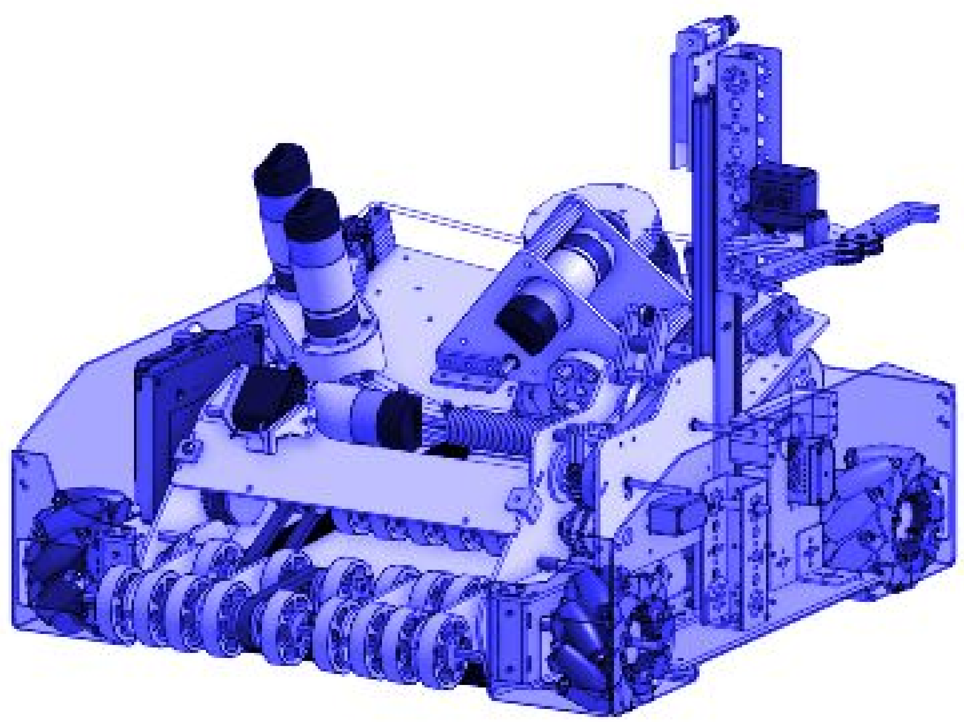


FTC Team 15317 Error Code 404

Engineering Notebook

2020-2021

Table of Contents:

Important/Notable Programs:

(Please explore our team's full code by visiting our GitHub Repository.)

Teleop2021.java
Teleop2021 controls our robot during the TeleOp period. It includes all of the driver-controlled functions of the robot, including driving, operating the claw and lift, collector, hopper, and shooter. This program also contains all of our driver controlled enhancements.

To see our synchronous state engine, which allows us to reuse our autonomous code to run automated sequences during Teleop at the push of a button, see **lines 126-177** below.

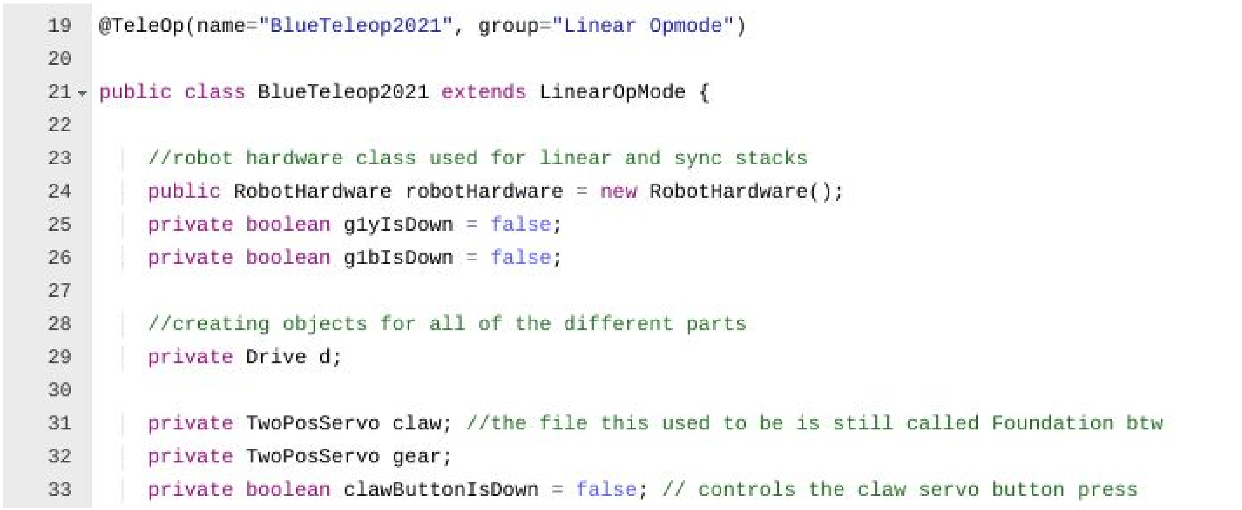
To see our auto-positioning feature which returns the robot to a set point in front of the power shots from anywhere on the field and sets the correct shooter angle and power for shooting the powershots, see **lines 148-177 below** This is an automated sequence that uses our synchronous state engine..

To see our strafing feature, which uses our autonomous code to strafe a set distance at the push of a button in order to aim between power shots see **lines 140-146** below.

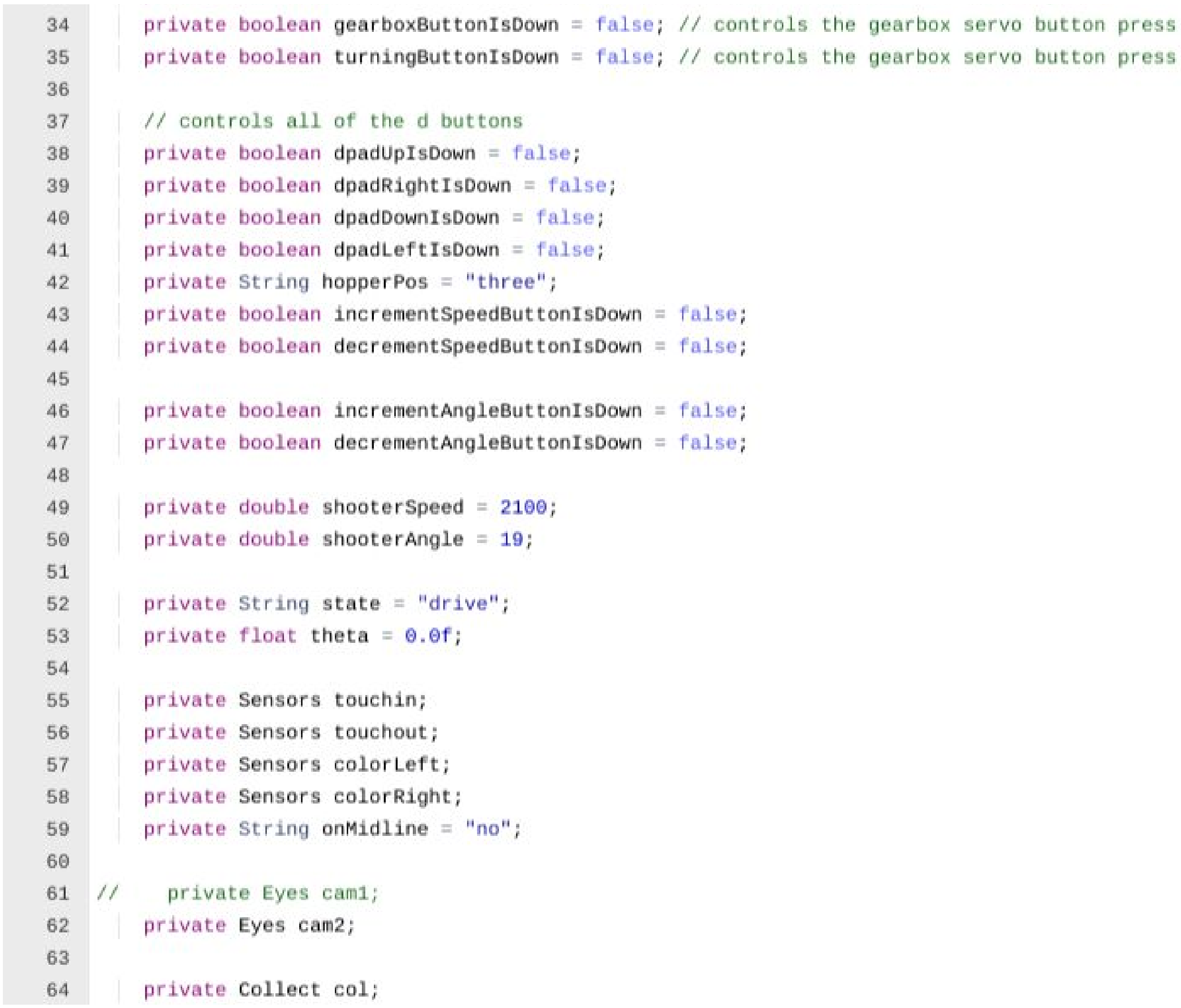
To see our auto-aiming feature, which lets the robot turn towards the goal, set an accurate shooter angle, and set an accurate shooting power with a push of a button from nearly anywhere on the field, see **lines 244-278** below.

To see our code for our targeting servo, which visually shows drivers when the robot sees a Vuforia target, see **lines 184-188** below.

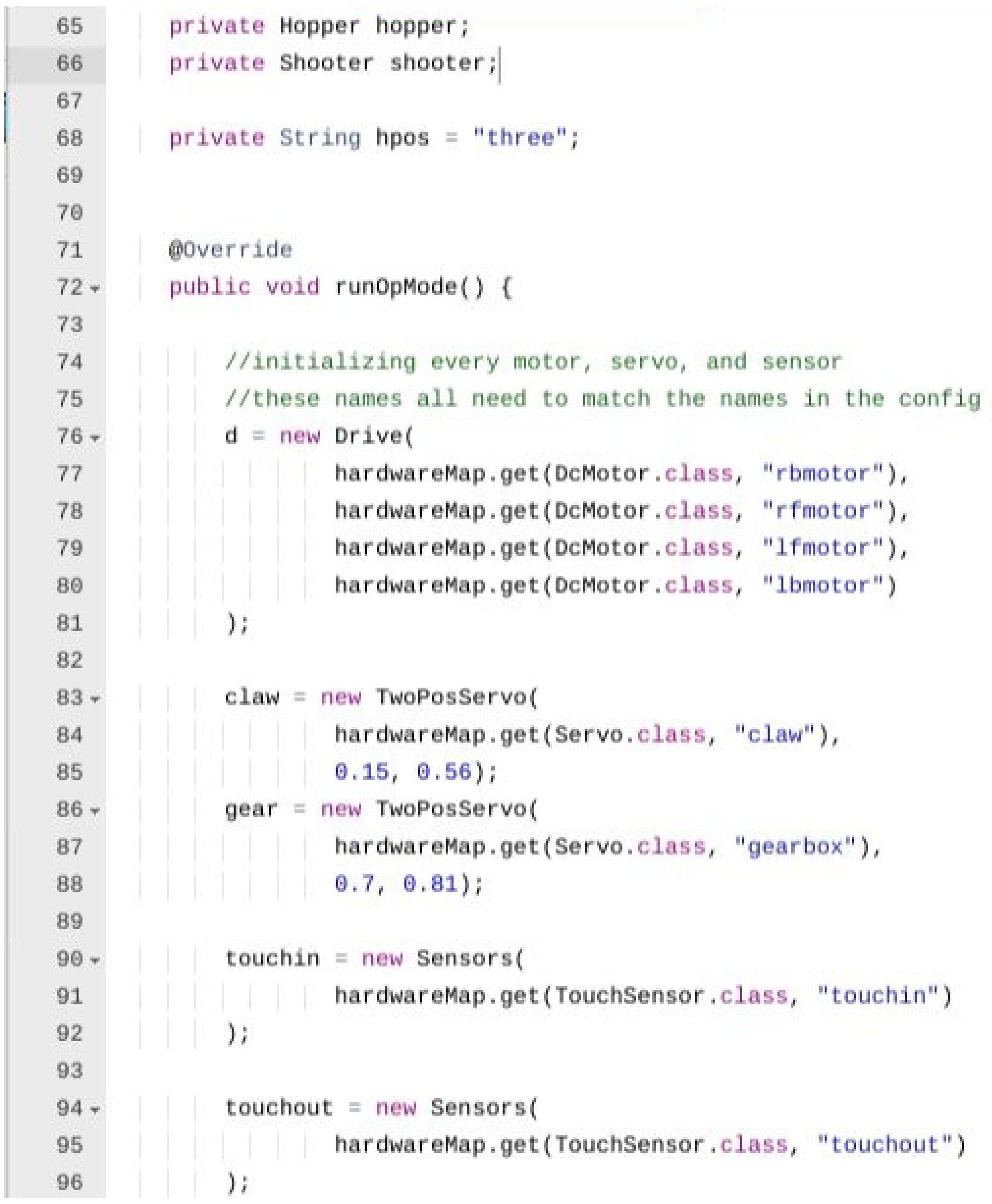
To see our automatic gear-shifting feature, which automatically meshes and retracts a gear depending on our drivers' gamepad actions to increase efficiency while switching between operating our lift and collector, see **lines 218-239** below.



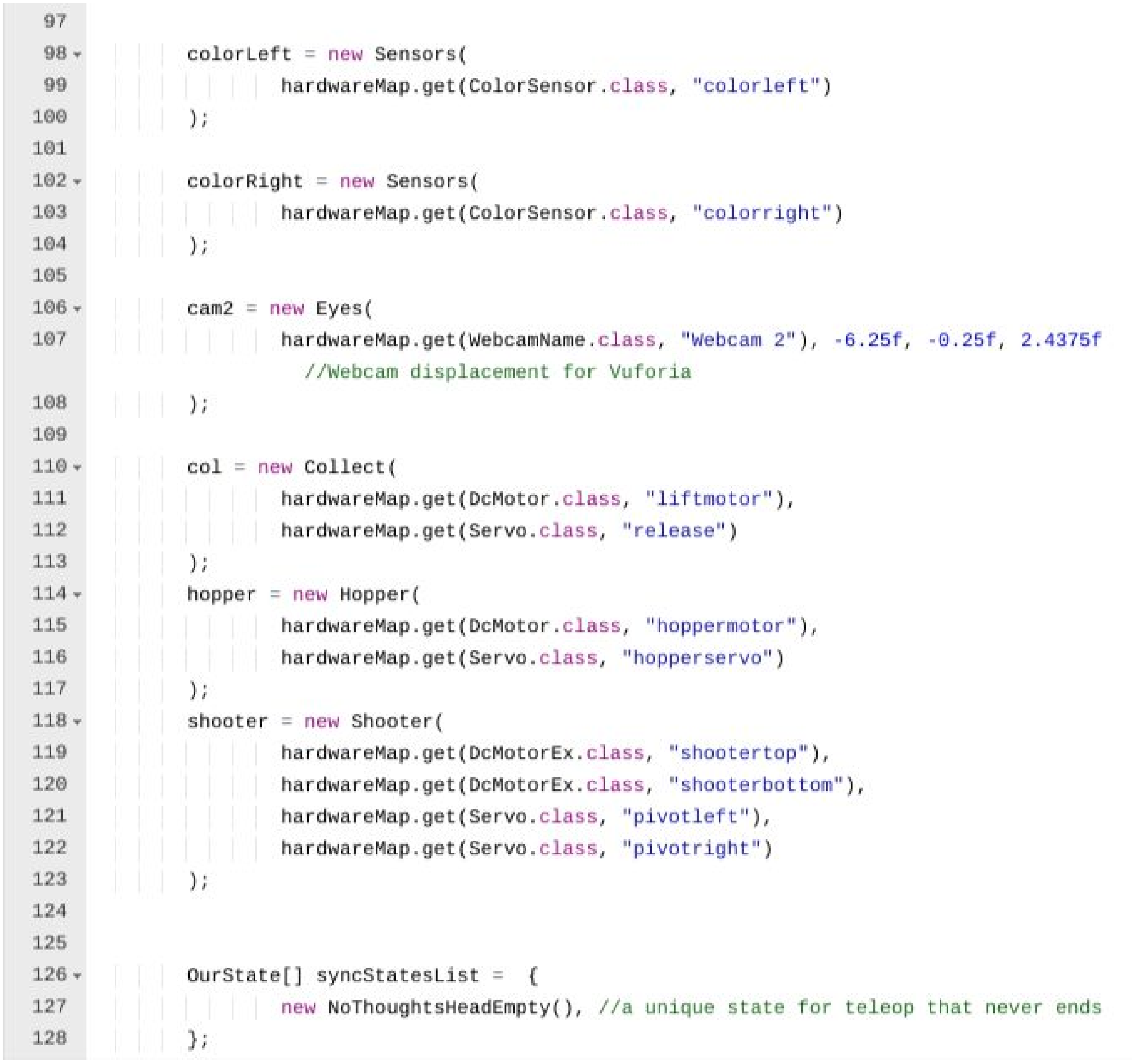
```java
@TeleOp(name="BlueTeleop2021", group="Linear Opmode")

public class BlueTeleop2021 extends LinearOpMode {

    //robot hardware class used for linear and sync stacks
    public RobotHardware robotHardware = new RobotHardware();
    private boolean g1yIsDown = false;
    private boolean g1bIsDown = false;

    //creating objects for all of the different parts
    private Drive d;

    private TwoPosServo claw; //the file this used to be is still called Foundation btw
    private TwoPosServo gear;
    private boolean clawButtonIsDown = false; // controls the claw servo button press
```



```java
    private boolean gearboxButtonIsDown = false; // controls the gearbox servo button press
    private boolean turningButtonIsDown = false; // controls the gearbox servo button press

    // controls all of the d buttons
    private boolean dpadUpIsDown = false;
    private boolean dpadRightIsDown = false;
    private boolean dpadDownIsDown = false;
    private boolean dpadLeftIsDown = false;
    private String hopperPos = "three";
    private boolean incrementSpeedButtonIsDown = false;
    private boolean decrementSpeedButtonIsDown = false;

    private boolean incrementAngleButtonIsDown = false;
    private boolean decrementAngleButtonIsDown = false;

    private double shooterSpeed = 2100;
    private double shooterAngle = 19;

    private String state = "drive";
    private float theta = 0.0f;

    private Sensors touchin;
    private Sensors touchout;
    private Sensors colorLeft;
    private Sensors colorRight;
    private String onMidline = "no";

//    private Eyes cam1;
    private Eyes cam2;

    private Collect col;
```

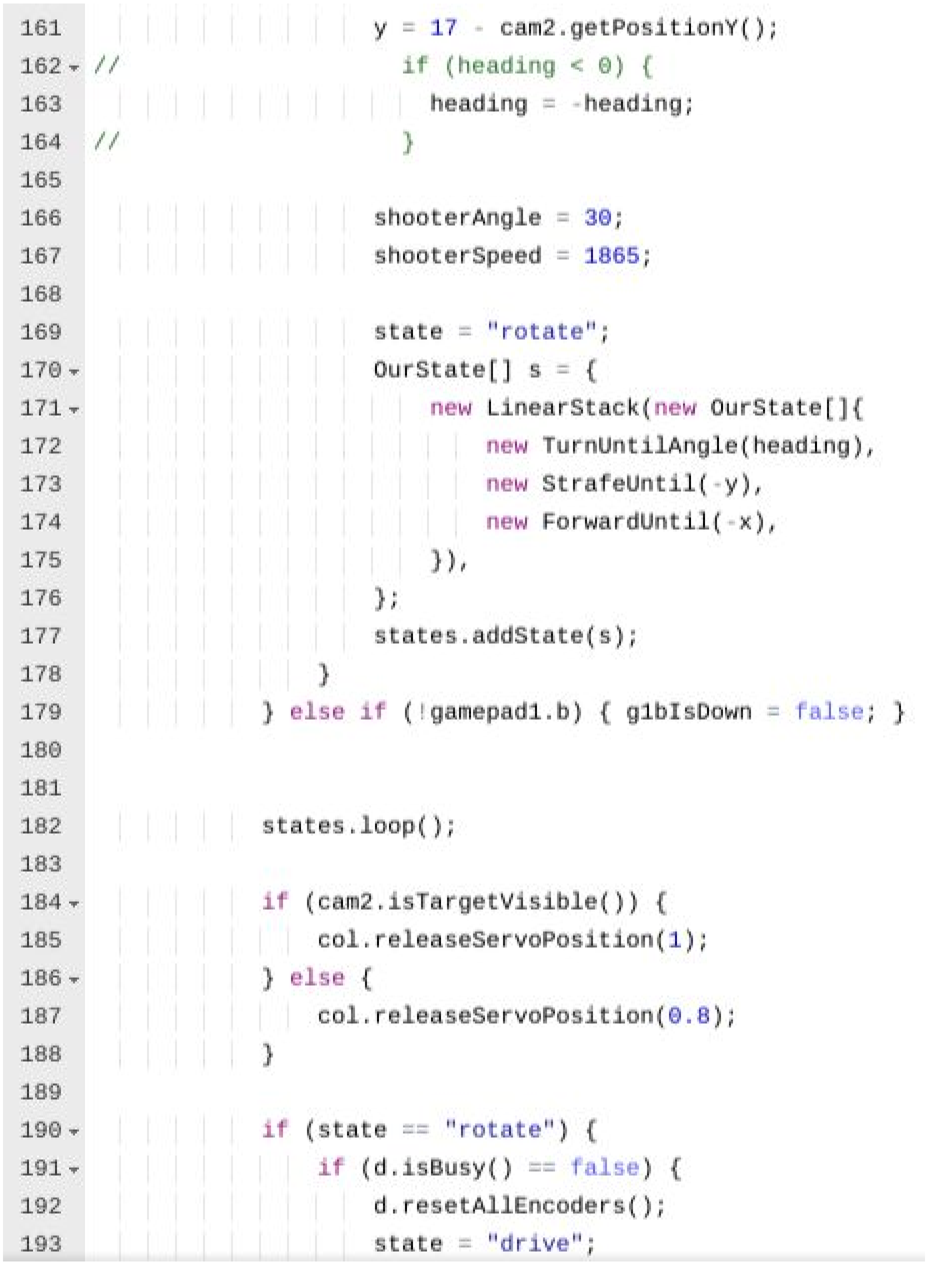


```java
    private Hopper hopper;
    private Shooter shooter;

    private String hpos = "three";


    @Override
    public void runOpMode() {

        //initializing every motor, servo, and sensor
        //these names all need to match the names in the config
        d = new Drive(
                hardwareMap.get(DcMotor.class, "rbmotor"),
                hardwareMap.get(DcMotor.class, "rfmotor"),
                hardwareMap.get(DcMotor.class, "lfmotor"),
                hardwareMap.get(DcMotor.class, "lbmotor")
        );

        claw = new TwoPosServo(
                hardwareMap.get(Servo.class, "claw"),
                0.15, 0.56);
        gear = new TwoPosServo(
                hardwareMap.get(Servo.class, "gearbox"),
                0.7, 0.81);

        touchin = new Sensors(
                hardwareMap.get(TouchSensor.class, "touchin")
        );

        touchout = new Sensors(
                hardwareMap.get(TouchSensor.class, "touchout")
        );
```

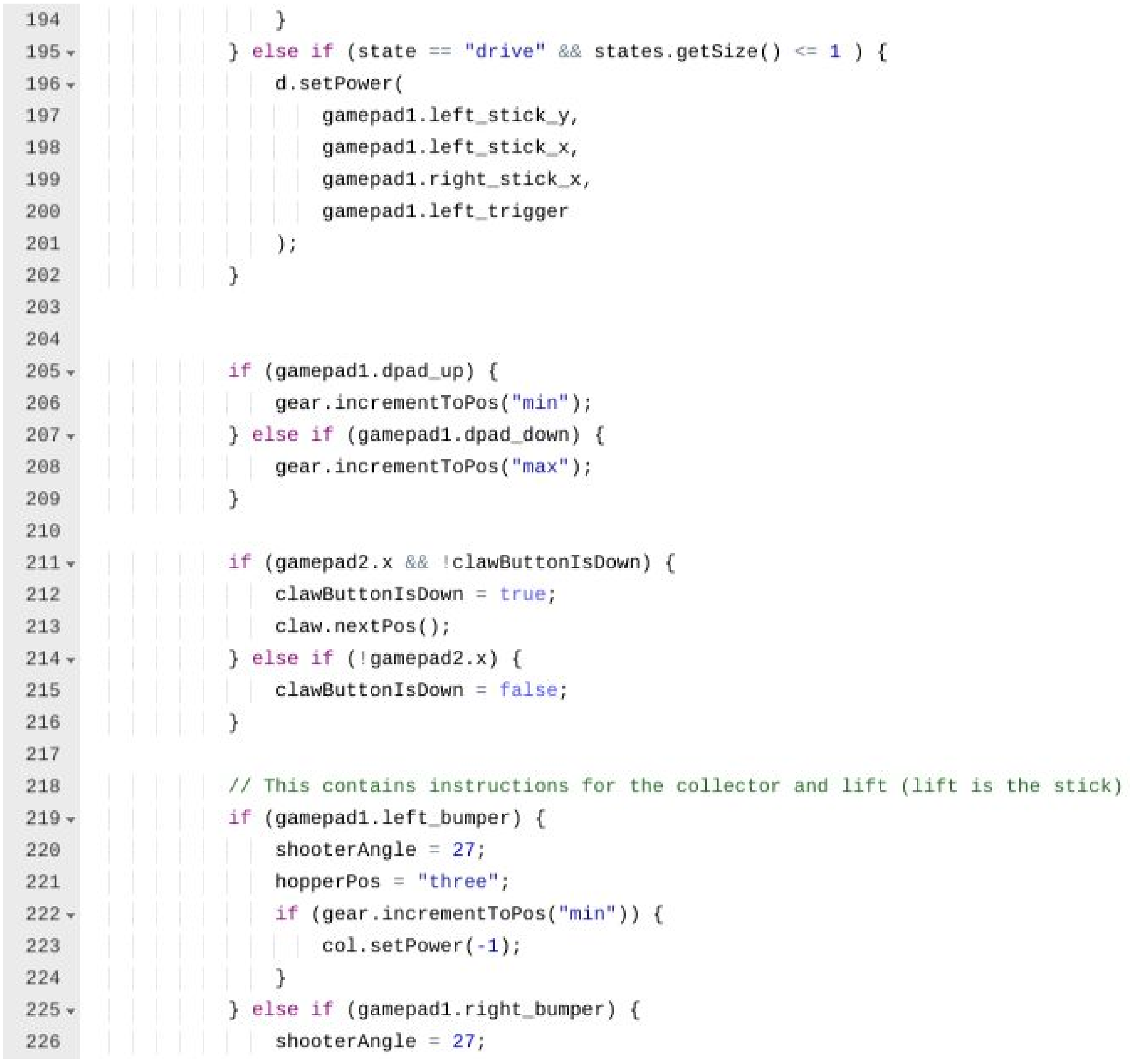


```

        colorLeft = new Sensors(
                hardwareMap.get(ColorSensor.class, "colorleft")
        );

        colorRight = new Sensors(
                hardwareMap.get(ColorSensor.class, "colorright")
        );

        cam2 = new Eyes(
                hardwareMap.get(WebcamName.class, "Webcam 2"), -6.25f, -0.25f, 2.4375f //Webcam displacement for Vuforia
        );

        col = new Collect(
                hardwareMap.get(DcMotor.class, "liftmotor"),
                hardwareMap.get(Servo.class, "release")
        );
        hopper = new Hopper(
                hardwareMap.get(DcMotor.class, "hoppermotor"),
                hardwareMap.get(Servo.class, "hopperservo")
        );
        shooter = new Shooter(
                hardwareMap.get(DcMotorEx.class, "shootertop"),
                hardwareMap.get(DcMotorEx.class, "shooterbottom"),
                hardwareMap.get(Servo.class, "pivotleft"),
                hardwareMap.get(Servo.class, "pivotright")
        );


        OurState[] syncStatesList =  {
                new NoThoughtsHeadEmpty(), //a unique state for teleop that never ends
        };
```

