PROGRAMMING

ABOUT

This section outlines our programming journey, starting the season with no programming experience; we went from a very unreliable moving foundation program to a much more complicated 2 skystone placing, moving foundation, and parking program. This section also highlights our current key concepts in both autonomous and tele-op and provides clear diagrams and descriptions of our 3 main autonomous programs.

TABLE OF CONTENTS

AUTONOMOUS | JOURNEY

Programming this season has been a little bit difficult because although 4 of our members have past FIRST experience, none of them had done programming. Despite this fact, we decided to set high goals and work as hard as we could to meet them. The first step in this was deciding what program to use:

1. **Android Studio**
2. **Blocks**
3. **On Bot Java**

We decided to use **Android studio** because we knew it gave us more options and flexibility. We considered what we would want to use down the road as we started implementing more complicated things such as motion profiling. Since we knew we were eventually going to have to use Android Studio in order to implement more complicated code, why not learn it from the beginning? So that's what we decided to do.

Isabel learned how to program by watching youtube videos, looking at the FTC SDK samples, and looking at other teams example code.

WHITE MOUNTAIN, ALAMOGORDO, SOCORRO

For White Mountain, our first qualifier, we only had about 3 days to program the whole robot. So Isabel was only able to figure out **RUN_TO_POSITION** with encoders. After White Mountain, we made a team decision to focus on mechanical aspects of the robot for our first few qualifiers, and code a little later, so wheel encoders were the only sensors used for Alamogordo and Socorro as well.

SENSORS USED

- Wheel encoders

MAIN METHODS USED:

- `public void Drive (int frontRightDistance, int frontLeftDistance, int rearRightDistance, int rearLeftDistance, double speed)`

AUTONOMOUS CAPABILITIES

1. Program 1: **15 Points**
    a. Reposition foundation
    b. Park
2. Program 2: **5 Points**
    a. Park

SANDIA PARK

For Sandia Park, we decided to integrate the rev imu (act as a gyro) into our autonomous so that the robot would drive straight and turn to precise degree values.

SENSORS USED

- Wheel Encoders
- Rev IMU
- Phone Camera -Open CV

MAIN METHODS USED

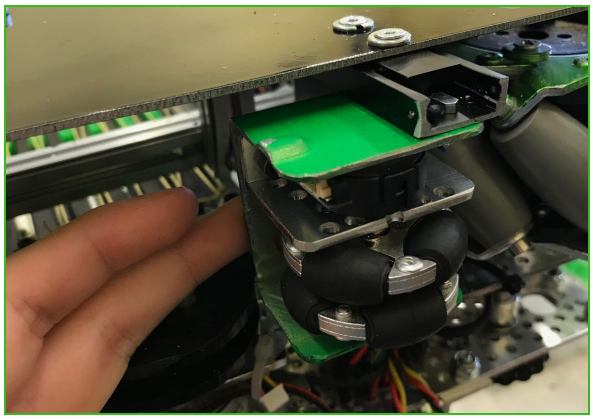
- `public void gyroDrive(double speed, int distance, double angle)`
- `public void gyroTurn(double speed, double angle)`
- `public void gyroHold(double speed, double angle, double holdTime)`
- `boolean onHeading(double speed, double angle, double PCoeff)`
- `public double getError(double targetAngle)`
- `public double getSteer(double error, double PCoeff)`

AUTONOMOUS CAPABILITIES

1. Program 1: **29 points**
    a. Deliver and place 1 skystone
    b. Reposition foundation
    c. Park
2. Program 2: **25 points**
    a. Deliver 2 skystones
    b. Park
3. Program 3: **15 points** (included the option to park far or near the skybridge)
    a. Reposition foundation
    b. Park
4. Program 4: **5 points**
    a. Park

CURRENT

We tried to implement odometry and motion profiling, but mechanical changes took longer than expected, leaving very little time for programming, so **we currently use the same sensors and methods as we did in Sandia Park.**



SENSORS USED

- Wheel Encoders
- Rev IMU
- Phone Camera -Open CV

MAIN METHODS USED

- `public void gyroDrive(double speed, int distance, double angle)`
- `public void gyroTurn(double speed, double angle)`
- `public void gyroHold(double speed, double angle, double holdTime)`
- `boolean onHeading(double speed, double angle, double PCoeff)`
- `public double getError(double targetAngle)`
- `public double getSteer(double error, double PCoeff)`

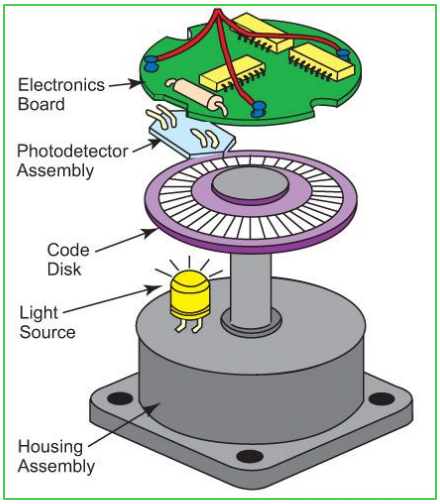
AUTONOMOUS CAPABILITIES

5. Program 1: **43 points**
    a. Deliver and place 2 skystones
    b. Reposition foundation
    c. Park
6. Program 2: **15 points** (included the option to park far or near the skybridge)
    a. Reposition foundation
    b. Park
7. Program 3: **5 points**
    a. Park

AUTONOMOUS | KEY CONCEPTS

Source: STEM Robotics

ENCODERS



From Isabel and Madelene's past FTC experience, the team knew we did not want to drive by time in autonomous. Encoders are much more accurate than drive by time, because they do not rely on the power of the battery. "Encoders attach to or are integrated into the drive motors and count the revolutions of the motor shaft either optically or magnetically. The DcMotor object has support for using encoder counts to control how long the motor will run when turned on." Encoders are used on all four of our drive motors in order to send the robot to specific positions on the field.

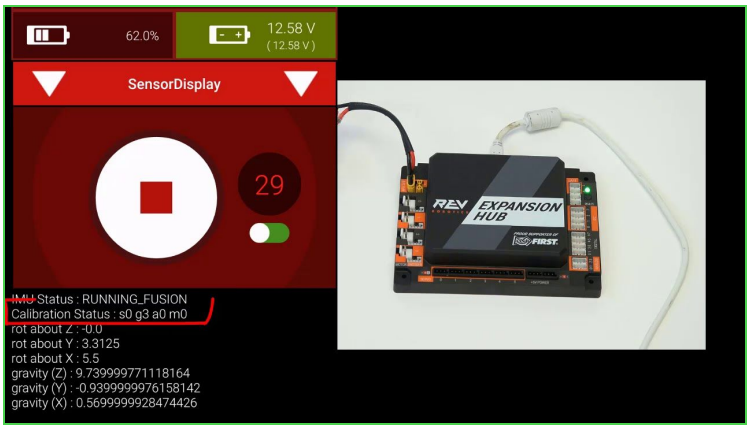
EXAMPLE CODE

```
@Override
public void runOpMode() {

//Sets direction of motor
motor.setDirection(DcMotor.Direction.REVERSE);

//Sets mode of motor
motor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

//Code to run once driver hits play
waitForStart();

//Resets encoder count -always want to start at 0
motor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

//Sets amount of ticks the robot should travel
motor.setTargetPosition(500);

//Sets mode of motor to "run" to target position
motor.setMode(DcMotor.RunMode.RUN_TO_POSITION);

//Gives power to the motor in order to reach target position
motor.setPower(0.3);
```

```
//While the motor is busy going to target position, do nothing
while (wheelFrontRight.isBusy() && opModeIsActive()) {
    }

//After the motor gets to its position, set power to 0
motor.setPower(0);

//Sets mode for next movement
motor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    sleep(500);
}
```

REV IMU & PID CONTROL

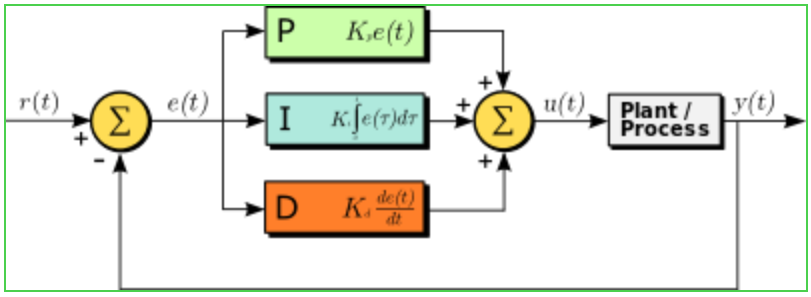
Source: STEM Robotics

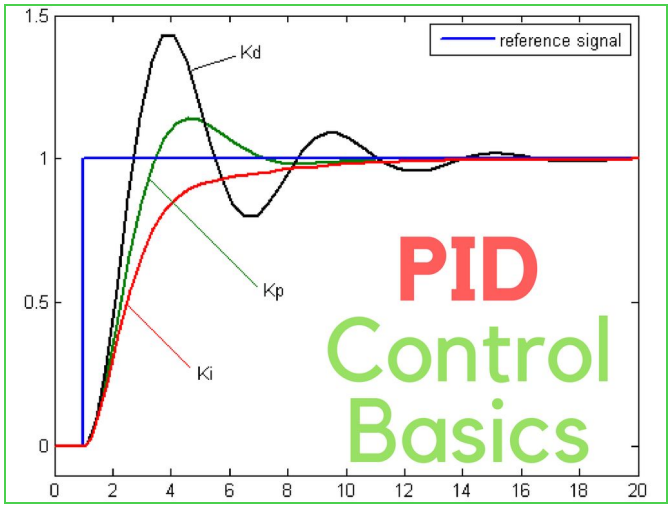


"The rev IMU is a sensor that can measure acceleration (movement) in several axes. It can be used in place of an external gyro. The IMU is not used in quite the same way as the gyro but is similar." We use the rev IMU to get the Z axis of our robot. This tells us the robot's heading, so if the robot turns ~90 degrees, the Z axis of the rev IMU will read ~90 degrees. This is how our robot drives straight and turns.

We have also included a **p loop** in our code to correct the robot if it starts driving off course or overturns. If we tell the robot to drive forward at 0 degrees, and it begins to unintentionally turn 5 degrees, the rev IMU will pick up this error and we can write a method to correct it, however, when correcting, the robot may overshoot. Same thing with turning. If we tell the robot to turn 90 degrees, it may turn 92 degrees because it overshoots depending on friction, motor power, etc. We could, of course, write a method to correct this, but it could overcorrect and still be at the wrong heading, thus the need for PID control is introduced.

"**PID** stands for **Proportional, Integral, Derivative**. The idea behind a PID controller is to take the desired state value, compare that to the actual (feedback) state value (the current gyro or IMU angle) and apply factors to the difference (called the error) that produce a proportional output value. the angle of turn is fed to the PID controller routine which measures error and produces a value (the turn power) at or near the starting (full) power. As the turn gets closer to 90 degrees, the PID routine starts to return smaller and smaller values thus reducing the power being applied and slowing the rate of turn. In theory this reduction in power and slowing rate of turn will eliminate the overshoot. The PID controller can also apply a tolerance margin that indicates when the actual value is within some percentage of the target to further control robot motors."





