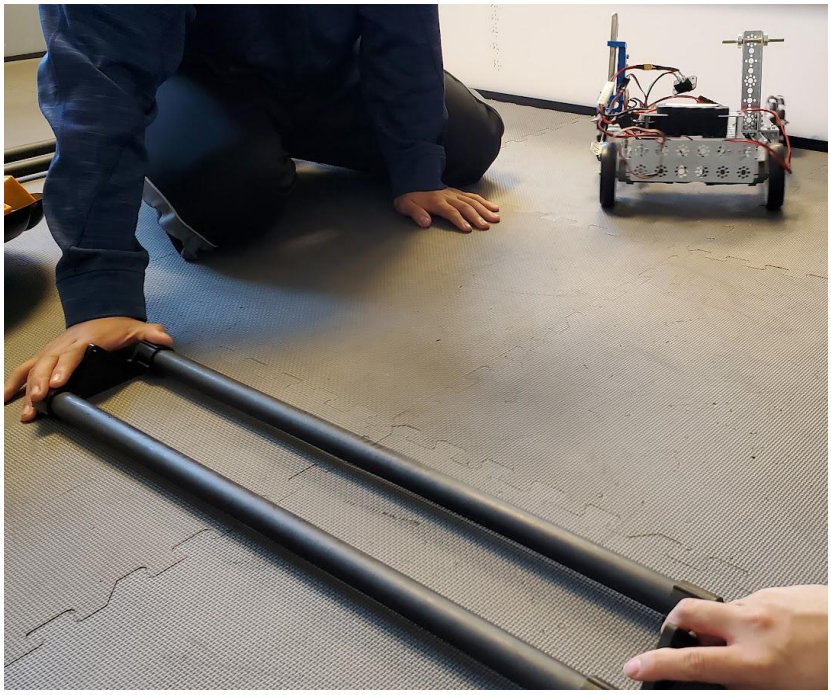
Design:

The wheels of the robot can get stuck in between the barriers. If this happens, the robot gets stuck. We possibly need bigger wheels or more traction on the wheels.

Test:



Signatures:_____Date:_____



Result:

If ready, write the TensorFlow code and test. Continue to experiment with the robot on the playing field.

Next Step:

Client has been successfully updated. (tutorial?) Webcam successfully connected. (tutorial?) Test Robot can go over the barriers but it can get stuck. Not consistent enough for competition use.

Signatures:_____Date:_____

MetroBotics Team #14212

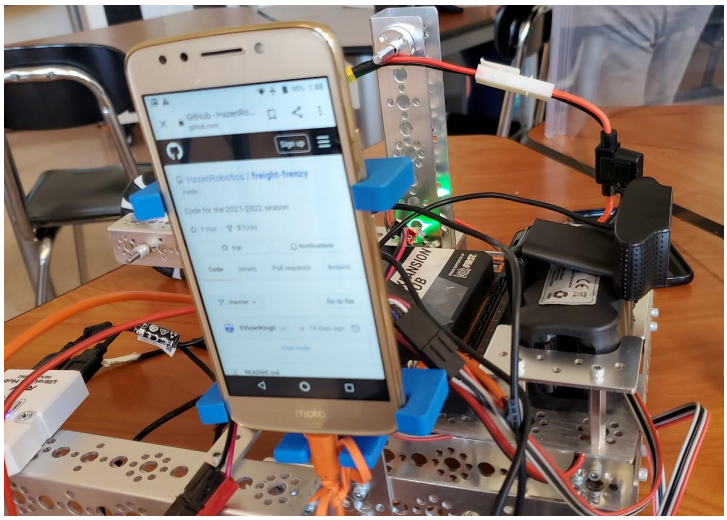
[TensorFlow Test 1] [Week of October 18th, 2021]

[Eric Guan, Samir Selmani, Sergio Alegre]

Think:

Design a Strategy for Autonomous Mode after reviewing the game manual.Testing the webcam using Tensorflow Object Detection (ultimate goal for now)

Design:



Test:

We have determined that the webcam is functioning properly. We are in the process of updating the controller to access the sample code for Freight Frenzy.

Result:

After reviewing the game manual, we are in the process of creating a strategy for autonomous mode. We are waiting for the build team to complete the robot to see what points are possible for us to obtain. We tested the webcam using the Ultimate Goal TensorFlow sample op-mode. We need to update the phones and apps to access the Freight Frenzy sample code.

Next Step:

Continue to create a strategy for Autonomous Mode. Update the phones and write Autonomous Code for Freight Frenzy

Signatures:_____Date:_____

MetroBotics Team #14212

[TensorFlow Test 2] [Week of October 25th, 2021]

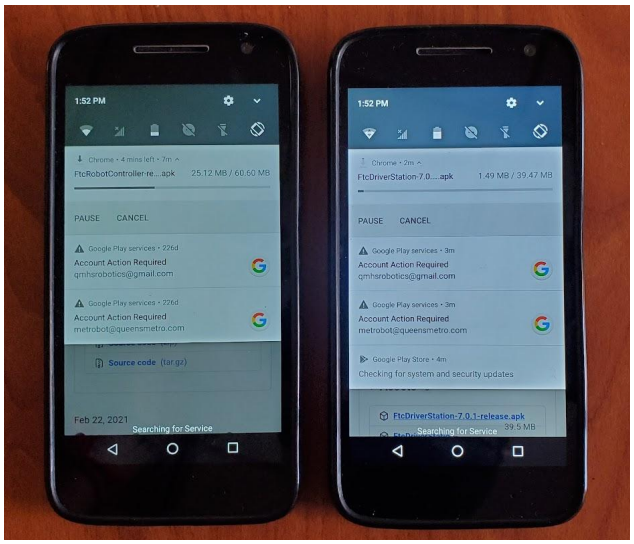
[Eric Guan, Sergio Alegre]

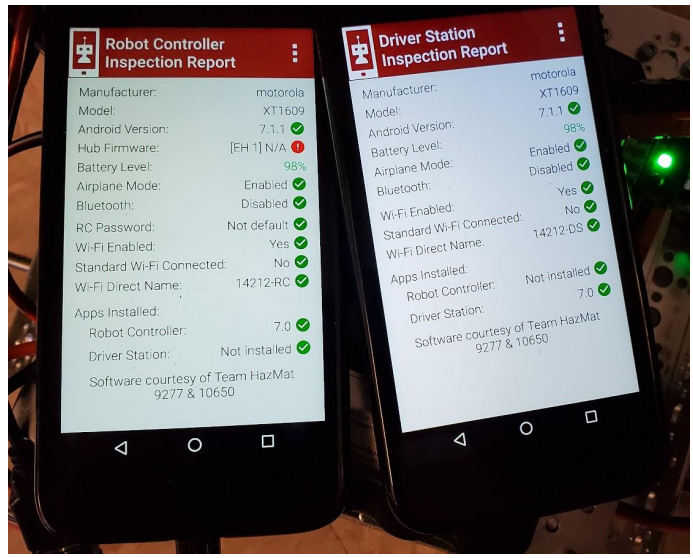
Think:

Continue to create a strategy for Autonomous Mode.Update the phones and test Autonomous Code for Freight Frenzy

Design:

The apps have been successfully installed:



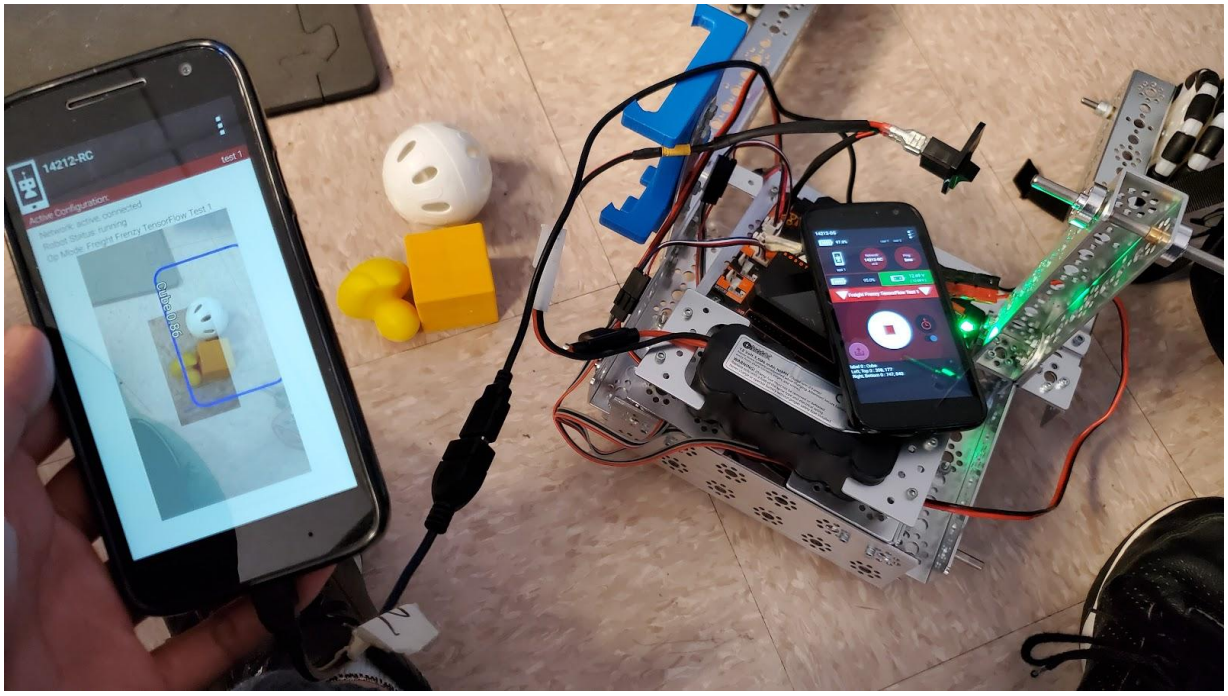


Signatures:_____Date:_____

MetroBotics Team #14212

Test:

Testing the FreightFrenzy TensorFlow Sample Op-mode



Result:

The updated apps for the Controller and Driver stations have been installed (Ver 7.0). We can now access the TensorFlow code for Freight Frenzy.We are testing the Object Detection in the sample TensorFlow op-mode. The objects aren't being detected quickly or reliably.

Next Step:

Continue to create a strategy for Autonomous Mode. Try to code a working autonomous mode implementing TensorFlow object detection. (use videos and previous years as reference) Learn how to increase the accuracy of TensorFlow and detect other objects.

Signatures:_____Date:_____

MetroBotics Team #14212

[TensorFlow Test 3] [Week of November 1st, 2021]

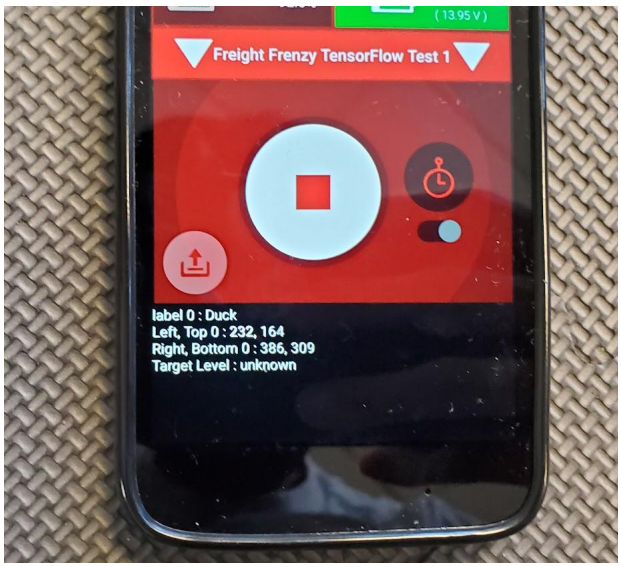
[Eric Guan, Samir Selmani, Sergio Alegre]

Think:

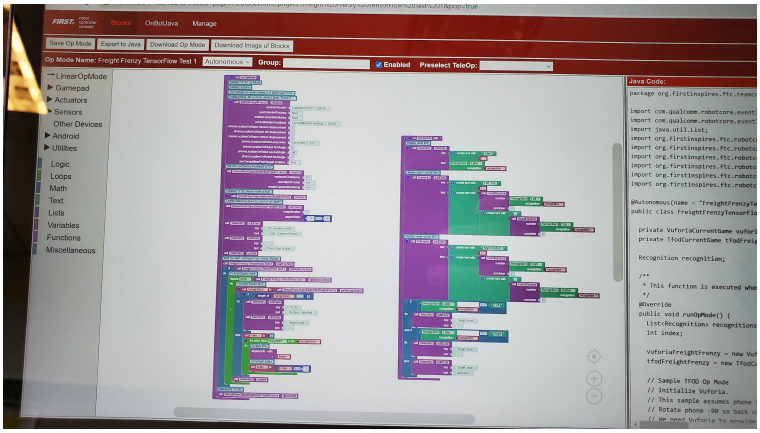
Continue to create a strategy for Autonomous Mode. Try to code a working autonomous mode implementing TensorFlow object detection. (use videos and previous years as reference) Learn how to increase the accuracy of TensorFlow and detect other objects if necessary.