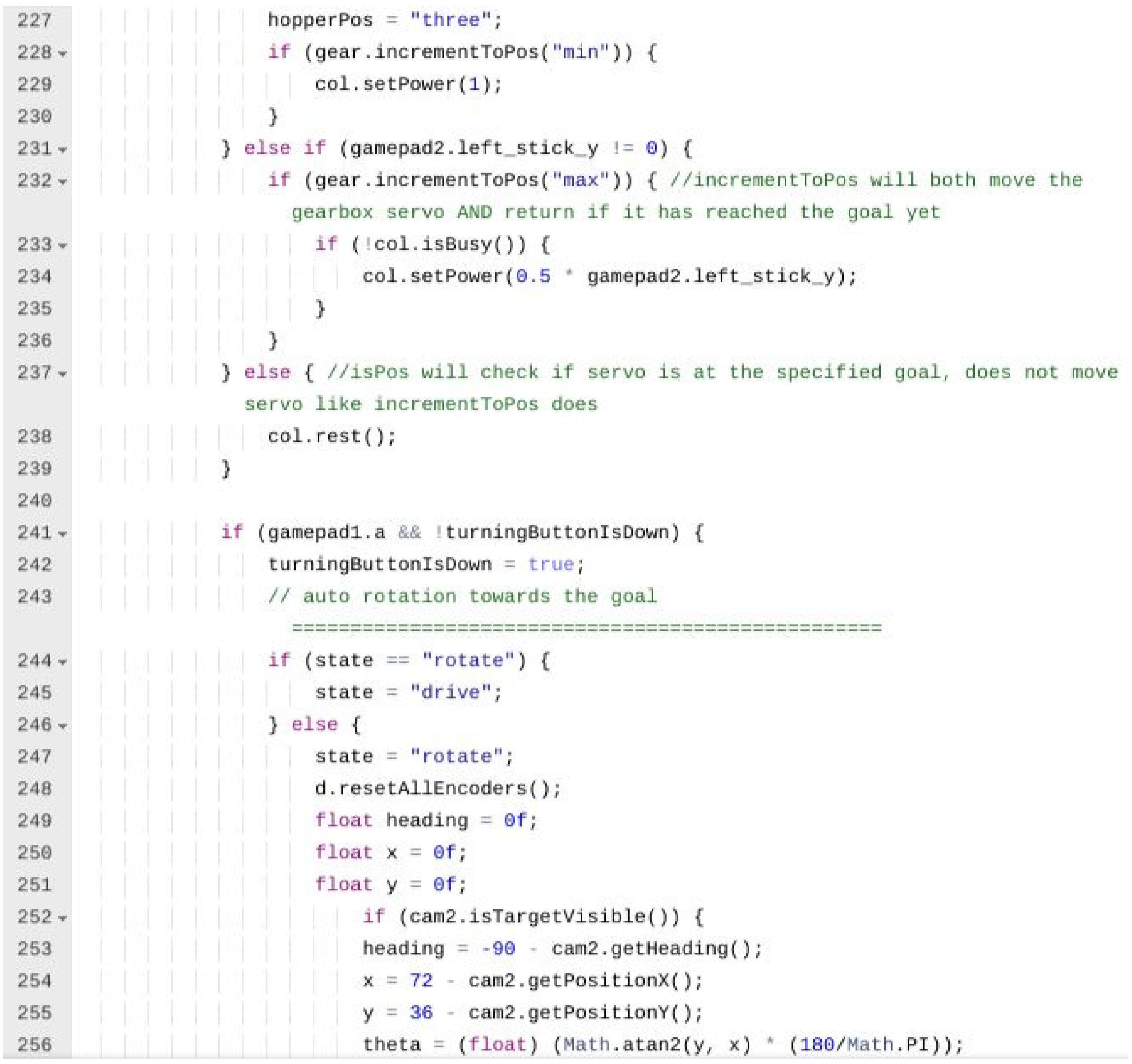
```java
            SynchronousStack states = new SynchronousStack(syncStatesList);
            robotHardware.build(hardwareMap);
            states.init(robotHardware);

            waitForStart();
            col.releaseCollector();

            while (opModeIsActive()) {



                    if (gamepad1.y && !g1yIsDown) { g1yIsDown = true;
                    state = "rotate";
                        OurState[] s = {
                                new StrafeUntil(8)
                        };
                        states.addState(s);
                    } else if (!gamepad1.y) { g1yIsDown = false; }

                    if (gamepad1.b && !g1bIsDown) { g1bIsDown = true;
                        // auto rotation towards the power shot ==================================================

                        state = "rotate";
                        float heading = 0f;
                        float x = 0f;
                        float y = 0f;


                        if (cam2.isTargetVisible()) {

                            heading = -90 - cam2.getHeading();
                            x = -39 - cam2.getPositionX();
```



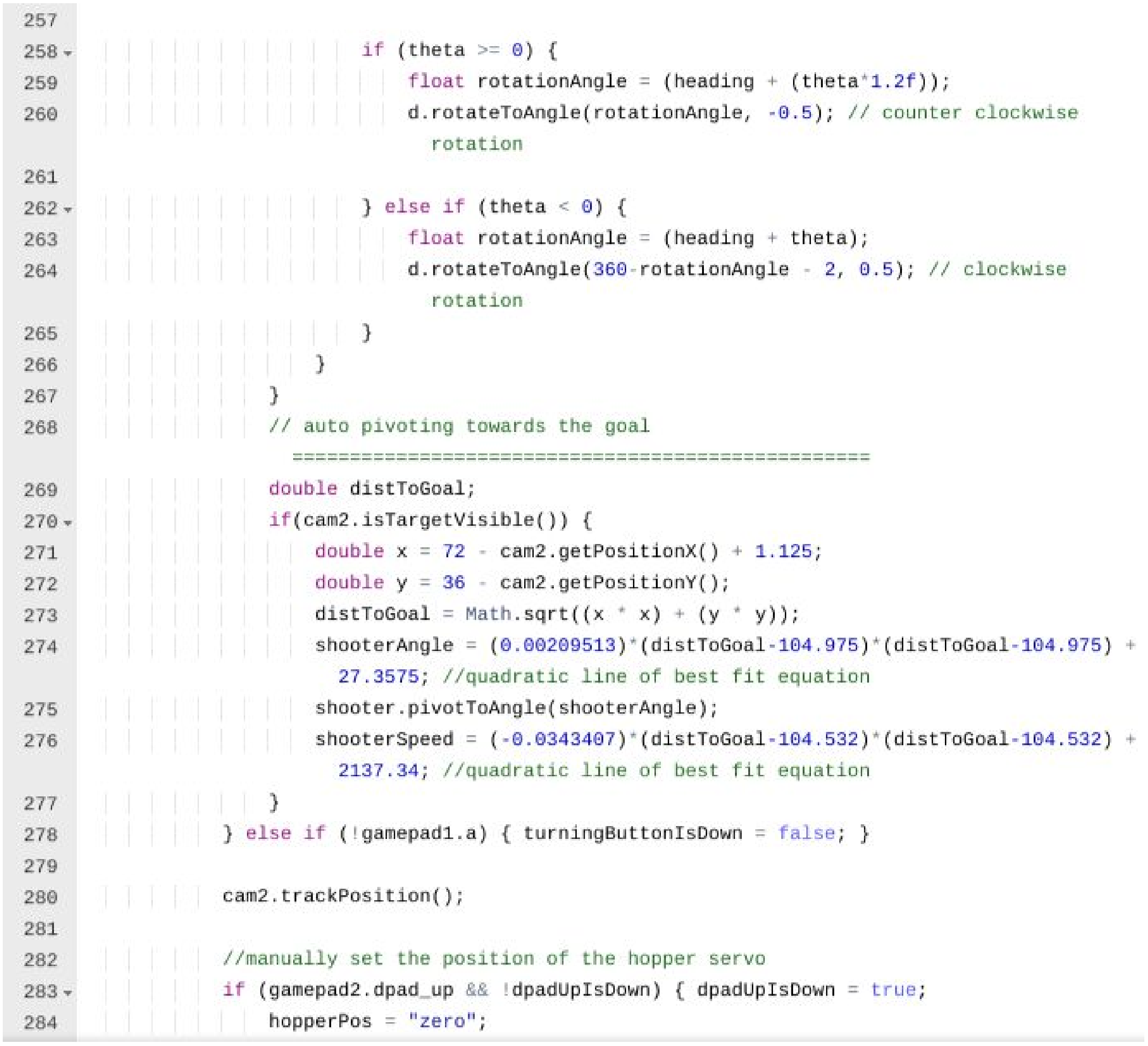
```
                    y = 17 - cam2.getPositionY();
//                    if (heading < 0) {
                        heading = -heading;
//                    }

                    shooterAngle = 30;
                    shooterSpeed = 1865;

                    state = "rotate";
                    OurState[] s = {
                        new LinearStack(new OurState[]{
                            new TurnUntilAngle(heading),
                            new StrafeUntil(-y),
                            new ForwardUntil(-x),
                        }),
                    };
                    states.addState(s);
                }
            } else if (!gamepad1.b) { g1bIsDown = false; }


            states.loop();

            if (cam2.isTargetVisible()) {
                col.releaseServoPosition(1);
            } else {
                col.releaseServoPosition(0.8);
            }

            if (state == "rotate") {
                if (d.isBusy() == false) {
                    d.resetAllEncoders();
                    state = "drive";
```



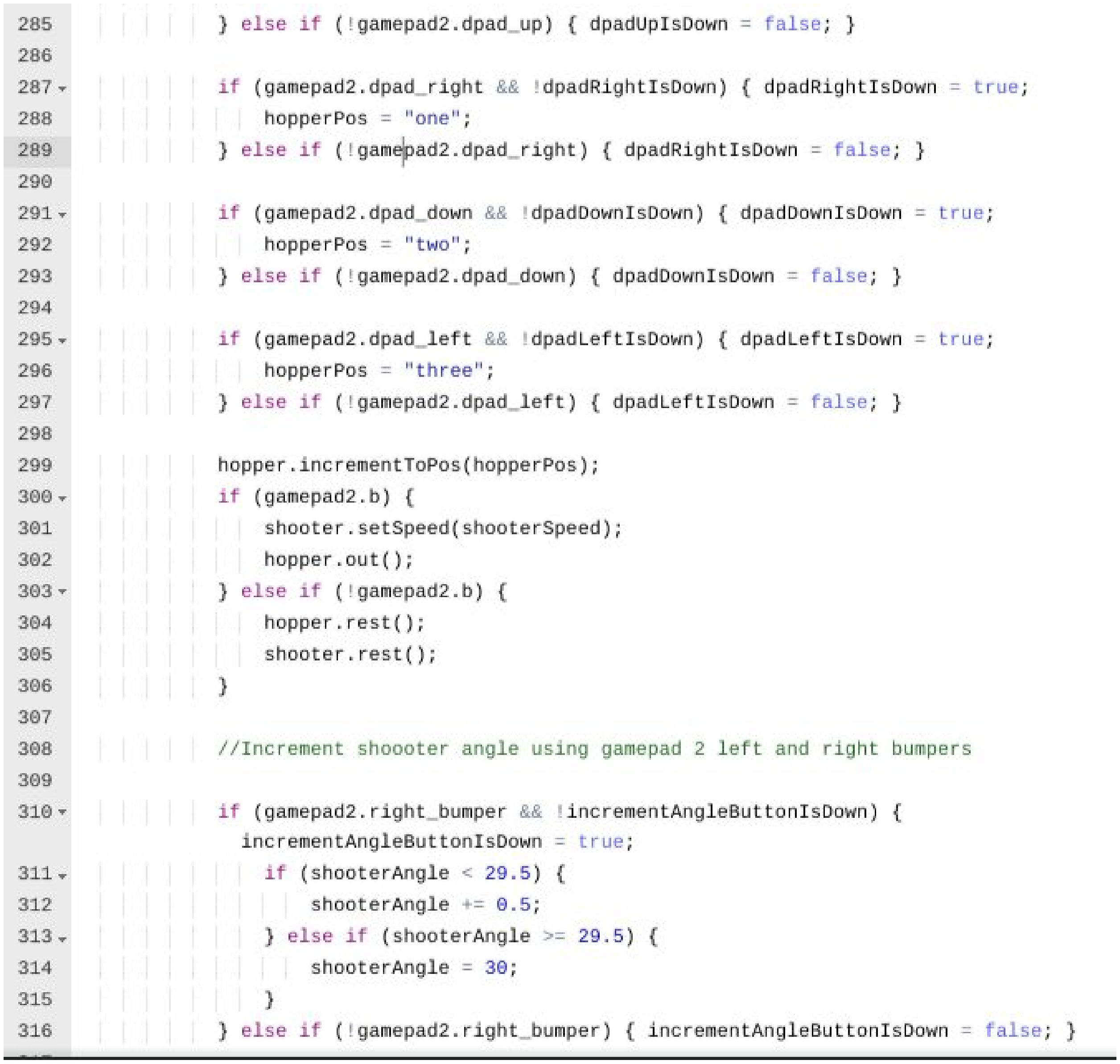
```
              }
            } else if (state == "drive" && states.getSize() <= 1 ) {
              d.setPower(
                gamepad1.left_stick_y,
                gamepad1.left_stick_x,
                gamepad1.right_stick_x,
                gamepad1.left_trigger
              );
            }


            if (gamepad1.dpad_up) {
              gear.incrementToPos("min");
            } else if (gamepad1.dpad_down) {
              gear.incrementToPos("max");
            }

            if (gamepad2.x && !clawButtonIsDown) {
              clawButtonIsDown = true;
              claw.nextPos();
            } else if (!gamepad2.x) {
              clawButtonIsDown = false;
            }

            // This contains instructions for the collector and lift (lift is the stick)
            if (gamepad1.left_bumper) {
              shooterAngle = 27;
              hopperPos = "three";
              if (gear.incrementToPos("min")) {
                col.setPower(-1);
              }
            } else if (gamepad1.right_bumper) {
              shooterAngle = 27;
```



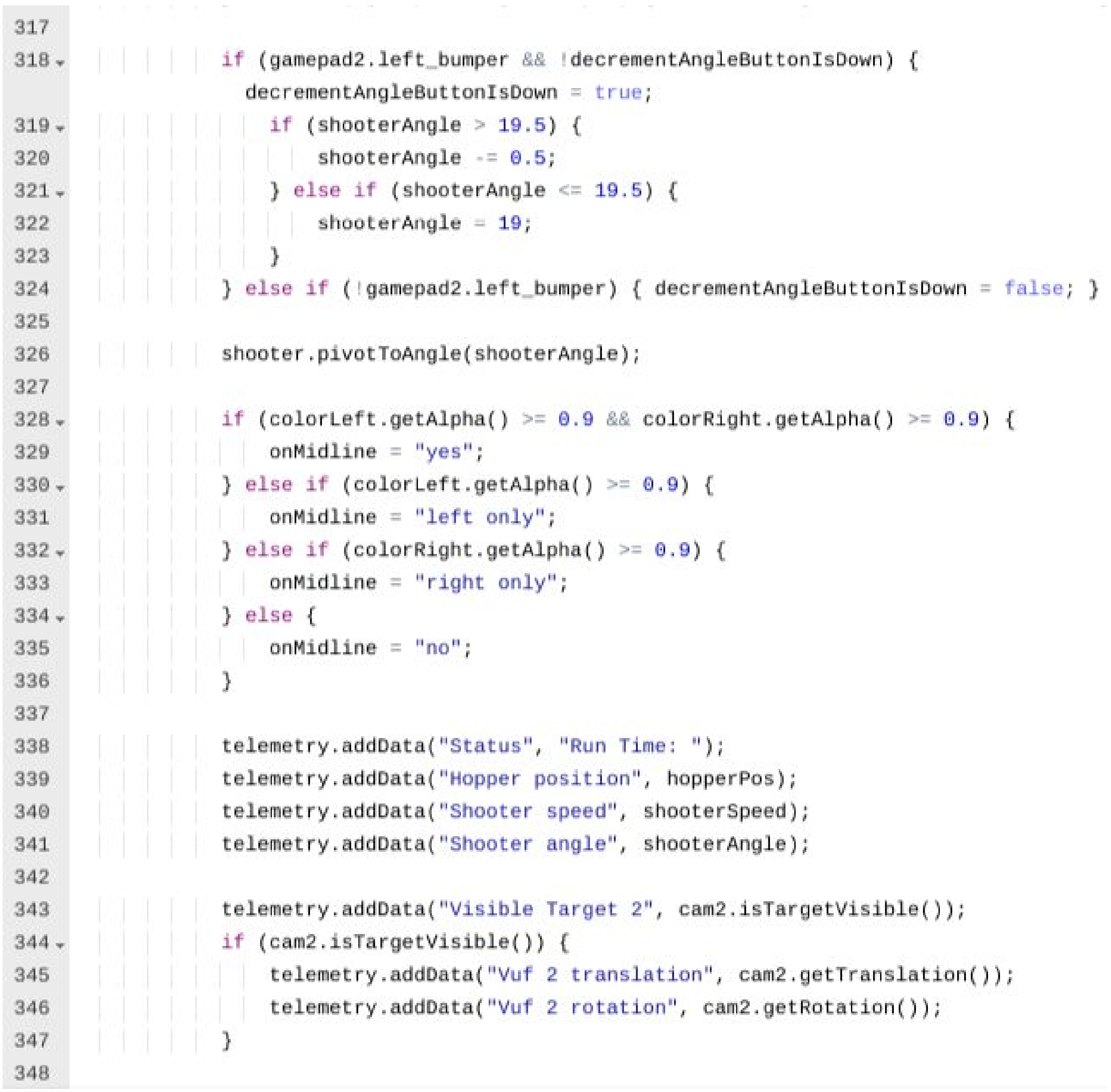
```
              hopperPos = "three";
              if (gear.incrementToPos("min")) {
                  col.setPower(1);
              }
          } else if (gamepad2.left_stick_y != 0) {
              if (gear.incrementToPos("max")) { //incrementToPos will both move the gearbox servo AND return if it has reached the goal yet
                  if (!col.isBusy()) {
                      col.setPower(0.5 * gamepad2.left_stick_y);
                  }
              }
          } else { //isPos will check if servo is at the specified goal, does not move servo like incrementToPos does
              col.rest();
          }

          if (gamepad1.a && !turningButtonIsDown) {
              turningButtonIsDown = true;
              // auto rotation towards the goal ==================================================
              if (state == "rotate") {
                  state = "drive";
              } else {
                  state = "rotate";
                  d.resetAllEncoders();
                  float heading = 0f;
                  float x = 0f;
                  float y = 0f;
                      if (cam2.isTargetVisible()) {
                      heading = -90 - cam2.getHeading();
                      x = 72 - cam2.getPositionX();
                      y = 36 - cam2.getPositionY();
                      theta = (float) (Math.atan2(y, x) * (180/Math.PI));
```



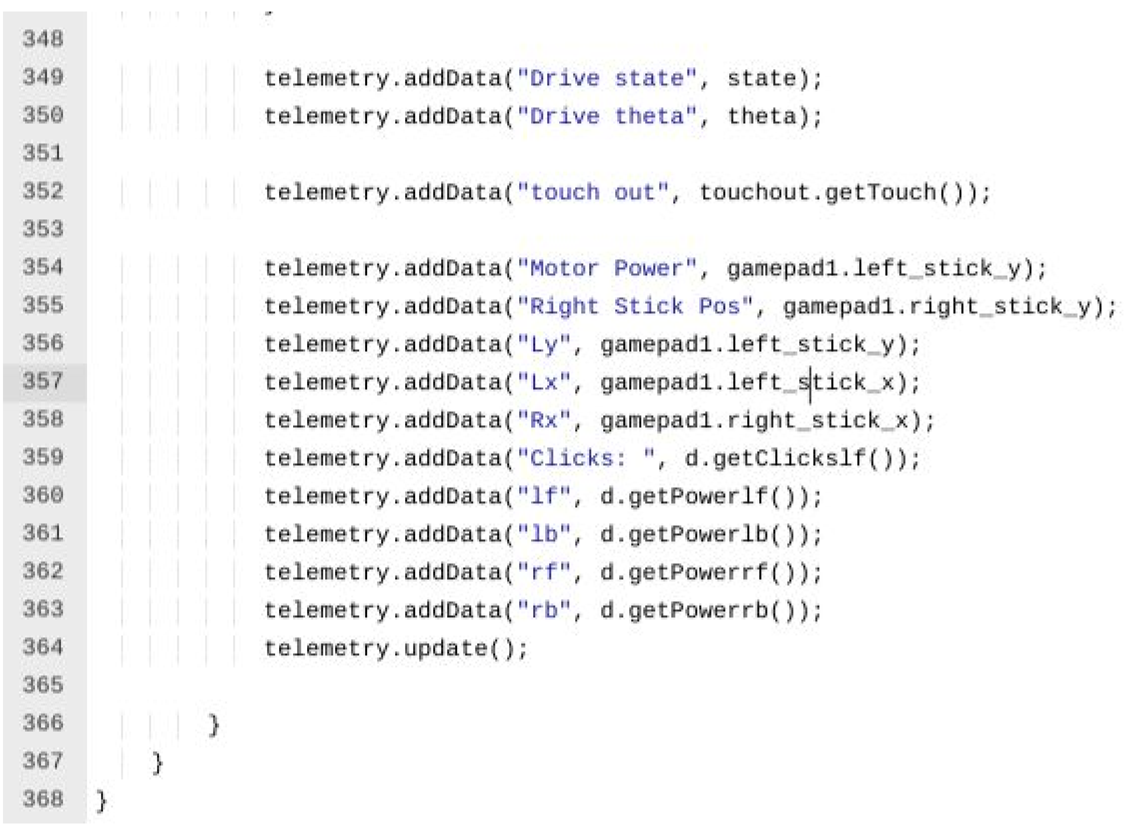
```
                      if (theta >= 0) {
                          float rotationAngle = (heading + (theta*1.2f));
                          d.rotateToAngle(rotationAngle, -0.5); // counter clockwise rotation

                      } else if (theta < 0) {
                          float rotationAngle = (heading + theta);
                          d.rotateToAngle(360-rotationAngle - 2, 0.5); // clockwise rotation
                      }
                  }
              }
              // auto pivoting towards the goal ==================================================
              double distToGoal;
              if(cam2.isTargetVisible()) {
                  double x = 72 - cam2.getPositionX() + 1.125;
                  double y = 36 - cam2.getPositionY();
                  distToGoal = Math.sqrt((x * x) + (y * y));
                  shooterAngle = (0.00209513)*(distToGoal-104.975)*(distToGoal-104.975) + 27.3575; //quadratic line of best fit equation
                  shooter.pivotToAngle(shooterAngle);
                  shooterSpeed = (-0.0343407)*(distToGoal-104.532)*(distToGoal-104.532) + 2137.34; //quadratic line of best fit equation
              }
          } else if (!gamepad1.a) { turningButtonIsDown = false; }

          cam2.trackPosition();

          //manually set the position of the hopper servo
          if (gamepad2.dpad_up && !dpadUpIsDown) { dpadUpIsDown = true;
              hopperPos = "zero";
```



```
          } else if (!gamepad2.dpad_up) { dpadUpIsDown = false; }

          if (gamepad2.dpad_right && !dpadRightIsDown) { dpadRightIsDown = true;
              hopperPos = "one";
          } else if (!gamepad2.dpad_right) { dpadRightIsDown = false; }

          if (gamepad2.dpad_down && !dpadDownIsDown) { dpadDownIsDown = true;
              hopperPos = "two";
          } else if (!gamepad2.dpad_down) { dpadDownIsDown = false; }

          if (gamepad2.dpad_left && !dpadLeftIsDown) { dpadLeftIsDown = true;
              hopperPos = "three";
          } else if (!gamepad2.dpad_left) { dpadLeftIsDown = false; }

          hopper.incrementToPos(hopperPos);
          if (gamepad2.b) {
              shooter.setSpeed(shooterSpeed);
              hopper.out();
          } else if (!gamepad2.b) {
              hopper.rest();
              shooter.rest();
          }

          //Increment shoooter angle using gamepad 2 left and right bumpers

          if (gamepad2.right_bumper && !incrementAngleButtonIsDown) { incrementAngleButtonIsDown = true;
              if (shooterAngle < 29.5) {
                  shooterAngle += 0.5;
              } else if (shooterAngle >= 29.5) {
                  shooterAngle = 30;
              }
          } else if (!gamepad2.right_bumper) { incrementAngleButtonIsDown = false; }
```