EXAMPLE CODE

```
public class PIDControlExample extends LinearOpMode {

   /* Declare OpMode members. */
   private DcMotor wheelFrontRight;
   private DcMotor wheelFrontLeft;
   private DcMotor wheelRearRight;
   private DcMotor wheelRearLeft;
   BNO055IMU imu;                // Additional Gyro device
   Orientation angles;


   // These constants define the desired driving/control characteristics
   // The can/should be tweaked to suit the specific robot drive train.
   static final double DRIVE_SPEED = 0.7;     // Nominal speed for better accuracy.
   static final double TURN_SPEED = 0.6;      // Nominal half speed for better accuracy.
```

```
    static final double HEADING_THRESHOLD = 1;      // As tight as we can make it with an integer
gyro
    static final double P_TURN_COEFF = 0.05;     // Larger is more responsive, but also less stable
    static final double P_DRIVE_COEFF = 0.08;     // Larger is more responsive, but also less stable
    @Override
    public void runOpMode() {

        //Initialize the standard drive system variables.
        wheelFrontRight = hardwareMap.get(DcMotor.class, "wheelFrontRight");
        wheelFrontLeft = hardwareMap.get(DcMotor.class, "wheelFrontLeft");
        wheelRearRight = hardwareMap.get(DcMotor.class, "wheelRearRight");
        wheelRearLeft = hardwareMap.get(DcMotor.class, "wheelRearLeft");

        wheelFrontRight.setDirection(DcMotor.Direction.REVERSE);
        wheelRearRight.setDirection(DcMotor.Direction.REVERSE);

        //Setting motor mode regarding encoders
        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        BNO055IMU.Parameters parameters = new BNO055IMU.Parameters();
        parameters.mode = BNO055IMU.SensorMode.IMU;
        parameters.angleUnit = BNO055IMU.AngleUnit.DEGREES;
        parameters.calibrationDataFile = "BNO055IMUCalibration.json";

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

        // Send telemetry message to alert driver that we are calibrating;
        telemetry.addData("Mode", "calibrating...");
        telemetry.update();

        sleep(300);

        telemetry.addData("Mode", "waiting for start");
```

```
        telemetry.addData("imu calib status", imu.getCalibrationStatus().toString());
        telemetry.update();

        waitForStart();
        // Step through each leg of the path,
        // Note: Reverse movement is obtained by setting a negative distance (not speed)

        //Drive Forward
        gyroDrive(DRIVE_SPEED, -450, 0.0);    // Drive FWD 48 inches
        sleep(100);

        //Turn right
        gyroTurn(TURN_SPEED, -90.0);          // Turn  CCW to -45 Degrees
        gyroHold(TURN_SPEED, -90.0, 0.5);     // Hold -45 Deg heading for a 1/2 second
        sleep(100)
        telemetry.addData("Path", "Complete");
        telemetry.update();
        sleep(300);
        angles = imu.getAngularOrientation(AxesReference.INTRINSIC, AxesOrder.ZYX,
AngleUnit.DEGREES);
        this.imu.getPosition();
        telemetry.addData("Z:", angles.firstAngle);
        telemetry.addData("Y:", angles.secondAngle);
        telemetry.addData("X:", angles.thirdAngle);
        telemetry.update();
        sleep(1000);
    }
   /**
     * Method to drive on a fixed compass bearing (angle), based on encoder counts.
     * Move will stop if either of these conditions occur:
     * 1) Move gets to the desired position
     * 2) Driver stops the opmode running.
     *
     * @param speed    Target speed for forward motion.  Should allow for _/- variance for adjusting
heading
     * @param distance Distance (in inches) to move from current position.  Negative distance means
move backwards.
     * @param angle    Absolute Angle (in Degrees) relative to last gyro reset.
     *                 0 = fwd. +ve is CCW from fwd. -ve is CW from forward.
```

```
     *                  If a relative angle is required, add/subtract from current heading.
     */
    public void gyroDrive(double speed,
                          int distance,
                          double angle) {
        double max;
        double error;
        double steer;
        double leftSpeed;
        double rightSpeed;

        // Ensure that the opmode is still active
        if (opModeIsActive()) {

            // Determine new target position, and pass to motor controller
            wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

            wheelFrontRight.setTargetPosition(distance);
            wheelFrontLeft.setTargetPosition(distance);
            wheelRearRight.setTargetPosition(distance);
            wheelRearLeft.setTargetPosition(distance);

            wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            wheelFrontRight.setPower(speed);
            wheelFrontLeft.setPower(speed);
            wheelRearRight.setPower(speed);
            wheelRearLeft.setPower(speed);

            // keep looping while we are still active, and BOTH motors are running.
            while (opModeIsActive() &&
```

```
                (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() && wheelRearRight.isBusy()
&& wheelRearLeft.isBusy())) {

            // adjust relative speed based on heading error.
            error = getError(angle);
            steer = getSteer(error, P_DRIVE_COEFF);

            // if driving in reverse, the motor correction also needs to be reversed
            if (distance < 0)
                steer *= -1.0;

            leftSpeed = speed - steer;
            rightSpeed = speed + steer;

            // Normalize speeds if either one exceeds +/- 1.0;
            max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));
            if (max > 1.0) {
                leftSpeed /= max;
                rightSpeed /= max;
            }

            wheelFrontLeft.setPower(leftSpeed);
            wheelRearLeft.setPower(leftSpeed);
            wheelFrontRight.setPower(rightSpeed);
            wheelRearRight.setPower(rightSpeed);

            // Display drive status for the driver.
            telemetry.addData("Err/St", "%5.1f/%5.1f", error, steer);
            telemetry.addData("Speed", "%5.2f:%5.2f", leftSpeed, rightSpeed);
            telemetry.update();
        }

        // Stop all motion;
        wheelFrontRight.setPower(0);
        wheelRearRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearLeft.setPower(0);
```

```
        // Turn off RUN_TO_POSITION
        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    }
}

public void gyroStrafe(double speed,
                       int distance,
                       double angle) {
    double max;
    double error;
    double steer;
    double leftSpeed;
    double rightSpeed;

    // Ensure that the opmode is still active
    if (opModeIsActive()) {

        // Determine new target position, and pass to motor controller
        wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        wheelFrontRight.setTargetPosition(-distance);
        wheelFrontLeft.setTargetPosition(distance);
        wheelRearRight.setTargetPosition(distance);
        wheelRearLeft.setTargetPosition(-distance);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        wheelFrontRight.setPower(speed);
        wheelFrontLeft.setPower(speed);
```

```
            wheelRearRight.setPower(speed);
            wheelRearLeft.setPower(speed);

            // keep looping while we are still active, and BOTH motors are running.
            while (opModeIsActive() &&
                   (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() && wheelRearRight.isBusy()
&& wheelRearLeft.isBusy())) {

                // adjust relative speed based on heading error.
                error = getError(angle);
                steer = getSteer(error, P_DRIVE_COEFF);

                // if driving in reverse, the motor correction also needs to be reversed
                if (distance < 0)
                    steer *= -1.0;

                leftSpeed = speed - steer;
                rightSpeed = speed + steer;

                // Normalize speeds if either one exceeds +/- 1.0;
                max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));
                if (max > 1.0) {
                    leftSpeed /= max;
                    rightSpeed /= max;
                }

                wheelFrontLeft.setPower(leftSpeed);
                wheelRearLeft.setPower(leftSpeed);
                wheelFrontRight.setPower(rightSpeed);
                wheelRearRight.setPower(rightSpeed);

                // Display drive status for the driver.
                telemetry.addData("Err/St", "%5.1f/%5.1f", error, steer);
                telemetry.addData("Speed", "%5.2f:%5.2f", leftSpeed, rightSpeed);
                telemetry.update();
            }

            // Stop all motion;
```

```
        wheelFrontRight.setPower(0);
        wheelRearRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearLeft.setPower(0);

        // Turn off RUN_TO_POSITION
        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    }
}


/**
 * Method to spin on central axis to point in a new direction.
 * Move will stop if either of these conditions occur:
 * 1) Move gets to the heading (angle)
 * 2) Driver stops the opmode running.
 *
 * @param speed Desired speed of turn.
 * @param angle Absolute Angle (in Degrees) relative to last gyro reset.
 *              0 = fwd. +ve is CCW from fwd. -ve is CW from forward.
 *              If a relative angle is required, add/subtract from current heading.
 */
public void gyroTurn(double speed, double angle) {

    // keep looping while we are still active, and not on heading.
    while (opModeIsActive() && !onHeading(speed, angle, P_TURN_COEFF)) {
        // Update telemetry & Allow time for other processes to run.
        telemetry.update();
    }
}

  /**
 * Method to obtain & hold a heading for a finite amount of time
 * Move will stop once the requested time has elapsed
 *
```

```
     * @param speed    Desired speed of turn.
     * @param angle    Absolute Angle (in Degrees) relative to last gyro reset.
     *                 0 = fwd. +ve is CCW from fwd. -ve is CW from forward.
     *                 If a relative angle is required, add/subtract from current heading.
     * @param holdTime Length of time (in seconds) to hold the specified heading.
     */
    public void gyroHold(double speed, double angle, double holdTime) {

        ElapsedTime holdTimer = new ElapsedTime();

        // keep looping while we have time remaining.
        holdTimer.reset();
        while (opModeIsActive() && (holdTimer.time() < holdTime)) {
            // Update telemetry & Allow time for other processes to run.
            onHeading(speed, angle, P_TURN_COEFF);
            telemetry.update();
        }

        // Stop all motion;
        wheelFrontRight.setPower(0);
        wheelRearRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearLeft.setPower(0);

    }

    /**
     * Perform one cycle of closed loop heading control.
     *
     * @param speed  Desired speed of turn.
     * @param angle  Absolute Angle (in Degrees) relative to last gyro reset.
     *               0 = fwd. +ve is CCW from fwd. -ve is CW from forward.
     *               If a relative angle is required, add/subtract from current heading.
     * @param PCoeff Proportional Gain coefficient
     * @return
     */
    boolean onHeading(double speed, double angle, double PCoeff) {
        double error;
```

```
        double steer;
        boolean onTarget = false;
        double leftSpeed;
        double rightSpeed;

        // determine turn power based on +/- error
        error = getError(angle);

        if (Math.abs(error) <= HEADING_THRESHOLD) {
            steer = 0.0;
            leftSpeed = 0.0;
            rightSpeed = 0.0;
            onTarget = true;
        } else {
            steer = getSteer(error, PCoeff);
            rightSpeed = speed * steer;
            leftSpeed = -rightSpeed;
        }

        // Send desired speeds to motors.
        wheelFrontLeft.setPower(leftSpeed);
        wheelRearLeft.setPower(leftSpeed);
        wheelFrontRight.setPower(rightSpeed);
        wheelRearRight.setPower(rightSpeed);

        // Display it for the driver.
        telemetry.addData("Target", "%5.2f", angle);
        telemetry.addData("Err/St", "%5.2f/%5.2f", error, steer);
        telemetry.addData("Speed.", "%5.2f:%5.2f", leftSpeed, rightSpeed);

        return onTarget;
    }

    /**
     * getError determines the error between the target angle and the robot's current heading
     *
     * @param targetAngle Desired angle (relative to global reference established at last Gyro
Reset).
```

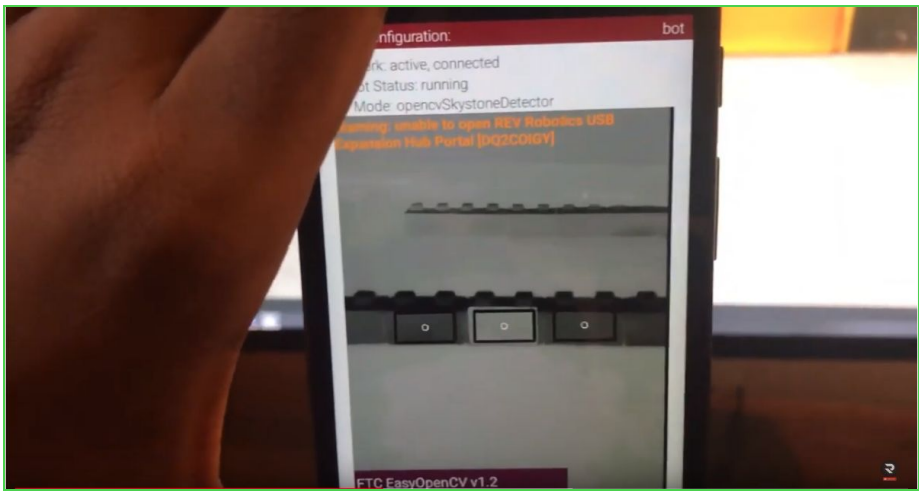
```
     * @return error angle: Degrees in the range +/- 180. Centered on the robot's frame of reference
     * +ve error means the robot should turn LEFT (CCW) to reduce error.
     */
    public double getError(double targetAngle) {

        double robotError;
        angles = imu.getAngularOrientation(AxesReference.INTRINSIC, AxesOrder.ZYX,
AngleUnit.DEGREES);
        this.imu.getPosition();
        // calculate error in -179 to +180 range  (
        robotError = targetAngle - angles.firstAngle;
        while (robotError > 180) robotError -= 360;
        while (robotError <= -180) robotError += 360;
        return robotError;
    }

    /**
     * returns desired steering force.  +/- 1 range.  +ve = steer left
     *
     * @param error  Error angle in robot relative degrees
     * @param PCoeff Proportional Gain Coefficient
     * @return
     */
    public double getSteer(double error, double PCoeff) {
        return Range.clip(error * PCoeff, -1, 1);
    }

}
```

OPEN CV

Source: OpenCV -about

“OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.... These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.”

We use OpenCV in our autonomous program to detect the skystones. We were able to utilize the code that team **#3939 Bolts of Steel** so kindly shared, with a few adjustments. This code is extremely reliable, working 100% of the time.



This code creates three rectangles on the phone with a circle in the center. These shapes are not trying to find stones to focus on and they do not move at any point, so the phone and three rectangles must initially be set up in line with the three prefered stones.

When the drivers press play, each of the three circles is looking at the shade difference on a grey scale. If the center circle detects a more white shade, and the other two on each side detect a darker gray, the program will return:

```
valMid = 255

valLeft = 0

valRight = 0
```

If `valMid` returns `255`, and `ValLeft` and `ValRight` both return `0`, we know that the skystone is in the middle. Same for the other combinations. We then use this information to construct if statements. For example:

```
if (valMid == 255 && valLeft == 0 && valRight == 0) {

//Do this

}
```

The same can be done for the other two combinations.

EXAMPLE CODE

Credit: #3939, Bolts of Steel

```
public class EasyOpenCVTest extends LinearOpMode {
       private ElapsedTime runtime = new ElapsedTime();

       //0 means skystone, 1 means yellow stone
       //-1 for debug, but we can keep it like this because if it works, it should change to either
0 or 255
       private static int valMid = -1;
       private static int valLeft = -1;
       private static int valRight = -1;

       private static float rectHeight = .6f/8f;
       private static float rectWidth = 1.5f/8f;

       private static float offsetX = 2f/10f;//changing this moves the three rects and the three
circles left or right, range : (-2, 2) not inclusive
       private static float offsetY = 0f/8f;//changing this moves the three rects and circles up or
down, range: (-4, 4) not inclusive

       private static float[] midPos = {4f/8f+offsetX, 4f/8f+offsetY};//0 = col, 1 = row
       private static float[] leftPos = {2f/8f+offsetX, 4f/8f+offsetY};
       private static float[] rightPos = {6f/8f+offsetX, 4f/8f+offsetY};
       //moves all rectangles right or left by amount. units are in ratio to monitor

   private final int rows = 640;
   private final int cols = 480;

   OpenCvCamera phoneCam;

   @Override
   public void runOpMode() throws InterruptedException {

       int cameraMonitorViewId =
hardwareMap.appContext.getResources().getIdentifier("cameraMonitorViewId", "id",
hardwareMap.appContext.getPackageName());
       phoneCam = new OpenCvInternalCamera(OpenCvInternalCamera.CameraDirection.BACK,
cameraMonitorViewId);
       phoneCam.openCameraDevice();//open camera
       phoneCam.setPipeline(new StageSwitchingPipeline());//different stages
       phoneCam.startStreaming(rows, cols, OpenCvCameraRotation.SIDEWAYS_LEFT);//display on RC
       //width, height
       //width = height in this case, because camera is in portrait mode.
```

```
        waitForStart();
        runtime.reset();

        while (opModeIsActive()) {
            telemetry.addData("Values", valLeft+"   "+valMid+"   "+valRight);
            telemetry.addData("Height", rows);
            telemetry.addData("Width", cols);

            telemetry.update();
            sleep(100);
            //call movement functions
//              strafe(0.4, 200);
//              moveDistance(0.4, 700);

        }
    }

    //detection pipeline
    static class StageSwitchingPipeline extends OpenCvPipeline
    {
        Mat yCbCrChan2Mat = new Mat();
        Mat thresholdMat = new Mat();
        Mat all = new Mat();
        List<MatOfPoint> contoursList = new ArrayList<>();

        enum Stage
        {//color difference. greyscale
            detection,//includes outlines
            THRESHOLD,//b&w
            RAW_IMAGE,//displays raw view
        }

        private Stage stageToRenderToViewport = Stage.detection;
        private Stage[] stages = Stage.values();

        @Override
        public void onViewportTapped()
        {
            /*
             * Note that this method is invoked from the UI thread
             * so whatever we do here, we must do quickly.
             */

            int currentStageNum = stageToRenderToViewport.ordinal();

            int nextStageNum = currentStageNum + 1;

            if(nextStageNum >= stages.length)
            {
                nextStageNum = 0;
            }

            stageToRenderToViewport = stages[nextStageNum];
        }

        @Override
        public Mat processFrame(Mat input)
        {
            contoursList.clear();
            /*
             * This pipeline finds the contours of yellow blobs such as the Gold Mineral
             * from the Rover Ruckus game.
             */

            //color diff cb.
```

```
            //lower cb = more blue = skystone = white
            //higher cb = less blue = yellow stone = grey
            Imgproc.cvtColor(input, yCbCrChan2Mat, Imgproc.COLOR_RGB2YCrCb);//converts rgb to ycrcb
            Core.extractChannel(yCbCrChan2Mat, yCbCrChan2Mat, 2);//takes cb difference and stores

            //b&w
            Imgproc.threshold(yCbCrChan2Mat, thresholdMat, 102, 255, Imgproc.THRESH_BINARY_INV);

            //outline/contour
            Imgproc.findContours(thresholdMat, contoursList, new Mat(), Imgproc.RETR_LIST,
Imgproc.CHAIN_APPROX_SIMPLE);
            yCbCrChan2Mat.copyTo(all);//copies mat object
            //Imgproc.drawContours(all, contoursList, -1, new Scalar(255, 0, 0), 3, 8);//draws blue
contours


            //get values from frame
            double[] pixMid = thresholdMat.get((int)(input.rows()* midPos[1]), (int)(input.cols()*
midPos[0]));//gets value at circle
            valMid = (int)pixMid[0];

            double[] pixLeft = thresholdMat.get((int)(input.rows()* leftPos[1]), (int)(input.cols()*
leftPos[0]));//gets value at circle
            valLeft = (int)pixLeft[0];

            double[] pixRight = thresholdMat.get((int)(input.rows()* rightPos[1]),
(int)(input.cols()* rightPos[0]));//gets value at circle
            valRight = (int)pixRight[0];

            //create three points
            Point pointMid = new Point((int)(input.cols()* midPos[0]), (int)(input.rows()*
midPos[1]));
            Point pointLeft = new Point((int)(input.cols()* leftPos[0]), (int)(input.rows()*
leftPos[1]));
            Point pointRight = new Point((int)(input.cols()* rightPos[0]), (int)(input.rows()*
rightPos[1]));

            //draw circles on those points
            Imgproc.circle(all, pointMid,5, new Scalar( 255, 0, 0 ),1 );//draws circle
            Imgproc.circle(all, pointLeft,5, new Scalar( 255, 0, 0 ),1 );//draws circle
            Imgproc.circle(all, pointRight,5, new Scalar( 255, 0, 0 ),1 );//draws circle

            //draw 3 rectangles
            Imgproc.rectangle(//1-3
                    all,
                    new Point(
                            input.cols()*(leftPos[0]-rectWidth/2),
                            input.rows()*(leftPos[1]-rectHeight/2)),
                    new Point(
                            input.cols()*(leftPos[0]+rectWidth/2),
                            input.rows()*(leftPos[1]+rectHeight/2)),
                    new Scalar(0, 255, 0), 3);
            Imgproc.rectangle(//3-5
                    all,
                    new Point(
                            input.cols()*(midPos[0]-rectWidth/2),
                            input.rows()*(midPos[1]-rectHeight/2)),
                    new Point(
                            input.cols()*(midPos[0]+rectWidth/2),
                            input.rows()*(midPos[1]+rectHeight/2)),
                    new Scalar(0, 255, 0), 3);
            Imgproc.rectangle(//5-7
                    all,
                    new Point(
                            input.cols()*(rightPos[0]-rectWidth/2),
```