Design:

Telemetry Data



Test:



Our current autonomous code using tensorflow.

Signatures:_____Date:_____

Result:

the TensorFlow code, since the rubber ducks could be placed on any of the 3 dots, we think we can use the coordinates received by the telemetry data and boolean statements to determine which code to run (3 options) which we will attempt next week.

Next Step:

Continue to create a strategy for Autonomous Mode. Try to code a working autonomous mode implementing TensorFlow object detection and use videos and previous years for inspiration and reference.

Signatures:_____Date:_____

MetroBotics Team #14212

[Scrimmage Preparation] [Week of November 8th and November 15th]

[Eric Guan, Alex Wu, Kevin Xia]

Think:

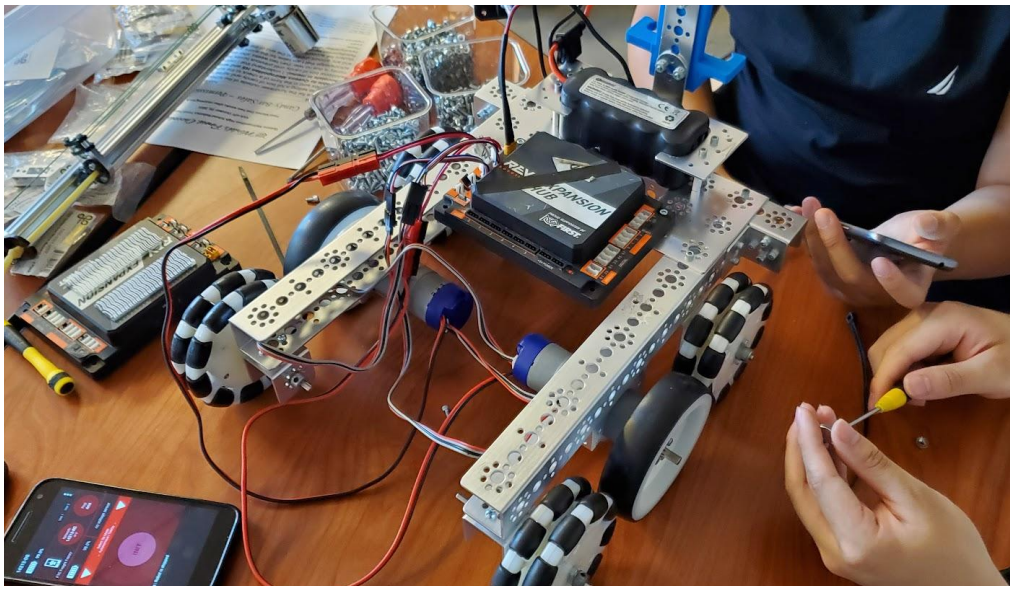
Create the Autonomous and Op-Mode for the Scrimmage Robot. Program the Arm. Set-up the Double Expansion Hub

Design:

No designs were available this week

Test:

Configuring the Phones with the new robot (naming the motors)



Result:

Double expansion hub is installed and works properly (update expansion hub 2 after the scrimmage) The basic Op-mode is complete (with arm). Autonomous mode is being worked on.

Next Step:

Use knowledge from the Scrimmage to make changes to our robot code. Improve the autonomous mode and include TensorFlow. Update the 2nd Expansion Hub after the scrimmage.

Signatures:_____Date:_____

MetroBotics Team #14212

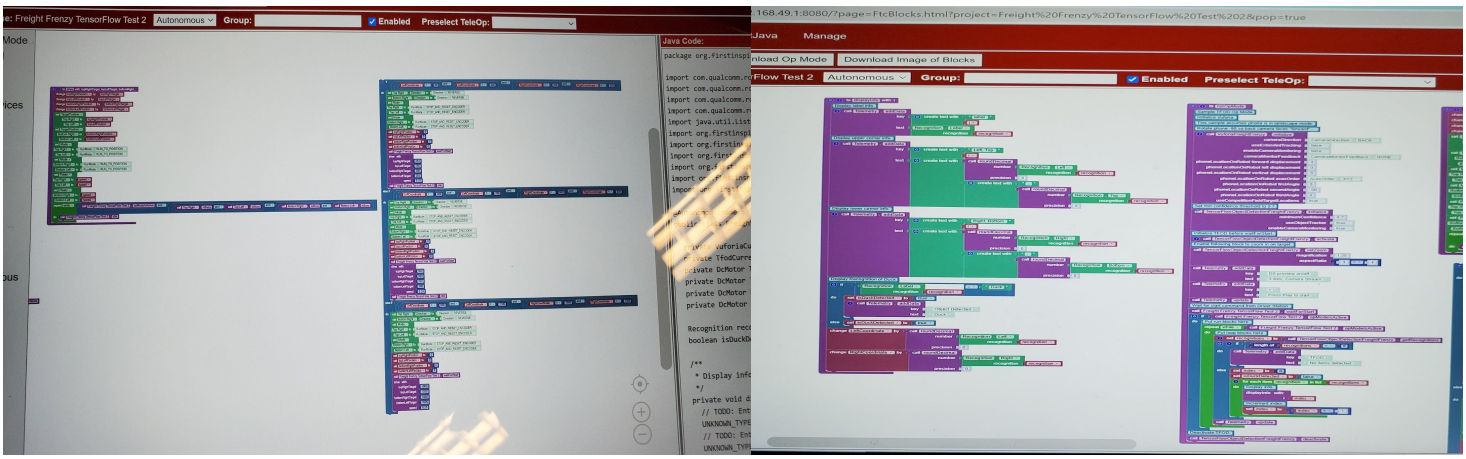
[TensorFlow Test 4] [November-22-2021]

[Eric, Sergio, Samir]

Think:

The robot does not have autonomous mode and the development of TensorFlow code will be conducted through the week and following after the scrimmage; Update the 2nd expansion hub.

Design:



Autonomous Code(With TensorFlow) Skeleton.

Test:

The camera could detect what its coordinates were but could not perform actions.

Result:

The camera had difficulty detecting objects and needed to be reangled for better detection of objects.

Next Step:

Find a better spot or angle for the camera to be positioned in a area where tensorflow can detect the objects the best.hold off on designing the custom shipping element until the code can work on objects.Update the 2nd Expansion hub after the scrimmage and get the code ready before the December 4th competition

Signatures:_____Date:_____

MetroBotics Team #14212

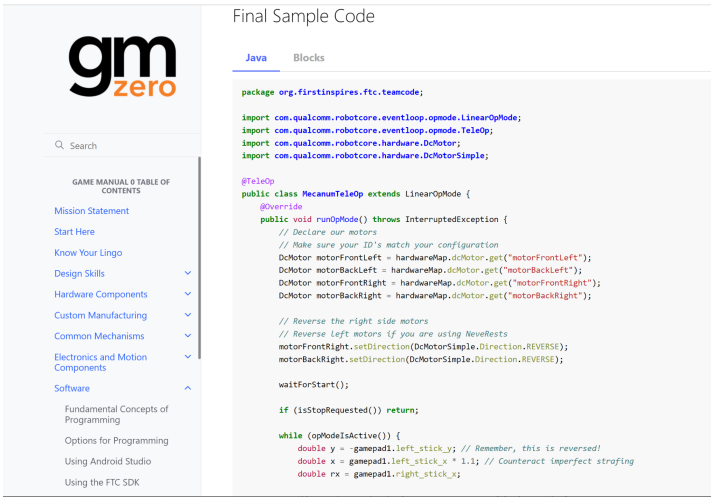
[Android Studio] [December 6,2021]

[Eric, Samir, Sergio, Kareem, Jake]

Think:

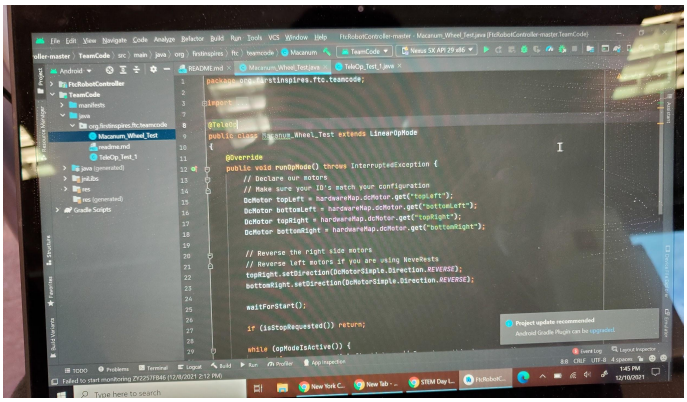
After the qualifiers,We asked the top team(Atomic Theory) for advice,They recommended we switch to android studio for more utility, and use a website(GM0.org) for more information.Install and setup Android Studio,and learn how to use Android Studio.Learn about Custom object detection.

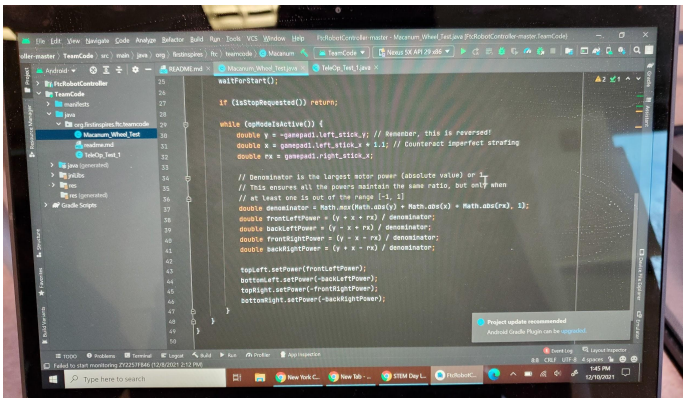
Design:



Sample code from GM0.org that was referenced for the test code

Test:





Experimental code for Android Studio

Signatures:_____Date:_____

Result:

We imported the FTC SDK into the Java code and started testing and experimenting with Android Studios.

Next Step:

Continue experimenting and studying Android Studio,Continue research for the implementation of custom object detection.

Signatures:_____Date:_____

[Software Implementation] [Week of January 10th and 17th]

[Eric, Sergio, Kevin, Samir, Kareem]

Think:

To create the process for the intake to be more efficient,We decided to implement toggling for the intake.In order to simplify the driver's task;Toggling eliminates the need for awareness over picking up objects while driving.TensorFlow had difficulty detecting our custom object in autonomous mode,thus we elected to switch to OpenCV.

Design:

Test code of the Intake Toggling with comments:

```
//code to toggle the intake using the A button
//uses the boolean "onOff" to remember if the button has been pressed before or
not. The sleep command allows the code to temporarily stop in order to allow
the onOff variable to switch. It gives us a 0.2 second window to press the
button.
//this code allows the variable to switch when the button is pressed.
            if (gamepad1.a && !onOff)
            {
                onOff = true;
                Thread.sleep(200);
            }
            else if (gamepad1.a && onOff)
            {
                onOff = false;
                Thread.sleep(200);
            }

            if(onOff)
            {
                intake.setPower(-1.0);
            }
            else if(!onOff)
            {
                intake.setPower(0.0);
            }

            if(gamepad1.b)
            {
                intake.setPower(1);
            }
            else
            {
                intake.setPower(0);
```

Signatures:_____Date:_____

```
        }
```

Testing the OpenCV Code:
We used a Sample code provided by PinkToTheFuture found on the FTC Reddit. It utilizes a range of colors to detect the custom shipping elements and a Contour Pipeline to draw the rectangle.