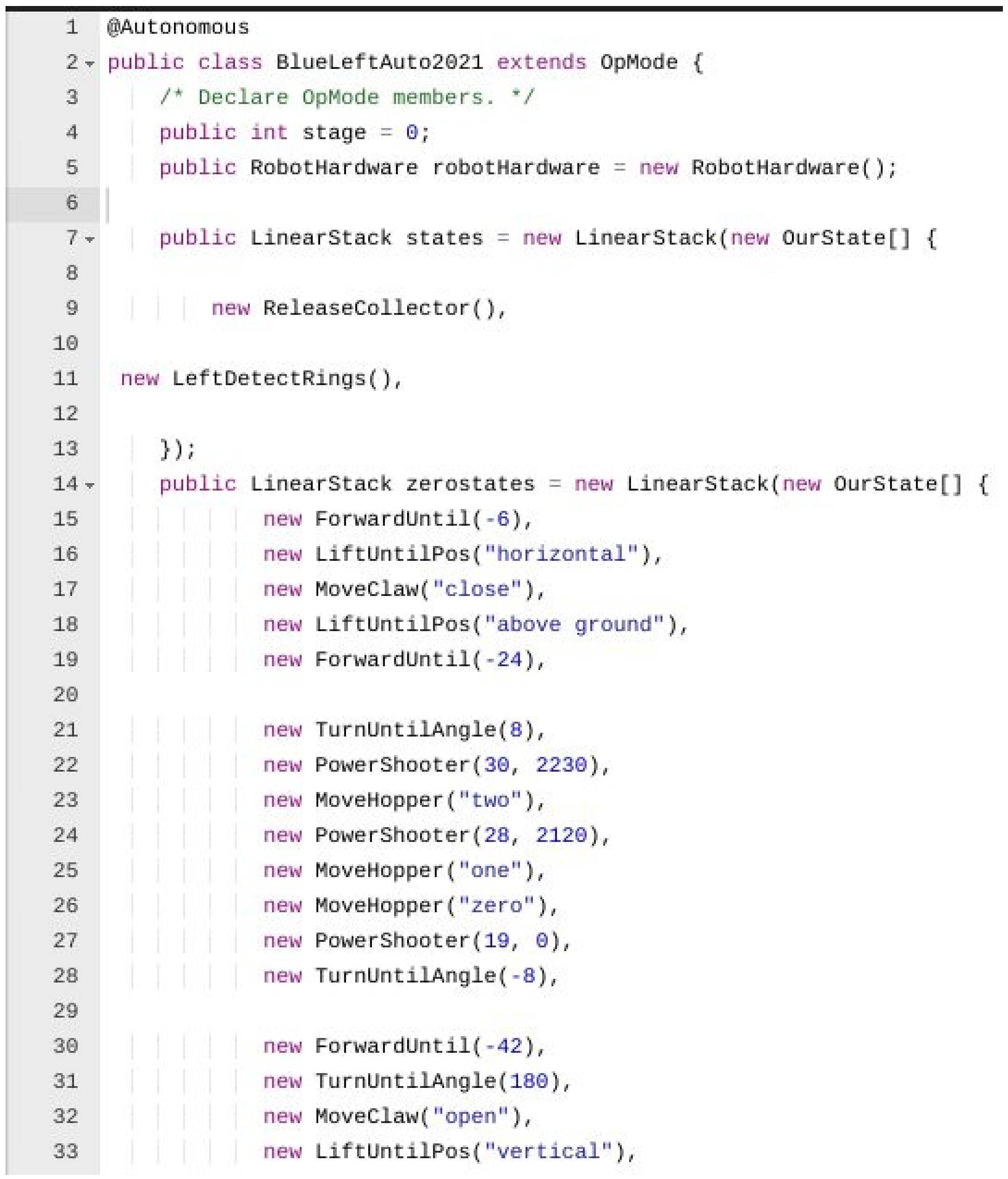
```

          if (gamepad2.left_bumper && !decrementAngleButtonIsDown) {
           decrementAngleButtonIsDown = true;
            if (shooterAngle > 19.5) {
              shooterAngle -= 0.5;
            } else if (shooterAngle <= 19.5) {
              shooterAngle = 19;
            }
          } else if (!gamepad2.left_bumper) { decrementAngleButtonIsDown = false; }

          shooter.pivotToAngle(shooterAngle);

          if (colorLeft.getAlpha() >= 0.9 && colorRight.getAlpha() >= 0.9) {
            onMidline = "yes";
          } else if (colorLeft.getAlpha() >= 0.9) {
            onMidline = "left only";
          } else if (colorRight.getAlpha() >= 0.9) {
            onMidline = "right only";
          } else {
            onMidline = "no";
          }

          telemetry.addData("Status", "Run Time: ");
          telemetry.addData("Hopper position", hopperPos);
          telemetry.addData("Shooter speed", shooterSpeed);
          telemetry.addData("Shooter angle", shooterAngle);

          telemetry.addData("Visible Target 2", cam2.isTargetVisible());
          if (cam2.isTargetVisible()) {
            telemetry.addData("Vuf 2 translation", cam2.getTranslation());
            telemetry.addData("Vuf 2 rotation", cam2.getRotation());
          }

```



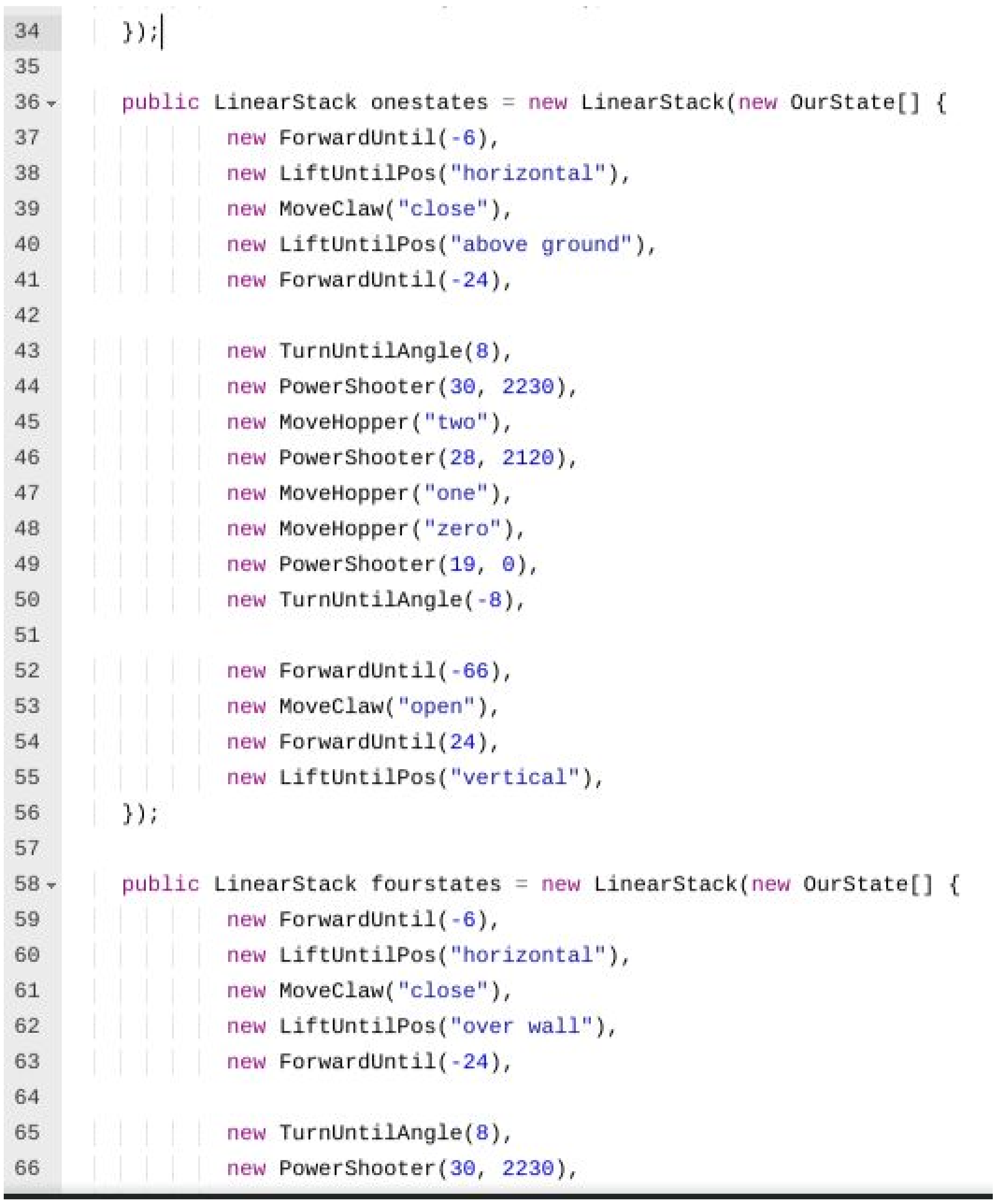
```

                        telemetry.addData("Drive state", state);
                        telemetry.addData("Drive theta", theta);

                        telemetry.addData("touch out", touchout.getTouch());

                        telemetry.addData("Motor Power", gamepad1.left_stick_y);
                        telemetry.addData("Right Stick Pos", gamepad1.right_stick_y);
                        telemetry.addData("Ly", gamepad1.left_stick_y);
                        telemetry.addData("Lx", gamepad1.left_stick_x);
                        telemetry.addData("Rx", gamepad1.right_stick_x);
                        telemetry.addData("Clicks: ", d.getClickslf());
                        telemetry.addData("lf", d.getPowerlf());
                        telemetry.addData("lb", d.getPowerlb());
                        telemetry.addData("rf", d.getPowerrf());
                        telemetry.addData("rb", d.getPowerrb());
                        telemetry.update();

                }
        }
}
```

BlueLeftAuto2021.java

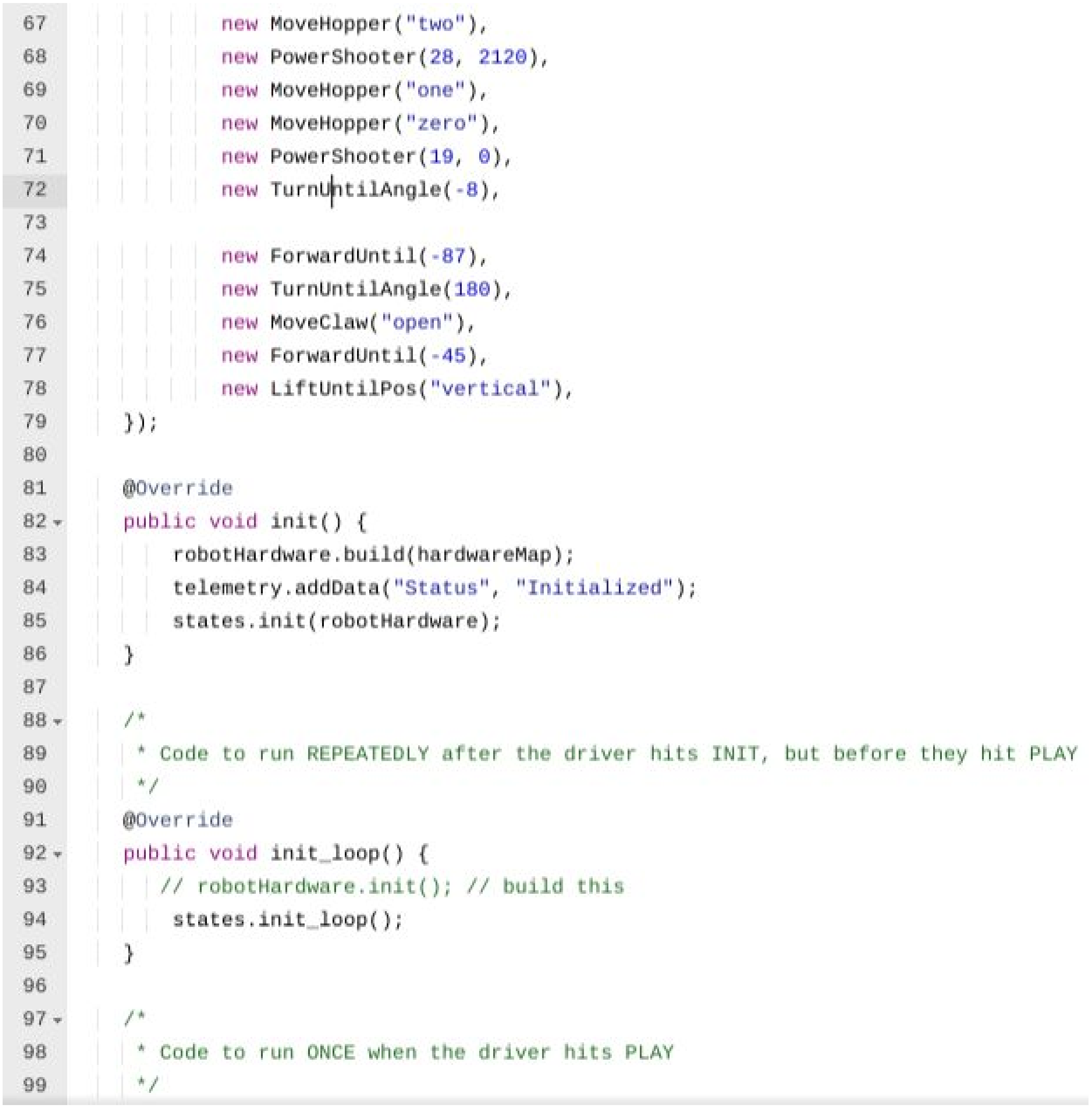
BlueLeftAuto2021 controls our robot during the Autonomous period when we start in an outer starting position. We use a state engine system that is easy to manipulate. By calling various states that each correspond to a certain task for the robot's hardware, we avoid getting stuck in "while" loops and create an efficient program. In the program, you can see our linear state engine structure as well as the various states we call.



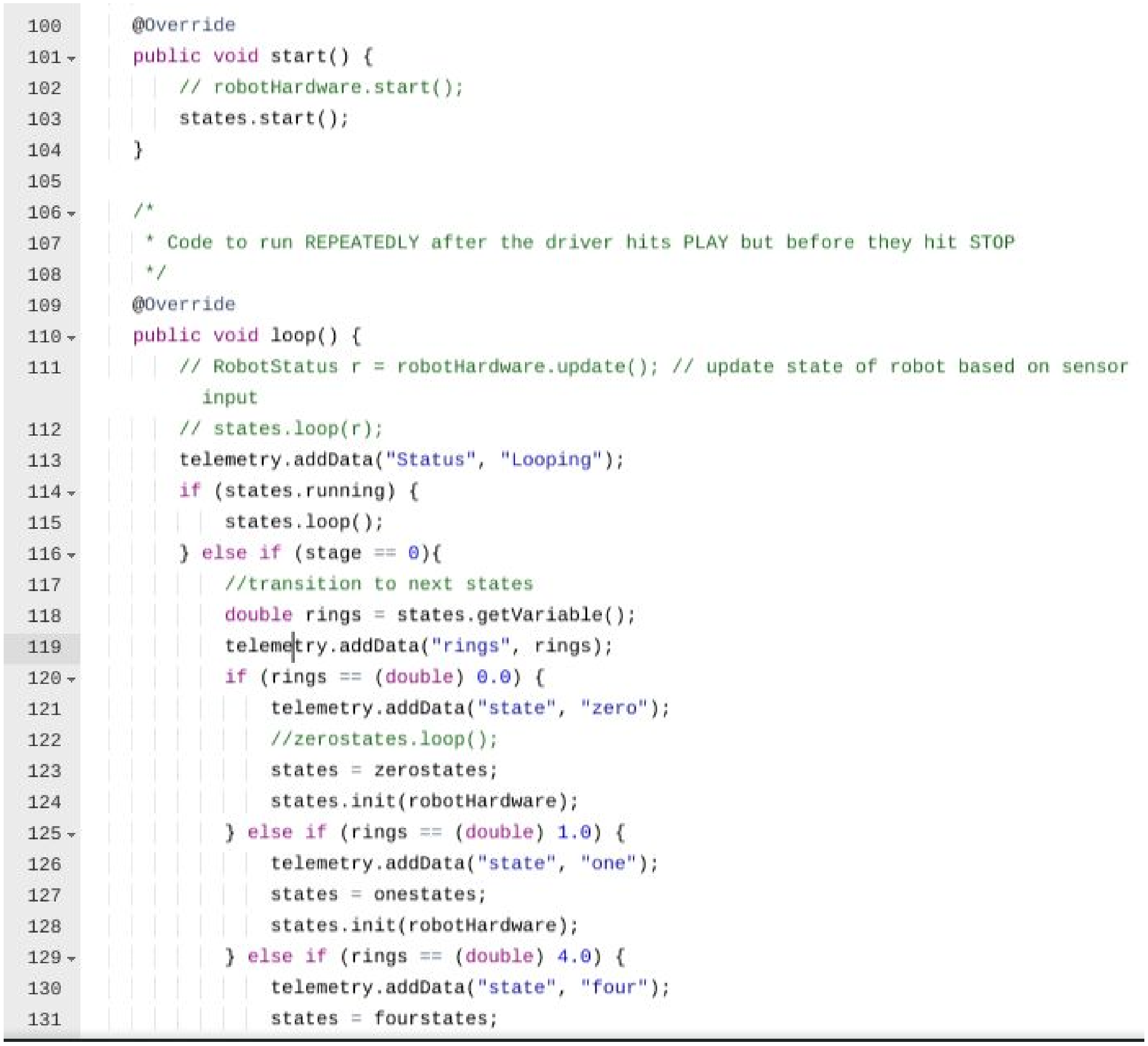
```
@Autonomous
public class BlueLeftAuto2021 extends OpMode {
    /* Declare OpMode members. */
    public int stage = 0;
    public RobotHardware robotHardware = new RobotHardware();

    public LinearStack states = new LinearStack(new OurState[] {

            new ReleaseCollector(),

 new LeftDetectRings(),

    });
    public LinearStack zerostates = new LinearStack(new OurState[] {
                new ForwardUntil(-6),
                new LiftUntilPos("horizontal"),
                new MoveClaw("close"),
                new LiftUntilPos("above ground"),
                new ForwardUntil(-24),

                new TurnUntilAngle(8),
                new PowerShooter(30, 2230),
                new MoveHopper("two"),
                new PowerShooter(28, 2120),
                new MoveHopper("one"),
                new MoveHopper("zero"),
                new PowerShooter(19, 0),
                new TurnUntilAngle(-8),

                new ForwardUntil(-42),
                new TurnUntilAngle(180),
                new MoveClaw("open"),
                new LiftUntilPos("vertical"),
```



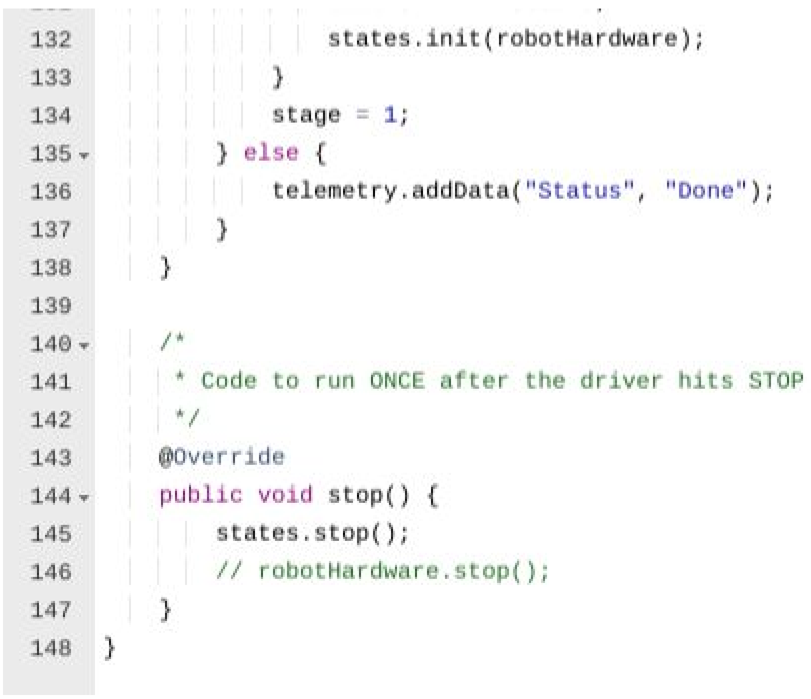
```
    });

    public LinearStack onestates = new LinearStack(new OurState[] {
            new ForwardUntil(-6),
            new LiftUntilPos("horizontal"),
            new MoveClaw("close"),
            new LiftUntilPos("above ground"),
            new ForwardUntil(-24),

            new TurnUntilAngle(8),
            new PowerShooter(30, 2230),
            new MoveHopper("two"),
            new PowerShooter(28, 2120),
            new MoveHopper("one"),
            new MoveHopper("zero"),
            new PowerShooter(19, 0),
            new TurnUntilAngle(-8),

            new ForwardUntil(-66),
            new MoveClaw("open"),
            new ForwardUntil(24),
            new LiftUntilPos("vertical"),
    });

    public LinearStack fourstates = new LinearStack(new OurState[] {
            new ForwardUntil(-6),
            new LiftUntilPos("horizontal"),
            new MoveClaw("close"),
            new LiftUntilPos("over wall"),
            new ForwardUntil(-24),

            new TurnUntilAngle(8),
            new PowerShooter(30, 2230),
```



```
                    new MoveHopper("two"),
                    new PowerShooter(28, 2120),
                    new MoveHopper("one"),
                    new MoveHopper("zero"),
                    new PowerShooter(19, 0),
                    new TurnUntilAngle(-8),

                    new ForwardUntil(-87),
                    new TurnUntilAngle(180),
                    new MoveClaw("open"),
                    new ForwardUntil(-45),
                    new LiftUntilPos("vertical"),
    });

    @Override
    public void init() {
        robotHardware.build(hardwareMap);
        telemetry.addData("Status", "Initialized");
        states.init(robotHardware);
    }

    /*
     * Code to run REPEATEDLY after the driver hits INIT, but before they hit PLAY
     */
    @Override
    public void init_loop() {
       // robotHardware.init(); // build this
        states.init_loop();
    }

    /*
     * Code to run ONCE when the driver hits PLAY
     */
```



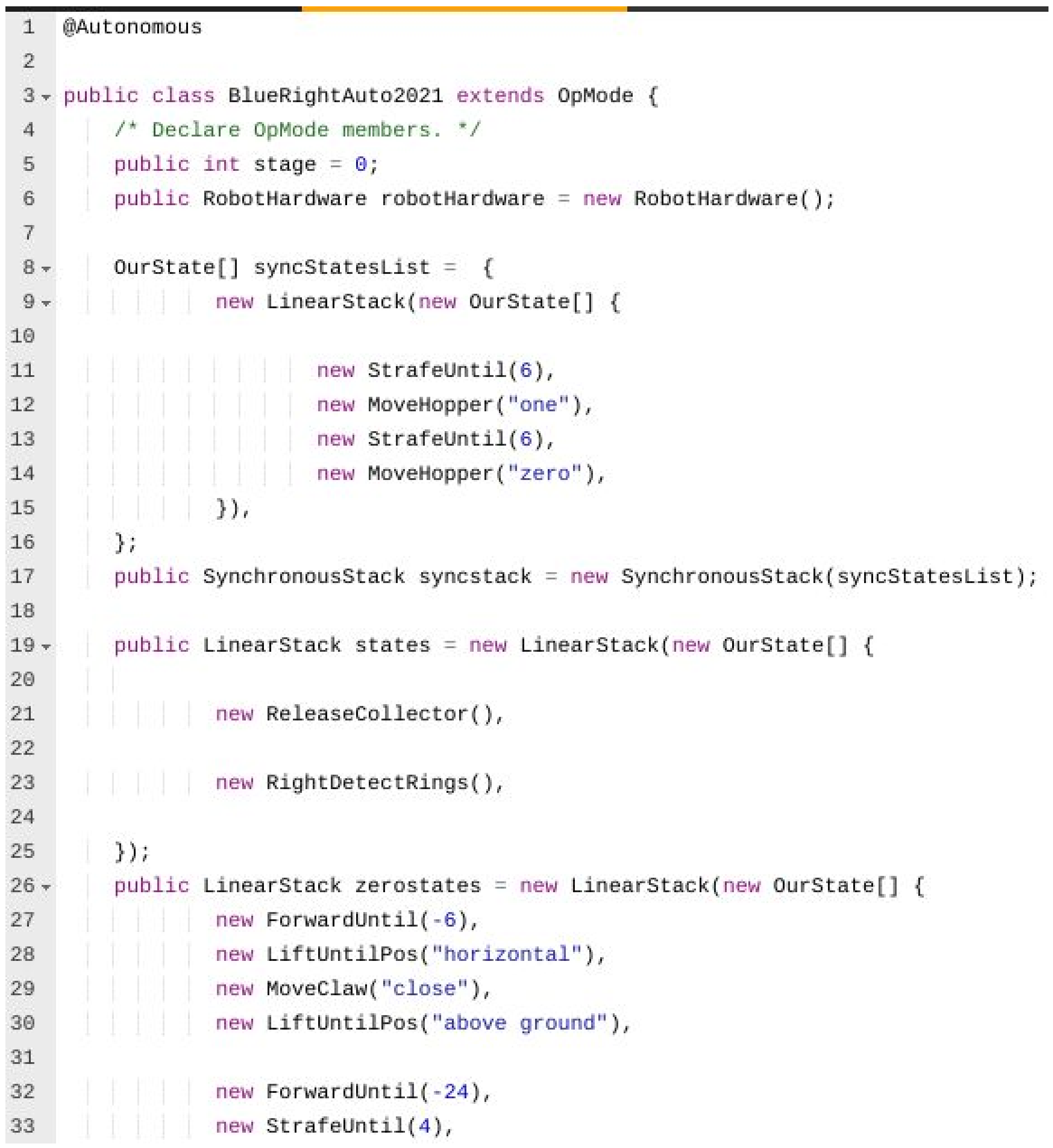
```
    @Override
    public void start() {
        // robotHardware.start();
        states.start();
    }

    /*
     * Code to run REPEATEDLY after the driver hits PLAY but before they hit STOP
     */
    @Override
    public void loop() {
        // RobotStatus r = robotHardware.update(); // update state of robot based on sensor
          input
        // states.loop(r);
        telemetry.addData("Status", "Looping");
        if (states.running) {
            states.loop();
        } else if (stage == 0){
            //transition to next states
            double rings = states.getVariable();
            telemetry.addData("rings", rings);
            if (rings == (double) 0.0) {
                telemetry.addData("state", "zero");
                //zerostates.loop();
                states = zerostates;
                states.init(robotHardware);
            } else if (rings == (double) 1.0) {
                telemetry.addData("state", "one");
                states = onestates;
                states.init(robotHardware);
            } else if (rings == (double) 4.0) {
                telemetry.addData("state", "four");
                states = fourstates;
```



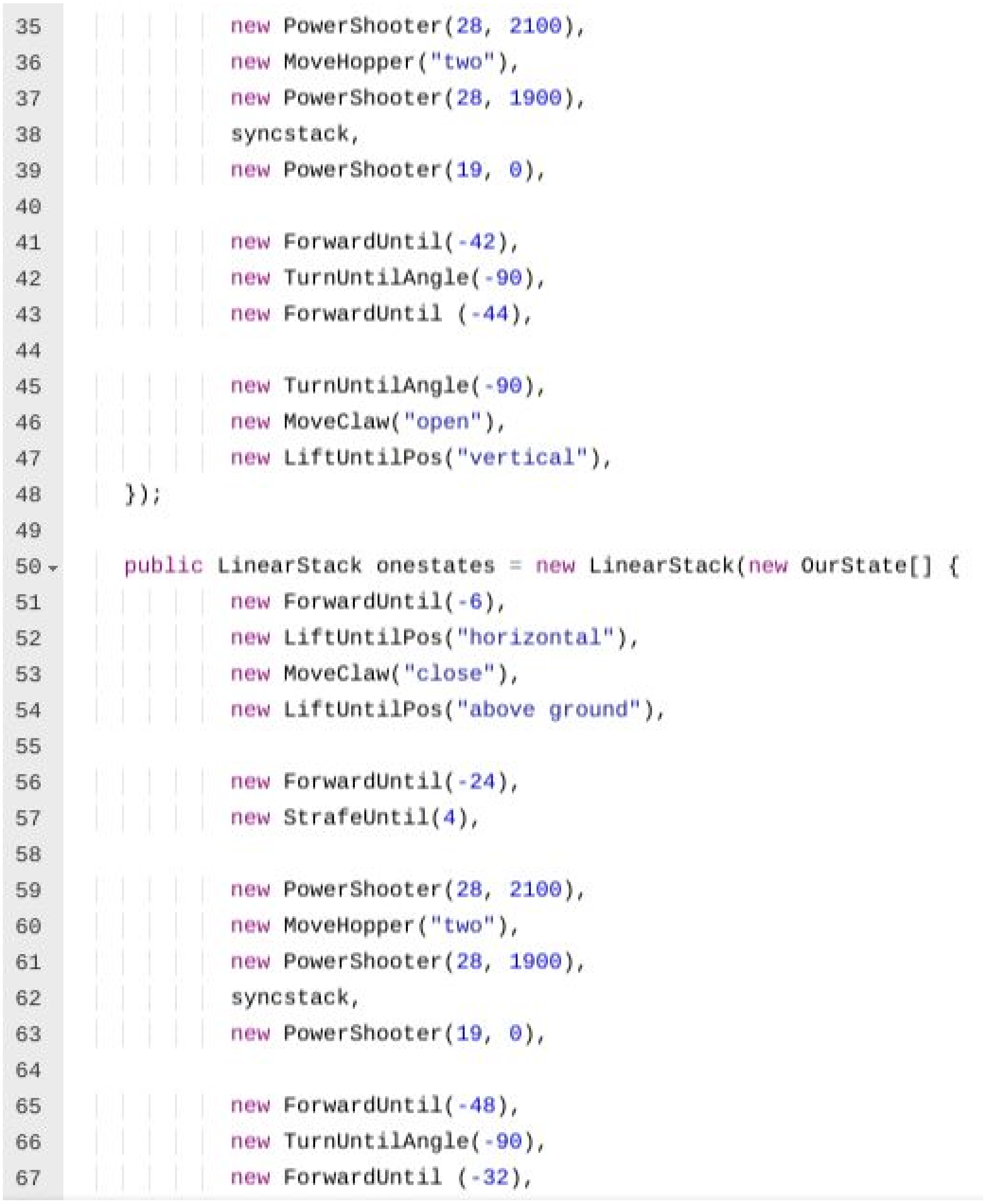
```
                states.init(robotHardware);
            }
            stage = 1;
        } else {
            telemetry.addData("Status", "Done");
        }
    }

    /*
     * Code to run ONCE after the driver hits STOP
     */
    @Override
    public void stop() {
        states.stop();
        // robotHardware.stop();
    }
}
```

<u>BlueRightAuto2021.java</u>
BlueRightAuto2021 controls our robot during the Autonomous period when we start in an inner starting position. In addition to our linear state engine, it contains a synchronous state stack which we insert into the code. This synchronous engine allows us to run multiple states at once. We have omitted the latter part of this program, which is identical to that of BlueLeftAuto2021.java.