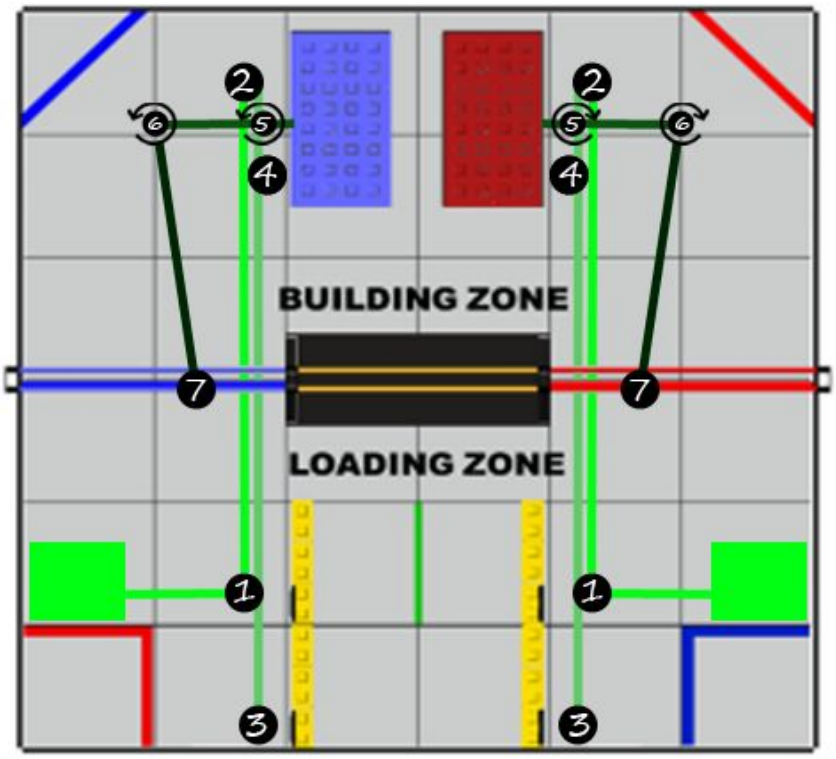
```
                        input.rows()*(rightPos[1]-rectHeight/2)),
                new Point(
                        input.cols()*(rightPos[0]+rectWidth/2),
                        input.rows()*(rightPos[1]+rectHeight/2)),
                new Scalar(0, 255, 0), 3);

        switch (stageToRenderToViewport)
        {
            case THRESHOLD:
            {
                return thresholdMat;
            }

            case detection:
            {
                return all;
            }

            case RAW_IMAGE:
            {
                return input;
            }

            default:
            {
                return input;
            }
        }
    }

}
```

AUTONOMOUS OPTIONS DIAGRAM

PROGRAM 1 | DELIVER & PLACE BOTH SKYSTONES, REPOSITION FOUNDATION, PARK

ACTIONS (red alliance)

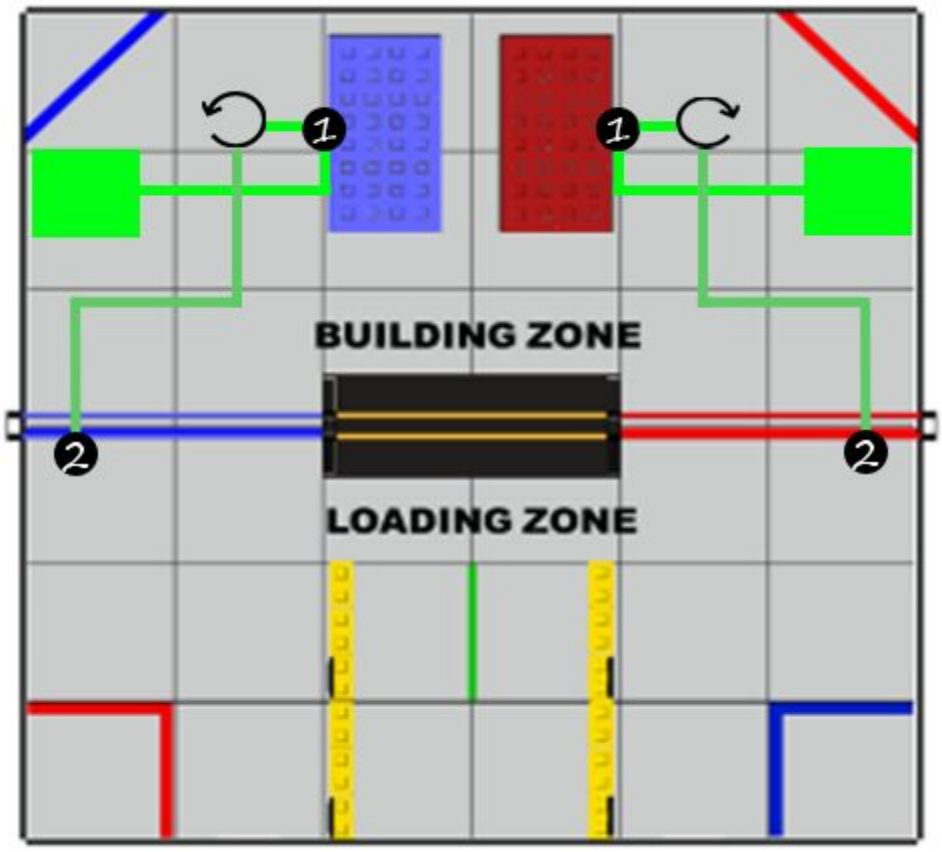
1. Strafe to skystone as claw rotates down
2. Claw grabs skystone
3. Drive forward (towards foundation) and swing claw & stone out of the way of the skybridge
4. Place skystone on foundation
5. Drive backwards (to 2nd skystone) and swing claw out of the way of the skybridge
6. Swing claw back out and grab second stone
7. Drive forward (towards foundation) and swing claw & stone out of the way of the skybridge
8. Place skystone on foundation
9. Turn clockwise 90 degrees
10. Drive forwards and clamp onto foundation
11. Drive backwards (towards building zone)
12. Turn clockwise 90 degrees (rotating foundation into build side)
13. Let go of foundation
14. Turn clockwise 5 degrees
15. Extend tape measure (park)



PROGRAM 2 | REPOSITION FOUNDATION

ACTIONS (red alliance)

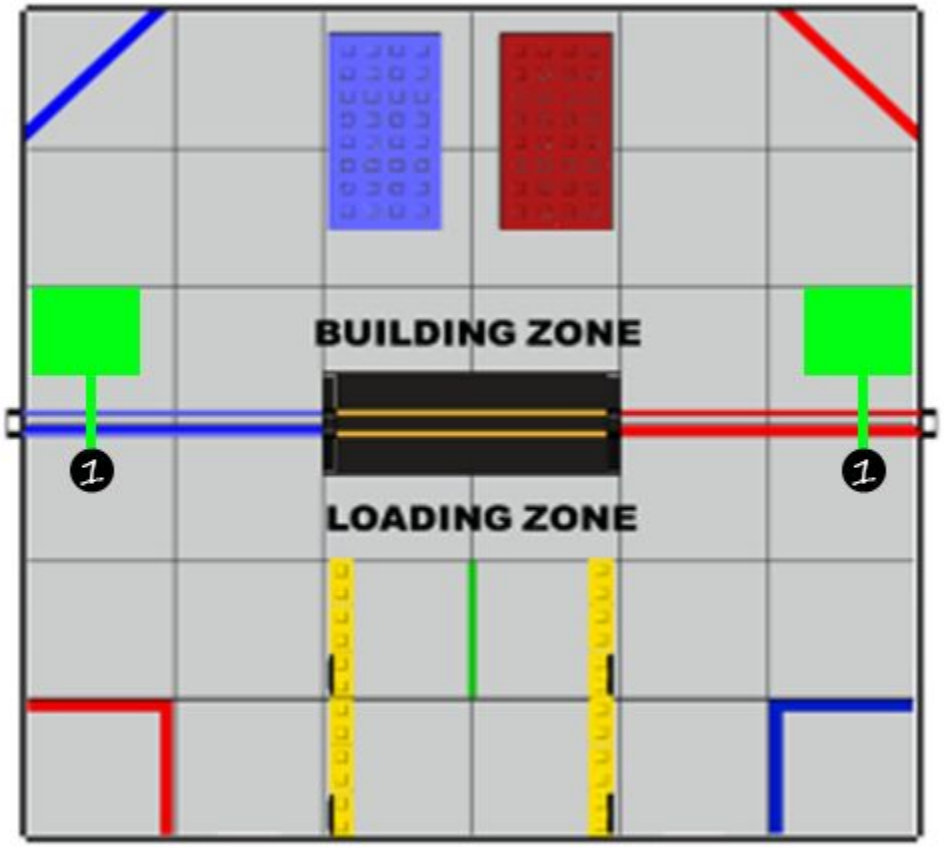
16. Drive forward
17. Clamp foundation
18. Strafe right
19. Drive backwards
20. Rotate clockwise 90 degrees
21. Foundation movers let go
22. Drive backwards
23. Strafe left
24. Drive forward (park)



PROGRAM 3 | PARK

ACTIONS

1. Tape measure extend



TELE OP | KEY CONCEPTS

LESSONS LEARNED

BRAKE MODE

At our first few qualifiers, the robot's horizontal and vertical slide motors were drifting after power was applied and then suddenly taken away. This was causing the lifts to overshoot, and it was slowing us down. After Alamogordo, we learned that there is a mode you can set the motors to called brake mode. This causes the motors to come to a complete stop immediately after power is taken away.

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

This additional line of code cut off multiple seconds from our driver controlled cycles because the slides stopped when the driver "told" them to stop instead of drifting and then stopping.

LIMIT SWITCHES

When retracting vertical lifts, there was always a chance of the string coming off of the spool when reversing the motor too far. Thus, the driver had to be super careful to run it just the right amount, which wasted time during driver controlled. Additionally, the driver had to be careful not to retract the horizontal slides too far back, which also wasted time. Thus, we found the need for limit switches. We decided to use two rev magnetic limit switches: one for the bottom of the vertical slides, and one for the back of the horizontal slides.

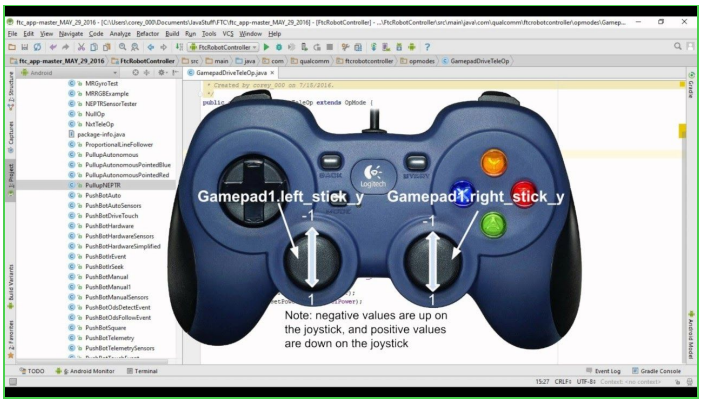
CODE | VERTICAL LIFTS

```
double motorVerticalLiftRightPower = -gamepad2.right_stick_y;
double motorVerticalLiftLeftPower = -gamepad2.right_stick_y;

if (!magneticLimitSwitchY.isPressed()) {
   motorVerticalLiftRightPower = Range.clip(motorVerticalLiftRightPower, -1, 1);
   motorVerticalLiftLeftPower = Range.clip(motorVerticalLiftLeftPower, -1, 1);
   motorVerticalLiftRight.setPower(motorVerticalLiftRightPower);
   motorVerticalLiftLeft.setPower(motorVerticalLiftLeftPower);
} else {
   motorVerticalLiftRightPower = Range.clip(motorVerticalLiftRightPower, 0, 1);
   motorVerticalLiftLeftPower = Range.clip(motorVerticalLiftLeftPower, 0, 1);
   motorVerticalLiftRight.setPower(motorVerticalLiftRightPower);
   motorVerticalLiftLeft.setPower(motorVerticalLiftLeftPower);
}
```

CODE | HORIZONTAL SLIDES

```
double motorHorizontalSlidePower = -gamepad2.left_stick_y;

if (!magneticLimitSwitchX.isPressed()) {
    motorHorizontalSlidePower = Range.clip(motorHorizontalSlidePower, -1, 1);
    motorHorizontalSlide.setPower(motorHorizontalSlidePower);
} else {
    motorHorizontalSlidePower = Range.clip(motorHorizontalSlidePower, 0, 1);
    motorHorizontalSlide.setPower(motorHorizontalSlidePower);
}
```

SETTING POWER -JOYSTICKS



Joysticks return a value of -1 to 1. Motors run off of the same range, so power can be set to the motors using the values that the joysticks return. The one thing to note is that (positive) 1 is returned when the joystick is pushed all the way down, and -1 is returned when the joystick is pushed all the way up. If power is set the motor like this:

```
wheelFrontRight.setPower = gamepad2.right_stick_y;
```

The motor will run in reverse when pushing the joystick forwards, and forwards when the joystick is pushed back; this is counter intuitive, so the way to fix it is to set the motor power like this:

```
wheelFrontRight.setPower = -gamepad2.right_stick_y;
```

This sets the motor power to the opposite sign of the returned joystick value, so -1, or forwards on the joystick, will multiply by the negative sign in front of `gamepad2.right_stick_y` and become (positive) 1, which will send positive power to the motor.

PROGRAMMING MECANUM DRIVE

Programming our mecanum drive was a big challenge for us the first few attempts. After writing multiple programs, it was either driving backwards, not strafing correctly, or all of the controls were inversed. Thanks to team **#9794, Wizardz.exe** for posting a tutorial on programming a mecanum drive, because we were able to adapt their blocks code to android studio, and get it working!

```
double vertical = -gamepad1.left_stick_y;
double horizontal = gamepad1.left_stick_x;
double pivot = gamepad1.right_stick_x;

double wheelFrontRightPower = -pivot + vertical - horizontal;
double wheelFrontLeftPower = -pivot + vertical + horizontal;
double wheelRearRightPower = pivot + vertical + horizontal;
double wheelRearLeftPower = pivot + vertical - horizontal;
```

CONTROLLER SCHEMATIC

GAMEPAD ONE

RIGHT STICK X AXIS	DRIVE: ROTATE
LEFT STICK X AXIS	DRIVE: STRAFE
LEFT STICK Y AXIS	DRIVE: FORWARD & BACKWARD

GAMEPAD TWO

RIGHT STICK Y AXIS	VERTICAL LIFTS
LEFT STICK Y AXIS	HORIZONTAL SLIDES

FULL CODE: 2 SKYSTONES DELIVERED AND PLACED, MOVED FOUNDATION, PARKED

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

import com.qualcomm.hardware.bosch.BNO055IMU;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.util.Range;

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.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.MatOfPoint2f;
import org.opencv.core.Point;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.imgproc.Imgproc;
import org.openftc.easyopencv.OpenCvCamera;
import org.openftc.easyopencv.OpenCvCameraRotation;
import org.openftc.easyopencv.OpenCvInternalCamera;
import org.openftc.easyopencv.OpenCvPipeline;

import java.util.ArrayList;
import java.util.List;


/**
* Created by maryjaneb  on 11/13/2016.
*
* nerverest ticks
* 60 1680
* 40 1120
* 20 560
*
* monitor: 640 x 480
*YES
*/
@Autonomous(name= "Blue: skystone")
//@Disabled
public class SkystoneeBlue extends LinearOpMode {
   private ElapsedTime runtime = new ElapsedTime();

   private static int valMid = -1;
   private static int valLeft = -1;
   private static int valRight = -1;
```

```
   private static float rectHeight = .6f/8f;
   private static float rectWidth = 1.5f/8f;

   private static float offsetX = 0f/10f;//changing this moves the three rects and the three
circles left or right, range : (-2, 2) not inclusive
   private static float offsetY = 0.5f/8f;//changing this moves the three rects and circles
up or down, range: (-4, 4) not inclusive

   private static float[] midPos = {4f/8f+offsetX, 4f/8f+offsetY};//0 = col, 1 = row
   private static float[] leftPos = {2f/8f+offsetX, 4f/8f+offsetY};
   private static float[] rightPos = {6f/8f+offsetX, 4f/8f+offsetY};

   //moves all rectangles right or left by amount. units are in ratio to monitor
   private final int rows = 640;
   private final int cols = 480;


   static final double DRIVE_SPEED = .8;     // Nominal speed for better accuracy.
   static final double TURN_SPEED = 0.6;     // Nominal half speed for better accuracy.

   static final double HEADING_THRESHOLD = 1;      // As tight as we can make it with an
integer gyro
   static final double P_TURN_COEFF = 0.05;     // Larger is more responsive, but also less
stable
   static final double P_DRIVE_COEFF = 0.04;     // Larger is more responsive, but also
less stable

   private static final double AUTO_CLAW_SPIN_IN = .50;
   private static final double AUTO_CLAW_SPIN_OUT_LEFT = 1;
   private static final double AUTO_CLAW_SPIN_OUT_RIGHT = .06;
   private static final double AUTO_CLAW_SPIN_LETGO = .41;
   private static final double AUTO_CLAW_ROTATE_UPUP = .54;
   private static final double AUTO_CLAW_ROTATE_INIT = 1;
   private static final double AUTO_CLAW_ROTATE_DOWN = .21;
   private static final double AUTO_CLAW_ROTATE_UP = .38;
   private static final double AUTO_CLAW_CLAMP_INIT = .31;
   private static final double AUTO_CLAW_CLAMP_DOWN = .15;
   private static final double AUTO_CLAW_CLAMP_UP = .70;
   private static final double AUTO_CLAW_CLAMP_LETGO = .33;

   private static final double FOUNDATION_ARM_OUT_ONE = .84;
   private static final double FOUNDATION_ARM_IN_ONE = .52;

   private static final double FOUNDATION_ARM_OUT_TWO = 0;
   private static final double FOUNDATION_ARM_IN_TWO = .30;

   private DcMotor wheelFrontRight;
   private DcMotor wheelFrontLeft;
   private DcMotor wheelRearRight;
   private DcMotor wheelRearLeft;
   private Servo foundationServoOne;
   private Servo foundationServoTwo;
   private Servo autoClawRotate;
   private Servo autoClawClamp;
   private Servo autoClawSpin;
   private DcMotor tapeMeasure;
```