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

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.util.Range;

import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.opencv.core.Scalar;
import org.openftc.easyopencv.OpenCvCamera;
import org.openftc.easyopencv.OpenCvCameraFactory;
import org.openftc.easyopencv.OpenCvCameraRotation;

//@Config
@Autonomous(name="OpenCV_Test", group="Tutorials")

public class OpenCV_Tutorial_Freight_Frenzy extends LinearOpMode {

   private OpenCvCamera webcam;
   private ContourPipeline pipeline;

   private double crThreshHigh = 150;
   private double crThreshLow = 120;
   private double cbThreshHigh = 255;
   private double cbThreshLow = 255;

   private int minRectangleArea = 2000;
   private double leftBarcodeRangeBoundary = 0.3; //i.e 30% of the way across
the frame from the left
   private double rightBarcodeRangeBoundary = 0.6; //i.e 60% of the way across
the frame from the left

   private double lowerRuntime = 0;
   private double upperRuntime = 0;

   // Green Range                                        Y      Cr     Cb
   public static Scalar scalarLowerYCrCb = new Scalar(  0.0, 0.0, 0.0);
   public static Scalar scalarUpperYCrCb = new Scalar(255.0, 120.0, 120.0);

   @Override
   public void runOpMode() throws InterruptedException
   {
```

Signatures:_____Date:_____

```
        // OpenCV webcam
        int cameraMonitorViewId =
hardwareMap.appContext.getResources().getIdentifier("cameraMonitorViewId",
"id", hardwareMap.appContext.getPackageName());
        webcam =
OpenCvCameraFactory.getInstance().createWebcam(hardwareMap.get(WebcamName.class
, "Webcam 1"), cameraMonitorViewId);
        //OpenCV Pipeline

        pipeline = new ContourPipeline(0.2, 0.2, 0.2, 0.2);

pipeline.configureScalarLower(scalarLowerYCrCb.val[0],scalarLowerYCrCb.val[1],s
calarLowerYCrCb.val[2]);

pipeline.configureScalarUpper(scalarUpperYCrCb.val[0],scalarUpperYCrCb.val[1],s
calarUpperYCrCb.val[2]);

        webcam.setPipeline(pipeline);

        // Webcam Streaming
        webcam.openCameraDeviceAsync(new OpenCvCamera.AsyncCameraOpenListener()
{
            @Override
            public void onOpened() {
                webcam.startStreaming(800, 450, OpenCvCameraRotation.UPRIGHT);
            }
            //use this to change the dimensions of the rectangle

            @Override
            public void onError(int errorCode) {

            }
        });

        waitForStart();

        if(isStopRequested()) return;

        while (opModeIsActive())
        {
            if(pipeline.error){
                telemetry.addData("Exception: ",
pipeline.debug.getStackTrace());
            }


            double rectangleArea = pipeline.getRectArea();

            //Print out the area of the rectangle that is found.
```

```
            telemetry.addData("Rectangle Area", rectangleArea);

            //Check to see if the rectangle has a large enough area to be a
marker.
            if(rectangleArea > minRectangleArea){
                //Then check the location of the rectangle to see which barcode
it is in.
                if(pipeline.getRectMidpointX() > rightBarcodeRangeBoundary *
pipeline.getRectWidth())
                {
                    telemetry.addData("Barcode Position", "Right");
                }
                else if(pipeline.getRectMidpointX() < leftBarcodeRangeBoundary *
pipeline.getRectWidth())
                {
                    telemetry.addData("Barcode Position", "Left");
                }
                else
                {
                    telemetry.addData("Barcode Position", "Center");
                }
            }

            telemetry.update();
        }
    }
    public void testing(ContourPipeline pipeline){
        if(lowerRuntime + 0.05 < getRuntime()){
            crThreshLow += -gamepad1.left_stick_y;
            cbThreshLow += gamepad1.left_stick_x;
            lowerRuntime = getRuntime();
        }
        if(upperRuntime + 0.05 < getRuntime()){
            crThreshHigh += -gamepad1.right_stick_y;
            cbThreshHigh += gamepad1.right_stick_x;
            upperRuntime = getRuntime();
        }

        crThreshLow = inValues(crThreshLow, 0, 255);
        crThreshHigh = inValues(crThreshHigh, 0, 255);
        cbThreshLow = inValues(cbThreshLow, 0, 255);
        cbThreshHigh = inValues(cbThreshHigh, 0, 255);

        pipeline.configureScalarLower(0.0, crThreshLow, cbThreshLow);
        pipeline.configureScalarUpper(255.0, crThreshHigh, cbThreshHigh);

        telemetry.addData("lowerCr ", crThreshLow);
        telemetry.addData("lowerCb ", cbThreshLow);
        telemetry.addData("UpperCr ", crThreshHigh);
```

Signatures:_____Date:_____

```
        telemetry.addData("UpperCb ", cbThreshHigh);
    }
    public double inValues(double value, double min, double max){
        if(value < min){ value = min; }
        if(value > max){ value = max; }
        return value;
    }
}
```

First Encoder Arm Test:
To increase efficiency, we would like to change the level of the arm with a single button. To make sure the arm goes to a precise location, encoders would be needed.

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

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;

@TeleOp
@Disabled

public class Encoder_Arm_Test_2 extends LinearOpMode
{
    static final double     COUNTS_PER_MOTOR_REV        = 751.8 ;    // goBilda
5203 223rpm ticks per revolution
    static final double     DRIVE_GEAR_REDUCTION        = 1.0 ;      // This is
< 1.0 if geared UP
    static final double     PULLY_HUB_DIAMETER_INCHES   = 1.375 ;  // For
figuring circumference
    static final double     COUNTS_PER_INCH             = (COUNTS_PER_MOTOR_REV
* DRIVE_GEAR_REDUCTION) /
            (PULLY_HUB_DIAMETER_INCHES * 3.1415);
    static final double     ARM_SPEED                   = 1;
    static final double     FIRST_LEVEL_INCHES          = 4.2096;
    static final double     SECOND_LEVEL_INCHES         = 9.2341;
    static final double     THIRD_LEVEL_INCHES          = 15.2090;

    @Override
    public void runOpMode() throws InterruptedException {
        // Declare our motors
        // Make sure your ID's match your configuration
        DcMotor armMotor = hardwareMap.dcMotor.get("armMotor");
        armMotor.setDirection(DcMotorSimple.Direction.REVERSE);
        armMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
```

```
        armMotor.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);

        waitForStart();

        if (isStopRequested()) return;

        while (opModeIsActive())
        {
            //test code for the arm
            if(gamepad1.a)
            {
                armMotor.setTargetPosition( (int) (FIRST_LEVEL_INCHES *
COUNTS_PER_INCH) );
                armMotor.setPower(ARM_SPEED);
                armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);

                while(armMotor.isBusy())
                {
                    telemetry.addData("Status", "Running the Arm");
                    telemetry.update();
                }

                armMotor.setPower(0);
            }

            if(gamepad1.b)
            {
                armMotor.setTargetPosition( (int) (SECOND_LEVEL_INCHES *
COUNTS_PER_INCH) );
                armMotor.setPower(ARM_SPEED);
                armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);

                while(armMotor.isBusy())
                {
                    telemetry.addData("Status", "Running the Arm");
                    telemetry.update();
                }

                armMotor.setPower(0);
            }

            if(gamepad1.y)
            {
                armMotor.setTargetPosition( (int) (THIRD_LEVEL_INCHES *
COUNTS_PER_INCH) );
                armMotor.setPower(ARM_SPEED);
                armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);

                while(armMotor.isBusy())
```

```
                {
                    telemetry.addData("Status", "Running the Arm");
                    telemetry.update();
                }

                armMotor.setPower(0);
            }

            if(gamepad1.x)
            {
                armMotor.setTargetPosition(35);
                armMotor.setPower(ARM_SPEED);
                armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);

                while(armMotor.isBusy())
                {
                    telemetry.addData("Status", "Running the Arm");
                    telemetry.update();
                }

                armMotor.setPower(0);
            }

        }
    }
}
```

Test:

In testing, we found that a sleep time of 200 ms was a good window for our intake code. This allowed us to constantly toggle our intake for use. After attaching the camera to the robot and with testing, we found that the OpenCV code properly detected the color pink accurately. After conducting a trial for the arm, the arm code didn't function as planned due to a lack of knowledge on encoders.

Result:

The toggling intake code worked as planned.The OpenCV code functioned as planned and is in the process of customization.The test for the arm code had not functioned and is suspected that the encoders are needed to be further researched in order for future action.

Next Step:

Customize OpenCV code for our objectives and continue research on encoder-use in Tele-Op.

Signatures:_____Date:_____

MetroBotics Team #14212

[Color Sensor Testing] [Week on January 19th]

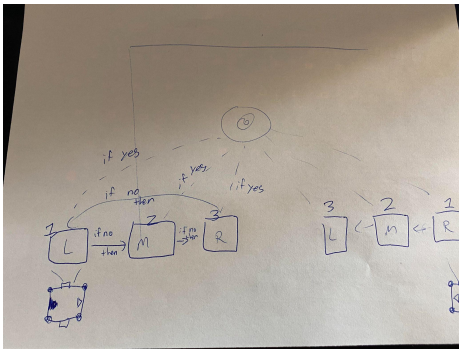
Kareem, Sergio, Samir

Think:

How can we use a color sensor to automate our robot to automatically detect and export our shipping element to the Hub.

Design:

Our robot should be able to detect from left to right or vice versa and then, if detected, deliver, otherwise move to the next object, else, repeat. In order to execute this well we have to know which color the team shipping element will be.

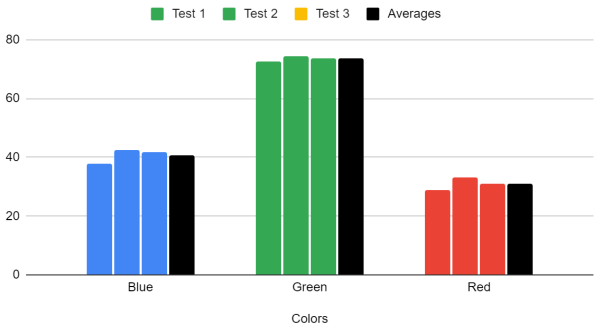


Test:

We test which color is the most easily detectable and then record and average them over the span of three tests which were then averaged.

Result:

Green scaled the highest for detection, when compared to other colors as seen on the second image (graph) where the average comparison was greater than the others.



Next Step:

After the result of one first color sensor we decided that we are going to add another color sensor to get the values from both sides of the robot. That way we can make sure that the robot drives the correct way for both sides if it is turned to a different direction.

Signatures:_____Date:_____

[Encoder Testing] [Week of January 24th]

[Eric Guan, Alex Wu, Kevin Xia]

Think:

After multiple failed trials with encoders,we discovered that the encoders in use was not the correct encoders for the motors.The motors required GoBilda specific encoders to function properly.We found this out after looking back at our Pitsco robot in where the encoders worked fine until we transitioned to the GoBilda Strafer Chassis.We are in process of switching the encoders before testing again.