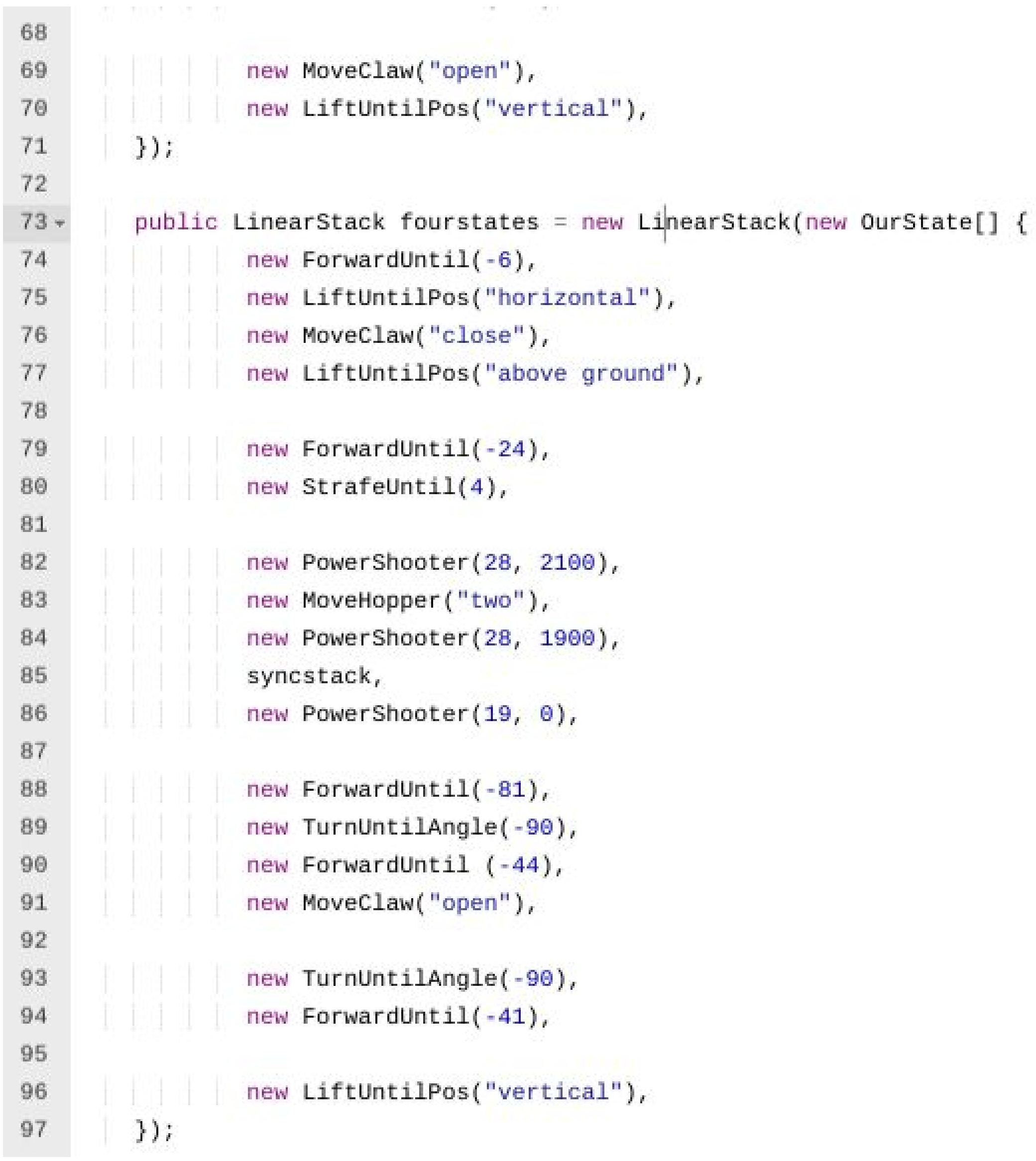
To see the implementation of our synchronous state engine in this autonomous program, see **lines 8-16** below.



```
1   @Autonomous
2
3   public class BlueRightAuto2021 extends OpMode {
4       /* Declare OpMode members. */
5       public int stage = 0;
6       public RobotHardware robotHardware = new RobotHardware();
7
8       OurState[] syncStatesList =  {
9               new LinearStack(new OurState[] {
10
11                      new StrafeUntil(6),
12                      new MoveHopper("one"),
13                      new StrafeUntil(6),
14                      new MoveHopper("zero"),
15              }),
16      };
17      public SynchronousStack syncstack = new SynchronousStack(syncStatesList);
18
19      public LinearStack states = new LinearStack(new OurState[] {
20
21              new ReleaseCollector(),
22
23              new RightDetectRings(),
24
25      });
26      public LinearStack zerostates = new LinearStack(new OurState[] {
27              new ForwardUntil(-6),
28              new LiftUntilPos("horizontal"),
29              new MoveClaw("close"),
30              new LiftUntilPos("above ground"),
31
32              new ForwardUntil(-24),
33              new StrafeUntil(4),
```

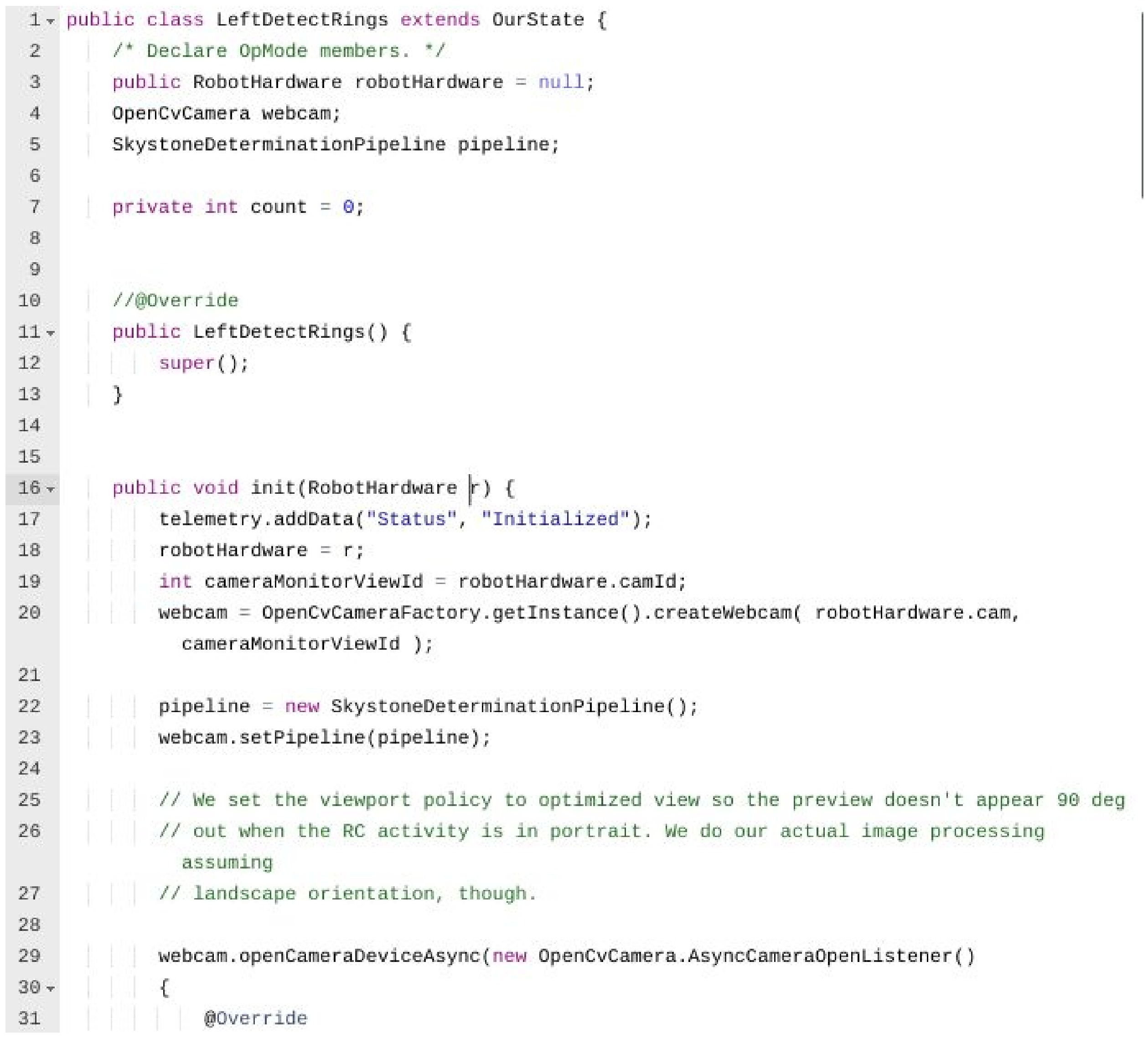


```
                new PowerShooter(28, 2100),
                new MoveHopper("two"),
                new PowerShooter(28, 1900),
                syncstack,
                new PowerShooter(19, 0),

                new ForwardUntil(-42),
                new TurnUntilAngle(-90),
                new ForwardUntil (-44),

                new TurnUntilAngle(-90),
                new MoveClaw("open"),
                new LiftUntilPos("vertical"),
    });

    public LinearStack onestates = new LinearStack(new OurState[] {
                new ForwardUntil(-6),
                new LiftUntilPos("horizontal"),
                new MoveClaw("close"),
                new LiftUntilPos("above ground"),

                new ForwardUntil(-24),
                new StrafeUntil(4),

                new PowerShooter(28, 2100),
                new MoveHopper("two"),
                new PowerShooter(28, 1900),
                syncstack,
                new PowerShooter(19, 0),

                new ForwardUntil(-48),
                new TurnUntilAngle(-90),
                new ForwardUntil (-32),
```

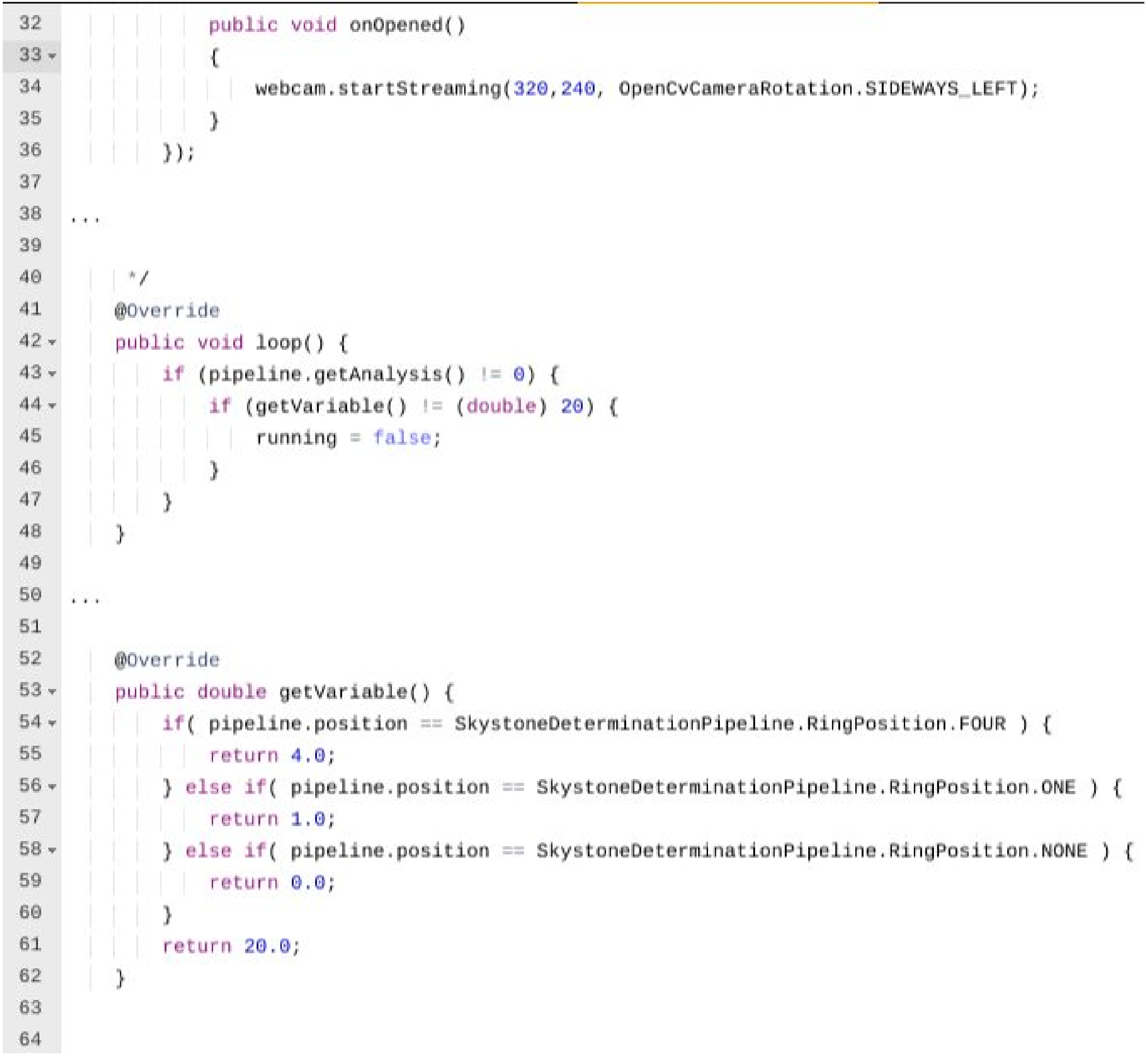


```

          new MoveClaw("open"),
          new LiftUntilPos("vertical"),
  });

  public LinearStack fourstates = new LinearStack(new OurState[] {
          new ForwardUntil(-6),
          new LiftUntilPos("horizontal"),
          new MoveClaw("close"),
          new LiftUntilPos("above ground"),

          new ForwardUntil(-24),
          new StrafeUntil(4),

          new PowerShooter(28, 2100),
          new MoveHopper("two"),
          new PowerShooter(28, 1900),
          syncstack,
          new PowerShooter(19, 0),

          new ForwardUntil(-81),
          new TurnUntilAngle(-90),
          new ForwardUntil (-44),
          new MoveClaw("open"),

          new TurnUntilAngle(-90),
          new ForwardUntil(-41),

          new LiftUntilPos("vertical"),
  });
```

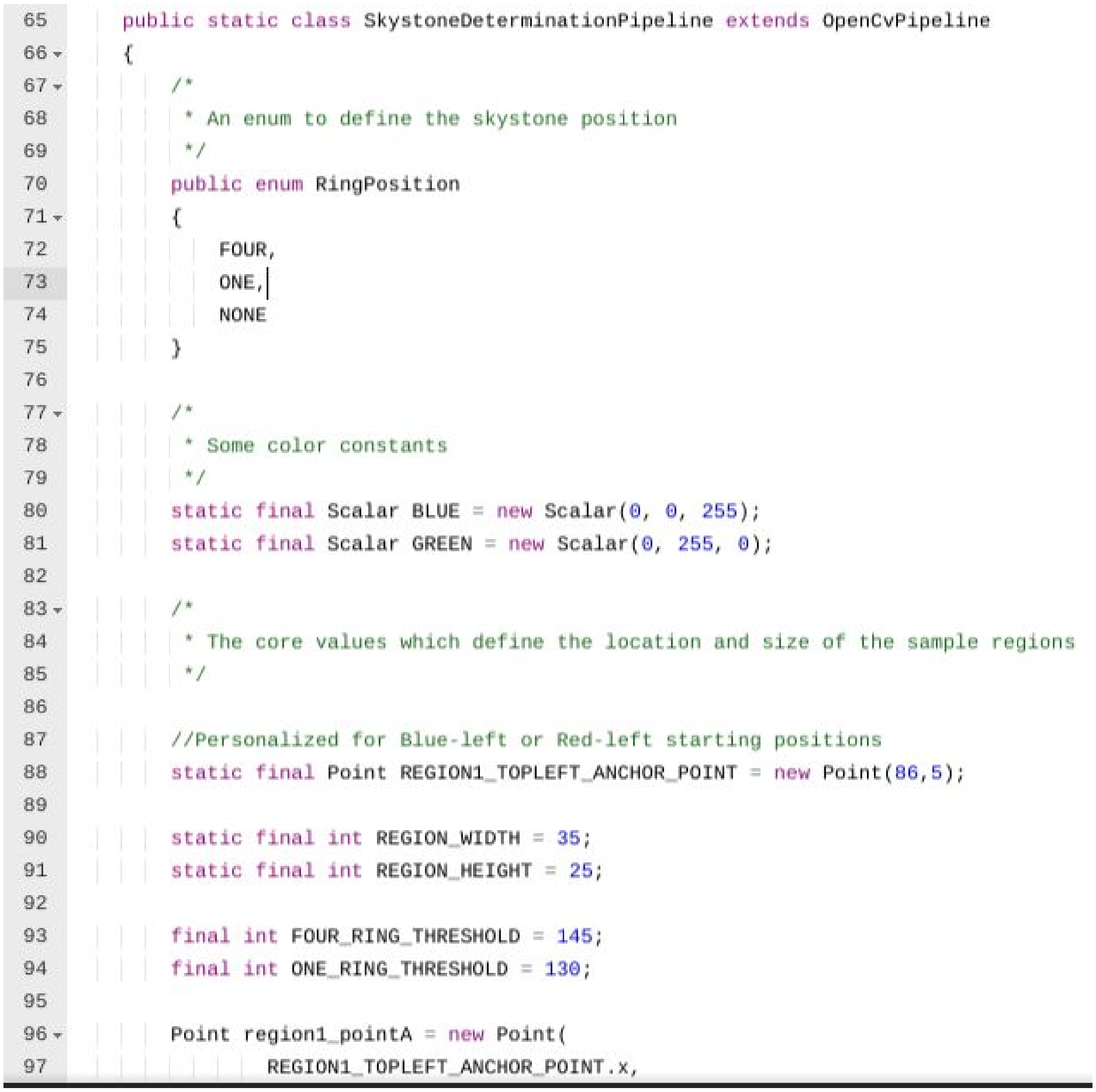
<u>DetectRings.java</u>
DetectRings is one of the states that we use as part of our Autonomous program. During this state, our robot uses OpenCV to detect the number of rings in the starting stack.
We do this by detecting the colors of the pixels in a targeted portion of a webcam feed (the webcam is pointed towards the starting ring stack). The higher the number of orange pixels, the more rings are in the stack. Please see more details for this state in the lines below.



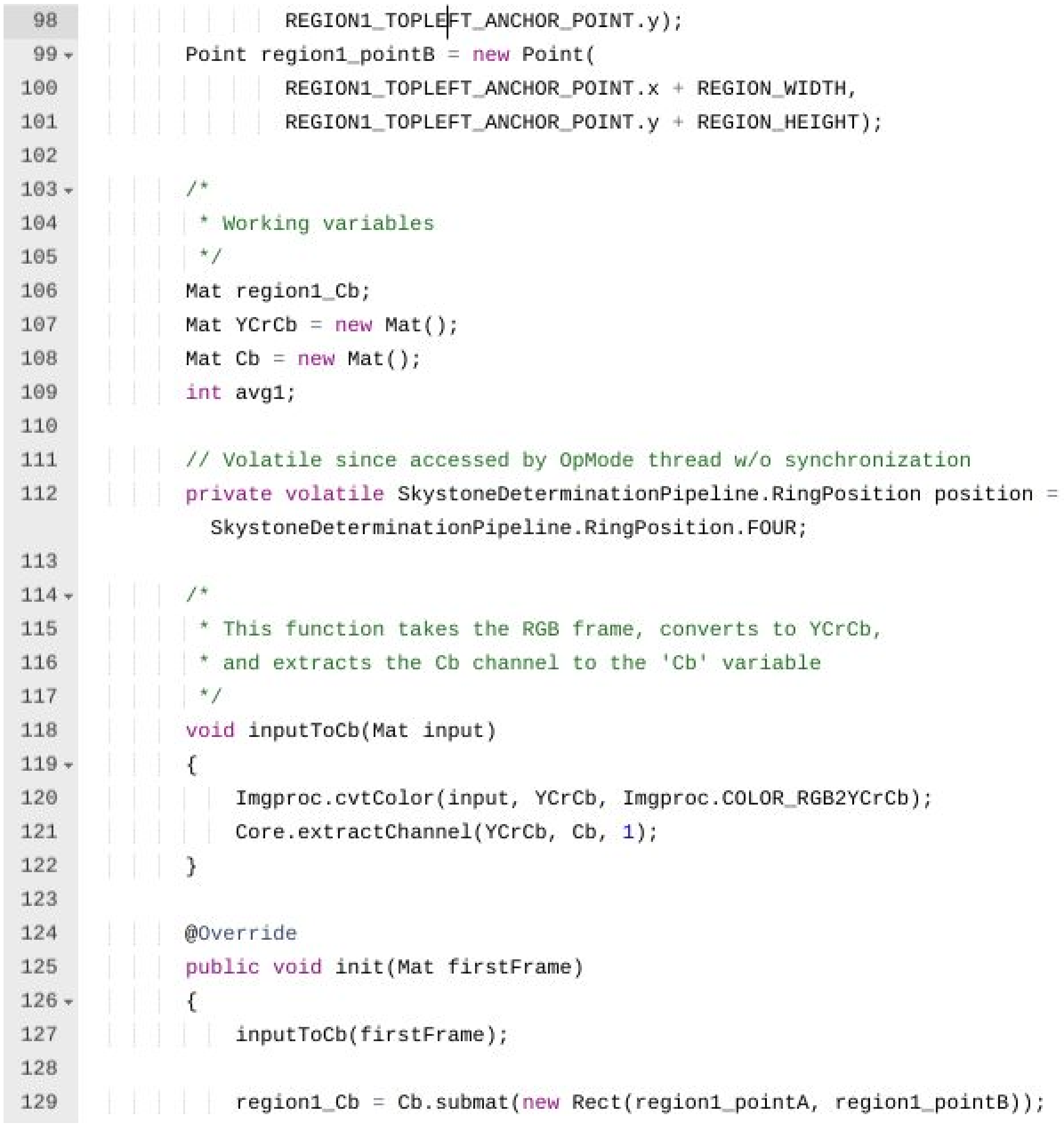
```java
public class LeftDetectRings extends OurState {
    /* Declare OpMode members. */
    public RobotHardware robotHardware = null;
    OpenCvCamera webcam;
    SkystoneDeterminationPipeline pipeline;

    private int count = 0;


    //@Override
    public LeftDetectRings() {
        super();
    }


    public void init(RobotHardware r) {
        telemetry.addData("Status", "Initialized");
        robotHardware = r;
        int cameraMonitorViewId = robotHardware.camId;
        webcam = OpenCvCameraFactory.getInstance().createWebcam( robotHardware.cam, cameraMonitorViewId );

        pipeline = new SkystoneDeterminationPipeline();
        webcam.setPipeline(pipeline);

        // We set the viewport policy to optimized view so the preview doesn't appear 90 deg
        // out when the RC activity is in portrait. We do our actual image processing assuming
        // landscape orientation, though.

        webcam.openCameraDeviceAsync(new OpenCvCamera.AsyncCameraOpenListener()
        {
            @Override
```



```
            public void onOpened()
            {
                webcam.startStreaming(320,240, OpenCvCameraRotation.SIDEWAYS_LEFT);
            }
        });

...

     */
    @Override
    public void loop() {
        if (pipeline.getAnalysis() != 0) {
            if (getVariable() != (double) 20) {
                running = false;
            }
        }
    }

...

    @Override
    public double getVariable() {
        if( pipeline.position == SkystoneDeterminationPipeline.RingPosition.FOUR ) {
            return 4.0;
        } else if( pipeline.position == SkystoneDeterminationPipeline.RingPosition.ONE ) {
            return 1.0;
        } else if( pipeline.position == SkystoneDeterminationPipeline.RingPosition.NONE ) {
            return 0.0;
        }
        return 20.0;
    }
```



```
public static class SkystoneDeterminationPipeline extends OpenCvPipeline
{
    /*
     * An enum to define the skystone position
     */
    public enum RingPosition
    {
        FOUR,
        ONE,
        NONE
    }

    /*
     * Some color constants
     */
    static final Scalar BLUE = new Scalar(0, 0, 255);
    static final Scalar GREEN = new Scalar(0, 255, 0);

    /*
     * The core values which define the location and size of the sample regions
     */

    //Personalized for Blue-left or Red-left starting positions
    static final Point REGION1_TOPLEFT_ANCHOR_POINT = new Point(86,5);

    static final int REGION_WIDTH = 35;
    static final int REGION_HEIGHT = 25;

    final int FOUR_RING_THRESHOLD = 145;
    final int ONE_RING_THRESHOLD = 130;

    Point region1_pointA = new Point(
            REGION1_TOPLEFT_ANCHOR_POINT.x,
```