```
    BNO055IMU imu;                  // Additional Gyro device
    Orientation angles;
    OpenCvCamera phoneCam;



    @Override
    public void runOpMode() throws InterruptedException {

        int cameraMonitorViewId =
hardwareMap.appContext.getResources().getIdentifier("cameraMonitorViewId", "id",
hardwareMap.appContext.getPackageName());
        phoneCam = new OpenCvInternalCamera(OpenCvInternalCamera.CameraDirection.BACK,
cameraMonitorViewId);
        phoneCam.openCameraDevice();//open camera
        phoneCam.setPipeline(new StageSwitchingPipeline());//different stages
        phoneCam.startStreaming(rows, cols, OpenCvCameraRotation.SIDEWAYS_LEFT);//display on
RC
        //width, height
        //width = height in this case, because camera is in portrait mode.

        wheelFrontRight = hardwareMap.get(DcMotor.class, "wheelFrontRight");
        wheelFrontLeft = hardwareMap.get(DcMotor.class, "wheelFrontLeft");
        wheelRearRight = hardwareMap.get(DcMotor.class, "wheelRearRight");
        wheelRearLeft = hardwareMap.get(DcMotor.class, "wheelRearLeft");

        autoClawClamp = hardwareMap.get(Servo.class, "autoClawClamp");
        autoClawRotate = hardwareMap.get(Servo.class, "autoClawRotate");
        autoClawSpin = hardwareMap.get(Servo.class, "autoClawSpin");

        tapeMeasure = hardwareMap.get(DcMotor.class, "tapeMeasure");

        wheelFrontRight.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        wheelFrontLeft.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        wheelRearRight.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        wheelRearLeft.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);


        wheelFrontRight.setDirection(DcMotor.Direction.REVERSE);
        wheelRearRight.setDirection(DcMotor.Direction.REVERSE);

        //Setting motor mode regarding encoders
        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        foundationServoOne = hardwareMap.get(Servo.class, "foundationServoOne");
        foundationServoTwo = hardwareMap.get(Servo.class, "foundationServoTwo");

        BNO055IMU.Parameters parameters = new BNO055IMU.Parameters();
        parameters.mode = BNO055IMU.SensorMode.IMU;
        parameters.angleUnit = BNO055IMU.AngleUnit.DEGREES;
        parameters.calibrationDataFile = "BNO055IMUCalibration.json";