Design:

The code we used before now works as intended: Encoder Arm Test

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

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;

@TeleOp
@Disabled

public class Encoder_Arm_Test_2 extends LinearOpMode
{
   static final double     COUNTS_PER_MOTOR_REV        = 751.8 ;    // goBilda
5203 223rpm ticks per revolution
   static final double     DRIVE_GEAR_REDUCTION        = 1.0 ;      // This is
< 1.0 if geared UP
   static final double     PULLY_HUB_DIAMETER_INCHES   = 1.375 ;  // For
figuring circumference
   static final double     COUNTS_PER_INCH             = (COUNTS_PER_MOTOR_REV
* DRIVE_GEAR_REDUCTION) /
           (PULLY_HUB_DIAMETER_INCHES * 3.1415);
   static final double     ARM_SPEED                   = 1;
   static final double     FIRST_LEVEL_INCHES          = 4.2096;
   static final double     SECOND_LEVEL_INCHES         = 9.2341;
   static final double     THIRD_LEVEL_INCHES          = 15.2090;

   @Override
   public void runOpMode() throws InterruptedException {
       // Declare our motors
       // Make sure your ID's match your configuration
       DcMotor armMotor = hardwareMap.dcMotor.get("armMotor");
       armMotor.setDirection(DcMotorSimple.Direction.REVERSE);
```

Signatures:_____Date:_____

```
        armMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        armMotor.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);

        waitForStart();

        if (isStopRequested()) return;

        while (opModeIsActive())
        {
            //test code for the arm
            if(gamepad1.a)
            {
                armMotor.setTargetPosition( (int) (FIRST_LEVEL_INCHES *
COUNTS_PER_INCH) );
                armMotor.setPower(ARM_SPEED);
                armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);

                while(armMotor.isBusy())
                {
                    telemetry.addData("Status", "Running the Arm");
                    telemetry.update();
                }

                armMotor.setPower(0);
            }

            if(gamepad1.b)
            {
                armMotor.setTargetPosition( (int) (SECOND_LEVEL_INCHES *
COUNTS_PER_INCH) );
                armMotor.setPower(ARM_SPEED);
                armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);

                while(armMotor.isBusy())
                {
                    telemetry.addData("Status", "Running the Arm");
                    telemetry.update();
                }

                armMotor.setPower(0);
            }

            if(gamepad1.y)
            {
                armMotor.setTargetPosition( (int) (THIRD_LEVEL_INCHES *
COUNTS_PER_INCH) );
                armMotor.setPower(ARM_SPEED);
                armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
```

```
            while(armMotor.isBusy())
            {
                telemetry.addData("Status", "Running the Arm");
                telemetry.update();
            }

            armMotor.setPower(0);
        }

        if(gamepad1.x)
        {
            armMotor.setTargetPosition(35);
            armMotor.setPower(ARM_SPEED);
            armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            while(armMotor.isBusy())
            {
                telemetry.addData("Status", "Running the Arm");
                telemetry.update();
            }

            armMotor.setPower(0);
        }

    }
  }
}
```

Test:

Testing the arm code for finding the accurate distance of extending the arm to reach the alliance shipping hub.Testing went very well and the arm reached the intended target distance for unloading.

Result:

The encoder code functioned as planned and the data found through calculations were accurate in the process of extending the arm for unloading.While the arm is in motion,we found out that the drivetrain cannot function properly due to the “while motor is busy” loop.Reworking the code is necessary to prevent any accidents.

Next Step:

The issue with the arm functioning and the drivetrain not functioning in tandem is an issue that needs more testing and is suspected of a “while motor is busy” loop in the code.Revising of the code and testing is planned.

Signatures:_____Date:_____

[Field Centric Drive Train] [1/27/2022]

[Eric Guan, Alex Wu, Kevin Xia]

Think:

In order for the robot to be easier to drive, we explored Field-Centric drive trains. Field-centric drive trains allow movement to be relative to the driver, thus making the robot easier to control.

Design:

To code this type of drive train, we needed to utilize the IMU built in the expansion hub. This IMU tracks turning and angles relative to the hub. By getting the values from the IMU, we can tell the robot to correct its course to allow for a field centric drive train.

First Test of the Field Centric Drive Train:

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

import com.qualcomm.hardware.bosch.JustLoggingAccelerationIntegrator;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;
import com.qualcomm.hardware.bosch.BNO055IMU;
import com.qualcomm.robotcore.util.Range;

import org.firstinspires.ftc.robotcore.external.navigation.Acceleration;
import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.navigation.AxesOrder;
import org.firstinspires.ftc.robotcore.external.navigation.AxesReference;
import org.firstinspires.ftc.robotcore.external.navigation.Orientation;
import org.firstinspires.ftc.robotcore.external.navigation.Position;
import org.firstinspires.ftc.robotcore.external.navigation.Velocity;

import static
org.firstinspires.ftc.robotcore.external.navigation.AngleUnit.DEGREES;


@TeleOp(name = "DriverRelativeControl", group = "tutorial")
public class Field_Centric_Drive_Test extends LinearOpMode {


   private DcMotor frontLeft;
   private DcMotor frontRight;
```

Signatures:_____Date:_____

```
    private DcMotor backRight;
    private DcMotor backLeft;

    public BNO055IMU imu;

    Orientation angles;
    Acceleration gravity;

    @Override
    public void runOpMode() {


        double driveTurn;
        //double driveVertical;
        //double driveHorizontal;

        double gamepadXCoordinate;
        double gamepadYCoordinate;
        double gamepadHypot;
        double gamepadDegree;
        double robotDegree;
        double movementDegree;
        double gamepadXControl;
        double gamepadYControl;


        frontLeft = hardwareMap.dcMotor.get("frontLeft");
        frontRight = hardwareMap.dcMotor.get("frontRight");
        backRight = hardwareMap.dcMotor.get("backRight");
        backLeft = hardwareMap.dcMotor.get("backLeft");

        //might need to change the motors being reversed
        frontRight.setDirection(DcMotorSimple.Direction.REVERSE);
        backRight.setDirection(DcMotorSimple.Direction.REVERSE);

        BNO055IMU.Parameters parameters = new BNO055IMU.Parameters();
        parameters.angleUnit = BNO055IMU.AngleUnit.DEGREES;
        parameters.accelUnit = BNO055IMU.AccelUnit.METERS_PERSEC_PERSEC;
        parameters.calibrationDataFile = "BNO055IMUCalibration.json";
        parameters.loggingEnabled = true;
        parameters.loggingTag = "IMU";
        parameters.accelerationIntegrationAlgorithm = new
JustLoggingAccelerationIntegrator();

                imu = hardwareMap.get(BNO055IMU.class, "imu");
        imu.initialize(parameters);

        composeTelemetry();
```

```
        waitForStart();

        imu.startAccelerationIntegration(new Position(), new Velocity(), 1000);


        while (opModeIsActive()) {
            driveTurn = -gamepad1.left_stick_x;
            //driveVertical = -gamepad1.right_stick_y;
            //driveHorizontal = gamepad1.right_stick_x;

            gamepadXCoordinate = gamepad1.right_stick_x; //this simply gives our
x value relative to the driver
            gamepadYCoordinate = -gamepad1.right_stick_y; //this simply gives
our y vaue relative to the driver
            gamepadHypot = Range.clip(Math.hypot(gamepadXCoordinate,
gamepadYCoordinate), 0, 1);
            //finds just how much power to give the robot based on how much x
and y given by gamepad
            //range.clip helps us keep our power within positive 1
            // also helps set maximum possible value of 1/sqrt(2) for x and y
controls if at a 45 degree angle (which yields greatest possible value for y+x)
            gamepadDegree = Math.atan2(gamepadYCoordinate, gamepadXCoordinate);
            //the inverse tangent of opposite/adjacent gives us our gamepad
degree
            robotDegree = getAngle();
            //gives us the angle our robot is at
            movementDegree = gamepadDegree - robotDegree;
            //adjust the angle we need to move at by finding needed movement
degree based on gamepad and robot angles
            gamepadXControl = Math.cos(Math.toRadians(movementDegree)) *
gamepadHypot;
            //by finding the adjacent side, we can get our needed x value to
power our motors
            gamepadYControl = Math.sin(Math.toRadians(movementDegree)) *
gamepadHypot;
            //by finding the opposite side, we can get our needed y value to
power our motors



            //by mulitplying the gamepadYControl and gamepadXControl by their
respective absolute values, we can guarantee that our motor powers will not
exceed 1 without any driveTurn
            //since we've maxed out our hypot at 1, the greatest possible value
of x+y is (1/sqrt(2)) + (1/sqrt(2)) = sqrt(2)
            //since (1/sqrt(2))^2 = 1/2 = .5, we know that we will not exceed a
power of 1 (with no turn), giving us more precision for our driving
            frontRight.setPower(gamepadYControl * Math.abs(gamepadYControl) -
gamepadXControl * Math.abs(gamepadXControl) + driveTurn);
```

```
            backRight.setPower(gamepadYControl * Math.abs(gamepadYControl) +
gamepadXControl * Math.abs(gamepadXControl) + driveTurn);
            frontLeft.setPower(gamepadYControl * Math.abs(gamepadYControl) +
gamepadXControl * Math.abs(gamepadXControl) - driveTurn);
            backLeft.setPower(gamepadYControl * Math.abs(gamepadYControl) -
gamepadXControl * Math.abs(gamepadXControl) - driveTurn);

            /*frontRight.setPower(driveVertical - driveHorizontal + driveTurn);
            backRight.setPower(driveVertical + driveHorizontal + driveTurn);
            frontLeft.setPower(driveVertical + driveHorizontal - driveTurn);
            backLeft.setPower(driveVertical - driveHorizontal - driveTurn);*/
        }
        telemetry.update();
    }

    void composeTelemetry() {
        telemetry.addAction(new Runnable() {
            @Override
            public void run() {
                angles = imu.getAngularOrientation(AxesReference.INTRINSIC,
AxesOrder.ZYX, AngleUnit.DEGREES);
                gravity = imu.getGravity();
            }
        });
    }

    //allows us to quickly get our z angle
    public double getAngle() {
        return imu.getAngularOrientation(AxesReference.INTRINSIC, AxesOrder.ZYX,
AngleUnit.DEGREES).firstAngle;
    }
}
```

Test:

To test our drivetrain, we put the robot on the playing field and tried driving around. However, the code did not function correctly.

Result:

Although the system was adjusting the angle to align itself relative to the driver, the controls did not function as intended. The drive train didn't allow the robot to move in any direction other than forward.

Next Step:

We need to further research field-centric driving and the IMU built into the Expansion hub.

[Finite State Machine] [1/31/2022]

Signatures:_____Date:_____

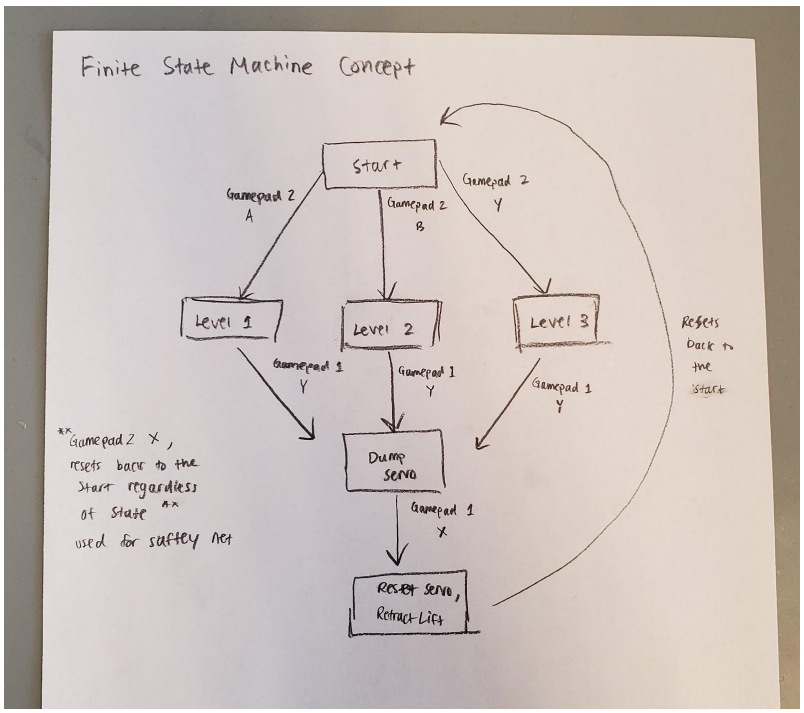
MetroBotics Team #14212

[Eric Guan, Alex Wu, Kevin Xia]

Think:

To fix the issue of the drivetrain and arm not being able to run together, we need to implement a state machine in our code.
This would allow us to switch the state of our robot with a single button, while also allowing us to move around while doing so. This increases our efficiency by allowing automation during our teleop phase.