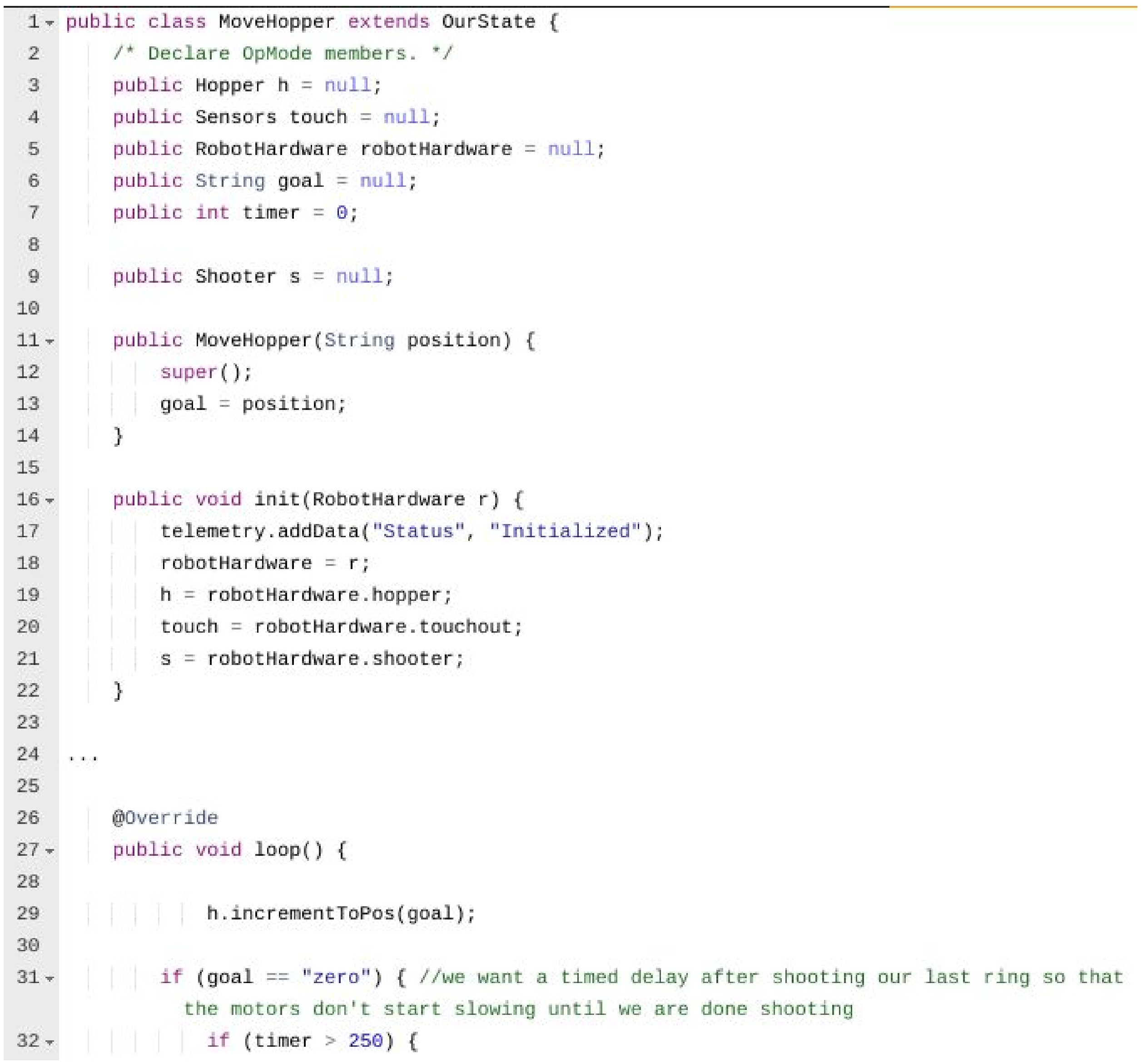
```
              REGION1_TOPLEFT_ANCHOR_POINT.y);
      Point region1_pointB = new Point(
              REGION1_TOPLEFT_ANCHOR_POINT.x + REGION_WIDTH,
              REGION1_TOPLEFT_ANCHOR_POINT.y + REGION_HEIGHT);

      /*
       * Working variables
       */
      Mat region1_Cb;
      Mat YCrCb = new Mat();
      Mat Cb = new Mat();
      int avg1;

      // Volatile since accessed by OpMode thread w/o synchronization
      private volatile SkystoneDeterminationPipeline.RingPosition position =
        SkystoneDeterminationPipeline.RingPosition.FOUR;

      /*
       * This function takes the RGB frame, converts to YCrCb,
       * and extracts the Cb channel to the 'Cb' variable
       */
      void inputToCb(Mat input)
      {
          Imgproc.cvtColor(input, YCrCb, Imgproc.COLOR_RGB2YCrCb);
          Core.extractChannel(YCrCb, Cb, 1);
      }

      @Override
      public void init(Mat firstFrame)
      {
          inputToCb(firstFrame);

          region1_Cb = Cb.submat(new Rect(region1_pointA, region1_pointB));
```

```java
        }

        @Override
        public Mat processFrame(Mat input)
        {
            inputToCb(input);

            avg1 = (int) Core.mean(region1_Cb).val[0];

            Imgproc.rectangle(
                    input, // Buffer to draw on
                    region1_pointA, // First point which defines the rectangle
                    region1_pointB, // Second point which defines the rectangle
                    BLUE, // The color the rectangle is drawn in
                    2); // Thickness of the rectangle lines

            position = SkystoneDeterminationPipeline.RingPosition.FOUR; // Record our analysis
            if(avg1 > FOUR_RING_THRESHOLD){
                position = SkystoneDeterminationPipeline.RingPosition.FOUR;
            }else if (avg1 > ONE_RING_THRESHOLD){
                position = SkystoneDeterminationPipeline.RingPosition.ONE;
            }else{
                position = SkystoneDeterminationPipeline.RingPosition.NONE;
            }

            Imgproc.rectangle(
                    input, // Buffer to draw on
                    region1_pointA, // First point which defines the rectangle
                    region1_pointB, // Second point which defines the rectangle
                    GREEN, // The color the rectangle is drawn in
                    -1); // Negative thickness means solid fill

            return input;
        }

        public int getAnalysis()
        {
            return avg1;
        }

    }

}
```

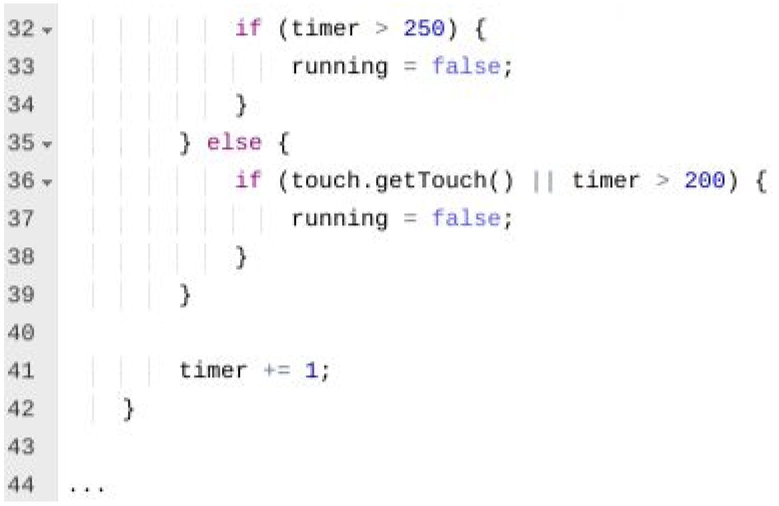
MoveHopper.java

MoveHopper is another state that we reference as part of our autonomous program. Since it moves the hopper, it is what we use to shoot during autonomous, It incorporates a touch sensor so that our robot can use that sensor input to track when rings have left the shooter. This allows us to properly time our robot’s movement while we are shooting during the autonomous period. Please see a brief overview of this state below.

As shown in **lines 36-38** this state waits for touch sensor input before progressing.



```
 1 public class MoveHopper extends OurState {
 2     /* Declare OpMode members. */
 3     public Hopper h = null;
 4     public Sensors touch = null;
 5     public RobotHardware robotHardware = null;
 6     public String goal = null;
 7     public int timer = 0;
 8 
 9     public Shooter s = null;
10 
11     public MoveHopper(String position) {
12         super();
13         goal = position;
14     }
15 
16     public void init(RobotHardware r) {
17         telemetry.addData("Status", "Initialized");
18         robotHardware = r;
19         h = robotHardware.hopper;
20         touch = robotHardware.touchout;
21         s = robotHardware.shooter;
22     }
23 
24 ...
25 
26     @Override
27     public void loop() {
28 
29                 h.incrementToPos(goal);
30 
31         if (goal == "zero") { //we want a timed delay after shooting our last ring so that the motors don't start slowing until we are done shooting
32             if (timer > 250) {
```
]



```
            if (timer > 250) {
                running = false;
            }
        } else {
            if (touch.getTouch() || timer > 200) {
                running = false;
            }
        }

        timer += 1;
    }

...
```

Points, Strategies:

Auto Points:

Start with 3 rings and wobble goal

Different starting field rings - 0,1,4

Wobble target to correct zone - 15 pts

Park over white line - 5 pts

Low Goal - 3 pts per ring

Medium goal - 6 pts per ring

High goal - 12pts per goal

Power Shot- 15pts per target

Shoot from behind launch line, except low goal

Human player is expected to return rings

Teleop Points:

Low goal - 2 pts per ring

Medium goal - 4 pts per ring

High goal - 6 pts per ring

Endgame Points:

Powershot - 15pts per target

Return wobble goal past start line - 5pts

Rong on wobble goal - 5pts per ring

Wobble goal in drop zone - 20pts

Perfect Auto-run (per robot) prediction

1. Shoot 3 preloaded discs at powershot
2. Move wobble goal to correct zone
3. Pick up rings from floor and shoot into high goal
4. Park on line

= 65, 77, or 113 pts

Perfect Teleop prediction (per robot):

1. Shoot all high goal targets ~ 8 cycles = 24 rings

=48 pts

Perfect Endgame prediction:

1. Shoot powershot targets
2. Continue shooting high goal ~ 1 or 2 cycles
3. Place wobble goal in drop zone

=108 pts

Total per robot = 221, 233, or 269 pts

Robot Goals (as of 9/20/20)

- Shooter Robot:
- Shoot upper goal and powershot
- Shoot from variety of positions (left, right, forward, backwards)
- Use code to aim
- Use as few motors for shooter as possible
- Use as less space as possible
- Reach:
- Shoot Mid-goal
- Shoot while intaking
- Wobble as 2nd Priority (changed to 1st priority for coders, 9/25/20)

Intake:

- Intake quickly, efficiently
- Stick outside of frame perimeter - easier to collect
- Durable
- Full Width
- Indexing:
- Hold 3 discs
- Efficient - No unnecessary parts
- Quick hand off from intake
- Quick handoff to shooter
- Prevent jamming

Prototyping Notes:

Random Robot Ideas:

Turret!!!!

One wheel Straight shooter

One wheel Arc shooter

2 wheel shooter

Catapult

Horizontal roller intake

Vertical Roller intake

Surgical Tube intake

Prototyping: Look at Kettering shooter - AndyMark Neverest 1:1 ~ 6600 rpm, 4in wheel

Shooter Variables:

Compression

Length of Contact with wheel

Wheel Size, Motor, Gearing - Linear speed of wheel

Straight or Arc

Accelerator wheel?

Secondary Wheel?

Number of Wheels?

Type of Wheel

Backing Material

Gearing Options for Shooter "Gearbox":

24:24

32:24

1:1.08 total = 1:3.7 - 32:16 - 32:16

1:0.81 total = 1:3.7 - 32:16 - 24:16

1:0.54 total = 1:3.7 - 32:16

1:0.72 total = 1:3.7 - 32:24 - 32:16

1:0.48 total = 1:3.7 - 32:24 - 32:24

1:0.36 total = 1:3.7 - 32:24

1:0.61 total = 1:3.7 - 24:16 - 24:16

1:0.41 total = 1:3.7 - 24:16

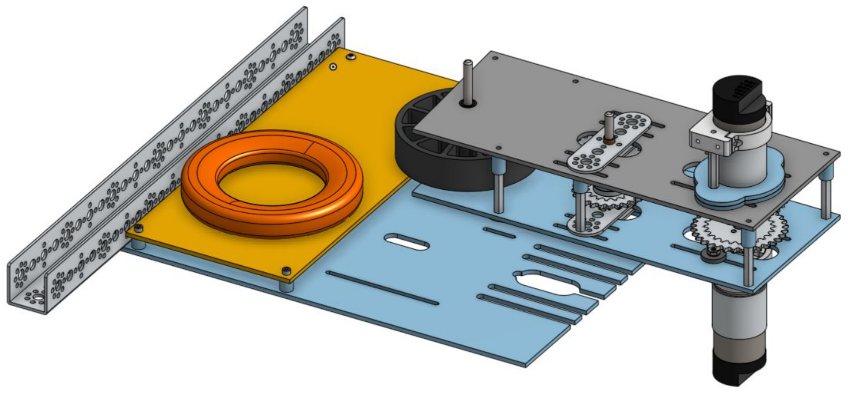
1:0.27 total = 1:3.7 - 24:24

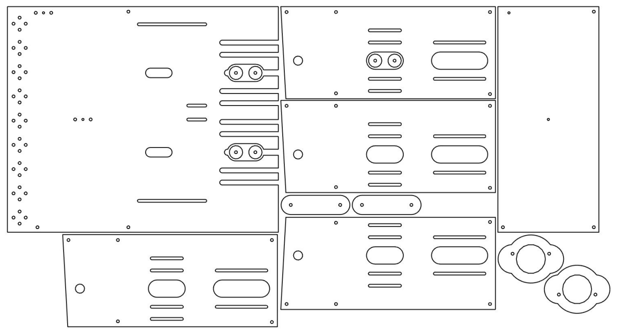
For first chain run(wheel to 1st stage) - 60 links

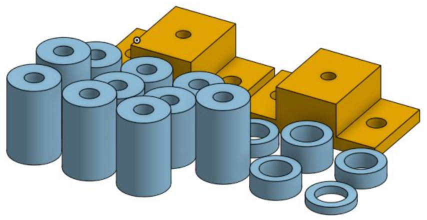
Second chain run(1st stage to 2nd stage) - 55 links

Intake Variables:
How many sets of wheels
Type of wheel/surgical tubing
Speed Compression

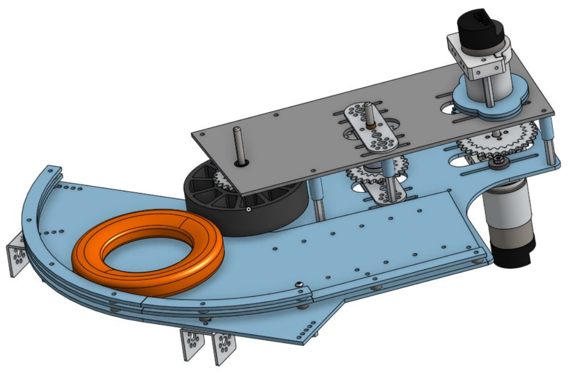
Designing:
Weekend Work Completed:
Straight shooter prototype is complete, and the inkscape file is ready for laser cutting. The spacers were simplified, and exported, so they are ready for 3d printing. Discussion needs to take place about whether the prototype needs a flywheel to maintain speed better, or have an external PID running to improve results.

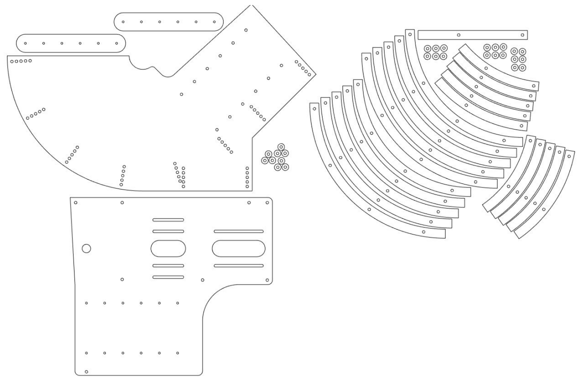




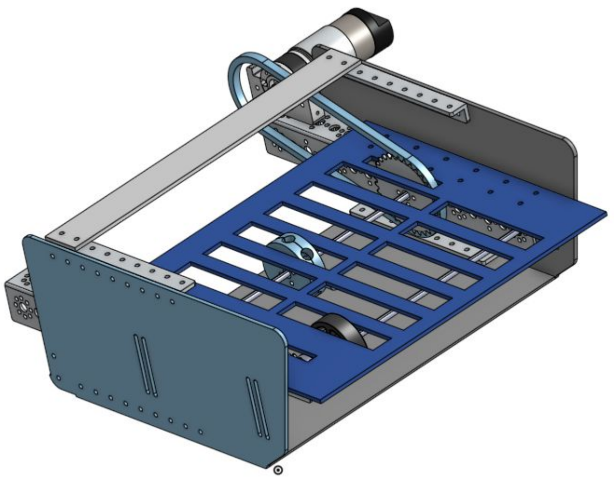


For the arc shooter prototype, a new bottom gearbox plate had to be made to include support for structure. The screws will be threaded directly in wood for simplicity, and the file is ready for laser cutting. The gearbox assembly is mostly the same as the other prototype.



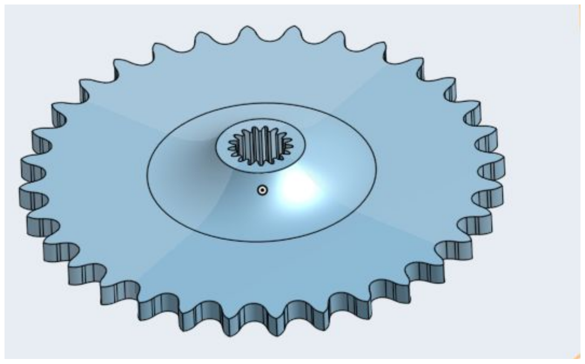


A chain run was added to the collector prototype, but it is still beefy, and needs optimization. more discussion needs to be done about testing procedures and motor specifications.

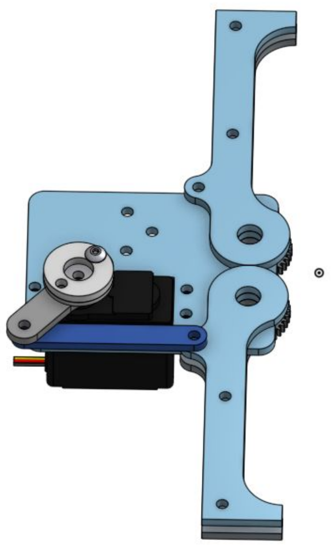


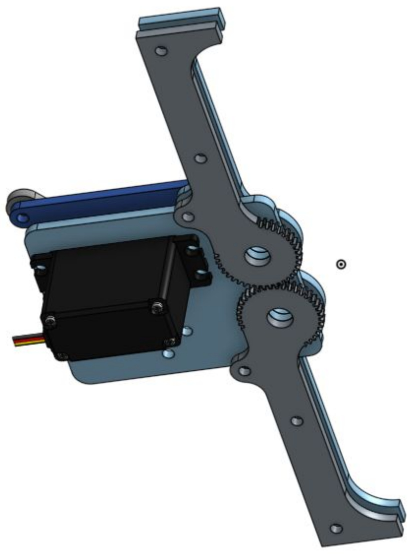
Afterwards, the file needs to be made ready to laser cut.

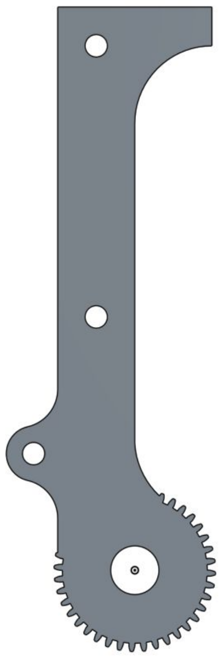
Custom 32t sprocket was made after the 24t sprocket printed successfully. These feature a specialized center that will interface with the pinion of the andymark 3.7 orbital motor.

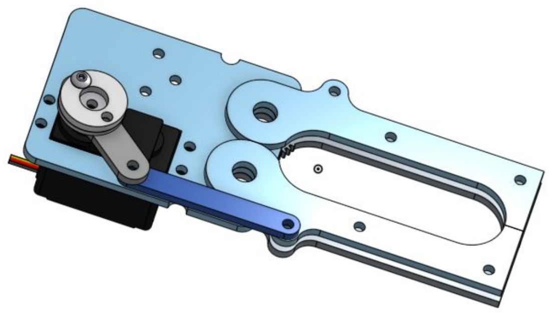


New claw shape was made for wobble goal grabbers. It is a slot design that allows easier grabbing of the wobble goal by allowing the goal to be a variety of distances away from the robot. This was created for a drawer slide lift, which was designed for the programmers to have a head start on autonomous testing. The autonomous chassis consists of a mecanum drive base, as well as a lift powered by string/spool. For the claw, it is powered by a servo and a 4 bar linkage. This means the servo won't take any of the sideways load of lifting the wobble goal. To actuate both arms at the same time, a custom gear mesh was created to allow the arms to move towards each other at the same rate. The gear is part of a 40DP 40T gear. The higher diametral pitch allows finer and more precise movement.

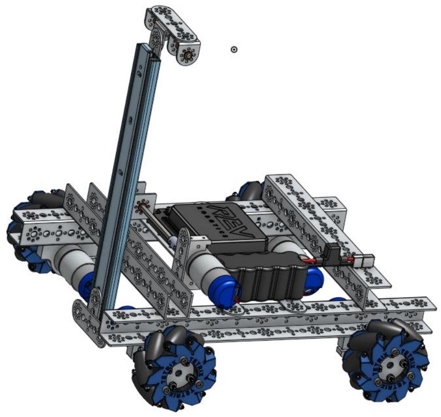




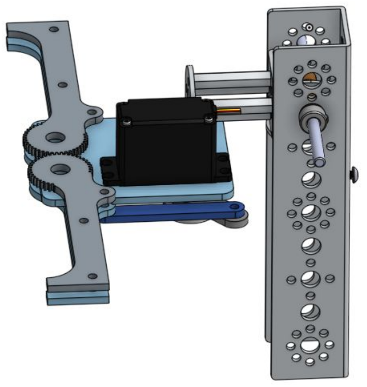


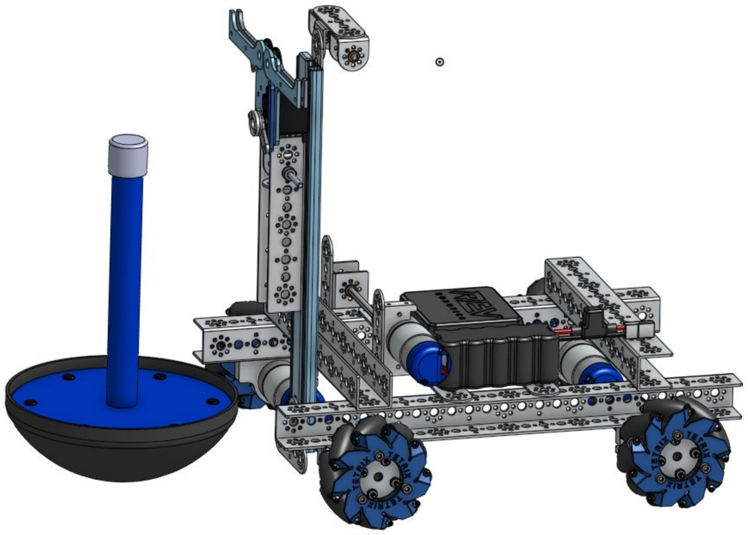


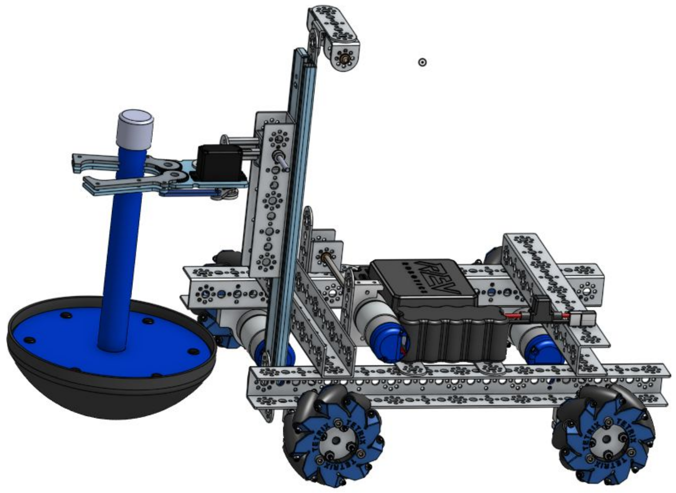
Auto chassis

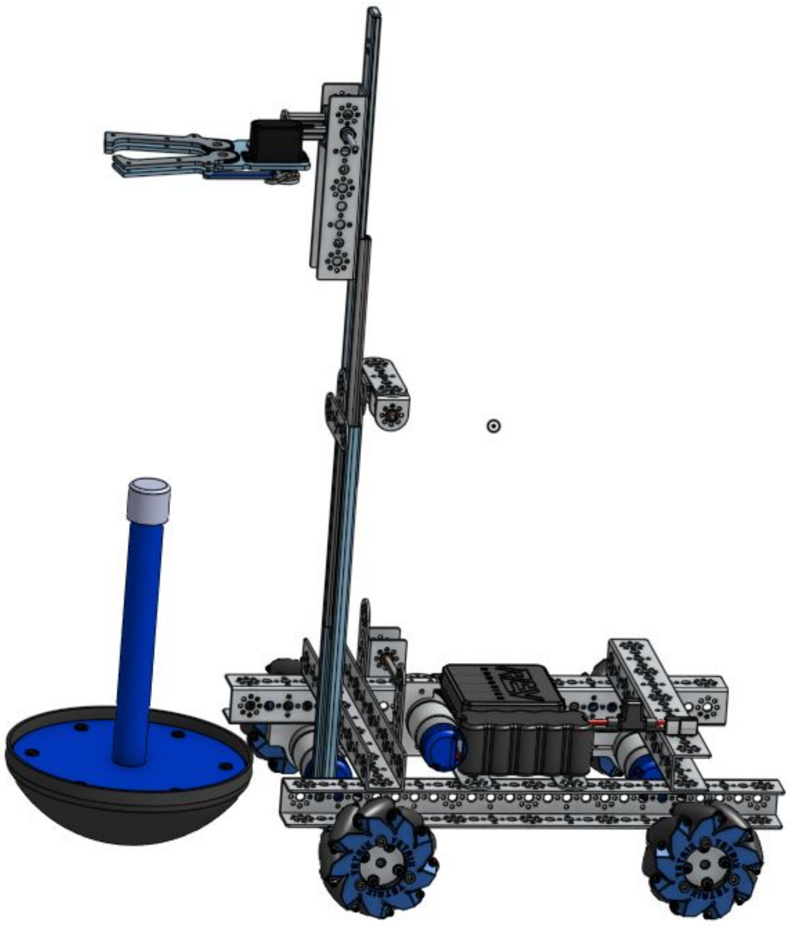


An arm was then made to connect the grabber to the lift of the chassis. The requirements were that it was simple, yet robust. It also had to be easy to assemble to accommodate shorter in person meetings. The design that was settled on was to make a swiveling arm that stayed upright in the beginning. The movement of the lift would then let the arm drop and hit a hard stop. The claw would then close around the wobble goal and lift it up. This was deemed a good solution for the programmers, as they would be able to proceed with their work. The design team will improve on this design, but this is a good step for