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

```
// Send telemetry message to alert driver that we are calibrating;
telemetry.addData("Mode", "calibrating...");
telemetry.update();

sleep(300);

telemetry.addData("Mode", "waiting for start");
telemetry.addData("imu calib status", imu.getCalibrationStatus().toString());
telemetry.update();

waitForStart();
runtime.reset();

while (opModeIsActive()) {
    telemetry.addData("Values", valLeft+"   "+valMid+"   "+valRight);
    telemetry.addData("Height", rows);
    telemetry.addData("Width", cols);

    telemetry.update();
    sleep(100);

    if (valLeft == 255 && valMid == 255 && valRight == 0) {

        //Rotates ClawRotate Down
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
        sleep(500);

        //Strafes over to first stone, clamp up, spin in
        Drive(-1230, 1230,1230, -1230, 0.5);

        //Corrects heading and clamps stone
        gyroTurn(TURN_SPEED, 0.8);
        sleep(100);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
        gyroHold(TURN_SPEED, 0.8, 0.03);
        sleep(700);


        //Drives towards foundation, rotates up, spins out, spins in
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UPUP);
        sleep(200);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_LEFT);
        sleep(200);
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
        sleep(200);
        gyroDrive(0.8, 4250, 0.8);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
        sleep(100);

        //Corrects heading, drops stone
        gyroTurn(TURN_SPEED, 0.2);
        sleep(600);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
        gyroHold(TURN_SPEED, 0.2, 0.03);
        sleep(300);
```

```
//Drives back to second stone, spins out, rotates down, spins in
autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_LEFT);
sleep(200);
autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
sleep(100);
gyroDriveClawBack(0.8, -5285, 0.2);

//sleep(100);

//Corrects heading and clamps stone
gyroTurn(TURN_SPEED, 0.4);
sleep(400);
autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
gyroHold(TURN_SPEED, 0.4, 0.03);
sleep(700);

//Drives back to foundation
autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UPUP);
sleep(150);
//autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_LEFT);
gyroDrive2(0.8, 4870, 0.4);
autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
sleep(100);

//Lets go
gyroTurn(TURN_SPEED, 0.4);
sleep(400);
autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
gyroHold(TURN_SPEED, 0.4, 0.03);
sleep(300);

//Turns to move foundation
foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
gyroTurn(0.5, 90);
gyroHold(0.5, 90, 0.5);

//Drives forwards
DriveStrafe(-450, -450, -450, -450, 0.2);
//gyroDrive(0.2, -450, 90);
foundationServoOne.setPosition(FOUNDATION_ARM_IN_ONE);
foundationServoTwo.setPosition(FOUNDATION_ARM_IN_TWO);
sleep(100);

gyroTurn(TURN_SPEED, 100);
gyroHold(TURN_SPEED, 100, 0.5);

//gyroTurn(TURN_SPEED, 80);
//Drives backwards, turns, strafes
gyroDrive(0.6, 1100, 100);
sleep(100);
gyroTurn(0.5, 175);

foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
sleep(200);
```

```
        gyroTurn(TURN_SPEED, 180);
        gyroHold(TURN_SPEED, 180, 0.5);
        //sleep(100);
        tapeMeasure.setPower(.8);
        sleep(1000);
        tapeMeasure.setPower(0);



    }

    sleep(200);

    if (valMid == 0 && valRight == 255 && valLeft == 255) {
        telemetry.addData("Status", "valMid has been detected");
        telemetry.update();
        sleep(100);

        //Rotates ClawRotate Down
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
        sleep(500);

        //Strafes over to first stone, clamp up, spin in
        Drive(-1230, 1230,1230, -1230, 0.5);

        gyroDrive(0.5, 330, 0);

        //Corrects heading and clamps stone
        gyroTurn(TURN_SPEED, 0.8);
        sleep(100);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
        gyroHold(TURN_SPEED, 0.8, 0.03);
        sleep(700);


        //Drives towards foundation, rotates up, spins out, spins in
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UPUP);
        sleep(200);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_LEFT);
        sleep(200);
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
        sleep(200);
        gyroDrive(0.8, 3600, 0.8);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
        sleep(100);

        //Corrects heading, drops stone
        gyroTurn(TURN_SPEED, 0.2);
        sleep(600);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
        gyroHold(TURN_SPEED, 0.2, 0.03);
        sleep(300);

        //Drives back to second stone, spins out, rotates down, spins in
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_LEFT);
        sleep(200);
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
```

```
sleep(100);
gyroDriveClawBack(0.8, -4715, 0.2);

//sleep(100);

//Corrects heading and clamps stone
gyroTurn(TURN_SPEED, 0.4);
sleep(400);
autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
gyroHold(TURN_SPEED, 0.4, 0.03);
sleep(700);

//Drives back to foundation
autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UPUP);
sleep(250);
autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_LEFT);
sleep(250);
autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
sleep(100);
gyroDrive(0.8, 4520, 0.4);
autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
sleep(100);

//Lets go
gyroTurn(TURN_SPEED, 0.4);
sleep(400);
autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
gyroHold(TURN_SPEED, 0.4, 0.03);
sleep(300);

//Turns to move foundation
foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
gyroTurn(0.5, 90);

//Drives forwards
DriveStrafe(-450, -450, -450, -450, 0.2);
//gyroDrive(0.2, -450, 90);
foundationServoOne.setPosition(FOUNDATION_ARM_IN_ONE);
foundationServoTwo.setPosition(FOUNDATION_ARM_IN_TWO);
sleep(100);

gyroTurn(TURN_SPEED, 100);

//gyroTurn(TURN_SPEED, 80);
//Drives backwards, turns, strafes
gyroDrive(0.6, 1100, 100);
gyroTurn(0.5, 175);

foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
sleep(200);

gyroTurn(TURN_SPEED, 180);
//sleep(100);
tapeMeasure.setPower(.8);
sleep(1000);
```

```
        tapeMeasure.setPower(0);

    }

    sleep(200);

    if (valRight == 255 && valMid == 255 && valLeft == 0) {
        telemetry.addData("Status", "valLeft has been detected");
        telemetry.update();
        sleep(100);

        //Rotates ClawRotate Down
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
        sleep(500);

        //Strafes over to first stone, clamp up, spin in
        Drive(-1215, 1215,1215, -1215, 0.5);

        gyroDrive(0.5, 620, 0);

        //Corrects heading and clamps stone
        gyroTurn(TURN_SPEED, 0.8);
        sleep(100);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
        gyroHold(TURN_SPEED, 0.8, 0.03);
        sleep(700);


        //Drives towards foundation, rotates up, spins out, spins in
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UPUP);
        sleep(200);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_LEFT);
        sleep(200);
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
        sleep(200);
        gyroDrive(0.8, 3400, 0.8);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
        sleep(100);

        //Corrects heading, drops stone
        gyroTurn(TURN_SPEED, 0.2);
        sleep(600);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
        gyroHold(TURN_SPEED, 0.2, 0.03);
        sleep(300);

        //Drives back to second stone, spins out, rotates down, spins in
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_LEFT);
        sleep(250);
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
        sleep(100);
        gyroDriveClawBack(0.8, -4540, 0.2);

        //sleep(100);

        //Corrects heading and clamps stone
        gyroTurn(TURN_SPEED, -1);
```

```
        sleep(400);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
        gyroHold(TURN_SPEED, -1, 0.03);
        sleep(700);

        //Drives back to foundation
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UPUP);
        sleep(250);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_LEFT);
        sleep(100);
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
        gyroDrive(0.8, 4300, -1);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
        sleep(100);

        //Lets go
        gyroTurn(TURN_SPEED, 0);
        sleep(400);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
        gyroHold(TURN_SPEED, 0, 0.03);
        sleep(300);

        //Turns to move foundation
        foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
        foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
        gyroTurn(0.5, 90);
        gyroHold(0.5, 90, 0.5);

        //Drives forwards
        DriveStrafe(-450, -450, -450, -450, 0.2);
        //gyroDrive(0.2, -450, 90);
        foundationServoOne.setPosition(FOUNDATION_ARM_IN_ONE);
        foundationServoTwo.setPosition(FOUNDATION_ARM_IN_TWO);
        sleep(100);

        gyroTurn(TURN_SPEED, 100);
        gyroHold(TURN_SPEED, 100, 0.5);

        //gyroTurn(TURN_SPEED, 80);
        //Drives backwards, turns, strafes
        gyroDrive(0.6, 950, 100);
        sleep(100);
        gyroTurn(0.5, 175);

        foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
        foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
        sleep(200);

        gyroTurn(TURN_SPEED, 180);
        //sleep(100);
        tapeMeasure.setPower(.8);
        sleep(1000);
        tapeMeasure.setPower(0);
        sleep(5000);


    }
```

```
        }
    }

    public void gyroDrive(double speed,
                          int distance,
                          double angle) {
        double max;
        double error;
        double steer;
        double leftSpeed;
        double rightSpeed;

        // Ensure that the opmode is still active
        if (opModeIsActive()) {

            // Determine new target position, and pass to motor controller
            wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

            wheelFrontRight.setTargetPosition(distance);
            wheelFrontLeft.setTargetPosition(distance);
            wheelRearRight.setTargetPosition(distance);
            wheelRearLeft.setTargetPosition(distance);

            wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            wheelFrontRight.setPower(speed);
            wheelFrontLeft.setPower(speed);
            wheelRearRight.setPower(speed);
            wheelRearLeft.setPower(speed);

            // keep looping while we are still active, and BOTH motors are running.
            while (opModeIsActive() &&
                    (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() &&
wheelRearRight.isBusy() && wheelRearLeft.isBusy())) {

                // adjust relative speed based on heading error.
                error = getError(angle);
                steer = getSteer(error, P_DRIVE_COEFF);

                // if driving in reverse, the motor correction also needs to be reversed
                if (distance < 0)
                    steer *= -1.0;

                leftSpeed = speed - steer;
                rightSpeed = speed + steer;

                // Normalize speeds if either one exceeds +/- 1.0;
                max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));
                if (max > 1.0) {
```

```
                leftSpeed /= max;
                rightSpeed /= max;
            }

            wheelFrontLeft.setPower(leftSpeed);
            wheelRearLeft.setPower(leftSpeed);
            wheelFrontRight.setPower(rightSpeed);
            wheelRearRight.setPower(rightSpeed);

            // Display drive status for the driver.
            telemetry.addData("Err/St", "%5.1f/%5.1f", error, steer);
            telemetry.addData("Speed", "%5.2f:%5.2f", leftSpeed, rightSpeed);
            telemetry.update();
        }

        // Stop all motion;
        wheelFrontRight.setPower(0);
        wheelRearRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearLeft.setPower(0);

        // Turn off RUN_TO_POSITION
        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    }
}

public void gyroDriveClaw2(double speed,
                           int distance,
                           double angle) {
    double max;
    double error;
    double steer;
    double leftSpeed;
    double rightSpeed;
    double remainingDistance;
    double positiveDistance;
    double positiveCurrentPosition;

    // Ensure that the opmode is still active
    if (opModeIsActive()) {

        // Determine new target position, and pass to motor controller
        wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        wheelFrontRight.setTargetPosition(distance);
        wheelFrontLeft.setTargetPosition(distance);
        wheelRearRight.setTargetPosition(distance);
        wheelRearLeft.setTargetPosition(distance);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
```

```
        wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        wheelFrontRight.setPower(speed);
        wheelFrontLeft.setPower(speed);
        wheelRearRight.setPower(speed);
        wheelRearLeft.setPower(speed);

        // keep looping while we are still active, and BOTH motors are running.
        while (opModeIsActive() &&
                (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() &&
wheelRearRight.isBusy() && wheelRearLeft.isBusy())) {

            // adjust relative speed based on heading error.
            error = getError(angle);
            steer = getSteer(error, P_DRIVE_COEFF);

            // if driving in reverse, the motor correction also needs to be reversed
            if (distance < 0)
                steer *= -1.0;

            leftSpeed = speed - steer;
            rightSpeed = speed + steer;

            // Normalize speeds if either one exceeds +/- 1.0;
            max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));
            if (max > 1.0) {
                leftSpeed /= max;
                rightSpeed /= max;
            }

            wheelFrontLeft.setPower(leftSpeed);
            wheelRearLeft.setPower(leftSpeed);
            wheelFrontRight.setPower(rightSpeed);
            wheelRearRight.setPower(rightSpeed);

            positiveDistance = distance * -1;
            positiveCurrentPosition = wheelFrontLeft.getCurrentPosition() * -1;

            remainingDistance = positiveDistance - positiveCurrentPosition;

            if (remainingDistance < 4300) {
                autoClawSpin.setPosition(AUTO_CLAW_SPIN_IN);
            } else {
                autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
            }

            // Display drive status for the driver.
            telemetry.addData("Err/St", "%5.1f/%5.1f", error, steer);
            telemetry.addData("Speed", "%5.2f:%5.2f", leftSpeed, rightSpeed);
            telemetry.update();
        }

        // Stop all motion;
        wheelFrontRight.setPower(0);
        wheelRearRight.setPower(0);
        wheelFrontLeft.setPower(0);
```

```
        wheelRearLeft.setPower(0);

        // Turn off RUN_TO_POSITION
        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    }
}

public void gyroTurnClaw(double speed, double angle) {

    autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);

    // keep looping while we are still active, and not on heading.
    while (opModeIsActive() && !onHeading(speed, angle, P_TURN_COEFF)) {
        // Update telemetry & Allow time for other processes to run.
        telemetry.update();
    }
}


public void gyroDriveClaw(double speed,
                          int distance,
                          double angle) {
    double max;
    double error;
    double steer;
    double leftSpeed;
    double rightSpeed;

    // Ensure that the opmode is still active
    if (opModeIsActive()) {

        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
        sleep(200);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
        // Determine new target position, and pass to motor controller
        wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        wheelFrontRight.setTargetPosition(distance);
        wheelFrontLeft.setTargetPosition(distance);
        wheelRearRight.setTargetPosition(distance);
        wheelRearLeft.setTargetPosition(distance);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        wheelFrontRight.setPower(speed);
        wheelFrontLeft.setPower(speed);
        wheelRearRight.setPower(speed);
        wheelRearLeft.setPower(speed);
```

```
            // keep looping while we are still active, and BOTH motors are running.
            while (opModeIsActive() &&
                    (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() &&
wheelRearRight.isBusy() && wheelRearLeft.isBusy())) {



                // adjust relative speed based on heading error.
                error = getError(angle);
                steer = getSteer(error, P_DRIVE_COEFF);

                // if driving in reverse, the motor correction also needs to be reversed
                if (distance < 0)
                    steer *= -1.0;

                leftSpeed = speed - steer;
                rightSpeed = speed + steer;

                // Normalize speeds if either one exceeds +/- 1.0;
                max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));
                if (max > 1.0) {
                    leftSpeed /= max;
                    rightSpeed /= max;
                }

                wheelFrontLeft.setPower(leftSpeed);
                wheelRearLeft.setPower(leftSpeed);
                wheelFrontRight.setPower(rightSpeed);
                wheelRearRight.setPower(rightSpeed);

                // Display drive status for the driver.
                telemetry.addData("Err/St", "%5.1f/%5.1f", error, steer);
                telemetry.addData("Speed", "%5.2f:%5.2f", leftSpeed, rightSpeed);
                telemetry.update();
            }

            autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);

            // Stop all motion;
            wheelFrontRight.setPower(0);
            wheelRearRight.setPower(0);
            wheelFrontLeft.setPower(0);
            wheelRearLeft.setPower(0);

            // Turn off RUN_TO_POSITION
            wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        }
    }


    public void gyroDriveClawBack(double speed,
                                  int distance,
                                  double angle) {
```

```
        double max;
        double error;
        double steer;
        double leftSpeed;
        double rightSpeed;
        double distanceLeft;
        double positiveDistance;
        double positiveCurrentPosition;



        // Ensure that the opmode is still active
        if (opModeIsActive()) {

            // Determine new target position, and pass to motor controller
            wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

            wheelFrontRight.setTargetPosition(distance);
            wheelFrontLeft.setTargetPosition(distance);
            wheelRearRight.setTargetPosition(distance);
            wheelRearLeft.setTargetPosition(distance);

            wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            wheelFrontRight.setPower(speed);
            wheelFrontLeft.setPower(speed);
            wheelRearRight.setPower(speed);
            wheelRearLeft.setPower(speed);

            // keep looping while we are still active, and BOTH motors are running.
            while (opModeIsActive() &&
                    (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() &&
wheelRearRight.isBusy() && wheelRearLeft.isBusy())) {


                // adjust relative speed based on heading error.
                error = getError(angle);
                steer = getSteer(error, P_DRIVE_COEFF);

                // if driving in reverse, the motor correction also needs to be reversed
                if (distance < 0)
                    steer *= -1.0;

                leftSpeed = speed - steer;
                rightSpeed = speed + steer;

                // Normalize speeds if either one exceeds +/- 1.0;
                max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));
                if (max > 1.0) {
                    leftSpeed /= max;
                    rightSpeed /= max;
```

```
                }

                wheelFrontLeft.setPower(leftSpeed);
                wheelRearLeft.setPower(leftSpeed);
                wheelFrontRight.setPower(rightSpeed);
                wheelRearRight.setPower(rightSpeed);

                //positiveDistance = distance * -1;
                //positiveCurrentPosition = wheelFrontLeft.getCurrentPosition() * -1;

                distanceLeft = distance - wheelFrontLeft.getCurrentPosition();

                if (distanceLeft > -1500) {
                    autoClawSpin.setPosition(AUTO_CLAW_SPIN_IN);
                } else {
                    autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_LEFT);
                }

                telemetry.addData("WheelCurrentPositiin",
wheelFrontLeft.getCurrentPosition());
                telemetry.addData("DistanceLeft", -5270 -
wheelFrontLeft.getCurrentPosition());
                telemetry.update();

            }

            // Stop all motion;
            wheelFrontRight.setPower(0);
            wheelRearRight.setPower(0);
            wheelFrontLeft.setPower(0);
            wheelRearLeft.setPower(0);

            // Turn off RUN_TO_POSITION
            wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        }
    }

    public void gyroDrive2(double speed,
                           int distance,
                           double angle) {
        double max;
        double error;
        double steer;
        double leftSpeed;
        double rightSpeed;
        double distanceLeft;



        // Ensure that the opmode is still active
        if (opModeIsActive()) {

            // Determine new target position, and pass to motor controller
            wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
```

```
        wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        wheelFrontRight.setTargetPosition(distance);
        wheelFrontLeft.setTargetPosition(distance);
        wheelRearRight.setTargetPosition(distance);
        wheelRearLeft.setTargetPosition(distance);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        wheelFrontRight.setPower(speed);
        wheelFrontLeft.setPower(speed);
        wheelRearRight.setPower(speed);
        wheelRearLeft.setPower(speed);

        // keep looping while we are still active, and BOTH motors are running.
        while (opModeIsActive() &&
                (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() &&
wheelRearRight.isBusy() && wheelRearLeft.isBusy())) {


            // adjust relative speed based on heading error.
            error = getError(angle);
            steer = getSteer(error, P_DRIVE_COEFF);

            // if driving in reverse, the motor correction also needs to be reversed
            if (distance < 0)
                steer *= -1.0;

            leftSpeed = speed - steer;
            rightSpeed = speed + steer;

            // Normalize speeds if either one exceeds +/- 1.0;
            max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));
            if (max > 1.0) {
                leftSpeed /= max;
                rightSpeed /= max;
            }

            wheelFrontLeft.setPower(leftSpeed);
            wheelRearLeft.setPower(leftSpeed);
            wheelFrontRight.setPower(rightSpeed);
            wheelRearRight.setPower(rightSpeed);

            //positiveDistance = distance * -1;
            //positiveCurrentPosition = wheelFrontLeft.getCurrentPosition() * -1;

            distanceLeft = distance - wheelFrontLeft.getCurrentPosition();

            if (distanceLeft < 4100) {
                autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_LEFT);
                autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
            } else {
```

```
                autoClawSpin.setPosition(AUTO_CLAW_SPIN_IN);
                autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UPUP);
            }

            telemetry.addData("WheelCurrentPositiin",
wheelFrontLeft.getCurrentPosition());
            telemetry.addData("DistanceLeft", -5270 -
wheelFrontLeft.getCurrentPosition());
            telemetry.update();

        }

        // Stop all motion;
        wheelFrontRight.setPower(0);
        wheelRearRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearLeft.setPower(0);

        // Turn off RUN_TO_POSITION
        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    }
}


   public void Drive (int frontRightDistance, int frontLeftDistance, int rearRightDistance,
int rearLeftDistance, double speed){


    autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
    autoClawSpin.setPosition(AUTO_CLAW_SPIN_IN);

    wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

    wheelFrontRight.setTargetPosition(frontRightDistance);
    wheelFrontLeft.setTargetPosition(frontLeftDistance);
    wheelRearRight.setTargetPosition(rearRightDistance);
    wheelRearLeft.setTargetPosition(rearLeftDistance);

    wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

    autoClawSpin.setPosition(AUTO_CLAW_SPIN_IN);
    autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);

    wheelFrontRight.setPower(speed);
    wheelFrontLeft.setPower(speed);
    wheelRearRight.setPower(speed);
    wheelRearLeft.setPower(speed);
```

```
        while (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() && wheelRearRight.isBusy()
&& wheelRearLeft.isBusy() && opModeIsActive()) {



        }

        wheelFrontRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearRight.setPower(0);
        wheelRearLeft.setPower(0);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(100);
    }

    public void DriveStrafe (int frontRightDistance, int frontLeftDistance, int
rearRightDistance, int rearLeftDistance, double speed){


        wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        wheelFrontRight.setTargetPosition(frontRightDistance);
        wheelFrontLeft.setTargetPosition(frontLeftDistance);
        wheelRearRight.setTargetPosition(rearRightDistance);
        wheelRearLeft.setTargetPosition(rearLeftDistance);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        wheelFrontRight.setPower(speed);
        wheelFrontLeft.setPower(speed);
        wheelRearRight.setPower(speed);
        wheelRearLeft.setPower(speed);


        while (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() && wheelRearRight.isBusy()
&& wheelRearLeft.isBusy() && opModeIsActive()) {



        }

        wheelFrontRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearRight.setPower(0);
        wheelRearLeft.setPower(0);
```

```
        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(100);
    }

    public void gyroTurn(double speed, double angle) {

        // keep looping while we are still active, and not on heading.
        while (opModeIsActive() && !onHeading(speed, angle, P_TURN_COEFF)) {
            // Update telemetry & Allow time for other processes to run.
            telemetry.update();
        }
    }


    public void gyroHold(double speed, double angle, double holdTime) {

        ElapsedTime holdTimer = new ElapsedTime();

        // keep looping while we have time remaining.
        holdTimer.reset();
        while (opModeIsActive() && (holdTimer.time() < holdTime)) {
            // Update telemetry & Allow time for other processes to run.
            onHeading(speed, angle, P_TURN_COEFF);
            telemetry.update();
        }

        // Stop all motion;
        wheelFrontRight.setPower(0);
        wheelRearRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearLeft.setPower(0);

    }

    public void gyroHoldClaw(double speed, double angle, double holdTime) {

        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
        ElapsedTime holdTimer = new ElapsedTime();

        // keep looping while we have time remaining.
        holdTimer.reset();
        while (opModeIsActive() && (holdTimer.time() < holdTime)) {
            // Update telemetry & Allow time for other processes to run.
            onHeading(speed, angle, P_TURN_COEFF);
            telemetry.update();
        }

        // Stop all motion;
        wheelFrontRight.setPower(0);
        wheelRearRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearLeft.setPower(0);
```

```
    }


    boolean onHeading(double speed, double angle, double PCoeff) {
        double error;
        double steer;
        boolean onTarget = false;
        double leftSpeed;
        double rightSpeed;

        // determine turn power based on +/- error
        error = getError(angle);

        if (Math.abs(error) <= HEADING_THRESHOLD) {
            steer = 0.0;
            leftSpeed = 0.0;
            rightSpeed = 0.0;
            onTarget = true;
        } else {
            steer = getSteer(error, PCoeff);
            rightSpeed = speed * steer;
            leftSpeed = -rightSpeed;
        }

        // Send desired speeds to motors.
        wheelFrontLeft.setPower(leftSpeed);
        wheelRearLeft.setPower(leftSpeed);
        wheelFrontRight.setPower(rightSpeed);
        wheelRearRight.setPower(rightSpeed);

        // Display it for the driver.
        telemetry.addData("Target", "%5.2f", angle);
        telemetry.addData("Err/St", "%5.2f/%5.2f", error, steer);
        telemetry.addData("Speed.", "%5.2f:%5.2f", leftSpeed, rightSpeed);

        return onTarget;
    }

    public double getError(double targetAngle) {

        double robotError;
        angles = imu.getAngularOrientation(AxesReference.INTRINSIC, AxesOrder.ZYX,
AngleUnit.DEGREES);
        this.imu.getPosition();
        // calculate error in -179 to +180 range  (
        robotError = targetAngle - angles.firstAngle;
        while (robotError > 180) robotError -= 360;
        while (robotError <= -180) robotError += 360;
        return robotError;
    }


    public double getSteer(double error, double PCoeff) {
        return Range.clip(error * PCoeff, -1, 1);
    }
```

```
    //detection pipeline
    static class StageSwitchingPipeline extends OpenCvPipeline
    {
        Mat yCbCrChan2Mat = new Mat();
        Mat thresholdMat = new Mat();
        Mat all = new Mat();
        List<MatOfPoint> contoursList = new ArrayList<>();

        enum Stage
        {//color difference. greyscale
            detection,//includes outlines
            THRESHOLD,//b&w
            RAW_IMAGE,//displays raw view
        }

        private Stage stageToRenderToViewport = Stage.detection;
        private Stage[] stages = Stage.values();

        @Override
        public void onViewportTapped()
        {
            /*
             * Note that this method is invoked from the UI thread
             * so whatever we do here, we must do quickly.
             */

            int currentStageNum = stageToRenderToViewport.ordinal();

            int nextStageNum = currentStageNum + 1;

            if(nextStageNum >= stages.length)
            {
                nextStageNum = 0;
            }

            stageToRenderToViewport = stages[nextStageNum];
        }

        @Override
        public Mat processFrame(Mat input)
        {
            contoursList.clear();
            /*
             * This pipeline finds the contours of yellow blobs such as the Gold Mineral
             * from the Rover Ruckus game.
             */

            //color diff cb.
            //lower cb = more blue = skystone = white
            //higher cb = less blue = yellow stone = grey
            Imgproc.cvtColor(input, yCbCrChan2Mat, Imgproc.COLOR_RGB2YCrCb);//converts rgb
to ycrcb
            Core.extractChannel(yCbCrChan2Mat, yCbCrChan2Mat, 2);//takes cb difference and
stores

            //b&w
```

```
            Imgproc.threshold(yCbCrChan2Mat, thresholdMat, 102, 255,
Imgproc.THRESH_BINARY_INV);

            //outline/contour
            Imgproc.findContours(thresholdMat, contoursList, new Mat(), Imgproc.RETR_LIST,
Imgproc.CHAIN_APPROX_SIMPLE);
            yCbCrChan2Mat.copyTo(all);//copies mat object
            //Imgproc.drawContours(all, contoursList, -1, new Scalar(255, 0, 0), 3,
8);//draws blue contours


            //get values from frame
            double[] pixMid = thresholdMat.get((int)(input.rows()* midPos[1]),
(int)(input.cols()* midPos[0]));//gets value at circle
            valMid = (int)pixMid[0];

            double[] pixLeft = thresholdMat.get((int)(input.rows()* leftPos[1]),
(int)(input.cols()* leftPos[0]));//gets value at circle
            valLeft = (int)pixLeft[0];

            double[] pixRight = thresholdMat.get((int)(input.rows()* rightPos[1]),
(int)(input.cols()* rightPos[0]));//gets value at circle
            valRight = (int)pixRight[0];

            //create three points
            Point pointMid = new Point((int)(input.cols()* midPos[0]), (int)(input.rows()*
midPos[1]));
            Point pointLeft = new Point((int)(input.cols()* leftPos[0]), (int)(input.rows()*
leftPos[1]));
            Point pointRight = new Point((int)(input.cols()* rightPos[0]),
(int)(input.rows()* rightPos[1]));

            //draw circles on those points
            Imgproc.circle(all, pointMid,5, new Scalar( 255, 0, 0 ),1 );//draws circle
            Imgproc.circle(all, pointLeft,5, new Scalar( 255, 0, 0 ),1 );//draws circle
            Imgproc.circle(all, pointRight,5, new Scalar( 255, 0, 0 ),1 );//draws circle

            //draw 3 rectangles
            Imgproc.rectangle(//1-3
                    all,
                    new Point(
                            input.cols()*(leftPos[0]-rectWidth/2),
                            input.rows()*(leftPos[1]-rectHeight/2)),
                    new Point(
                            input.cols()*(leftPos[0]+rectWidth/2),
                            input.rows()*(leftPos[1]+rectHeight/2)),
                    new Scalar(0, 255, 0), 3);
            Imgproc.rectangle(//3-5
                    all,
                    new Point(
                            input.cols()*(midPos[0]-rectWidth/2),
                            input.rows()*(midPos[1]-rectHeight/2)),
                    new Point(
                            input.cols()*(midPos[0]+rectWidth/2),
                            input.rows()*(midPos[1]+rectHeight/2)),
                    new Scalar(0, 255, 0), 3);
            Imgproc.rectangle(//5-7
```

```
                all,
                new Point(
                        input.cols()*(rightPos[0]-rectWidth/2),
                        input.rows()*(rightPos[1]-rectHeight/2)),
                new Point(
                        input.cols()*(rightPos[0]+rectWidth/2),
                        input.rows()*(rightPos[1]+rectHeight/2)),
                new Scalar(0, 255, 0), 3);

        switch (stageToRenderToViewport)
        {
            case THRESHOLD:
            {
                return thresholdMat;
            }

            case detection:
            {
                return all;
            }

            case RAW_IMAGE:
            {
                return input;
            }

            default:
            {
                return input;
            }
        }
    }

}
}
```

FULL CODE: MOVE FOUNDATION, PARK

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

import com.qualcomm.hardware.bosch.BNO055IMU;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.util.Range;