Design:



First State Machine Test Code:

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

import com.qualcomm.robotcore.eventloop.opmode.OpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.util.ElapsedTime;

@TeleOp(name="State Machine")
public class StateMachine_Testing extends OpMode
```

Signatures:_____Date:_____

```
{
   //arm encoder calculations
   static final double     COUNTS_PER_MOTOR_REV        = 751.8 ;    // goBilda
5203 223rpm ticks per revolution
   static final double     DRIVE_GEAR_REDUCTION        = 1.0 ;      // This is
< 1.0 if geared UP
   static final double     PULLY_HUB_DIAMETER_INCHES   = 1.375 ;  // For
figuring circumference
   static final double     COUNTS_PER_INCH             = (COUNTS_PER_MOTOR_REV
* DRIVE_GEAR_REDUCTION) /
           (PULLY_HUB_DIAMETER_INCHES * 3.1415);
   static final double     ARM_SPEED                   = 0.8;
   static final double     FIRST_LEVEL_INCHES          = 4.2096;
   static final double     SECOND_LEVEL_INCHES         = 9.2341;
   static final double     THIRD_LEVEL_INCHES          = 15.2090;

   public enum LiftState
   {
       LIFT_START,
       LIFT_DUMP,
       LIFT_DUMP_RETRACT,
       LIFT_RETRACT,
       LIFT_EXTEND_ONE,
       LIFT_EXTEND_TWO,
       LIFT_EXTEND_THREE
   }

   LiftState liftState = LiftState.LIFT_START;

   //arm motor
   public DcMotor armMotor;

   //mecanum wheels
   public DcMotor frontLeft;
   public DcMotor backLeft;
   public DcMotor frontRight;
   public DcMotor backRight;

   //dump servo
   public Servo Bucket;
   ElapsedTime liftTimer = new ElapsedTime();

   final double DUMP_IDLE = 0.61;
   final double DUMP_DEPOSIT = 0.0;
   final int LIFT_LEVEL_ZERO = 15;
   final int LIFT_LEVEL_ONE = (int) (FIRST_LEVEL_INCHES * COUNTS_PER_INCH);
   final int LIFT_LEVEL_TWO = (int) (SECOND_LEVEL_INCHES * COUNTS_PER_INCH);
   final int LIFT_LEVEL_THREE = (int) (THIRD_LEVEL_INCHES * COUNTS_PER_INCH);
```

```
public void init()
{
    frontLeft = hardwareMap.dcMotor.get("frontLeft");
    backLeft = hardwareMap.dcMotor.get("backLeft");
    frontRight = hardwareMap.dcMotor.get("frontRight");
    backRight = hardwareMap.dcMotor.get("backRight");
    armMotor = hardwareMap.dcMotor.get("armMotor");
    Bucket = hardwareMap.servo.get("Bucket");

    frontRight.setDirection(DcMotorSimple.Direction.REVERSE);
    backRight.setDirection(DcMotorSimple.Direction.REVERSE);
    armMotor.setDirection(DcMotorSimple.Direction.REVERSE);

    armMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    armMotor.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
}

public void loop()
{
    switch (liftState)
    {
        case LIFT_START:
            if(gamepad2.a)
            {
                armMotor.setTargetPosition(LIFT_LEVEL_ONE);
                armMotor.setPower(ARM_SPEED);
                armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
                liftState = LiftState.LIFT_EXTEND_ONE;
            }

            if(gamepad2.b)
            {
                armMotor.setTargetPosition(LIFT_LEVEL_TWO);
                armMotor.setPower(ARM_SPEED);
                armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
                liftState = LiftState.LIFT_EXTEND_TWO;
            }

            if(gamepad2.y)
            {
                armMotor.setTargetPosition(LIFT_LEVEL_THREE);
                armMotor.setPower(ARM_SPEED);
                armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
                liftState = LiftState.LIFT_EXTEND_THREE;
            }
            break;
        case LIFT_EXTEND_ONE:
```

```
                if(Math.abs(armMotor.getCurrentPosition() - LIFT_LEVEL_ONE) <
10)
                {
                    armMotor.setPower(0);
                    liftState = LiftState.LIFT_DUMP;
                }
                break;
            case LIFT_EXTEND_TWO:
                if(Math.abs(armMotor.getCurrentPosition() - LIFT_LEVEL_TWO) <
10)
                {
                    armMotor.setPower(0);
                    liftState = LiftState.LIFT_DUMP;
                }
                break;
            case LIFT_EXTEND_THREE:
                if(Math.abs(armMotor.getCurrentPosition() - LIFT_LEVEL_THREE) <
10)
                {
                    armMotor.setPower(0);
                    liftState = LiftState.LIFT_DUMP;
                }
                break;
            case LIFT_DUMP:
                if(gamepad1.y)
                {
                    Bucket.setPosition(DUMP_DEPOSIT);
                    liftState = LiftState.LIFT_DUMP_RETRACT;
                }
                break;
            case LIFT_DUMP_RETRACT:
                if(gamepad1.x)
                {
                    Bucket.setPosition(DUMP_IDLE);
                    try {
                        Thread.sleep(250);
                    } catch (InterruptedException e) { }
                    armMotor.setTargetPosition(LIFT_LEVEL_ZERO);
                    armMotor.setPower(ARM_SPEED);
                    armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
                    liftState = LiftState.LIFT_RETRACT;
                }
                break;
            case LIFT_RETRACT:
                if(Math.abs(armMotor.getCurrentPosition() - LIFT_LEVEL_ZERO) <
10)
                {
                    armMotor.setPower(0);
                    liftState = LiftState.LIFT_START;
```

```
                }
                break;
            default:
                liftState = LiftState.LIFT_START;
        }
        //safety button to reset the states
        if (gamepad2.x && liftState != LiftState.LIFT_START)
        {
            Bucket.setPosition(DUMP_IDLE);
            try {
                Thread.sleep(250);
            } catch (InterruptedException e) { }
            armMotor.setTargetPosition(LIFT_LEVEL_ZERO);
            armMotor.setPower(ARM_SPEED);
            armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            liftState = LiftState.LIFT_RETRACT;
        }

        //code for the mecanum wheels
        double y = -gamepad1.left_stick_y; // Remember, this is reversed!
        double x = gamepad1.left_stick_x * 1.1; // Counteract imperfect strafing
        double rx = gamepad1.right_stick_x;

        // Denominator is the largest motor power (absolute value) or 1
        // This ensures all the powers maintain the same ratio, but only when
        // at least one is out of the range [-1, 1]
        double denominator = Math.max(Math.abs(y) + Math.abs(x) + Math.abs(rx),
1);
        double frontLeftPower = (y + x + rx) / denominator;
        double backLeftPower = (y - x + rx) / denominator;
        double frontRightPower = (y - x - rx) / denominator;
        double backRightPower = (y + x - rx) / denominator;