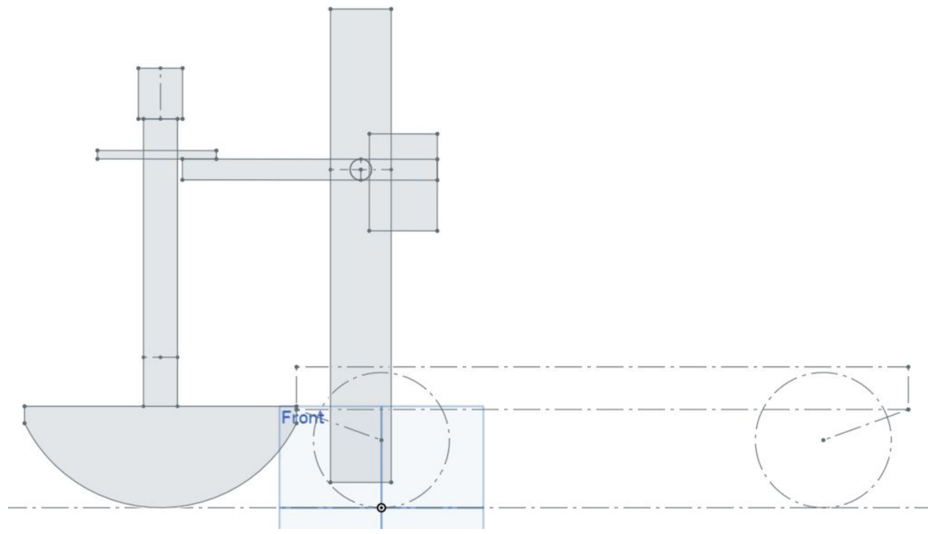


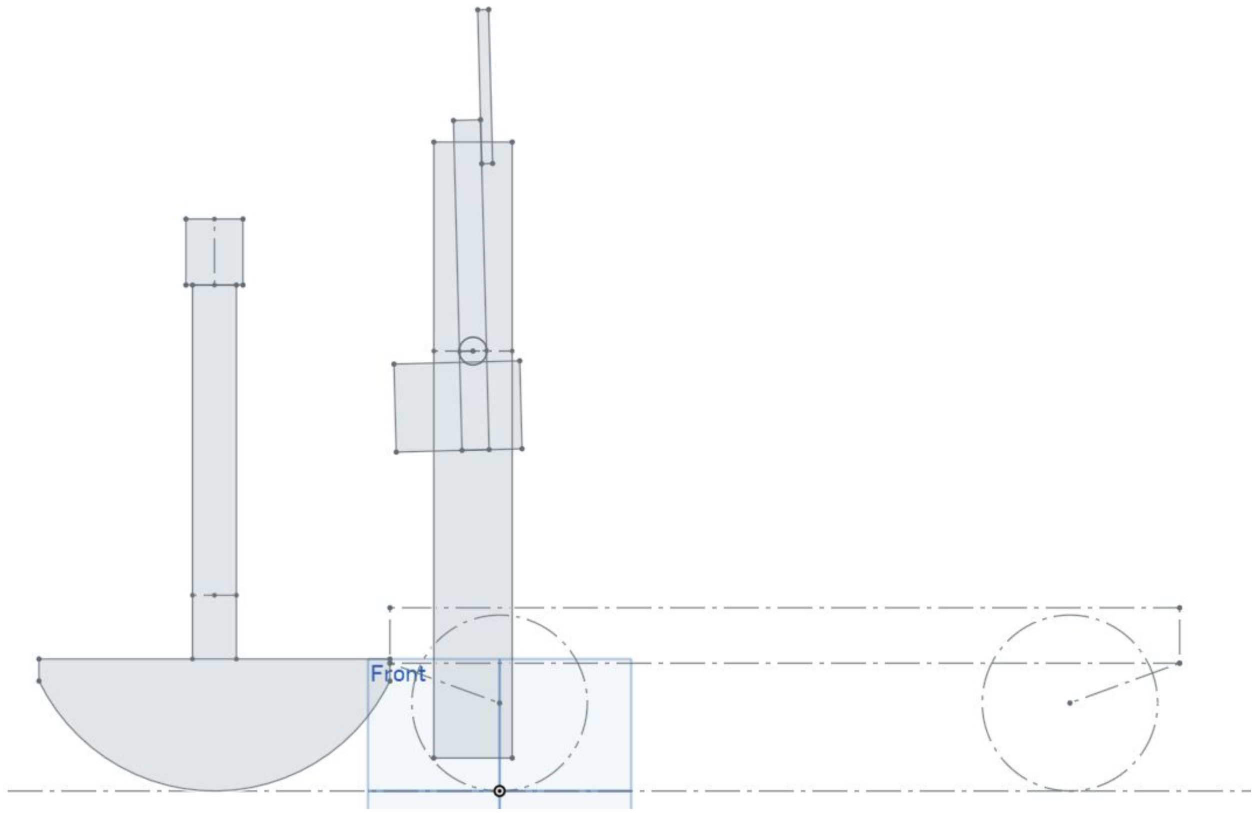






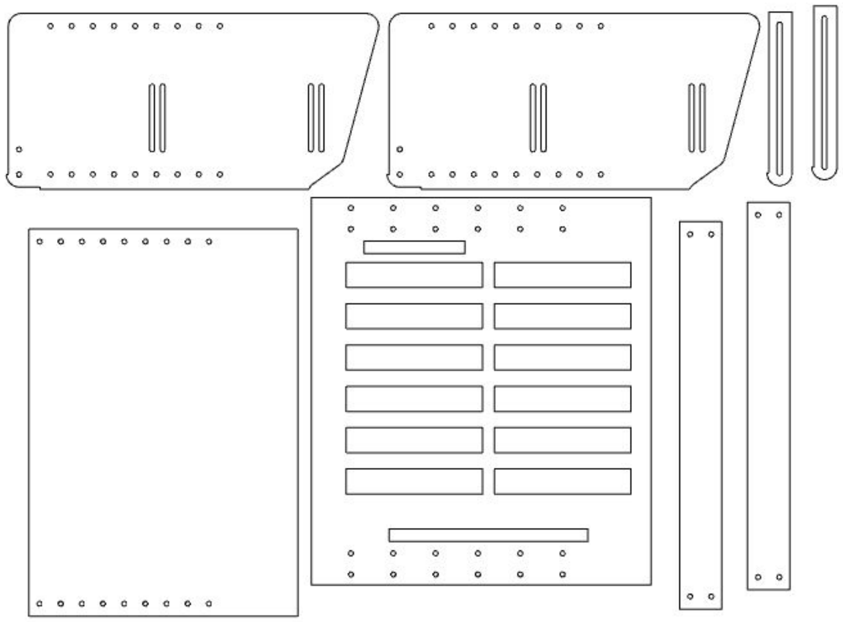
Another arm design idea that was thought of was a sideways elevator. This is similar to the current elevator, but it has different packaging than the current design. The idea could be considered if the center space is needed for other systems. A quick sketch of the arm on a lift shows the possible configurations.



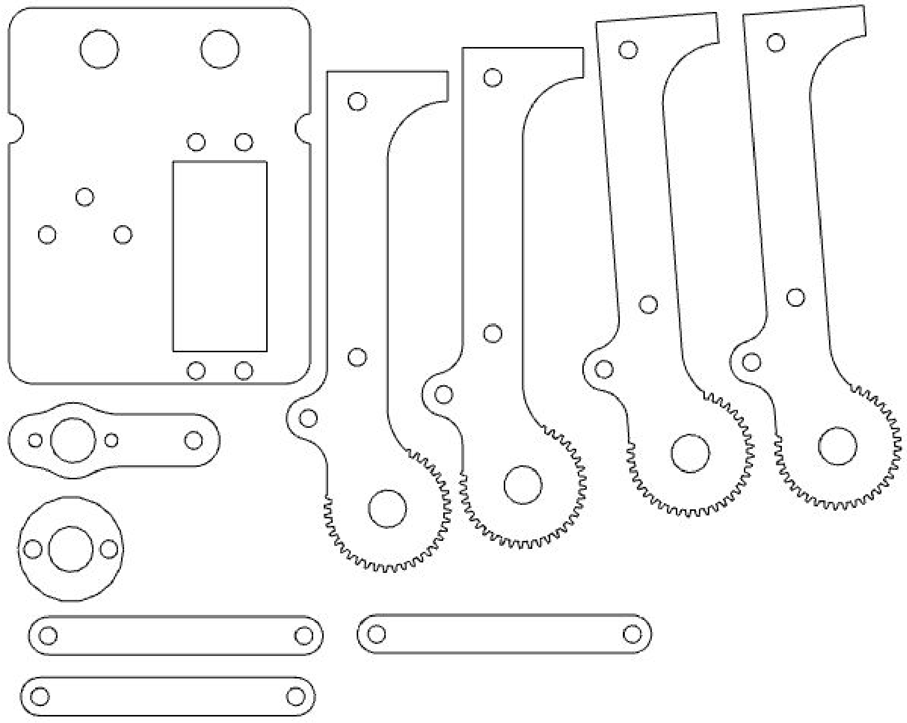


Monday work

The Laser Cutting file was created for the collector, as well as the claw. Both were transferred to inkscape to ensure smooth work on Tuesday in pe



Claw:

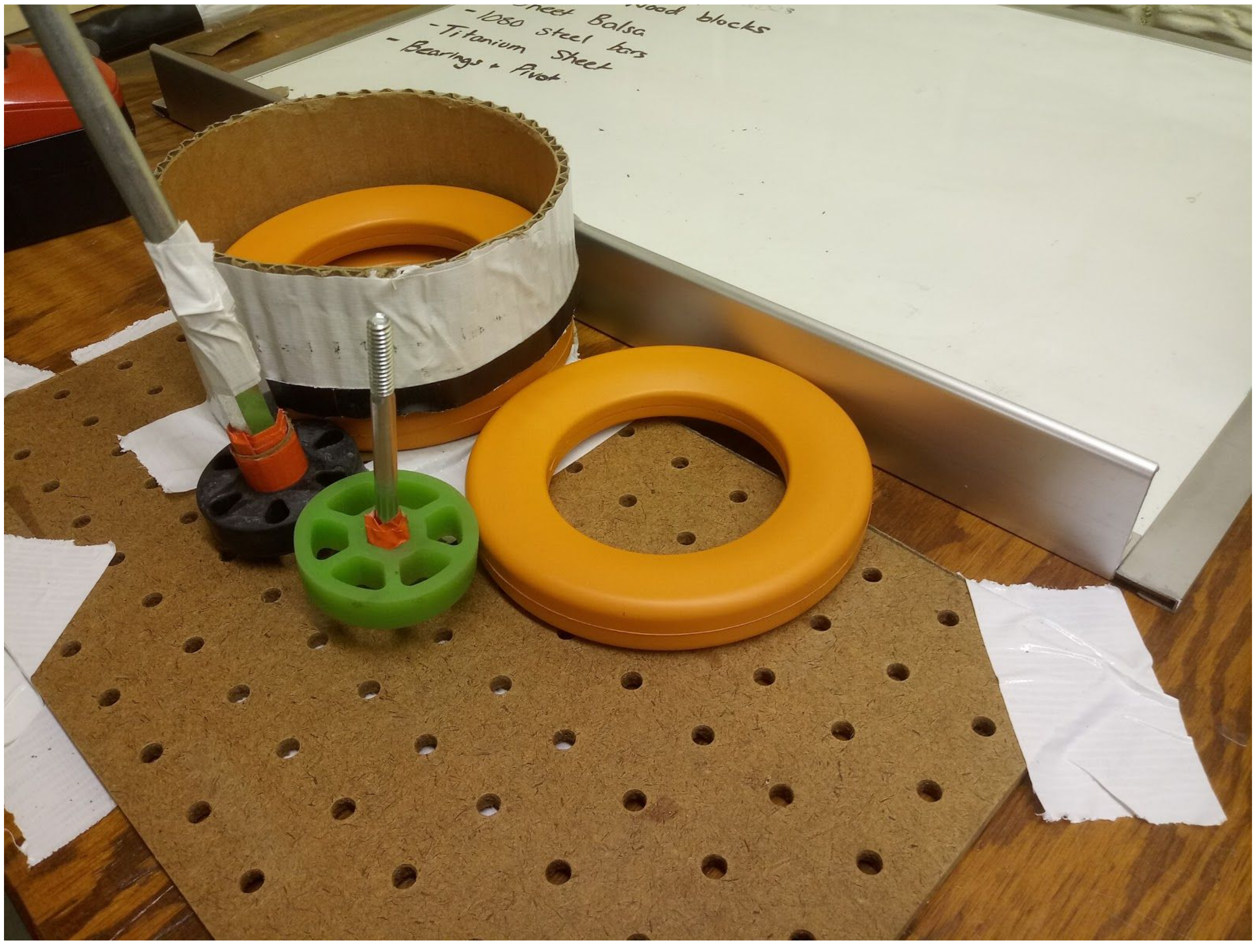


Tuesday:

In person meeting was cancelled and postponed due to not getting school permission to enter the building. We will set aside the current prototypes and focus on indexing solutions until we get permission to physically create the prototypes and test them.

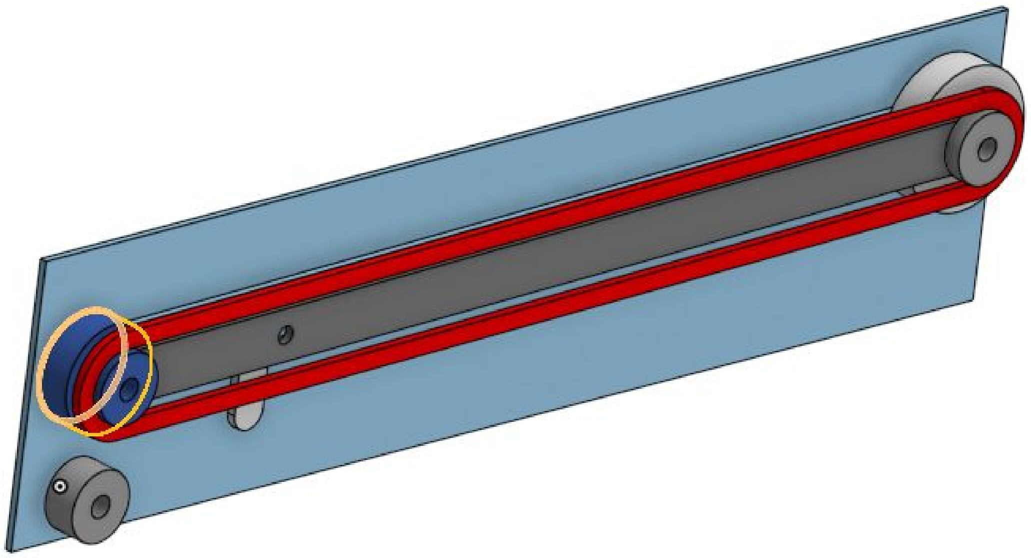
Monday, Oct. 12

Since we had several of the rings already, we went ahead and made a quick cardboard prototype of an indexing solution. This uses a cylinder shaped container that allows the discs to fit inside nicely. We then use two wheels to move the rings from there to another location. We are pleased with the speed, but not with the consistency.



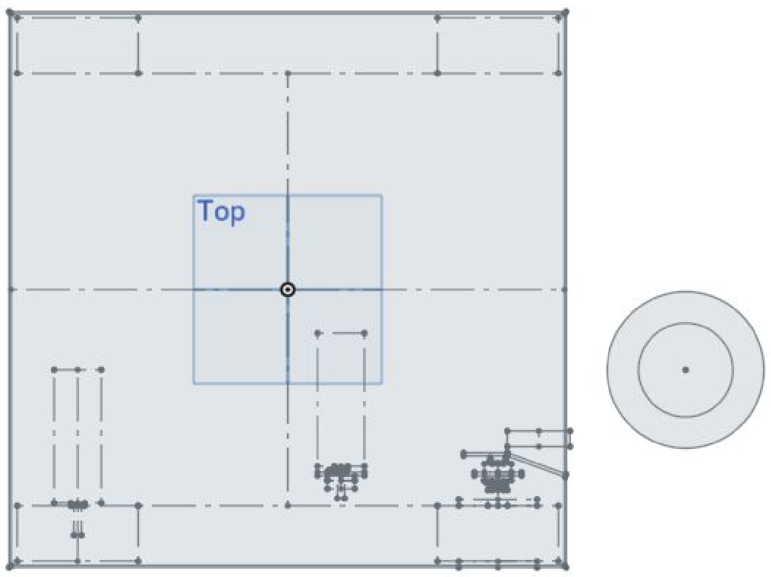
Thursday, Oct. 15

If our shooter requires 2 motors, the drive base needs 4, and the intake and indexer each need 1, there will be no motor available for the lift. We can solve this by using a PTO system that uses the power of one motor to power two things. The shooter motors will most likely be vertical, so for simplicity, we will use the intake motor because it is already horizontal. A simple design uses a servo to engage the gear of the lift to the gear of the intake:



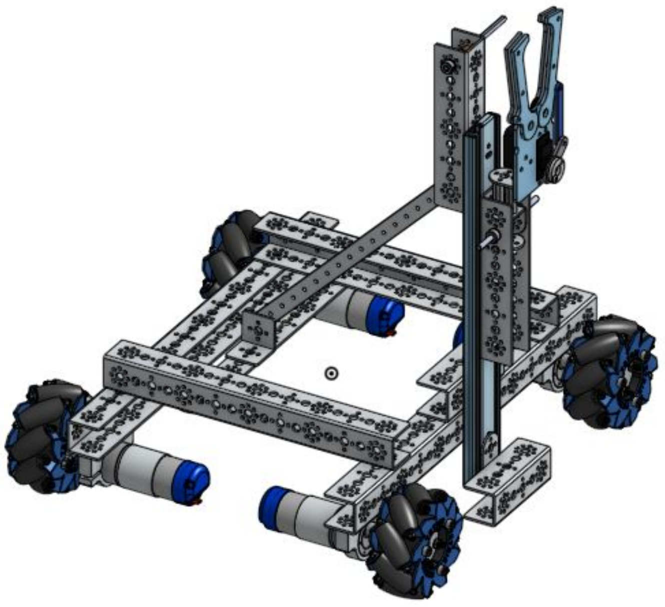
Friday, Oct 16

Even though we do not know the details of our subsystems, we can make a quick sketch of the known items and systems on our robot. This will give us an idea of where space is available, as well as warn us of system integration problems. To start, we know that we want our 4 mecanum wheels in the corners, yet if we directly drive the front two wheels, our collection ramp will be extremely steep. To go around this, we use chains to drive the front wheels. To support these wheels, we drew in all the bearings, spacers, and the sprocket itself to give us an idea of how wide it will be. Our end result leaves us with ~10.75 inches in the center to use for the collector.

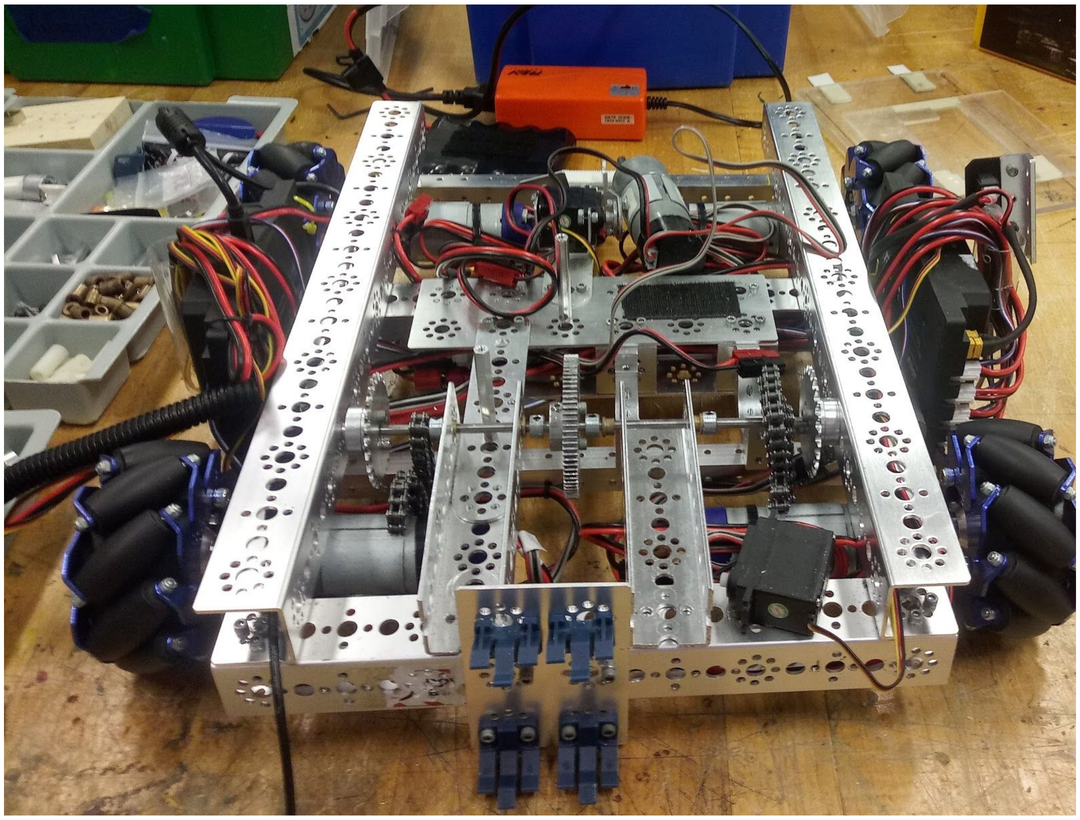


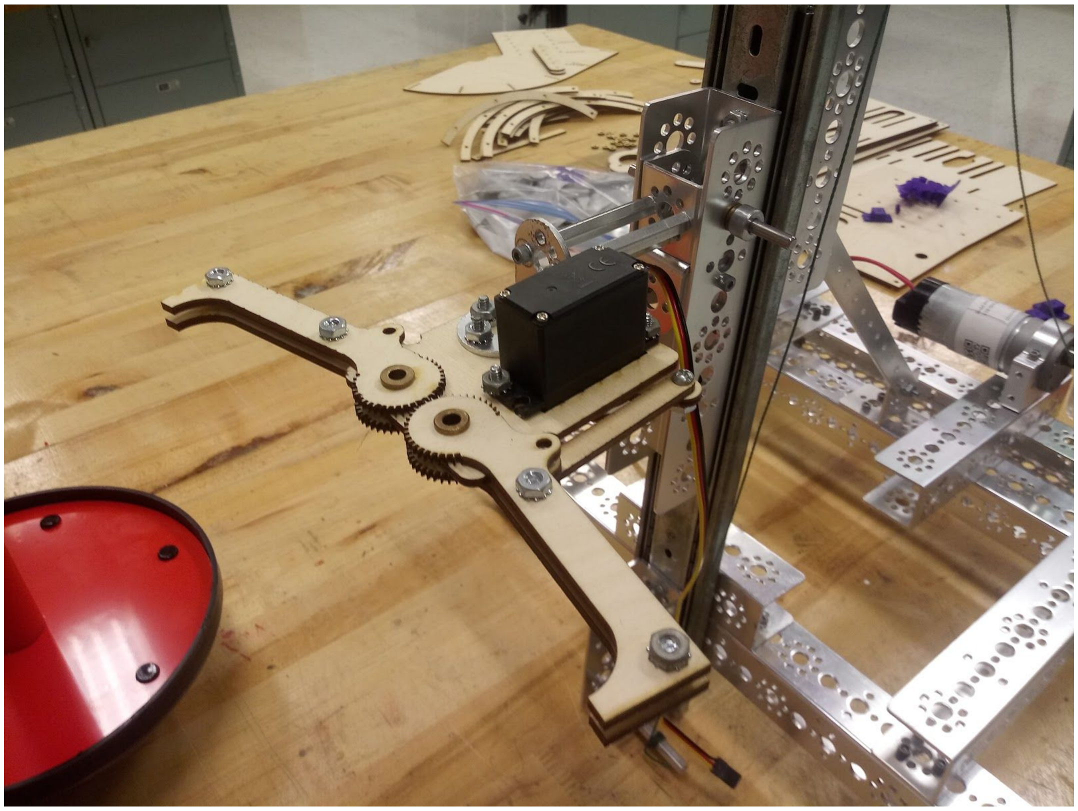
Monday, Oct 19

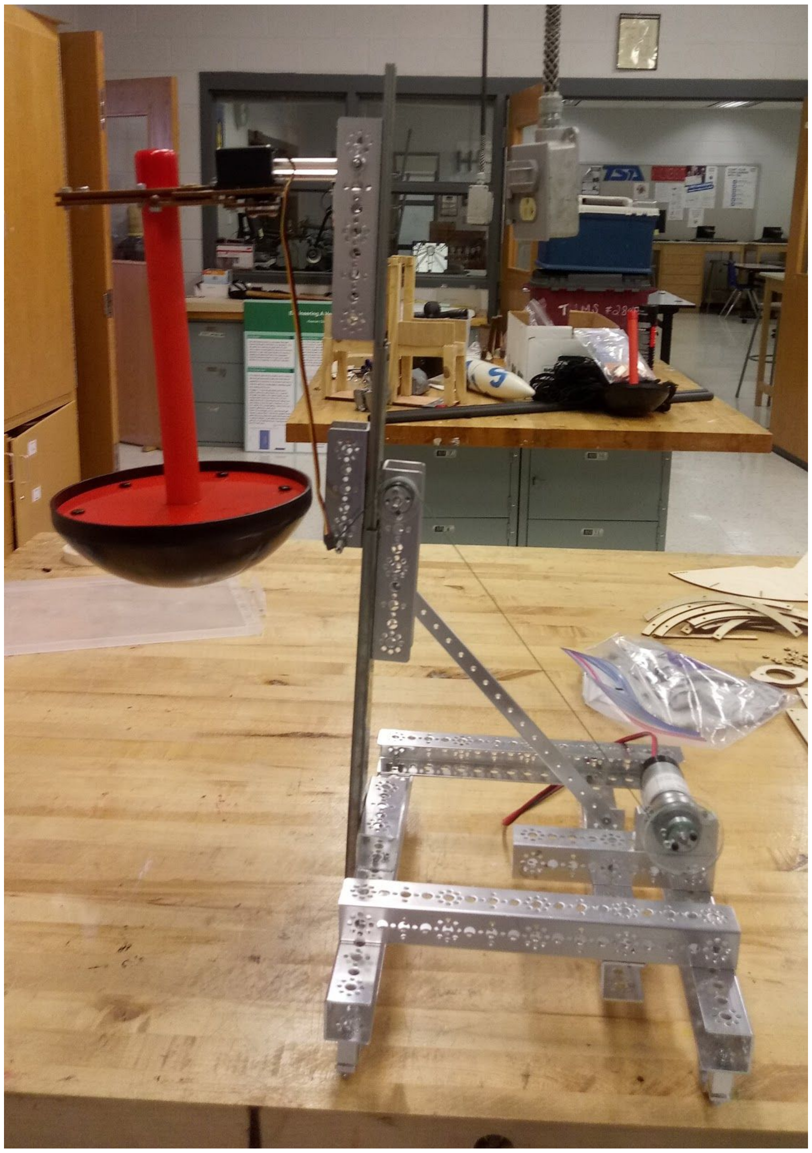
A new auto chassis was made for mounting the claw on the side. This will be assembled at the Tuesday meeting tomorrow.