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.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.MatOfPoint2f;
import org.opencv.core.Point;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.imgproc.Imgproc;
import org.openftc.easyopencv.OpenCvCamera;
import org.openftc.easyopencv.OpenCvCameraRotation;
import org.openftc.easyopencv.OpenCvInternalCamera;
import org.openftc.easyopencv.OpenCvPipeline;

import java.util.ArrayList;
import java.util.List;


/**
* Created by maryjaneb  on 11/13/2016.
*
* nerverest ticks
* 60 1680
* 40 1120
* 20 560
*
* monitor: 640 x 480
*YES
*/
@Autonomous(name= "Red: skystone")
//@Disabled
public class SkystoneRed extends LinearOpMode {
   private ElapsedTime runtime = new ElapsedTime();

   private static int valMid = -1;
   private static int valLeft = -1;
   private static int valRight = -1;

   private static float rectHeight = .6f/8f;
   private static float rectWidth = 1.5f/8f;
```

```
   private static float offsetX = 1f/10f;//changing this moves the three rects and the
three circles left or right, range : (-2, 2) not inclusive
   private static float offsetY = 0.5f/8f;//changing this moves the three rects and
circles up or down, range: (-4, 4) not inclusive

   private static float[] midPos = {4f/8f+offsetX, 4f/8f+offsetY};//0 = col, 1 = row
   private static float[] leftPos = {2f/8f+offsetX, 4f/8f+offsetY};
   private static float[] rightPos = {6f/8f+offsetX, 4f/8f+offsetY};

   //moves all rectangles right or left by amount. units are in ratio to monitor
   private final int rows = 640;
   private final int cols = 480;


   static final double DRIVE_SPEED = .8;     // Nominal speed for better accuracy.
   static final double TURN_SPEED = 0.6;     // Nominal half speed for better accuracy.

   static final double HEADING_THRESHOLD = 1;      // As tight as we can make it with an
integer gyro
   static final double P_TURN_COEFF = 0.05;     // Larger is more responsive, but also
less stable
   static final double P_DRIVE_COEFF = 0.05;     // Larger is more responsive, but also
less stable

   private static final double AUTO_CLAW_SPIN_IN = .50;
   private static final double AUTO_CLAW_SPIN_OUT_LEFT = 1;
   private static final double AUTO_CLAW_SPIN_OUT_RIGHT = .06;
   private static final double AUTO_CLAW_SPIN_LETGO = .59;
   private static final double AUTO_CLAW_ROTATE_UPUP = .54;
   private static final double AUTO_CLAW_ROTATE_INIT = 1;
   private static final double AUTO_CLAW_ROTATE_DOWN = .21;
   private static final double AUTO_CLAW_ROTATE_UP = .38;
   private static final double AUTO_CLAW_CLAMP_INIT = .31;
   private static final double AUTO_CLAW_CLAMP_DOWN = .16;
   private static final double AUTO_CLAW_CLAMP_UP = .70;
   private static final double AUTO_CLAW_CLAMP_LETGO = .33;

   private static final double FOUNDATION_ARM_OUT_ONE = .84;
   private static final double FOUNDATION_ARM_IN_ONE = .52;

   private static final double FOUNDATION_ARM_OUT_TWO = 0;
   private static final double FOUNDATION_ARM_IN_TWO = .30;

   private DcMotor wheelFrontRight;
   private DcMotor wheelFrontLeft;
   private DcMotor wheelRearRight;
   private DcMotor wheelRearLeft;
   private Servo foundationServoOne;
   private Servo foundationServoTwo;
   private Servo autoClawRotate;
   private Servo autoClawClamp;
   private Servo autoClawSpin;
   private DcMotor tapeMeasure;

   BNO055IMU imu;                // Additional Gyro device
   Orientation angles;
```

```
    OpenCvCamera phoneCam;


    @Override
    public void runOpMode() throws InterruptedException {

        int cameraMonitorViewId =
hardwareMap.appContext.getResources().getIdentifier("cameraMonitorViewId", "id",
hardwareMap.appContext.getPackageName());
        phoneCam = new OpenCvInternalCamera(OpenCvInternalCamera.CameraDirection.BACK,
cameraMonitorViewId);
        phoneCam.openCameraDevice();//open camera
        phoneCam.setPipeline(new StageSwitchingPipeline());//different stages
        phoneCam.startStreaming(rows, cols, OpenCvCameraRotation.SIDEWAYS_LEFT);//display
on RC
        //width, height
        //width = height in this case, because camera is in portrait mode.

        wheelFrontRight = hardwareMap.get(DcMotor.class, "wheelFrontRight");
        wheelFrontLeft = hardwareMap.get(DcMotor.class, "wheelFrontLeft");
        wheelRearRight = hardwareMap.get(DcMotor.class, "wheelRearRight");
        wheelRearLeft = hardwareMap.get(DcMotor.class, "wheelRearLeft");

        autoClawClamp = hardwareMap.get(Servo.class, "autoClawClamp");
        autoClawRotate = hardwareMap.get(Servo.class, "autoClawRotate");
        autoClawSpin = hardwareMap.get(Servo.class, "autoClawSpin");

        tapeMeasure = hardwareMap.get(DcMotor.class, "tapeMeasure");

        wheelFrontRight.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        wheelFrontLeft.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        wheelRearRight.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        wheelRearLeft.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);


        wheelFrontRight.setDirection(DcMotor.Direction.REVERSE);
        wheelRearRight.setDirection(DcMotor.Direction.REVERSE);

        //Setting motor mode regarding encoders
        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        foundationServoOne = hardwareMap.get(Servo.class, "foundationServoOne");
        foundationServoTwo = hardwareMap.get(Servo.class, "foundationServoTwo");

        BNO055IMU.Parameters parameters = new BNO055IMU.Parameters();
        parameters.mode = BNO055IMU.SensorMode.IMU;
        parameters.angleUnit = BNO055IMU.AngleUnit.DEGREES;
        parameters.calibrationDataFile = "BNO055IMUCalibration.json";

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

        // Send telemetry message to alert driver that we are calibrating;
```

```
telemetry.addData("Mode", "calibrating...");
telemetry.update();

sleep(300);

telemetry.addData("Mode", "waiting for start");
telemetry.addData("imu calib status", imu.getCalibrationStatus().toString());
telemetry.update();

waitForStart();
runtime.reset();

while (opModeIsActive()) {
    telemetry.addData("Values", valLeft+"   "+valMid+"   "+valRight);
    telemetry.addData("Height", rows);
    telemetry.addData("Width", cols);

    telemetry.update();
    sleep(100);

    if (valLeft == 0 && valMid == 255 && valRight == 255) {

        //Rotates ClawRotate Down
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
        sleep(500);

        //Strafes over to first stone, clamp up, spin in
        Drive(-1230, 1230,1230, -1230, 0.5);

        //Corrects heading and clamps stone
        gyroTurn(TURN_SPEED, 2);
        sleep(100);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
        gyroHold(TURN_SPEED, 2, 0.03);
        sleep(700);


        //Drives towards foundation, rotates up, spins out, spins in
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UPUP);
        sleep(200);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
        sleep(200);
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
        sleep(200);
        gyroDrive(0.8, -4250, 2);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
        sleep(100);

        //Corrects heading, drops stone
        gyroTurn(TURN_SPEED, 1.5);
        sleep(400);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
        gyroHold(TURN_SPEED, 1.5, 0.03);
        sleep(300);

        //Drives back to second stone, spins out, rotates down, spins in
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
```

```
sleep(200);
autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
sleep(100);
gyroDriveClawBack(0.8, 5270, 1.5);

//Corrects heading and clamps stone
gyroTurn(TURN_SPEED, 2);
sleep(400);
autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
gyroHold(TURN_SPEED, 2, 0.03);
sleep(700);

//Drives back to foundation
autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
sleep(150);
//autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
gyroDrive2(0.8, -4650, 2);
autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
sleep(100);

//Lets go
gyroTurn(TURN_SPEED, 1.5);
sleep(400);
autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
gyroHold(TURN_SPEED, 1.5, 0.03);
sleep(300);

//Turns to move foundation
foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
gyroTurn(0.5, 90);
gyroHold(0.5, 90, 0.5);

//Drives forwards
DriveStrafe(-450, -450, -450, -450, 0.2);
//gyroDrive(0.2, -450, 90);
foundationServoOne.setPosition(FOUNDATION_ARM_IN_ONE);
foundationServoTwo.setPosition(FOUNDATION_ARM_IN_TWO);
sleep(100);

gyroTurn(TURN_SPEED, 80);
gyroHold(TURN_SPEED, 80, 0.5);

//gyroTurn(TURN_SPEED, 80);
//Drives backwards, turns, strafes
gyroDrive(0.6, 1100, 80);
sleep(100);
gyroTurn(0.5, 5);

foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
sleep(200);

gyroTurn(TURN_SPEED, -20);
gyroHold(TURN_SPEED, -20, 0.5);
//sleep(100);
tapeMeasure.setPower(.8);
```

```
        sleep(1000);
        tapeMeasure.setPower(0);



    }

    sleep(200);

    if (valMid == 0 && valRight == 255 && valLeft == 255) {
        telemetry.addData("Status", "valMid has been detected");
        telemetry.update();
        sleep(100);

        //Rotates ClawRotate Down
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
        sleep(500);

        //Strafes over to first stone, clamp up, spin in
        Drive(-1255, 1255,1255, -1255, 0.5);
        sleep(100);

        gyroDrive(0.5, -370, 0);

        //Corrects heading and clamps stone
        gyroTurn(TURN_SPEED, 2);
        sleep(100);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
        gyroHold(TURN_SPEED, 2, 0.03);
        sleep(700);


        //Drives towards foundation, rotates up, spins out, spins in
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UPUP);
        sleep(200);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
        sleep(200);
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
        sleep(200);
        gyroDrive(0.8, -3600, 2);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
        sleep(100);

        //Corrects heading, drops stone
        gyroTurn(TURN_SPEED, 0.6);
        sleep(500);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
        gyroHold(TURN_SPEED, 0.6, 0.03);
        sleep(300);

        //Drives back to second stone, spins out, rotates down, spins in
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
        sleep(200);
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
        sleep(100);
        gyroDriveClawBack(0.6, 4700, 0.7);
```

```
//Corrects heading and clamps stone
gyroTurn(TURN_SPEED, 0);
sleep(400);
autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
gyroHold(TURN_SPEED, 0, 0.03);
sleep(700);

//Drives back to foundation
autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
sleep(150);
autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
gyroDrive(0.8, -4400, 0);
autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
sleep(100);

//Lets go
gyroTurn(TURN_SPEED, 1.5);
sleep(500);
autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
gyroHold(TURN_SPEED, 1.5, 0.03);
sleep(300);

//Turns to move foundation
foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
gyroTurn(0.5, 90);
gyroHold(0.5, 90, 0.5);

//Drives forwards
DriveStrafe(-450, -450, -450, -450, 0.2);
//gyroDrive(0.2, -450, 90);
foundationServoOne.setPosition(FOUNDATION_ARM_IN_ONE);
foundationServoTwo.setPosition(FOUNDATION_ARM_IN_TWO);
sleep(100);

gyroTurn(TURN_SPEED, 80);
gyroHold(TURN_SPEED, 80, 0.5);

//gyroTurn(TURN_SPEED, 80);
//Drives backwards, turns, strafes
gyroDrive(0.6, 1000, 80);
sleep(100);
gyroTurn(0.5, 5);

foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
sleep(200);

gyroTurn(TURN_SPEED, -20);
gyroHold(TURN_SPEED, -20, 0.5);
//sleep(100);

tapeMeasure.setPower(.8);
sleep(1000);
tapeMeasure.setPower(0);
```

```
    }

    sleep(200);

    if (valRight == 0 && valMid == 255 && valLeft == 255) {
        telemetry.addData("Status", "valRight has been detected");
        telemetry.update();
        sleep(100);

        //Rotates ClawRotate Down
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
        sleep(500);

        //Strafes over to first stone, clamp up, spin in
        Drive(-1230, 1230,1230, -1230, 0.5);
        sleep(100);

        gyroDrive(0.5, -740, 0);

        //Corrects heading and clamps stone
        gyroTurn(TURN_SPEED, 2);
        sleep(100);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
        gyroHold(TURN_SPEED, 2, 0.03);
        sleep(700);


        //Drives towards foundation, rotates up, spins out, spins in
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UPUP);
        sleep(200);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
        sleep(200);
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
        sleep(200);
        gyroDrive(0.8, -3250, 2);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
        sleep(100);

        //Corrects heading, drops stone
        gyroTurn(TURN_SPEED, 0.7);
        sleep(500);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
        gyroHold(TURN_SPEED, 0.7, 0.03);
        sleep(300);

        //Drives back to second stone, spins out, rotates down, spins in
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
        sleep(200);
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
        sleep(100);
        gyroDriveClawBack(0.8, 4320, 0.7);

        //Corrects heading and clamps stone
        gyroTurn(TURN_SPEED, 0);
        sleep(400);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
        gyroHold(TURN_SPEED, 0, 0.03);
```

```
            sleep(700);

            //Drives back to foundation
            autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
            sleep(150);
            autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
            gyroDrive(0.8, -3900, 0);
            autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
            sleep(100);

            //Lets go
            gyroTurn(TURN_SPEED, 1.5);
            sleep(500);
            autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
            gyroHold(TURN_SPEED, 1.5, 0.03);
            sleep(300);

            //Turns to move foundation
            foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
            foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
            gyroTurn(0.5, 90);
            gyroHold(0.5, 90, 0.5);

            //Drives forwards
            DriveStrafe(-450, -450, -450, -450, 0.2);
            //gyroDrive(0.2, -450, 90);
            foundationServoOne.setPosition(FOUNDATION_ARM_IN_ONE);
            foundationServoTwo.setPosition(FOUNDATION_ARM_IN_TWO);
            sleep(100);

            gyroTurn(TURN_SPEED, 80);
            gyroHold(TURN_SPEED, 80, 0.5);

            //gyroTurn(TURN_SPEED, 80);
            //Drives backwards, turns, strafes
            gyroDrive(0.6, 1000, 80);
            sleep(100);
            gyroTurn(0.5, 5);

            foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
            foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
            sleep(200);

            gyroTurn(TURN_SPEED, -20);
            gyroHold(TURN_SPEED, -20, 0.5);
            //sleep(100);

            tapeMeasure.setPower(.8);
            sleep(1000);
            tapeMeasure.setPower(0);


        }


    }
}
```

```
    public void gyroDrive(double speed,
                          int distance,
                          double angle) {
        double max;
        double error;
        double steer;
        double leftSpeed;
        double rightSpeed;

        // Ensure that the opmode is still active
        if (opModeIsActive()) {

            // Determine new target position, and pass to motor controller
            wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

            wheelFrontRight.setTargetPosition(distance);
            wheelFrontLeft.setTargetPosition(distance);
            wheelRearRight.setTargetPosition(distance);
            wheelRearLeft.setTargetPosition(distance);

            wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            wheelFrontRight.setPower(speed);
            wheelFrontLeft.setPower(speed);
            wheelRearRight.setPower(speed);
            wheelRearLeft.setPower(speed);

            // keep looping while we are still active, and BOTH motors are running.
            while (opModeIsActive() &&
                    (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() &&
wheelRearRight.isBusy() && wheelRearLeft.isBusy())) {

                // adjust relative speed based on heading error.
                error = getError(angle);
                steer = getSteer(error, P_DRIVE_COEFF);

                // if driving in reverse, the motor correction also needs to be reversed
                if (distance < 0)
                    steer *= -1.0;

                leftSpeed = speed - steer;
                rightSpeed = speed + steer;

                // Normalize speeds if either one exceeds +/- 1.0;
                max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));
                if (max > 1.0) {
                    leftSpeed /= max;
                    rightSpeed /= max;
                }
```

```
            wheelFrontLeft.setPower(leftSpeed);
            wheelRearLeft.setPower(leftSpeed);
            wheelFrontRight.setPower(rightSpeed);
            wheelRearRight.setPower(rightSpeed);

            // Display drive status for the driver.
            telemetry.addData("Err/St", "%5.1f/%5.1f", error, steer);
            telemetry.addData("Speed", "%5.2f:%5.2f", leftSpeed, rightSpeed);
            telemetry.update();
        }

        // Stop all motion;
        wheelFrontRight.setPower(0);
        wheelRearRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearLeft.setPower(0);

        // Turn off RUN_TO_POSITION
        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    }
}

public void gyroDrive2(double speed,
                       int distance,
                       double angle) {
    double max;
    double error;
    double steer;
    double leftSpeed;
    double rightSpeed;
    double distanceLeft;