        frontLeft.setPower(-frontLeftPower/1.5);
        backLeft.setPower(-backLeftPower/1.5);
        frontRight.setPower(-frontRightPower/1.5);
        backRight.setPower(-backRightPower/1.5);
    }
}
```

Test:

To test the code, we simply observed whether the arm and servo is moving as intended. Once the arm worked properly, we combined drive train movement while we moved the arm to see if we resolved the issue.

Result:

Signatures:_____Date:_____

The code worked as intended. Using this State Machine, we can move the drivetrain while we lift or retract the arm. The code runs smoothly and in a controlled manner.

Next Step:

We need to implement the finite state machine into our actual code used for the competition, then test how the code performs in a real game.

Think:

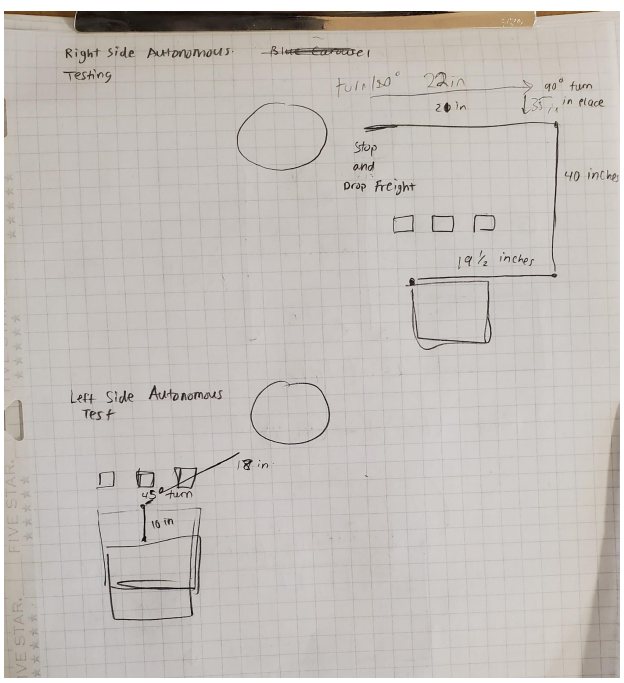
In order to detect our custom shipping element, we used OpenCV. We chose to use OpenCV as our object detection method because OpenCV can detect our shipping element using color and position on the camera screen.

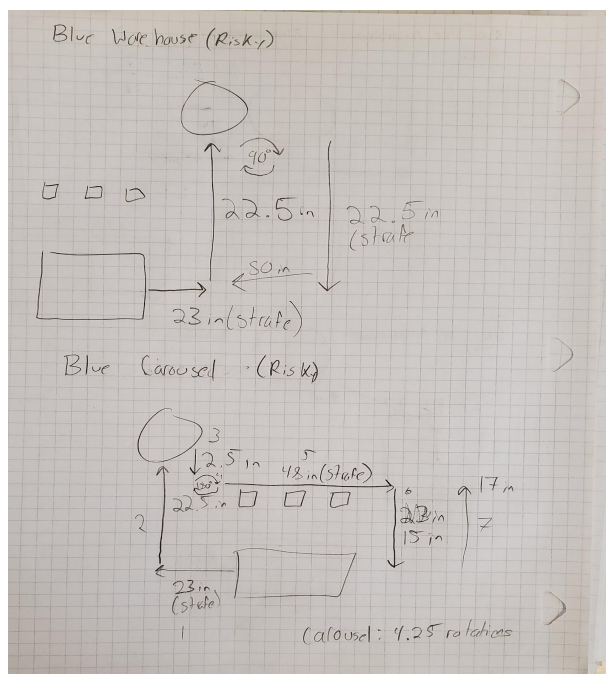
Design:

In order to avoid creating a custom model for our shipping element, we decided to use OpenCVs color sensing capabilities.
To know which position the shipping element is on, we divided up the camera screen into three positions and created if-statements that checked which one the shipping element was on.
After the location is detected, we need to run our autonomous code.

Place Codes here:

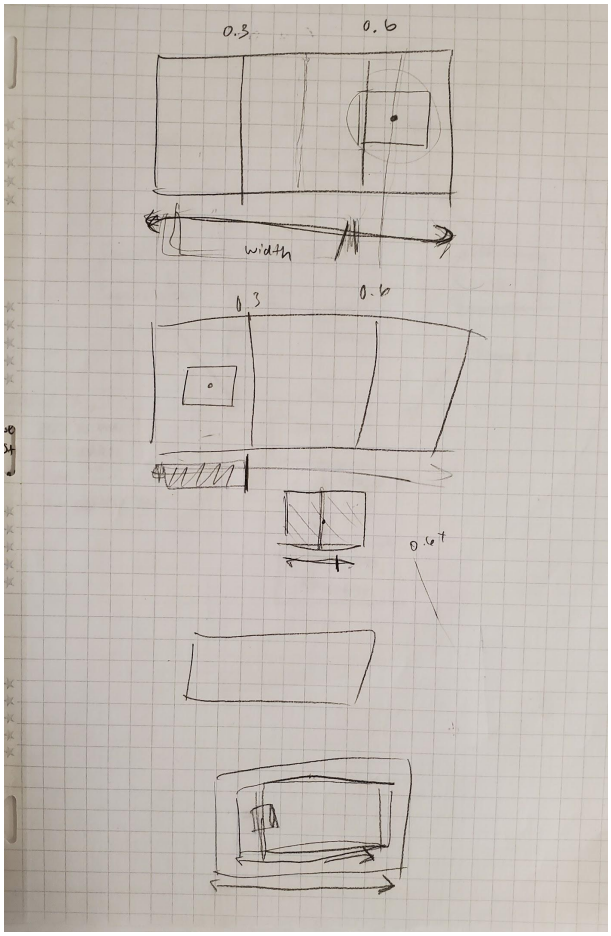
Calculations of the paths we drew out:





Picture of the OpenCV Concept Drawings:

Signatures:_____Date:_____



Test:

We tested the accuracy of our detection code and got it to be almost 100% accurate.

When testing the autonomous movement, we saw that the mats resulted in the robot losing a bit of its accuracy when using encoders (slipping). However, it is accurate for the most part.

Trials of how many times the final code worked:

Signatures:_____Date:_____

MetroBotics Team #14212

Object Detection Trials

Trial #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Left	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Center	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Right	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

The camera works 100% of the time and can detect the custom shipping element easily in any position and orientation.

Autonomous Mode Trials

Blue Carousel Side

Trial #	1	2	3	4	5
Left	✔	✔	✔	✔	✔
Center	✔	✔	✔	✔	✔
Right	✔	✔	✔	✔	✔

Red Carousel Side

Trial #	1	2	3	4	5
Left	✔	✔	✔	✔	✔
Center	✔	✔	✔	✔	✔
Right	✔	✔	✔	✔	✔

Blue Warehouse Side

Trial #	1	2	3	4	5
Left	✔	✔	✔	✔	✔
Center	✔	✔	✔	✔	✔
Right	✔	✔	✔	✔	✔

Signatures:_____Date:_____

Red Warehouse Side

Trial #	1	2	3	4	5
Left	✔	✔	✔	✔	✔
Center	✔	✔	✔	✔	✔
Right	✔	✔	✔	✔	✔

After doing some tests, our autonomous mode works nearly 100% of the time for all four starting locations on the field. However, we can't be completely confident that this is fail proof since we only used encoders so we have to completely make sure that the robot starts at the center of it's starting position.

Result:

The object detection software worked very well. In any orientation, the object can be detected and the correct position is returned.

The autonomous movement that proceeds the object detection also worked well but was limited due to the fact that we only relied on the encoders. Although they should move precise distances, the mats cause slipping and potential inaccuracies. However, this wasn't enough to warrant any significant issues.

Next Step:

Possibly study odometry after this week's qualifier to increase the accuracy of our code.

Signatures:_____Date:_____

[Implementing FTC Dashboard] [February 7th, 2022]

[Eric Guan]

Think:

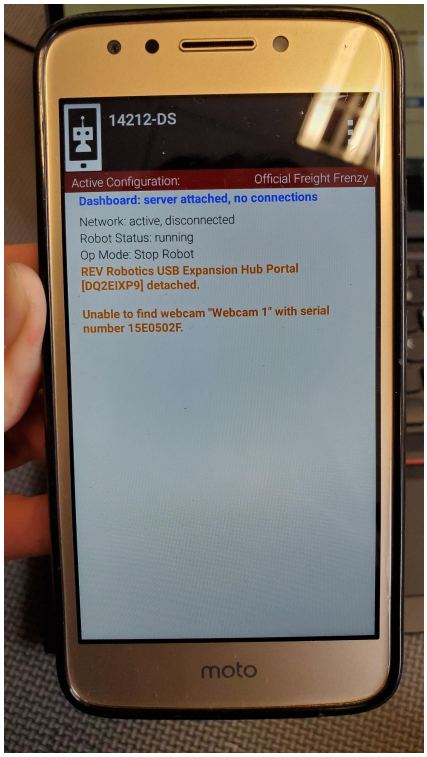
When testing our code, we would usually have to set up our devices and have the robot perform the actions in real life. This often took too much time and it didn't allow us to get useful telemetry for debugging. Because of this, we decided to implement FTC Dashboard, which allows us to test code and view telemetry remotely. This would increase our efficiency and allow us to better debug.

Design:

Because this is our first time using FTC Dashboard, we tested our webcam code using FTCDashboard to see if it worked. We added the files into the gradle script and uploaded the code to our phones

Test:

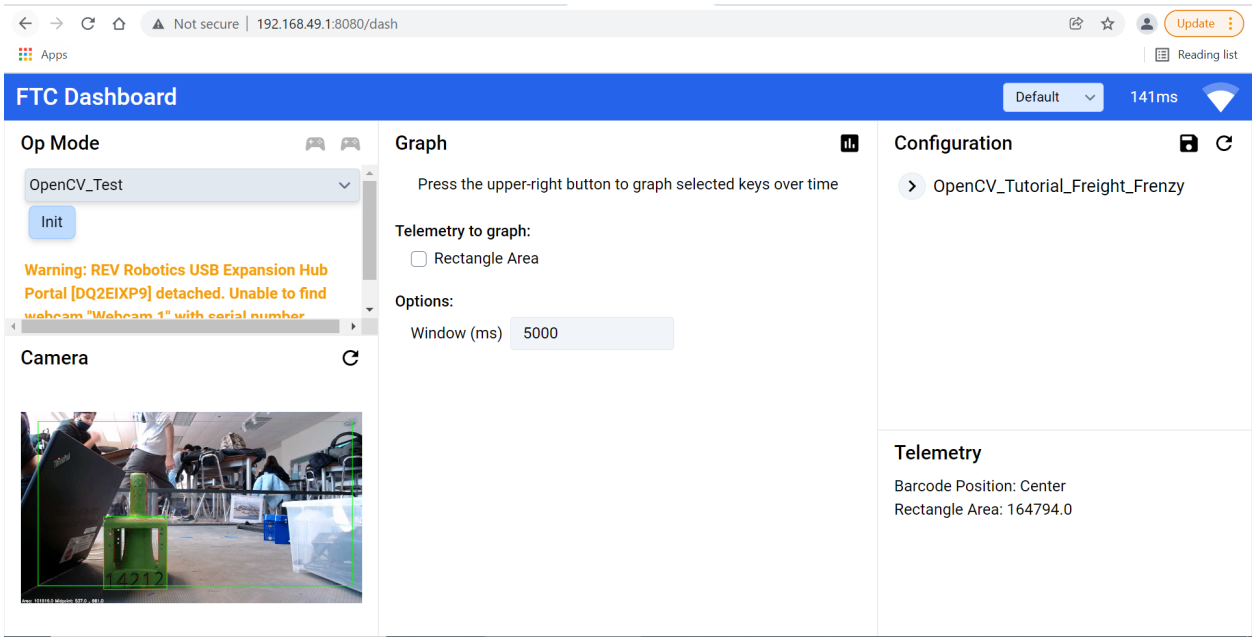
After uploading the FTCDashboard files onto the phones, we get this line of text on our controller phone.



We then ran our webcam code with the FTCDashboard initialization code and opened the dashboard website. To connect the phones to the laptop, we used the phone wifi

Signatures:_____Date:_____

signal.



Result:

The result of the FTCDashboard testing went well. We were able to see the webcam and the telemetry data remotely (as seen in the image above). From what we saw, FTCDashboard would definitely help us in the debugging process.

Next Step:

When testing Roadrunner for autonomous pathing, we can use FTCDashboard to facilitate the debugging and testing process. Our next steps would be to implement Roadrunner.

Signatures:_____Date:_____

[Tuning Roadrunner] [Week of February 7th & 14th, 2022]
[Eric Guan]

Think:

From Qualifier #4, we were able to implement a working autonomous mode. However, it was very simple and couldn't perform any complex actions. Roadrunner would be the next step in improving our autonomous mode since it offers commands to create complex autonomous paths.

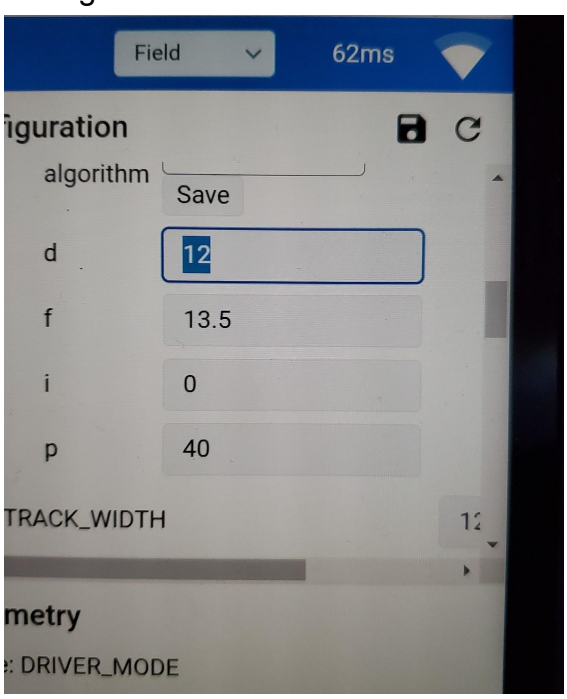
Design:

Because this is our first time using Roadrunner, we need to install the RoadRunner SDK into Android Studio.

We used the Roadrunner quickstart as well as FTC Dashboard to calibrate and tune our robot to work with the roadrunner algorithms. This required mapping our hardware, adjusting PID values, and running various op-modes to test the calibration.

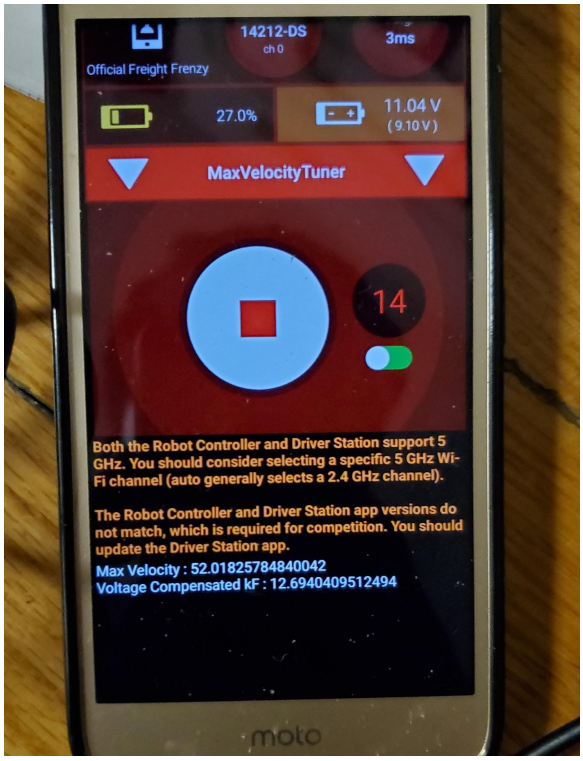
Pictures of the tuning process:

Tuning PIDF values on the FTC Dashboard:



Running Roadrunner Tuning Op-modes:

Signatures:_____Date:_____



Final Tuned Values:

```
public static PIDFCoefficients MOTOR_VELO_PID = new PIDFCoefficients(40, 0, 15, 13.5);

public static double WHEEL_RADIUS = 1.88976; // in
public static double GEAR_RATIO = 0.99; // output (wheel) speed / input (motor) speed
public static double TRACK_WIDTH = 13.13; // in

public static double MAX_VEL = 47;
public static double MAX_ACCEL = 30;
public static double MAX_ANG_VEL = Math.toRadians(180);
public static double MAX_ANG_ACCEL = Math.toRadians(180);

public static PIDCoefficients TRANSLATIONAL_PID = new PIDCoefficients(7, 0, 0);
public static PIDCoefficients HEADING_PID = new PIDCoefficients(8, 0, 0);

public static double LATERAL_MULTIPLIER = 1.122;