Tuesday, Oct. 20
We finally met in person at the school. Thomas was in charge of taking apart the old robot and sorting parts, while Niranjan needed to cut out the prototyping files and get started on the test chassis.

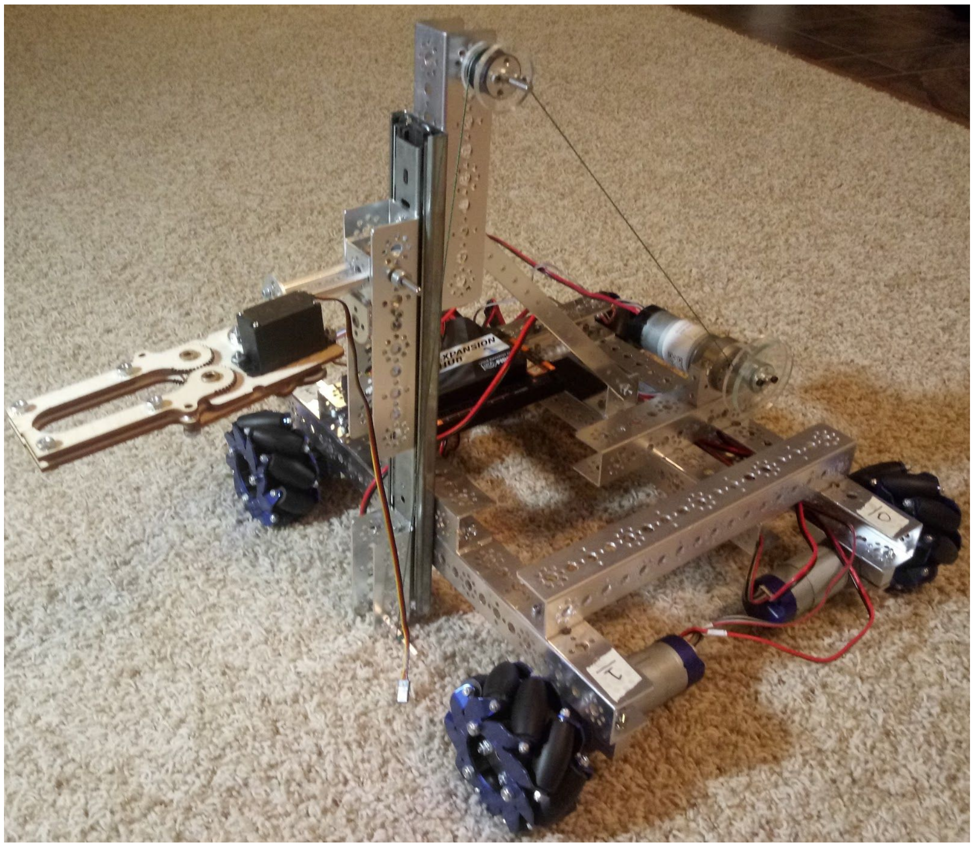


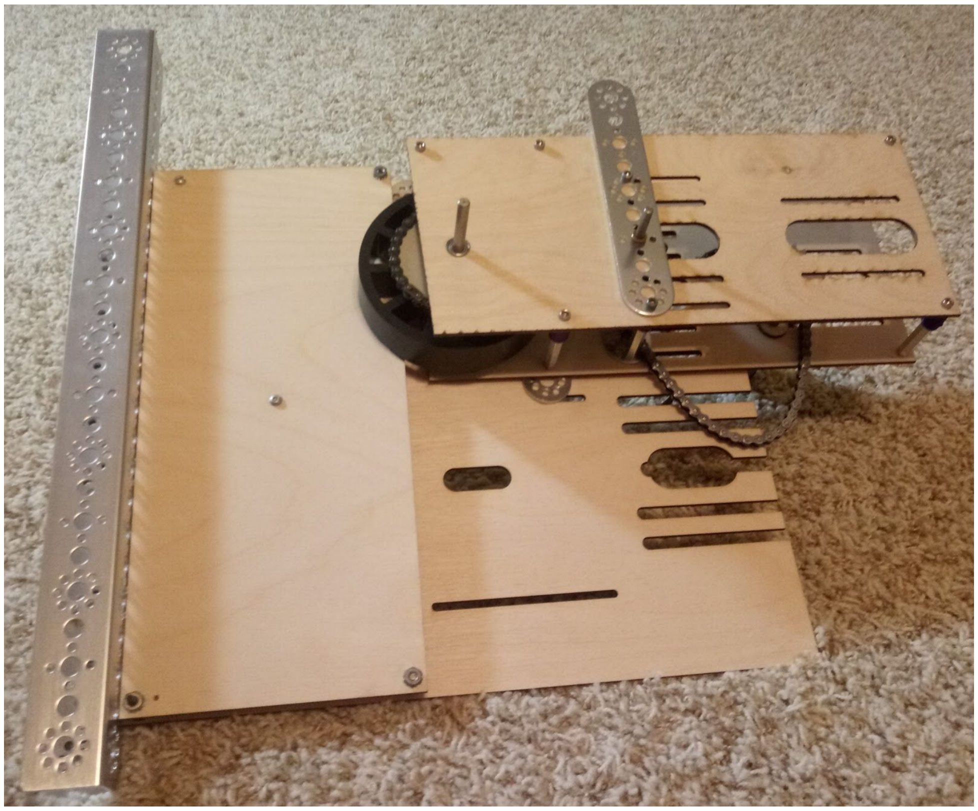




<u>Thursday, Oct 22</u>

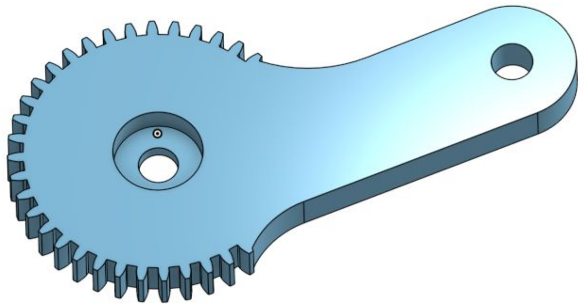
Today we focused on completely taking apart the old robot, and trying your best to finish the test chassis for the programmers. We succeeded in taking apart the robot, but Niranjan has taken home the test chassis to finish wiring and will then let his brother Anish collaborate with Carlo to program it. We also got some progress towards assembling our first shooter prototype. Most of it is complete, except for adding the correct lengths of chain. Niranjan has also taken that home where he will finish it and start testing.

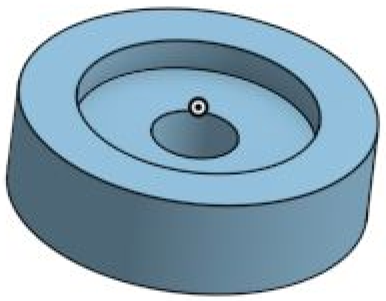




Saturday Oct. 24

The two main problems with the claw were that it was not stiff enough, and that the gears weren't meshing properly, since they weren't cut properly on the laser cutter. To solve both of these problems, we are 3d printing gear meshes that will allow the gear profile to be sized correctly, and allow bolts to go through the pivot, stiffening the claw.



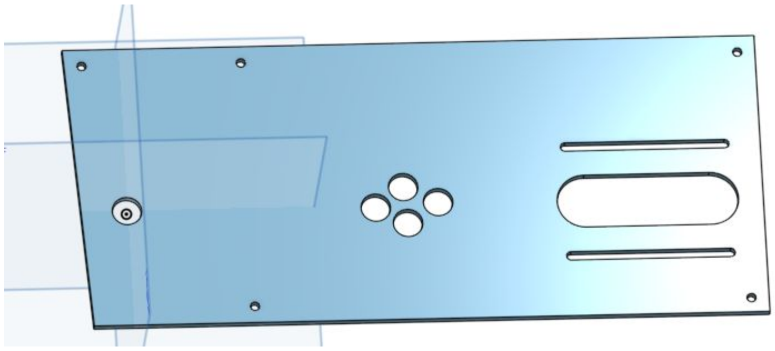


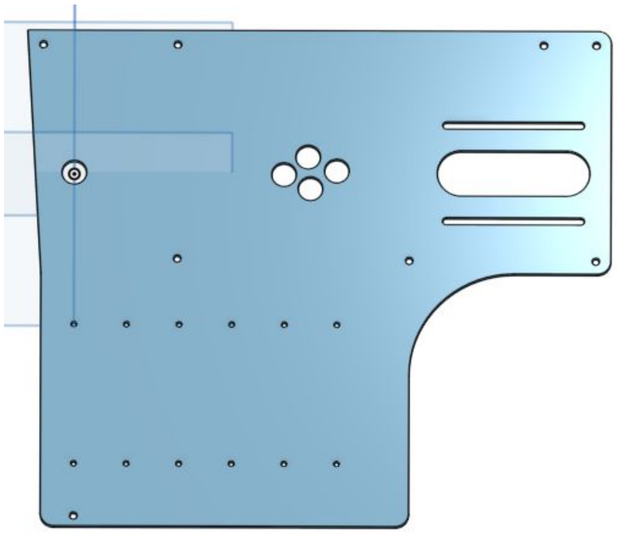
Sunday, Oct 25

The shooter testing did not go as planned, because of how hard the chain was to properly tension. The set-up was a lot better. Our main focus is the exit velocity of the disc, and so we have a lined background and timer running with an Ipad on Slo-mo taking video. This allows us to know distance over time, a velocity. We set up the shooter at a 30 degree angle, and are also taking distance, though not as accurately. The motor is powered by an old 12V DC converter to consistency. The supply is wired with a switch inline, and is capable of providing 1.8 Amps, enough to power the motor, but not enough if the motor stalls.

Monday, Oct 26

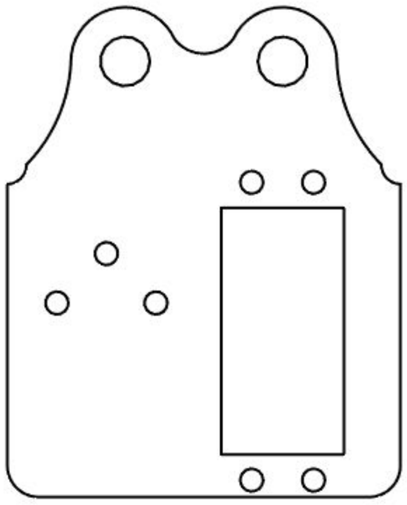
The main focuses for the next meeting are fixing the shooter, and making a new claw. A new claw was needed to reduce friction, yet maintain stiffness. This version won't require tightening the pivot to increase rigidity. The shooter was also re-designed to eliminate the need to tension the intermediary shaft. The new version boasts fixed center-to-center holes calculated with an online chain calculator. This shaft will now use bearings to reduce even more friction.



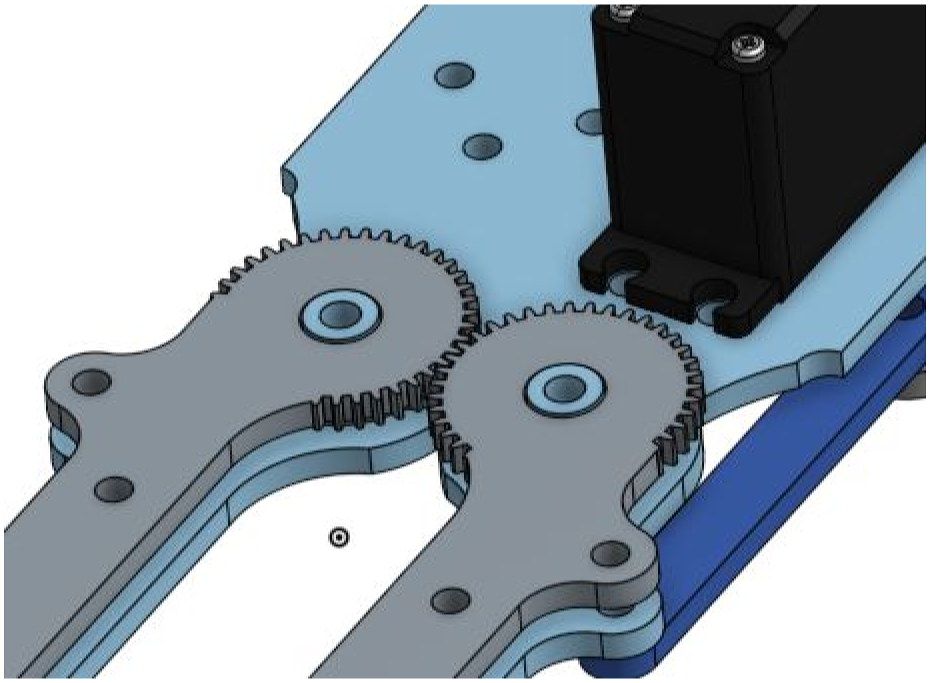


Tuesday, Oct 27

We made some slight differences in the design of the claw that should help there to be less friction and more stiffness.



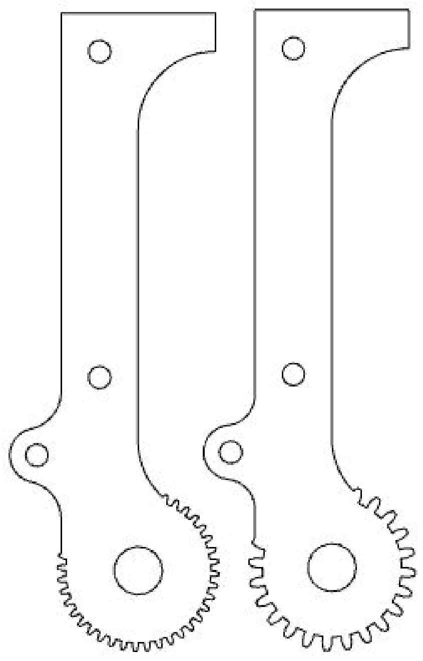
The main plate of the claw was redesigned to have less surface area between the claws and the plate itself. This plate was recut out of 1/8 inch plywood and replaced on the claw assembly.



The center brass bushings were also replaced with 3D printing inserts. These inserts are kept in place with some washers and a bolt. Both of these changes help stiffen the arms while reducing friction. Some further adjustments we want to make to the claw is changing the teeth on the gears. When laser cut, the fine gears we have been using don't work very well. Replacing the fine teeth with larger teeth will be smoother and more reliable.

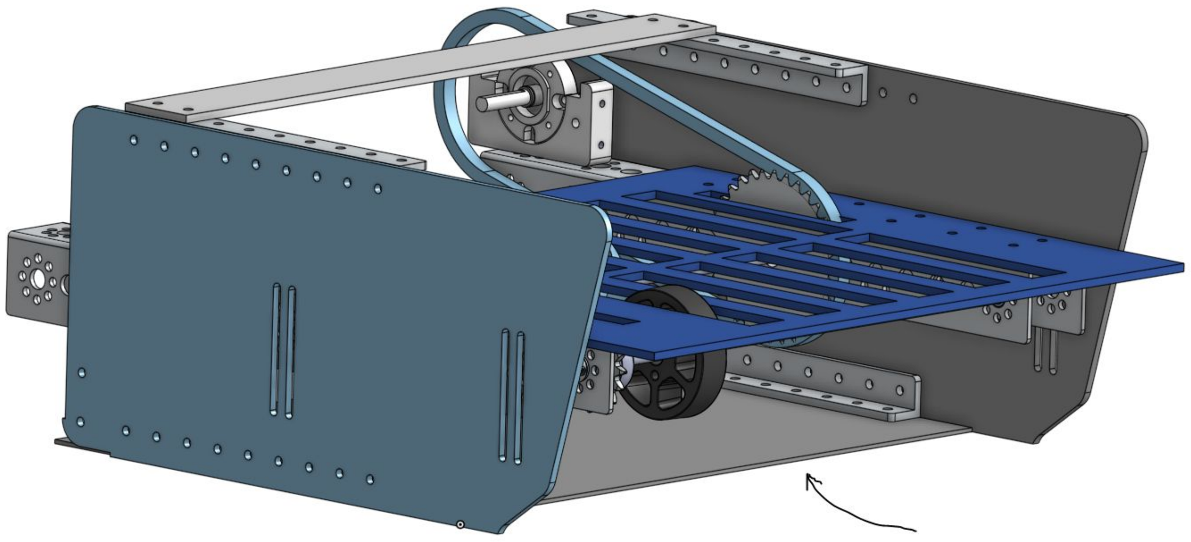
<u>Wednesday, Oct 28</u>

To prepare for tomorrow's in person meeting, we made new arms for the claw with a larger gear profile. The earlier gears were 40DP gears, which did not lasercut properly and stripped easily. The new gear profile is 24 DP, with the same pitch diameter. The part also implements a gear calculator, so switching between different profiles is as easy as typing in a new number.

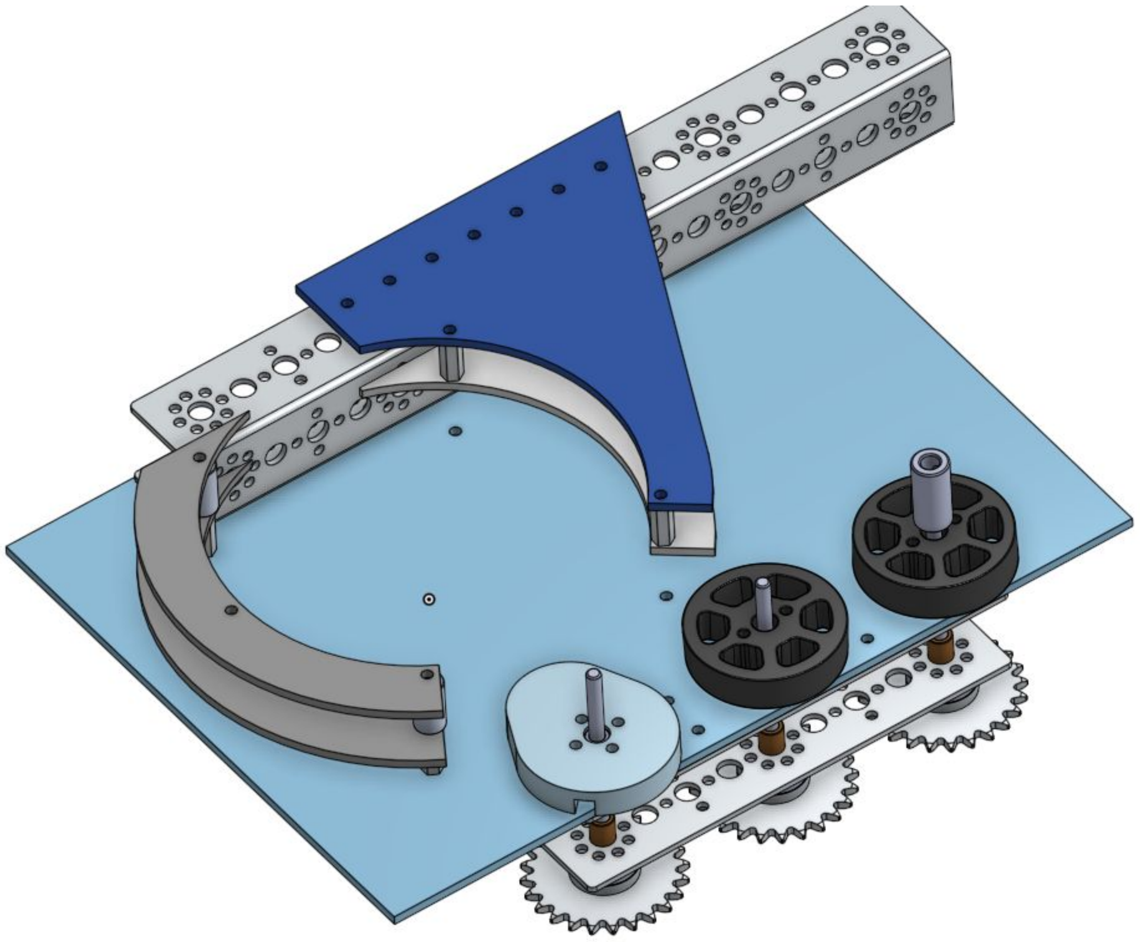


<u>Thursday, Oct 29</u>

The collector was assembled, and a couple problems were noticed: The bottom plate was too far forward, and was obstructing rings as they tried to be collected. So, we will shift it backwards and sand an edge to deal with this. For the final version we would probably use as thin of a material to fix this.



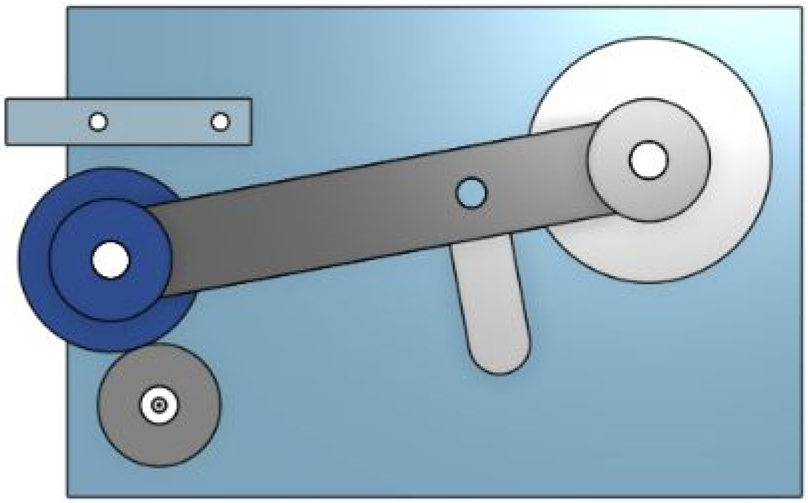
Also, we designed a hopper prototype during the meeting and laser-cut the parts needed. This is a progression of the prototype Niranjan made out of cardboard. It will have a cylindrical storage, and have sets of wheels to dispense the rings. One addition is that we will include a motor to agitate and pull in the rings as they come into the storage container. Here is a CAD model that created after the meeting:



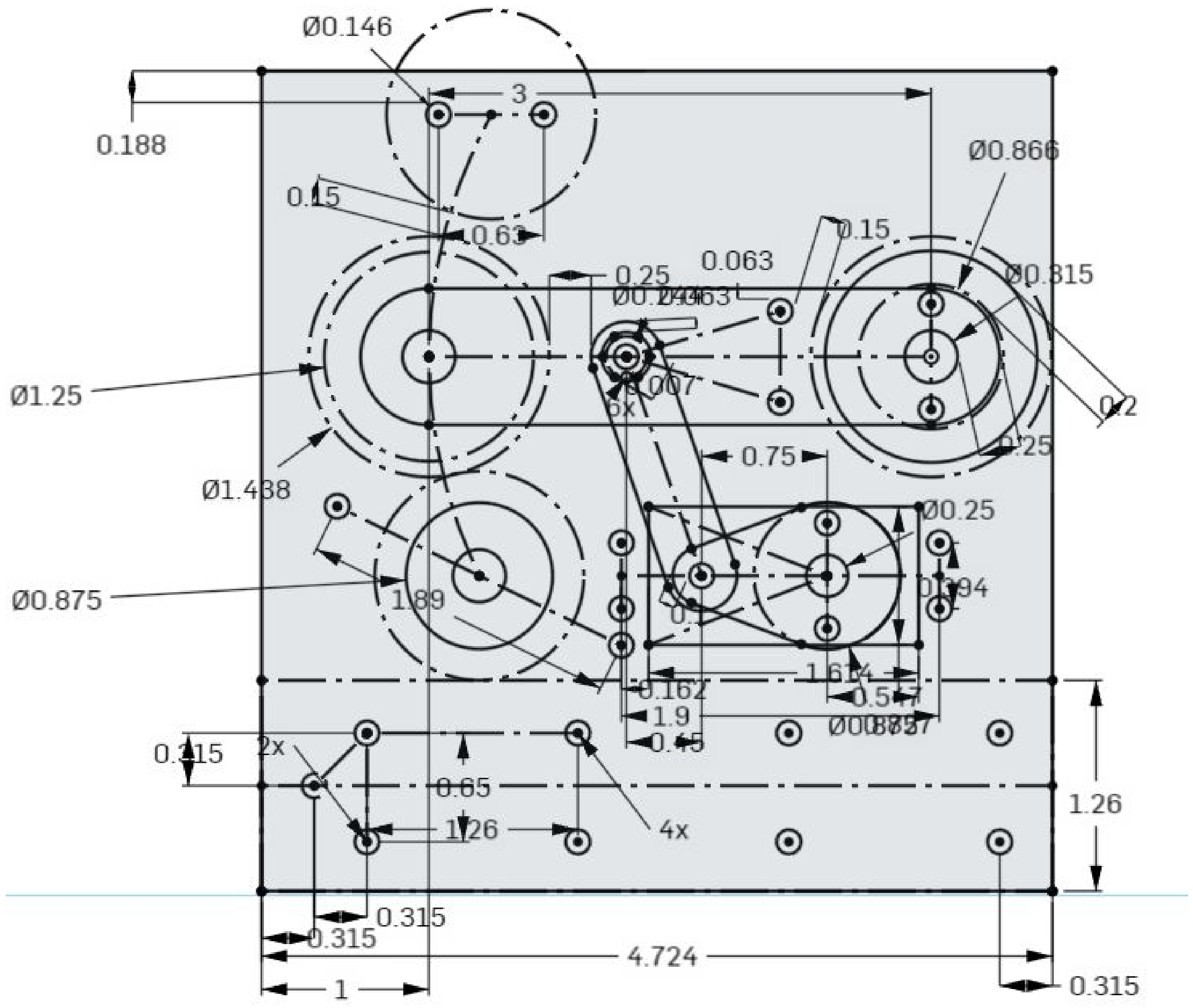
This uses a cam to push the ring out, which will then be pulled out with the wheels. The one problem that we foresee is that the chain will not be tensioned properly.