    // Ensure that the opmode is still active
    if (opModeIsActive()) {

        // Determine new target position, and pass to motor controller
        wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        wheelFrontRight.setTargetPosition(distance);
        wheelFrontLeft.setTargetPosition(distance);
        wheelRearRight.setTargetPosition(distance);
        wheelRearLeft.setTargetPosition(distance);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        wheelFrontRight.setPower(speed);
```

```
            wheelFrontLeft.setPower(speed);
            wheelRearRight.setPower(speed);
            wheelRearLeft.setPower(speed);

            // keep looping while we are still active, and BOTH motors are running.
            while (opModeIsActive() &&
                    (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() &&
wheelRearRight.isBusy() && wheelRearLeft.isBusy())) {


                // adjust relative speed based on heading error.
                error = getError(angle);
                steer = getSteer(error, P_DRIVE_COEFF);

                // if driving in reverse, the motor correction also needs to be reversed
                if (distance < 0)
                    steer *= -1.0;

                leftSpeed = speed - steer;
                rightSpeed = speed + steer;

                // Normalize speeds if either one exceeds +/- 1.0;
                max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));
                if (max > 1.0) {
                    leftSpeed /= max;
                    rightSpeed /= max;
                }

                wheelFrontLeft.setPower(leftSpeed);
                wheelRearLeft.setPower(leftSpeed);
                wheelFrontRight.setPower(rightSpeed);
                wheelRearRight.setPower(rightSpeed);

                //positiveDistance = distance * -1;
                //positiveCurrentPosition = wheelFrontLeft.getCurrentPosition() * -1;

                distanceLeft = distance - wheelFrontLeft.getCurrentPosition();

                if (distanceLeft > -4100) {
                    autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
                    autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
                } else {
                    autoClawSpin.setPosition(AUTO_CLAW_SPIN_IN);
                    autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UPUP);
                }

                telemetry.addData("WheelCurrentPositiin",
wheelFrontLeft.getCurrentPosition());
                telemetry.addData("DistanceLeft", -5270 -
wheelFrontLeft.getCurrentPosition());
                telemetry.update();

            }

            // Stop all motion;
            wheelFrontRight.setPower(0);
            wheelRearRight.setPower(0);
```

```
            wheelFrontLeft.setPower(0);
            wheelRearLeft.setPower(0);

            // Turn off RUN_TO_POSITION
            wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        }
    }

    public void gyroDriveClaw2(double speed,
                         int distance,
                         double angle) {
        double max;
        double error;
        double steer;
        double leftSpeed;
        double rightSpeed;
        double remainingDistance;
        double positiveDistance;
        double positiveCurrentPosition;

        // Ensure that the opmode is still active
        if (opModeIsActive()) {

            // Determine new target position, and pass to motor controller
            wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

            wheelFrontRight.setTargetPosition(distance);
            wheelFrontLeft.setTargetPosition(distance);
            wheelRearRight.setTargetPosition(distance);
            wheelRearLeft.setTargetPosition(distance);

            wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            wheelFrontRight.setPower(speed);
            wheelFrontLeft.setPower(speed);
            wheelRearRight.setPower(speed);
            wheelRearLeft.setPower(speed);

            // keep looping while we are still active, and BOTH motors are running.
            while (opModeIsActive() &&
                   (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() &&
wheelRearRight.isBusy() && wheelRearLeft.isBusy())) {

                // adjust relative speed based on heading error.
                error = getError(angle);
                steer = getSteer(error, P_DRIVE_COEFF);

                // if driving in reverse, the motor correction also needs to be reversed
```

```
            if (distance < 0)
                steer *= -1.0;

            leftSpeed = speed - steer;
            rightSpeed = speed + steer;

            // Normalize speeds if either one exceeds +/- 1.0;
            max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));
            if (max > 1.0) {
                leftSpeed /= max;
                rightSpeed /= max;
            }

            wheelFrontLeft.setPower(leftSpeed);
            wheelRearLeft.setPower(leftSpeed);
            wheelFrontRight.setPower(rightSpeed);
            wheelRearRight.setPower(rightSpeed);

            positiveDistance = distance * -1;
            positiveCurrentPosition = wheelFrontLeft.getCurrentPosition() * -1;

            remainingDistance = positiveDistance - positiveCurrentPosition;

            if (remainingDistance < 4300) {
                autoClawSpin.setPosition(AUTO_CLAW_SPIN_IN);
            } else {
                autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
            }

            // Display drive status for the driver.
            telemetry.addData("Err/St", "%5.1f/%5.1f", error, steer);
            telemetry.addData("Speed", "%5.2f:%5.2f", leftSpeed, rightSpeed);
            telemetry.update();
        }

        // Stop all motion;
        wheelFrontRight.setPower(0);
        wheelRearRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearLeft.setPower(0);

        // Turn off RUN_TO_POSITION
        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    }
}

public void gyroTurnClaw(double speed, double angle) {

    autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);

    // keep looping while we are still active, and not on heading.
    while (opModeIsActive() && !onHeading(speed, angle, P_TURN_COEFF)) {
        // Update telemetry & Allow time for other processes to run.
        telemetry.update();
```

```
        }
    }


    public void gyroDriveClaw(double speed,
                              int distance,
                              double angle) {
        double max;
        double error;
        double steer;
        double leftSpeed;
        double rightSpeed;

        // Ensure that the opmode is still active
        if (opModeIsActive()) {

            autoClawRotate.setPosition(AUTO_CLAW_ROTATE_UP);
            sleep(200);
            autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
            // Determine new target position, and pass to motor controller
            wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
            wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

            wheelFrontRight.setTargetPosition(distance);
            wheelFrontLeft.setTargetPosition(distance);
            wheelRearRight.setTargetPosition(distance);
            wheelRearLeft.setTargetPosition(distance);

            wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            wheelFrontRight.setPower(speed);
            wheelFrontLeft.setPower(speed);
            wheelRearRight.setPower(speed);
            wheelRearLeft.setPower(speed);

            // keep looping while we are still active, and BOTH motors are running.
            while (opModeIsActive() &&
                    (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() &&
wheelRearRight.isBusy() && wheelRearLeft.isBusy())) {



                // adjust relative speed based on heading error.
                error = getError(angle);
                steer = getSteer(error, P_DRIVE_COEFF);

                // if driving in reverse, the motor correction also needs to be reversed
                if (distance < 0)
                    steer *= -1.0;

                leftSpeed = speed - steer;
                rightSpeed = speed + steer;
```

```
            // Normalize speeds if either one exceeds +/- 1.0;
            max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));
            if (max > 1.0) {
                leftSpeed /= max;
                rightSpeed /= max;
            }

            wheelFrontLeft.setPower(leftSpeed);
            wheelRearLeft.setPower(leftSpeed);
            wheelFrontRight.setPower(rightSpeed);
            wheelRearRight.setPower(rightSpeed);

            // Display drive status for the driver.
            telemetry.addData("Err/St", "%5.1f/%5.1f", error, steer);
            telemetry.addData("Speed", "%5.2f:%5.2f", leftSpeed, rightSpeed);
            telemetry.update();
        }

        autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);

        // Stop all motion;
        wheelFrontRight.setPower(0);
        wheelRearRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearLeft.setPower(0);

        // Turn off RUN_TO_POSITION
        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    }
}


public void gyroDriveClawBack(double speed,
                              int distance,
                              double angle) {
    double max;
    double error;
    double steer;
    double leftSpeed;
    double rightSpeed;
    double distanceLeft;



    // Ensure that the opmode is still active
    if (opModeIsActive()) {

        // Determine new target position, and pass to motor controller
        wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
```

```
            wheelFrontRight.setTargetPosition(distance);
            wheelFrontLeft.setTargetPosition(distance);
            wheelRearRight.setTargetPosition(distance);
            wheelRearLeft.setTargetPosition(distance);

            wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            wheelFrontRight.setPower(speed);
            wheelFrontLeft.setPower(speed);
            wheelRearRight.setPower(speed);
            wheelRearLeft.setPower(speed);

            // keep looping while we are still active, and BOTH motors are running.
            while (opModeIsActive() &&
                    (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() &&
wheelRearRight.isBusy() && wheelRearLeft.isBusy())) {


                // adjust relative speed based on heading error.
                error = getError(angle);
                steer = getSteer(error, P_DRIVE_COEFF);

                // if driving in reverse, the motor correction also needs to be reversed
                if (distance < 0)
                    steer *= -1.0;

                leftSpeed = speed - steer;
                rightSpeed = speed + steer;

                // Normalize speeds if either one exceeds +/- 1.0;
                max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));
                if (max > 1.0) {
                    leftSpeed /= max;
                    rightSpeed /= max;
                }

                wheelFrontLeft.setPower(leftSpeed);
                wheelRearLeft.setPower(leftSpeed);
                wheelFrontRight.setPower(rightSpeed);
                wheelRearRight.setPower(rightSpeed);

                distanceLeft = distance - wheelFrontLeft.getCurrentPosition();

                if (distanceLeft < 1500) {
                    autoClawSpin.setPosition(AUTO_CLAW_SPIN_IN);
                } else {
                    autoClawSpin.setPosition(AUTO_CLAW_SPIN_OUT_RIGHT);
                }

                // Display drive status for the driver.
                telemetry.addData("Err/St", "%5.1f/%5.1f", error, steer);
                telemetry.addData("Speed", "%5.2f:%5.2f", leftSpeed, rightSpeed);
                telemetry.update();
            }
```

```
            // Stop all motion;
            wheelFrontRight.setPower(0);
            wheelRearRight.setPower(0);
            wheelFrontLeft.setPower(0);
            wheelRearLeft.setPower(0);

            // Turn off RUN_TO_POSITION
            wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        }
    }


    public void Drive (int frontRightDistance, int frontLeftDistance, int
rearRightDistance, int rearLeftDistance, double speed){


        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);
        autoClawSpin.setPosition(AUTO_CLAW_SPIN_IN);

        wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        wheelFrontRight.setTargetPosition(frontRightDistance);
        wheelFrontLeft.setTargetPosition(frontLeftDistance);
        wheelRearRight.setTargetPosition(rearRightDistance);
        wheelRearLeft.setTargetPosition(rearLeftDistance);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        autoClawSpin.setPosition(AUTO_CLAW_SPIN_IN);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_UP);

        wheelFrontRight.setPower(speed);
        wheelFrontLeft.setPower(speed);
        wheelRearRight.setPower(speed);
        wheelRearLeft.setPower(speed);


        while (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() &&
wheelRearRight.isBusy() && wheelRearLeft.isBusy() && opModeIsActive()) {



        }

        wheelFrontRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearRight.setPower(0);
```

```
        wheelRearLeft.setPower(0);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(100);
    }

    public void DriveStrafe (int frontRightDistance, int frontLeftDistance, int
rearRightDistance, int rearLeftDistance, double speed){


        wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        wheelFrontRight.setTargetPosition(frontRightDistance);
        wheelFrontLeft.setTargetPosition(frontLeftDistance);
        wheelRearRight.setTargetPosition(rearRightDistance);
        wheelRearLeft.setTargetPosition(rearLeftDistance);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        wheelFrontRight.setPower(speed);
        wheelFrontLeft.setPower(speed);
        wheelRearRight.setPower(speed);
        wheelRearLeft.setPower(speed);


        while (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() &&
wheelRearRight.isBusy() && wheelRearLeft.isBusy() && opModeIsActive()) {



        }

        wheelFrontRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearRight.setPower(0);
        wheelRearLeft.setPower(0);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(100);
    }

    public void gyroTurn(double speed, double angle) {
```

```
    // keep looping while we are still active, and not on heading.
    while (opModeIsActive() && !onHeading(speed, angle, P_TURN_COEFF)) {
        // Update telemetry & Allow time for other processes to run.
        telemetry.update();
    }
}


public void gyroHold(double speed, double angle, double holdTime) {

    ElapsedTime holdTimer = new ElapsedTime();

    // keep looping while we have time remaining.
    holdTimer.reset();
    while (opModeIsActive() && (holdTimer.time() < holdTime)) {
        // Update telemetry & Allow time for other processes to run.
        onHeading(speed, angle, P_TURN_COEFF);
        telemetry.update();
    }

    // Stop all motion;
    wheelFrontRight.setPower(0);
    wheelRearRight.setPower(0);
    wheelFrontLeft.setPower(0);
    wheelRearLeft.setPower(0);

}

public void gyroHoldClaw(double speed, double angle, double holdTime) {

    autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);
    ElapsedTime holdTimer = new ElapsedTime();

    // keep looping while we have time remaining.
    holdTimer.reset();
    while (opModeIsActive() && (holdTimer.time() < holdTime)) {
        // Update telemetry & Allow time for other processes to run.
        onHeading(speed, angle, P_TURN_COEFF);
        telemetry.update();
    }

    // Stop all motion;
    wheelFrontRight.setPower(0);
    wheelRearRight.setPower(0);
    wheelFrontLeft.setPower(0);
    wheelRearLeft.setPower(0);

}



boolean onHeading(double speed, double angle, double PCoeff) {
    double error;
    double steer;
    boolean onTarget = false;
    double leftSpeed;
    double rightSpeed;
```

```
        // determine turn power based on +/- error
        error = getError(angle);

        if (Math.abs(error) <= HEADING_THRESHOLD) {
            steer = 0.0;
            leftSpeed = 0.0;
            rightSpeed = 0.0;
            onTarget = true;
        } else {
            steer = getSteer(error, PCoeff);
            rightSpeed = speed * steer;
            leftSpeed = -rightSpeed;
        }

        // Send desired speeds to motors.
        wheelFrontLeft.setPower(leftSpeed);
        wheelRearLeft.setPower(leftSpeed);
        wheelFrontRight.setPower(rightSpeed);
        wheelRearRight.setPower(rightSpeed);

        // Display it for the driver.
        telemetry.addData("Target", "%5.2f", angle);
        telemetry.addData("Err/St", "%5.2f/%5.2f", error, steer);
        telemetry.addData("Speed.", "%5.2f:%5.2f", leftSpeed, rightSpeed);

        return onTarget;
    }

    public double getError(double targetAngle) {

        double robotError;
        angles = imu.getAngularOrientation(AxesReference.INTRINSIC, AxesOrder.ZYX,
AngleUnit.DEGREES);
        this.imu.getPosition();
        // calculate error in -179 to +180 range  (
        robotError = targetAngle - angles.firstAngle;
        while (robotError > 180) robotError -= 360;
        while (robotError <= -180) robotError += 360;
        return robotError;
    }


    public double getSteer(double error, double PCoeff) {
        return Range.clip(error * PCoeff, -1, 1);
    }

    //detection pipeline
    static class StageSwitchingPipeline extends OpenCvPipeline
    {
        Mat yCbCrChan2Mat = new Mat();
        Mat thresholdMat = new Mat();
        Mat all = new Mat();
        List<MatOfPoint> contoursList = new ArrayList<>();

        enum Stage
        {//color difference. greyscale
```

```
        detection,//includes outlines
        THRESHOLD,//b&w
        RAW_IMAGE,//displays raw view
    }

    private Stage stageToRenderToViewport = Stage.detection;
    private Stage[] stages = Stage.values();

    @Override
    public void onViewportTapped()
    {
        /*
         * Note that this method is invoked from the UI thread
         * so whatever we do here, we must do quickly.
         */

        int currentStageNum = stageToRenderToViewport.ordinal();

        int nextStageNum = currentStageNum + 1;

        if(nextStageNum >= stages.length)
        {
            nextStageNum = 0;
        }

        stageToRenderToViewport = stages[nextStageNum];
    }

    @Override
    public Mat processFrame(Mat input)
    {
        contoursList.clear();
        /*
         * This pipeline finds the contours of yellow blobs such as the Gold Mineral
         * from the Rover Ruckus game.
         */

        //color diff cb.
        //lower cb = more blue = skystone = white
        //higher cb = less blue = yellow stone = grey
        Imgproc.cvtColor(input, yCbCrChan2Mat, Imgproc.COLOR_RGB2YCrCb);//converts rgb
to ycrcb
        Core.extractChannel(yCbCrChan2Mat, yCbCrChan2Mat, 2);//takes cb difference and
stores

        //b&w
        Imgproc.threshold(yCbCrChan2Mat, thresholdMat, 102, 255,
Imgproc.THRESH_BINARY_INV);

        //outline/contour
        Imgproc.findContours(thresholdMat, contoursList, new Mat(), Imgproc.RETR_LIST,
Imgproc.CHAIN_APPROX_SIMPLE);
        yCbCrChan2Mat.copyTo(all);//copies mat object
        //Imgproc.drawContours(all, contoursList, -1, new Scalar(255, 0, 0), 3,
8);//draws blue contours
```