BNO055IMUUtil.remapZAxis(imu, AxisDirection.NEG_Y);

leftFront.setDirection(DcMotorSimple.Direction.REVERSE);
leftRear.setDirection(DcMotorSimple.Direction.REVERSE);
```

Signatures:_____Date:_____

Test:

After everything was fine tuned, we ran the recommended testing op-modes to see if the tuning was effective. We saw that for the most part, our robot followed the paths very well. With the values we used to tune our robot, it can accurately correct itself and stay on path.

Result:

From this experience, we learned how to tune our robot for Roadrunner. This was a lengthy process because getting the best PID values took a lot of time. However, the more we practice this, the quicker we will become. We also learned the importance of FTC Dashboard in facilitating this process as well as how to use it more in depth. Although we aren't using odometry, the drive encoders still proved to be accurate.

Next Step:

With Roadrunner fully calibrated, we can start programming our own paths for autonomous. We may create a finite state machine for autonomous mode to shift between actions.

MetroBotics Team #14212

[Eric Guan, Kevin Xia]

Think:

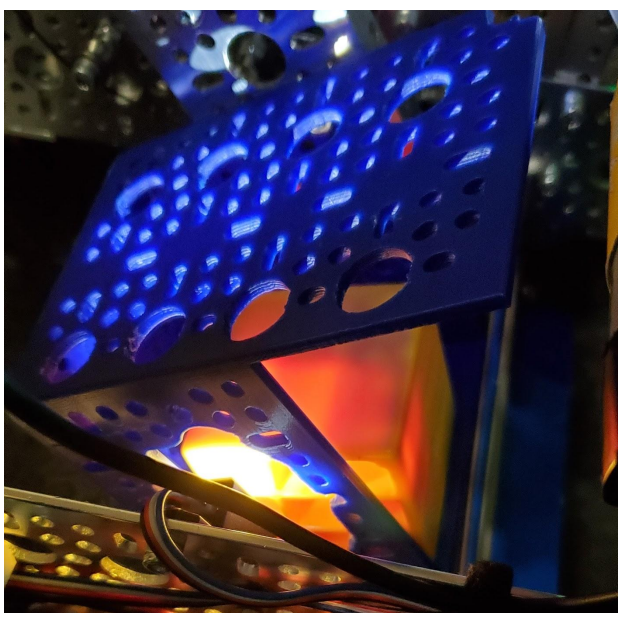
In order to score freight in autonomous mode, the robot needs to know when the freight has been picked up. This can be done with a color sensor. By stationing one near the bucket, it can detect whether a freight is present based on the color.

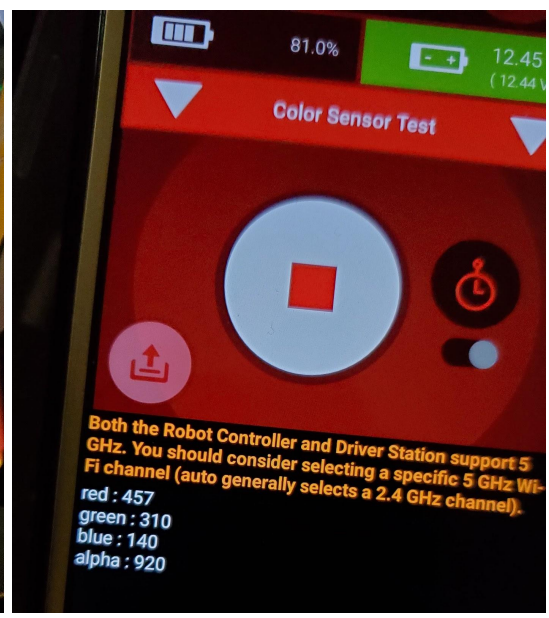
Design:

The color sensor will be attached next to the bucket.
Future plan for autonomous - If the color orange is detected (cube), stop the intake and continue the autonomous mode.
If the color white is detected (ball), stop the intake and continue the autonomous mode.
We need to program it to detect a range of colors to account for the lighting and shadows.

Test:

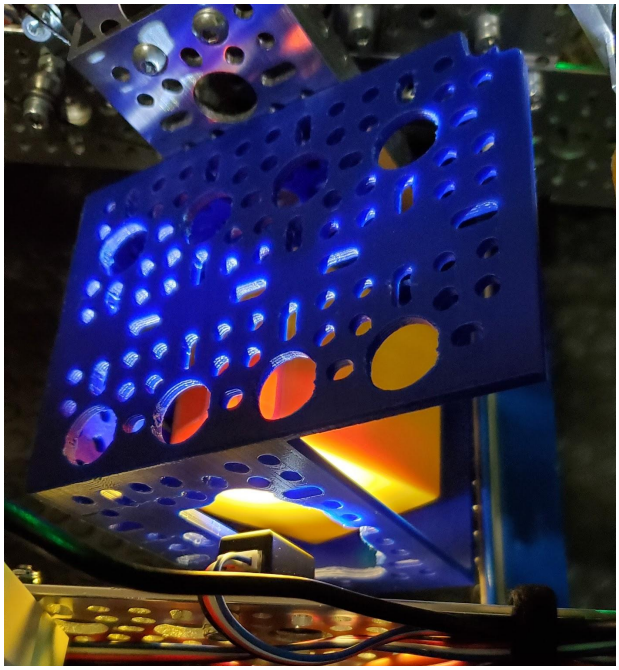
CUBE Testing:
Scenario 1 -

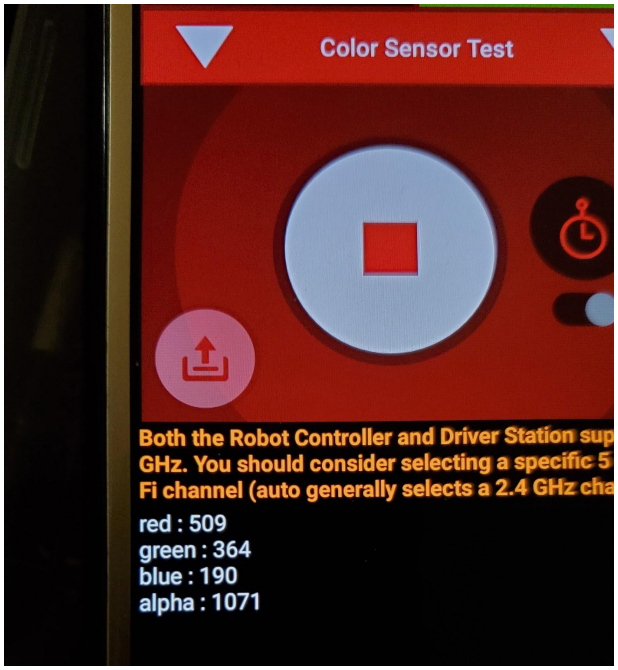




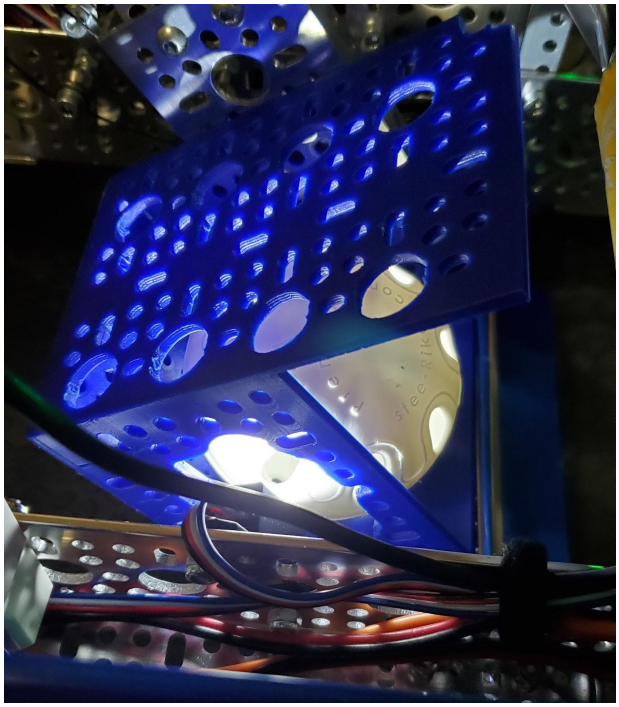
Scenario 2 -

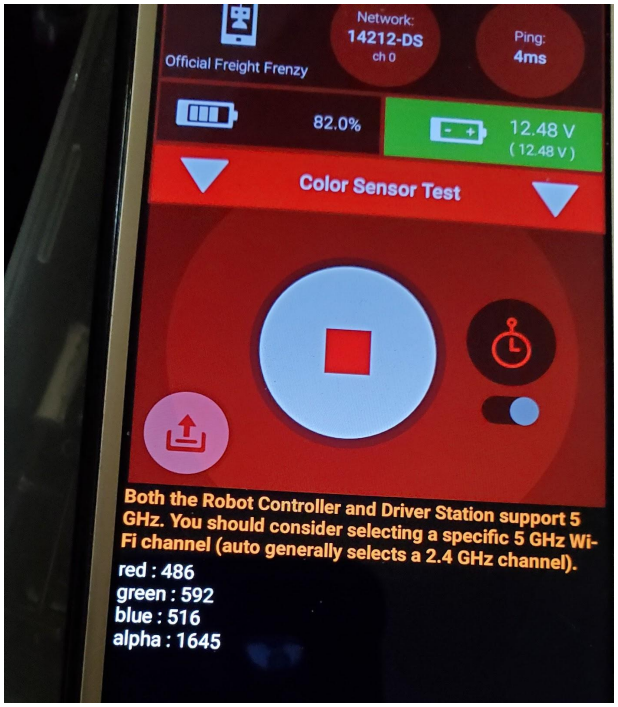
Signatures:_____Date:_____





BALL Testing:





Combining all the values into one algorithm for detecting if the object is present:

```
if( (intake_color.red() > 50 && intake_color.red() < 2250) &&
       (intake_color.green() > 30 && intake_color.green() < 2700) &&
       (intake_color.blue() > 10 && intake_color.blue() < 2300) &&
       (intake_color.alpha() > 100 && intake_color.alpha() < 7500))
```

Signatures:_____Date:_____