```
            //get values from frame
            double[] pixMid = thresholdMat.get((int)(input.rows()* midPos[1]),
(int)(input.cols()* midPos[0]));//gets value at circle
            valMid = (int)pixMid[0];

            double[] pixLeft = thresholdMat.get((int)(input.rows()* leftPos[1]),
(int)(input.cols()* leftPos[0]));//gets value at circle
            valLeft = (int)pixLeft[0];

            double[] pixRight = thresholdMat.get((int)(input.rows()* rightPos[1]),
(int)(input.cols()* rightPos[0]));//gets value at circle
            valRight = (int)pixRight[0];

            //create three points
            Point pointMid = new Point((int)(input.cols()* midPos[0]), (int)(input.rows()*
midPos[1]));
            Point pointLeft = new Point((int)(input.cols()* leftPos[0]),
(int)(input.rows()* leftPos[1]));
            Point pointRight = new Point((int)(input.cols()* rightPos[0]),
(int)(input.rows()* rightPos[1]));

            //draw circles on those points
            Imgproc.circle(all, pointMid,5, new Scalar( 255, 0, 0 ),1 );//draws circle
            Imgproc.circle(all, pointLeft,5, new Scalar( 255, 0, 0 ),1 );//draws circle
            Imgproc.circle(all, pointRight,5, new Scalar( 255, 0, 0 ),1 );//draws circle

            //draw 3 rectangles
            Imgproc.rectangle(//1-3
                    all,
                    new Point(
                            input.cols()*(leftPos[0]-rectWidth/2),
                            input.rows()*(leftPos[1]-rectHeight/2)),
                    new Point(
                            input.cols()*(leftPos[0]+rectWidth/2),
                            input.rows()*(leftPos[1]+rectHeight/2)),
                    new Scalar(0, 255, 0), 3);
            Imgproc.rectangle(//3-5
                    all,
                    new Point(
                            input.cols()*(midPos[0]-rectWidth/2),
                            input.rows()*(midPos[1]-rectHeight/2)),
                    new Point(
                            input.cols()*(midPos[0]+rectWidth/2),
                            input.rows()*(midPos[1]+rectHeight/2)),
                    new Scalar(0, 255, 0), 3);
            Imgproc.rectangle(//5-7
                    all,
                    new Point(
                            input.cols()*(rightPos[0]-rectWidth/2),
                            input.rows()*(rightPos[1]-rectHeight/2)),
                    new Point(
                            input.cols()*(rightPos[0]+rectWidth/2),
                            input.rows()*(rightPos[1]+rectHeight/2)),
                    new Scalar(0, 255, 0), 3);

            switch (stageToRenderToViewport)
            {
```

```
                case THRESHOLD:
                {
                    return thresholdMat;
                }

                case detection:
                {
                    return all;
                }

                case RAW_IMAGE:
                {
                    return input;
                }

                default:
                {
                    return input;
                }
            }
        }

    }
}
```

FULL CODE: PARK

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


import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.Servo;

@Autonomous (name = "Drive Forward")
//@Disabled


public class DriveForward extends LinearOpMode
{

   private DcMotor wheelFrontRight;
   private DcMotor wheelFrontLeft;
   private DcMotor wheelRearRight;
   private DcMotor wheelRearLeft;



   @Override
   public void runOpMode() {


       wheelFrontRight = hardwareMap.get(DcMotor.class,"wheelFrontRight");
       wheelFrontLeft = hardwareMap.get(DcMotor.class,"wheelFrontLeft");
       wheelRearRight = hardwareMap.get(DcMotor.class,"wheelRearRight");
       wheelRearLeft = hardwareMap.get(DcMotor.class,"wheelRearLeft");

       wheelFrontRight.setDirection(DcMotor.Direction.REVERSE);
       wheelRearRight.setDirection(DcMotor.Direction.REVERSE);

       wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
       wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
       wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
       wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);


       waitForStart();


       //Drive Forward
       Drive(-400, -400, -400, -400, 0.2);


   }

   public void Drive (int frontRightDistance, int frontLeftDistance, int rearRightDistance,
int rearLeftDistance, double speed){


       wheelFrontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
       wheelFrontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
```

```
        wheelRearRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        wheelFrontRight.setTargetPosition(frontRightDistance);
        wheelFrontLeft.setTargetPosition(frontLeftDistance);
        wheelRearRight.setTargetPosition(rearRightDistance);
        wheelRearLeft.setTargetPosition(rearLeftDistance);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        wheelFrontRight.setPower(speed);
        wheelFrontLeft.setPower(speed);
        wheelRearRight.setPower(speed);
        wheelRearLeft.setPower(speed);


        while (wheelFrontRight.isBusy() && wheelFrontLeft.isBusy() && wheelRearRight.isBusy()
&& wheelRearLeft.isBusy() && opModeIsActive()) {
            telemetry.addData("FrontRightPosition", wheelFrontRight.getCurrentPosition());
            telemetry.addData("FrontLeftPosition", wheelFrontLeft.getCurrentPosition());
            telemetry.update();
        }

        wheelFrontRight.setPower(0);
        wheelFrontLeft.setPower(0);
        wheelRearRight.setPower(0);
        wheelRearLeft.setPower(0);

        wheelFrontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelFrontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        wheelRearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(500);
    }

}
```

FULL CODE: TELE OP

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

import android.webkit.DownloadListener;

import com.qualcomm.hardware.rev.RevTouchSensor;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;
import com.qualcomm.robotcore.hardware.DigitalChannel;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.util.Range;


@TeleOp (name = "Enigma TeleOp")
//@Disabled
public class FirstTeleOpTryThree extends LinearOpMode {
   private ElapsedTime runtime = new ElapsedTime();
   //motors
   //mecanum wheels
   private DcMotor wheelFrontRight;
   private DcMotor wheelFrontLeft;
   private DcMotor wheelRearRight;
   private DcMotor wheelRearLeft;


   //vertical lift
   private DcMotor motorVerticalLiftRight;
   private DcMotor motorVerticalLiftLeft;

   //horizontal lift
   private DcMotor motorHorizontalSlide;

   //servos
   private CRServo motorIntakeRight;
   private CRServo motorIntakeLeft;
   private DcMotor tapeMeasure;
   private Servo frontClawServo;
   private Servo foundationServoOne;
   private Servo foundationServoTwo;
   private Servo pullPinServo;
   private Servo parkingServo;
   private Servo rearClawServo;
   private Servo autoClawRotate;
   private Servo autoClawClamp;
   private Servo autoClawSpin;

   private RevTouchSensor magneticLimitSwitchX;
   private RevTouchSensor magneticLimitSwitchY;

   private static final double FOUNDATION_ARM_OUT_ONE = .84;
   private static final double FOUNDATION_ARM_IN_ONE = .52;
```

```
private static final double FOUNDATION_ARM_OUT_TWO = 0;
private static final double FOUNDATION_ARM_IN_TWO = .30;

private static final double REAR_CLAW_OUT = .30;
private static final double REAR_CLAW_IN = .63;
private static final double REAR_CLAW_MID = .57;

private static final double PULL_PIN = 0;
private static final double PIN_INITIAL = 1;

private static final double FRONT_CLAW_IN = .46;
private static final double FRONT_CLAW_OUT = .85;
private static final double FRONT_CLAW_MID = .64;

private static final double PARKING_SERVO_INIT = 0;
private static final double PAKRING_SERVO_OUT = .90;

private static final double AUTO_CLAW_SPIN_IN = .50;
private static final double AUTO_CLAW_SPIN_OUT_LEFT = 1;
private static final double AUTO_CLAW_SPIN_OUT_RIGHT = .06;
private static final double AUTO_CLAW_SPIN_LETGO = .41;
private static final double AUTO_CLAW_ROTATE_UPUP = .54;
private static final double AUTO_CLAW_ROTATE_INIT = 1;
private static final double AUTO_CLAW_ROTATE_DOWN = .21;
private static final double AUTO_CLAW_ROTATE_UP = .38;
private static final double AUTO_CLAW_CLAMP_INIT = .31;
private static final double AUTO_CLAW_CLAMP_DOWN = .16;
private static final double AUTO_CLAW_CLAMP_UP = .70;
private static final double AUTO_CLAW_CLAMP_LETGO = .33;


@Override
public void runOpMode() {
    //motors
    //mecanum wheels
    wheelFrontRight = hardwareMap.get(DcMotor.class, "wheelFrontRight");
    wheelFrontLeft = hardwareMap.get(DcMotor.class, "wheelFrontLeft");
    wheelRearRight = hardwareMap.get(DcMotor.class, "wheelRearRight");
    wheelRearLeft = hardwareMap.get(DcMotor.class, "wheelRearLeft");

    wheelFrontRight.setDirection(DcMotor.Direction.REVERSE);
    wheelRearRight.setDirection(DcMotor.Direction.REVERSE);

    wheelFrontRight.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
    wheelFrontLeft.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
    wheelRearRight.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
    wheelRearLeft.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);

    //vertical lift
    motorVerticalLiftRight = hardwareMap.get(DcMotor.class, "motorVerticalLiftRight");
    motorVerticalLiftLeft = hardwareMap.get(DcMotor.class, "motorVerticalLiftLeft");

    motorVerticalLiftRight.setDirection(DcMotor.Direction.REVERSE);
    motorVerticalLiftRight.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
    motorVerticalLiftLeft.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
```

```
        //horizontal slide
        motorHorizontalSlide = hardwareMap.get(DcMotor.class, "motorHorizontalSlide");

        motorHorizontalSlide.setDirection(DcMotor.Direction.REVERSE);
        motorHorizontalSlide.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);


        //servos
        frontClawServo = hardwareMap.get(Servo.class, "frontClawServo");
        foundationServoOne = hardwareMap.get(Servo.class, "foundationServoOne");
        foundationServoTwo = hardwareMap.get(Servo.class, "foundationServoTwo");
        pullPinServo = hardwareMap.get(Servo.class, "pullPinServo");
        rearClawServo = hardwareMap.get(Servo.class, "rearClawServo");
        tapeMeasure = hardwareMap.get(DcMotor.class, "tapeMeasure");
        autoClawClamp = hardwareMap.get(Servo.class, "autoClawClamp");
        autoClawRotate = hardwareMap.get(Servo.class, "autoClawRotate");
        autoClawSpin = hardwareMap.get(Servo.class, "autoClawSpin");
        motorIntakeRight = hardwareMap.get(CRServo.class, "motorIntakeRight");
        motorIntakeLeft = hardwareMap.get(CRServo.class, "motorIntakeLeft");
        parkingServo = hardwareMap.get(Servo.class, "parkingServo");

        motorIntakeRight.setDirection(CRServo.Direction.REVERSE);

        magneticLimitSwitchX = hardwareMap.get(RevTouchSensor.class, "magneticLimitSwitchX");
        magneticLimitSwitchY = hardwareMap.get(RevTouchSensor.class, "magneticLimitSwitchY");


        //sets position of servo
        rearClawServo.setPosition(REAR_CLAW_OUT);
        frontClawServo.setPosition(FRONT_CLAW_OUT);

        foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
        foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);

        //parkingServo.setPosition(PARKING_SERVO_INIT);

        autoClawSpin.setPosition(AUTO_CLAW_SPIN_LETGO);
        autoClawRotate.setPosition(AUTO_CLAW_ROTATE_DOWN);
        autoClawClamp.setPosition(AUTO_CLAW_CLAMP_DOWN);

        telemetry.addData("Status", "UhhhUhhhH we haf do wAIt heow by du duow fo do
teachow");
        telemetry.update();

        //wait for driver to press start
        waitForStart();
        runtime.reset();


        //run until driver presses stop
        while (opModeIsActive()) {
            //mecanum drive

            double vertical = -gamepad1.left_stick_y;
            double horizontal = gamepad1.left_stick_x;
            double pivot = gamepad1.right_stick_x;
```

```
        double wheelFrontRightPower = -pivot + vertical - horizontal;
        double wheelFrontLeftPower = -pivot + vertical + horizontal;
        double wheelRearRightPower = pivot + vertical + horizontal;
        double wheelRearLeftPower = pivot + vertical - horizontal;

        if (gamepad1.right_bumper) {
            wheelFrontRightPower = Range.clip(wheelFrontRightPower, -.3, .3);
            wheelFrontLeftPower = Range.clip(wheelFrontLeftPower, -.3, .3);
            wheelRearRightPower = Range.clip(wheelRearRightPower, -.3, .3);
            wheelRearLeftPower = Range.clip(wheelRearLeftPower, -.3, .3);
        } else {
            wheelFrontRightPower = Range.clip(wheelFrontRightPower, -.95, .95);
            wheelFrontLeftPower = Range.clip(wheelFrontLeftPower, -.95, .95);
            wheelRearRightPower = Range.clip(wheelRearRightPower, -.95, .95);
            wheelRearLeftPower = Range.clip(wheelRearLeftPower, -.95, .95);
        }


        wheelFrontRight.setPower(wheelFrontRightPower);
        wheelRearRight.setPower(wheelFrontLeftPower);
        wheelFrontLeft.setPower(wheelRearRightPower);
        wheelRearLeft.setPower(wheelRearLeftPower);


        //vertical lifts
        double motorVerticalLiftRightPower = -gamepad2.right_stick_y;
        double motorVerticalLiftLeftPower = -gamepad2.right_stick_y;

        if (!magneticLimitSwitchY.isPressed()) {
            motorVerticalLiftRightPower = Range.clip(motorVerticalLiftRightPower, -1, 1);
            motorVerticalLiftLeftPower = Range.clip(motorVerticalLiftLeftPower, -0.90,
.93);
            motorVerticalLiftRight.setPower(motorVerticalLiftRightPower);
            motorVerticalLiftLeft.setPower(motorVerticalLiftLeftPower);
        } else {
            motorVerticalLiftRightPower = Range.clip(motorVerticalLiftRightPower, 0, 1);
            motorVerticalLiftLeftPower = Range.clip(motorVerticalLiftLeftPower, 0, .93);
            motorVerticalLiftRight.setPower(motorVerticalLiftRightPower);
            motorVerticalLiftLeft.setPower(motorVerticalLiftLeftPower);
        }


        //horizontal slide
        double motorHorizontalSlidePower = -gamepad2.left_stick_y;

        if (!magneticLimitSwitchX.isPressed()) {
            motorHorizontalSlidePower = Range.clip(motorHorizontalSlidePower, -.90, .90);
            motorHorizontalSlide.setPower(motorHorizontalSlidePower);
        } else {
            motorHorizontalSlidePower = Range.clip(motorHorizontalSlidePower, 0, .90);
            motorHorizontalSlide.setPower(motorHorizontalSlidePower);
        }

/*
        if (gamepad1.left_bumper) {
            motorIntakeRight.setPower(-.8);
```

```
            motorIntakeLeft.setPower(-.8);
        } else if (gamepad1.right_bumper) {
            motorIntakeRight.setPower(.8);
            motorIntakeLeft.setPower(.8);
        } else {
            motorIntakeRight.setPower(.8);
            motorIntakeLeft.setPower(.8);
        }

*/

if (gamepad1.left_bumper) {
   motorIntakeLeft.setPower(-0.8);
   motorIntakeRight.setPower(-0.8);
} else {
   motorIntakeRight.setPower(0);
   motorIntakeLeft.setPower(0);
}

        //servos
        if (gamepad1.b) {
            tapeMeasure.setPower(1);
        } else {
            tapeMeasure.setPower(0);
        }

        //foundation servos
        if (gamepad1.y) {
            foundationServoOne.setPosition(FOUNDATION_ARM_OUT_ONE);
            foundationServoTwo.setPosition(FOUNDATION_ARM_OUT_TWO);
        }


        if (gamepad1.a) {
            foundationServoOne.setPosition(FOUNDATION_ARM_IN_ONE);
            foundationServoTwo.setPosition(FOUNDATION_ARM_IN_TWO);
        }


        if (gamepad2.right_bumper) {
            pullPinServo.setPosition(PIN_INITIAL);
        } else {
            pullPinServo.setPosition(PULL_PIN);
        }

        if (gamepad2.x) {
            rearClawServo.setPosition(REAR_CLAW_OUT);
            frontClawServo.setPosition(FRONT_CLAW_OUT);
        }


        if (gamepad2.b) {
            rearClawServo.setPosition(REAR_CLAW_IN);
            frontClawServo.setPosition(FRONT_CLAW_IN);
        }
```

```
            if (gamepad2.y) {
                frontClawServo.setPosition(FRONT_CLAW_MID);
                rearClawServo.setPosition(REAR_CLAW_MID);
            }

            if (gamepad2.left_bumper) {
                parkingServo.setPosition(PAKRING_SERVO_OUT);
            } else {
                parkingServo.setPosition(PARKING_SERVO_INIT);
            }



            idle();


        }
    }
}
```