```
{
   telemetry.addLine("object detected");
}

telemetry.update();
```

Result:

Upon testing, we found it much easier to combine both the orange and white color values together for one algorithm, instead of differentiating between the two. Doing this wouldn't take anything away because we are only concerned whether or not there is an object in the intake. This resulted in accurate telemetry data when the object is in the bucket.

Next Step:

We need to incorporate the color sensor object detection algorithm into a larger intake system, which will later be transferred into the autonomous road runner code.

Signatures:_____Date:_____

MetroBotics Team #14212

[Improving Arm Finite State Machine] [February 23rd, 2022]

[Eric Guan]

Think:

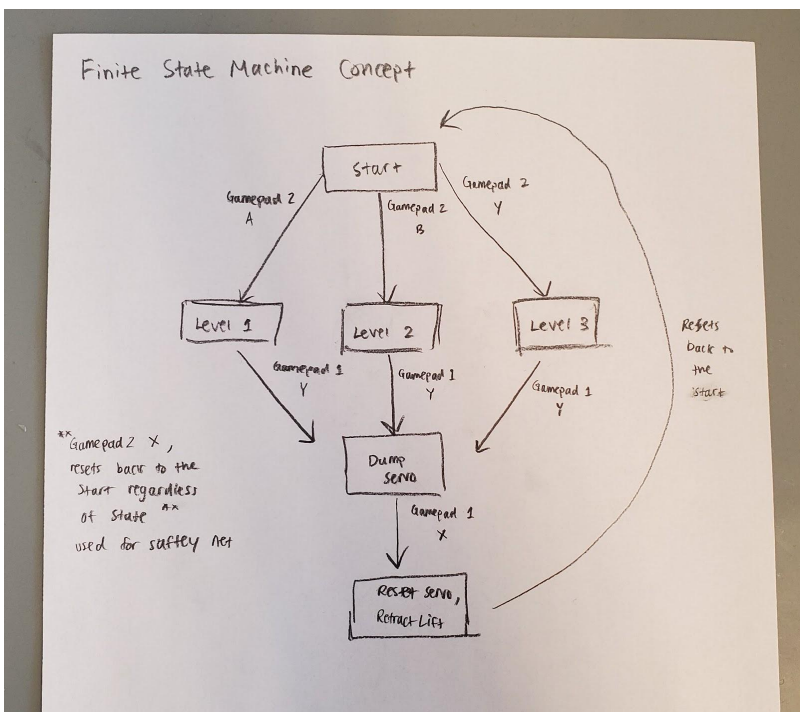
After watching the algorithm in action during the February 5th Qualifier, we noticed that some improvements could've been made.

In our first iteration of the algorithm, the driver had to press 'X' to reset the arm after the freight had been dumped. After reflecting on that decision, we decided that it was unnecessary and made it so one button would dump and reset the arm.

Additionally, we also wanted to make it so the arm could shift levels after we go to a certain level to correct for any errors in a quick and easy way.

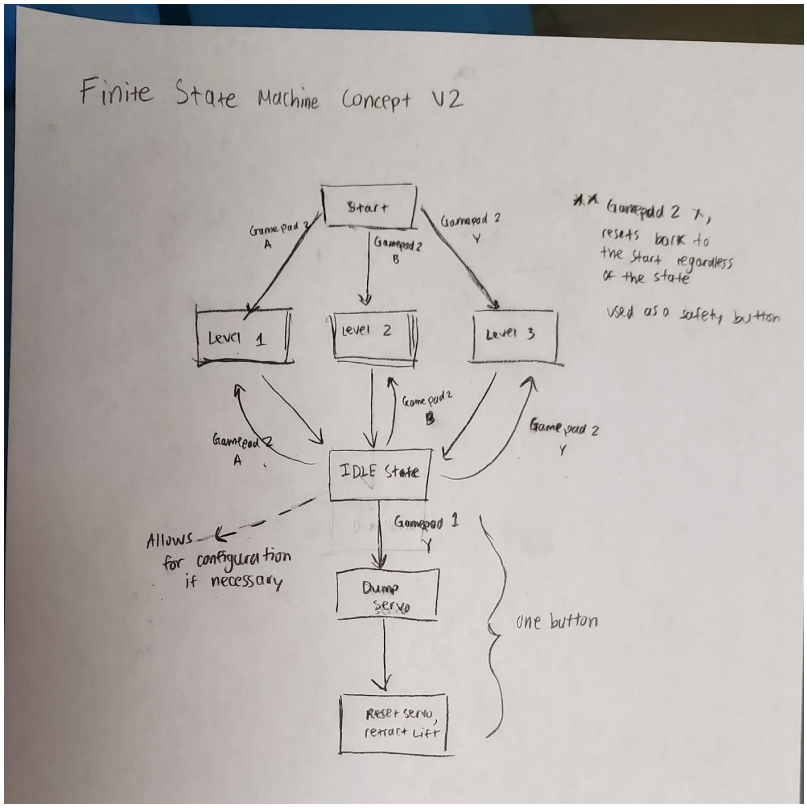
Design:

First Finite State Machine Arm State Diagram:



Second Finite State Machine Arm State Diagram (with all the intended changes):

Signatures:_____Date:_____



We then used this diagram to make adjustments to the code:

```
switch (liftState)
{
   case LIFT_START:
       if(gamepad2.a)
       {
           armMotor.setTargetPosition(LIFT_LEVEL_ONE);
           armMotor.setPower(ARM_SPEED);
           armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
           liftState = LiftState.LIFT_EXTEND_ONE;
       }

       if(gamepad2.b)
       {
           armMotor.setTargetPosition(LIFT_LEVEL_TWO);
           armMotor.setPower(ARM_SPEED);
           armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
           liftState = LiftState.LIFT_EXTEND_TWO;
       }

       if(gamepad2.y)
       {
           armMotor.setTargetPosition(LIFT_LEVEL_THREE);
           armMotor.setPower(ARM_SPEED);
```

Signatures:_____Date:_____

```
            armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            liftState = LiftState.LIFT_EXTEND_THREE;
        }
        break;
    case LIFT_EXTEND_ONE:
        if(Math.abs(armMotor.getCurrentPosition() - LIFT_LEVEL_ONE) < 10)
        {
            armMotor.setPower(0);
            liftState = LiftState.LIFT_DUMP;
        }
        break;
    case LIFT_EXTEND_TWO:
        if(Math.abs(armMotor.getCurrentPosition() - LIFT_LEVEL_TWO) < 10)
        {
            armMotor.setPower(0);
            liftState = LiftState.LIFT_DUMP;
        }
        break;
    case LIFT_EXTEND_THREE:
        if(Math.abs(armMotor.getCurrentPosition() - LIFT_LEVEL_THREE) < 10)
        {
            armMotor.setPower(0);
            liftState = LiftState.LIFT_DUMP;
        }
        break;
    case LIFT_DUMP:
        if(gamepad1.y)
        {
            Bucket.setPosition(DUMP_DEPOSIT);
            dumpTimer.reset();
            liftState = LiftState.LIFT_DUMP_TIMER;
        }

        if(gamepad2.a)
        {
            armMotor.setTargetPosition(LIFT_LEVEL_ONE);
            armMotor.setPower(ARM_SPEED);
            armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            liftState = LiftState.LIFT_EXTEND_ONE;
        }

        if(gamepad2.b)
        {
            armMotor.setTargetPosition(LIFT_LEVEL_TWO);
            armMotor.setPower(ARM_SPEED);
            armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            liftState = LiftState.LIFT_EXTEND_TWO;
        }
```

```
        if(gamepad2.y)
        {
            armMotor.setTargetPosition(LIFT_LEVEL_THREE);
            armMotor.setPower(ARM_SPEED);
            armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            liftState = LiftState.LIFT_EXTEND_THREE;
        }
        break;
    case LIFT_DUMP_TIMER:
        if(dumpTimer.seconds() >= DUMP_TIME)
        {
            liftState = LiftState.LIFT_DUMP_RETRACT;
        }
        break;
    case LIFT_DUMP_RETRACT:
        Bucket.setPosition(DUMP_IDLE);
        dumpResetTimer.reset();
        liftState = LiftState.LIFT_DUMP_RETRACT_TIMER;
        break;
    case LIFT_DUMP_RETRACT_TIMER:
        if(dumpResetTimer.seconds() >= DUMP_RETRACT_TIME)
        {
            armMotor.setTargetPosition(LIFT_LEVEL_ZERO);
            armMotor.setPower(ARM_SPEED);
            armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            liftState = LiftState.LIFT_RETRACT;
        }
        break;
    case LIFT_RETRACT:
        if(Math.abs(armMotor.getCurrentPosition() - LIFT_LEVEL_ZERO) < 10)
        {
            armMotor.setPower(0);
            liftState = LiftState.LIFT_START;
        }
        break;
    default:
        liftState = LiftState.LIFT_START;
}
//safety button to reset the states
if (gamepad2.x && liftState != LiftState.LIFT_START)
{
    Bucket.setPosition(DUMP_IDLE);
    dumpResetTimer.reset();
    liftState = LiftState.LIFT_DUMP_RETRACT_TIMER;
}
```

Test:

To test, we simply ran the program and saw how it performed in depositing the freight.

Signatures:_____Date:_____

We first tested it by observing the arm movement, then moved on to using actual freight and depositing it on the shipping hubs.

Result:

The new state machine went well. We were able to accurately change levels and the freight-depositing process is much more streamlined. We also adjusted the timing of the automation a little based on what we saw to make it safer and more accurate.

Next Step:

To further test the capabilities of the code, we would need to perform competition-like matches to see how it fairs in the real game. Adjustments may be made depending on what we see.

Signatures:_____Date:_____

MetroBotics Team #14212

[Intake Finite State Machine] [February 27th, 2022]

[Eric Guan]

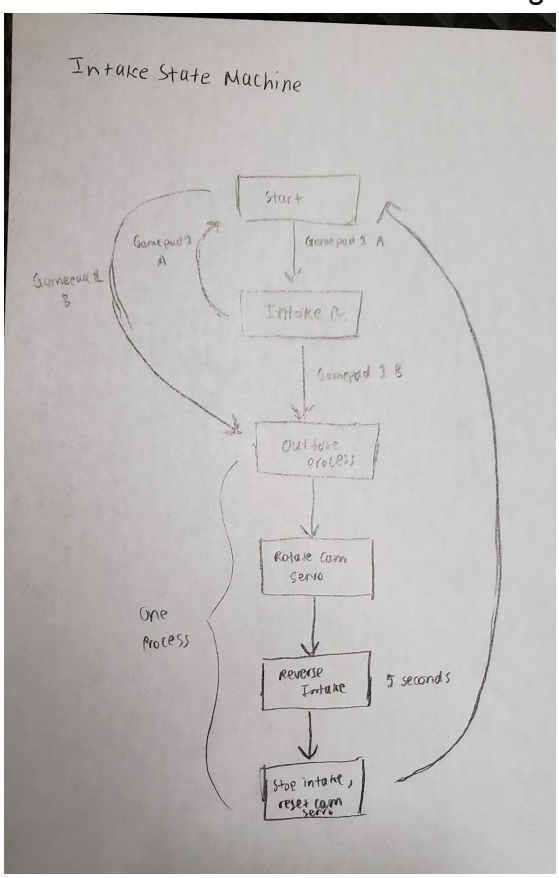
Think:

To increase the efficiency of our intake system, we decided to implement another finite state machine for the intake. This would allow us to automate the intake process and make it easier for the driver to intake freight.

Design:

Because we added a cam servo, we needed to include that into the pre-existing intake. Because Java allows us to have multiple enum states, we were able to simply create a new state machine and put it in our code.

Finite State Machine Intake State Diagram:



Using this state model, we then created it in Java:

```
switch (intakeState)
{
   case INTAKE_START:
```

Signatures:_____Date:_____

```
        if(gamepad1.a)
        {
            intake.setPower(1.0);
            intakePauser.reset();
            intakeState = IntakeState.INTAKE_PAUSE1;
        }

        if(gamepad1.b)
        {
            cam.setPosition(CAM_ON);
            intake.setPower(-1.0);
            intakeTimer.reset();
            intakeState = IntakeState.INTAKE_RESET;
        }
        break;
    case INTAKE_PAUSE1:
        if(intakePauser.seconds() >= INTAKE_PAUSE_TIME)
        {
            intakeState = IntakeState.INTAKE_ON;
        }
        break;
    case INTAKE_PAUSE2:
        if(intakePauser.seconds() >= INTAKE_PAUSE_TIME)
        {
            intakeState = IntakeState.INTAKE_START;
        }
        break;
    case INTAKE_ON:
        if(gamepad1.a)
        {
            intake.setPower(0.0);
            intakePauser.reset();
            intakeState = IntakeState.INTAKE_PAUSE2;
        }

        if(gamepad1.b)
        {
            cam.setPosition(CAM_ON);
            intake.setPower(-1.0);
            intakeTimer.reset();
            intakeState = IntakeState.INTAKE_RESET;
        }
        break;
    case INTAKE_RESET:
        if(intakeTimer.seconds() >= INTAKE_TIME)
        {
            intake.setPower(0.0);
            cam.setPosition(CAM_IDLE);
            intakeState = IntakeState.INTAKE_START;
```

```
        }
        break;
    default:
        intakeState = IntakeState.INTAKE_START;
}
//safety button to reset the states
if (gamepad1.x && liftState != LiftState.LIFT_START)
{
    intake.setPower(0.0);
    cam.setPosition(CAM_IDLE);
    intakeState = IntakeState.INTAKE_START;
}
```

Test:

To check if the code worked, we tested all the functions of the intake using the controllers. Because the intake is supposed to have automation while still allowing us to have complete control over the tele-op, we moved the drivetrain while controlling the intake. We then tested if it worked in intaking and outtaking freight.

Result:

The result of the testing was successful. Our code worked smoothly and as intended. When testing the practicality of the intake, we found it to work most of the time. It is able to efficiently intake the freight and outtake it if necessary.

Next Step:

We need to conduct further testing on the intake state machine in a real game situation. Additionally, we need to implement this intake system into our autonomous program.

Signatures:_____Date:_____