Tuesday, Nov 3

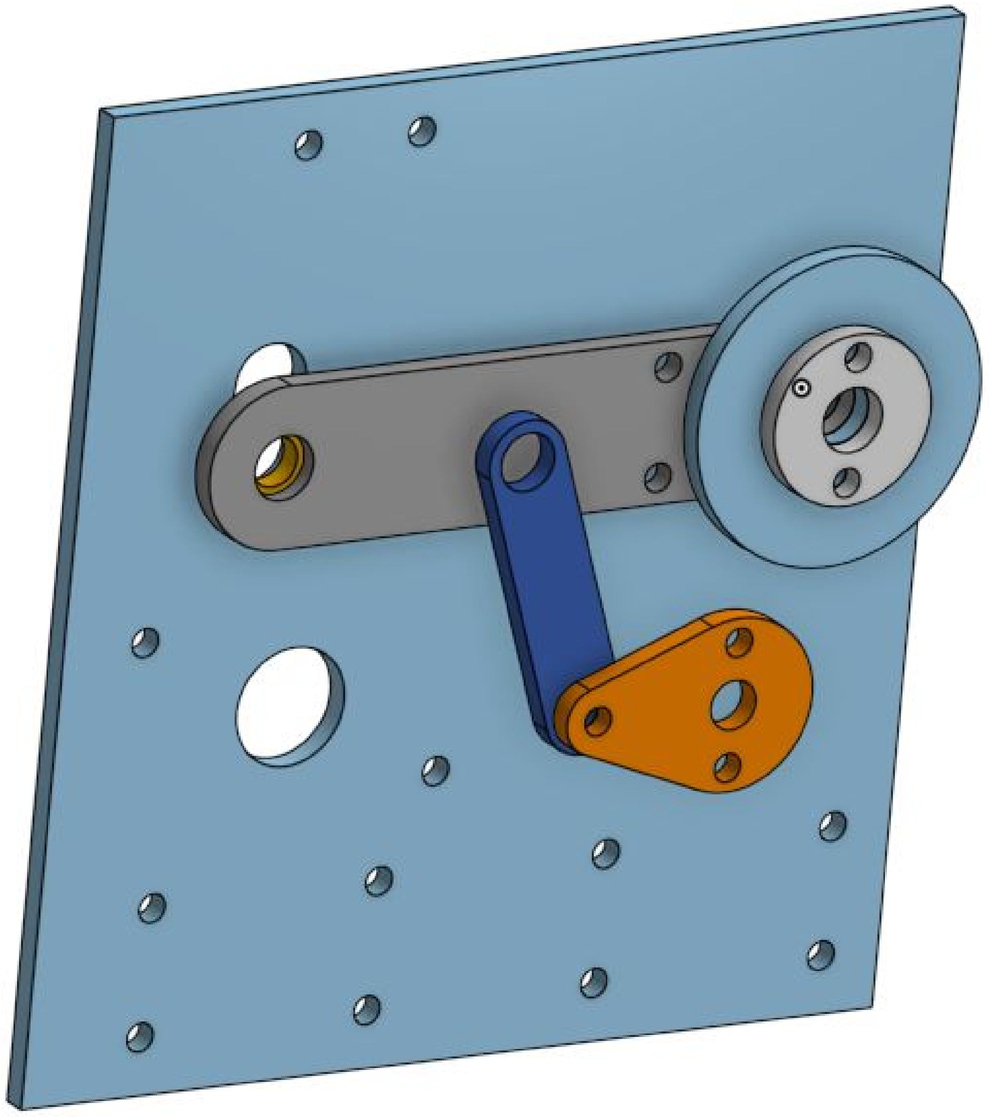
Today a lot of time was spent designing time. The first item that we designed was the PTO gearbox test. For this test, we wanted to have a motor running and have a servo be able to engage a gear and run the spool. We then wanted there to be a stationary gear that the servo could engage with to act as a brake. This all needs to be attached to the test chassis so the programmers can include it in their autonomous programs. First, we take our first concept sketch (pg. 40) and shorten it to make it more realistic. This gives us a reference for the design.



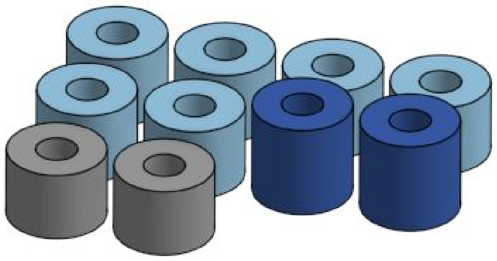
Next, we start making a base sketch. This includes every part of the PTO on the same plane. We like to design everything in one sketch so that we can have references between all parts, and be able to edit all changes within one sketch. To start, we know that we want to use the Tetrix gears, as well as custom gears with the same pitch diameter. We draw the pitch diameter of the motorized gear at the origin because it will be a fixed location, and is also an important point. We then create our shifting lever. This includes a gear one one end, that will engage with the motorized gear. The other side has a spool. We make the length 3.004, which is a proper center-to-center distance plus a little extra for the sprockets. Next, we draw the pitch diameter of the stationary gear. We then add mounting holes for the motor, as well as holes to secure everything with either standoffs, or the tetrix c-channel. Finally, we add a mounting spot for the servo, and create an arm to mount to it. Now, all our designing is complete, and all that is left is to extrude the proper sections and assemble it in CAD.

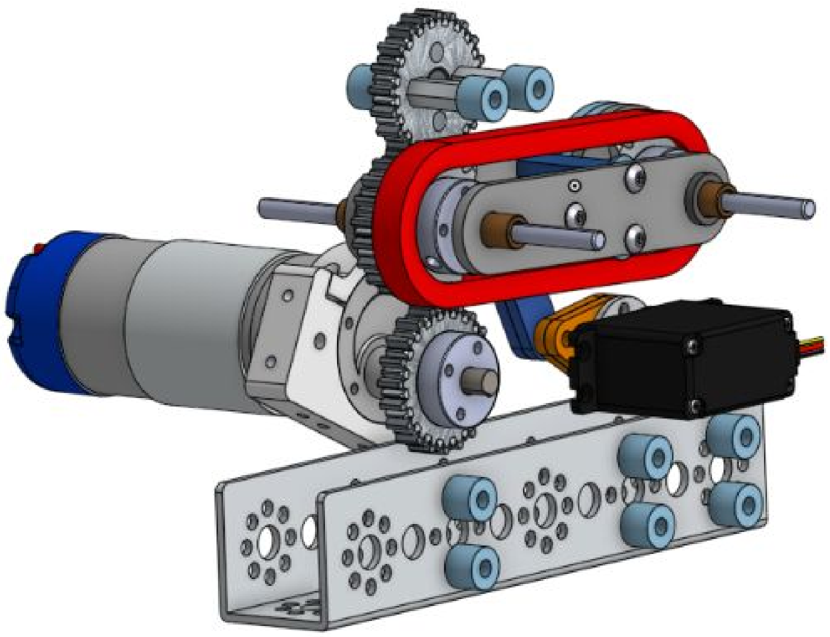

Ø0.146
3
0.188
Ø0.866
0.15
0.63
0.15
0.25
0.063
Ø0.315
6x
0.007
Ø1.25
0.2
0.25
0.75
Ø1.438
Ø0.25
Ø0.875
1.89
0.394
1.614
0.162
0.547
1.9
0.45
0.315
2x
0.65
1.26
1.26
4x
0.315
0.315
4.724
0.315
1

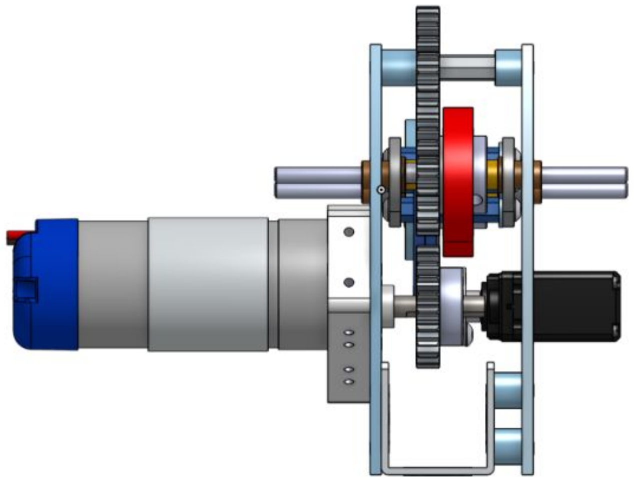
After extruding everything, our model looks like this:



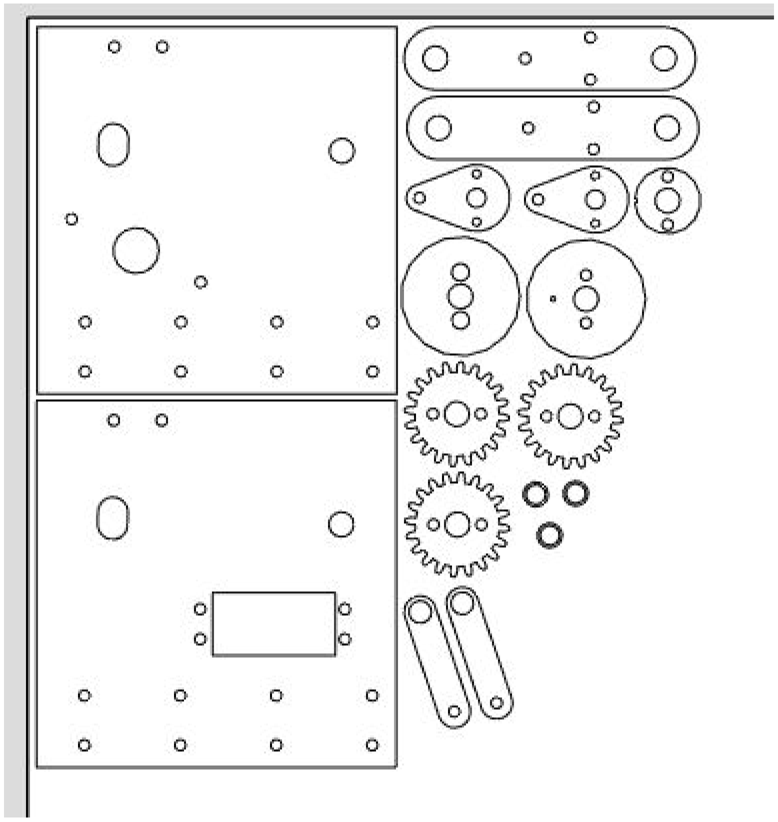
While assembling, we add in kit parts such as the motors, as well as determine the correct spacer widths. We create the spacers in a seperate part studio, and add them to the assembly. Our focus now is to use as many kit parts as possible during assembly, and minimize 3d printed parts, as they take longer to create.



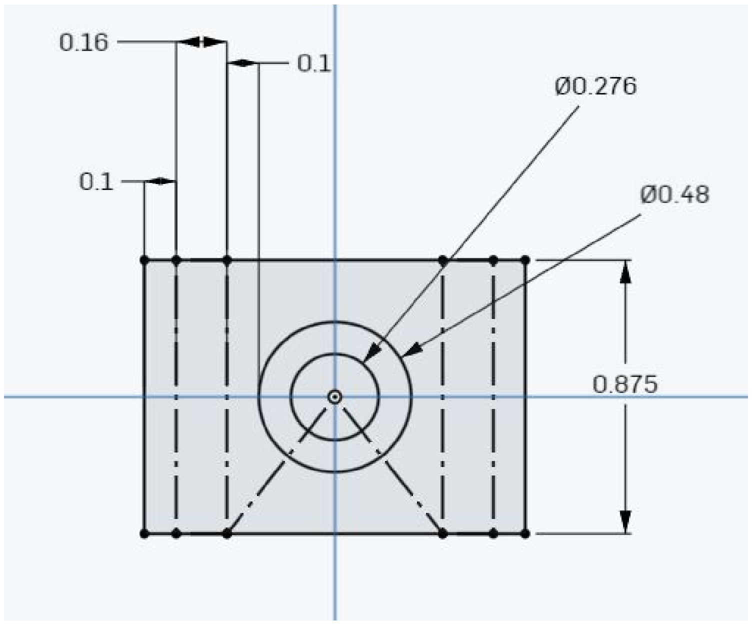




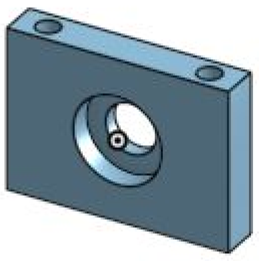
Our assembly is now complete. We now take all our parts and make them ready for manufacturing. For this design, that means getting the laser-cutting files ready.



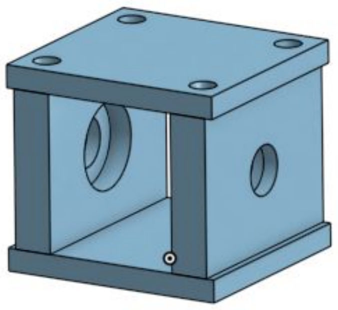
The next thing that we designed were bearing blocks. We also wanted to see if our custom solution would be sturdy enough to be on the chassis of our robot. We again started with a master sketch, except this time it looks a bit different, because we have features on several planes.

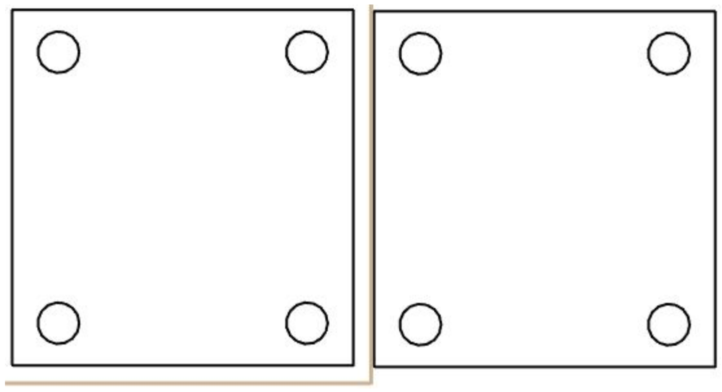


Because the bearing hole is the most important feature, we put it on the origin. We then use construction lines to represent the tapped holes we plan to put on the top and bottom. We also use construction lines to set both of these "tapped holes" equidistant from the center. We use construction lines instead of dimensions because this is something that should stay true no matter the dimensions. Finally, we add dimensions to constrain everything. We like to dimension things with a purpose, so that is why we dimensions off of the tangent of the circle. Finally, we extrude the block.

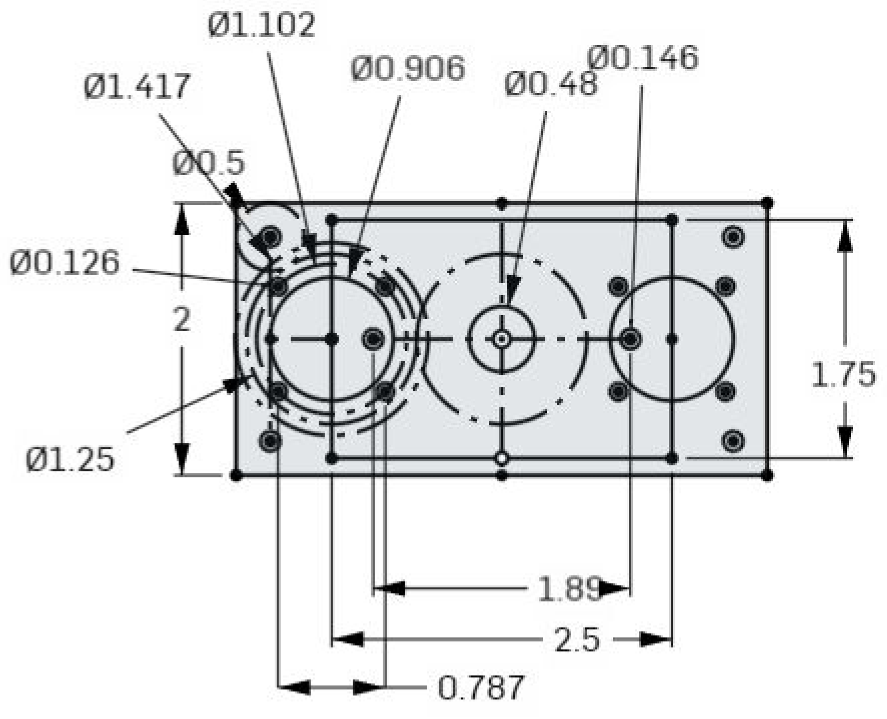


We decided to 3d print this object due to its multi-planar features. We also create an assembly to show how it will be assembled. The top and bottom plates will be laser-cut out of 1/8 in plywood, our standard thickness.

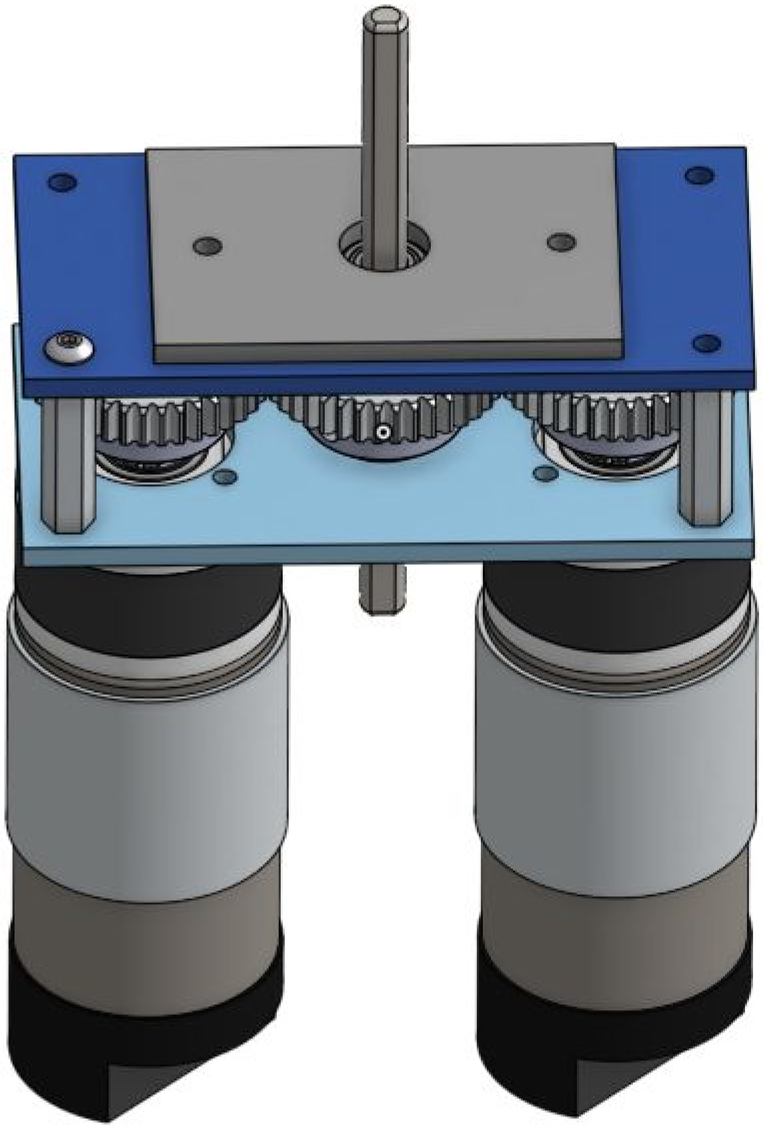




The last thing we need to design is a dual-motor gearbox. We want to try shooting with two motors, because it doubles the torque, so in theory, it should help us achieve a better rapid-fire. This is a simple assembly. We know that we want to use the 40t Tetrix gears, for simplicity. We also know that the center shaft will be supported with bearings, to reduce friction. So, we create a center rectangle, and place the bearing hole in the center. We also create pitch diameter circles of the tetrix gears. By making these tangents, we should achieve the proper spacing. We then create a mounting pattern for the motors, and add holes to assemble with standoffs. For the other side, we mirror over the holes, for simplicity. This also allows for easier changing of the design. Our sketch now looks like this:

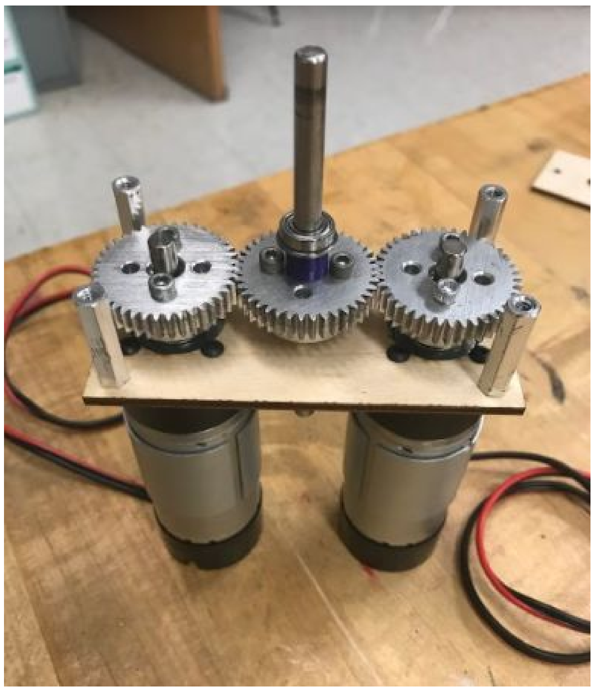
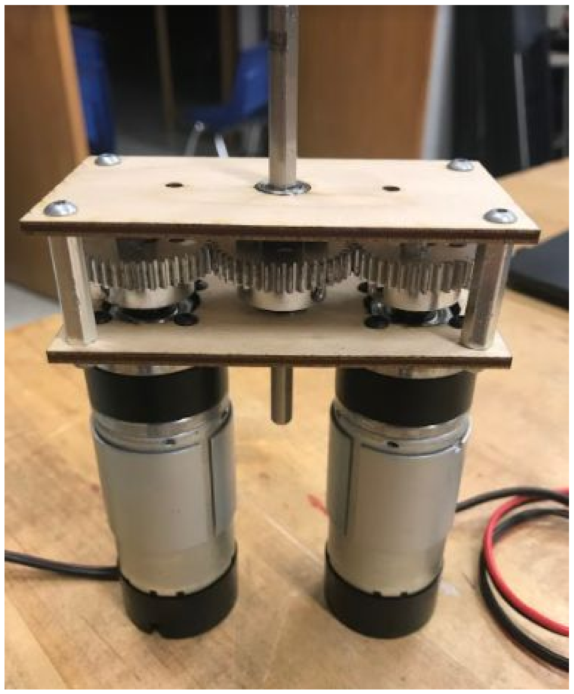


We then extrude all of the plates. While doing so, we noticed that another top plate needed to be added, so that the mounting screws didn't interfere with the shooter when we wanted to mount everything together. So, after extruding, we assemble everything together, with the motors and all, to ensure that our design will work.



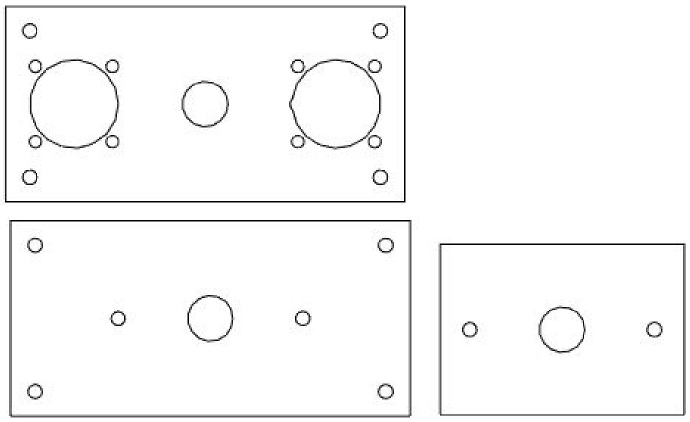
Thursday, Nov 5

With the final robot in mind we wanted to create the prototypes that we discussed and designed earlier. For the shooter, this was a dual motor gearbox, which should help us shoot the discs in rapid succession.

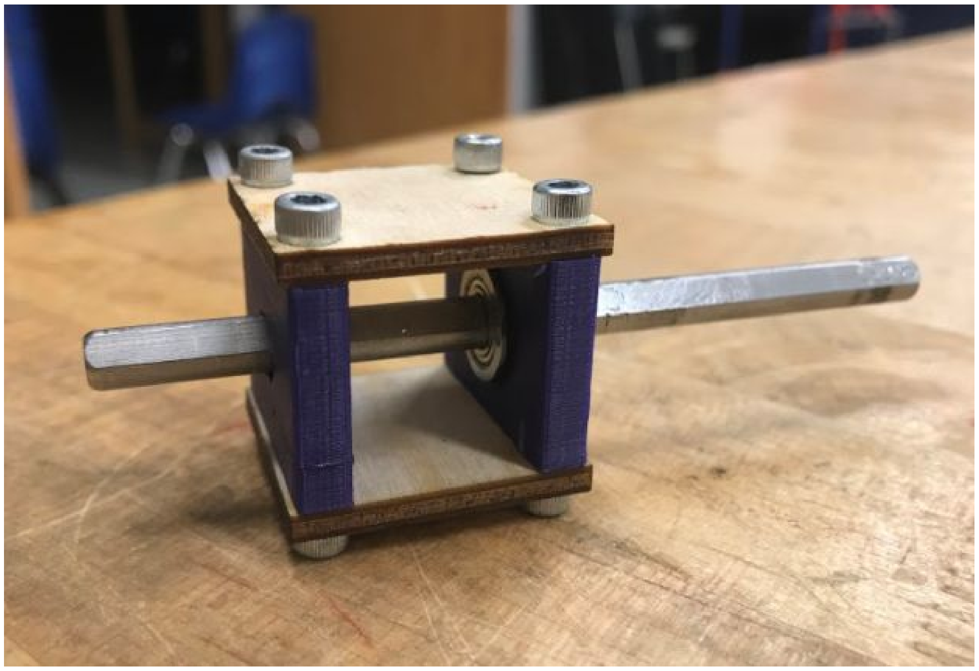



This is the final gearbox. At first, the gears were extremely tight and didn't run smoothly. To solve this we loosened the motor screws and used the extra tolerance to pull the motors apart slightly before tightening the screws again.

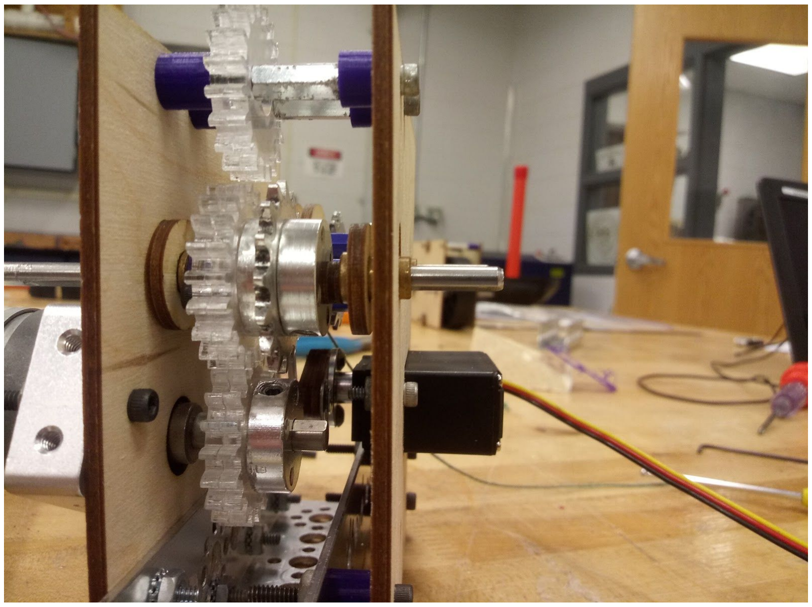
Here is the laser-cutting file for the gearbox. The plates were laser cut out of 1/8th inch plywood. The second test was to try and create a custom bearing block. We took the 3d printed blocks and assembled them with screws.

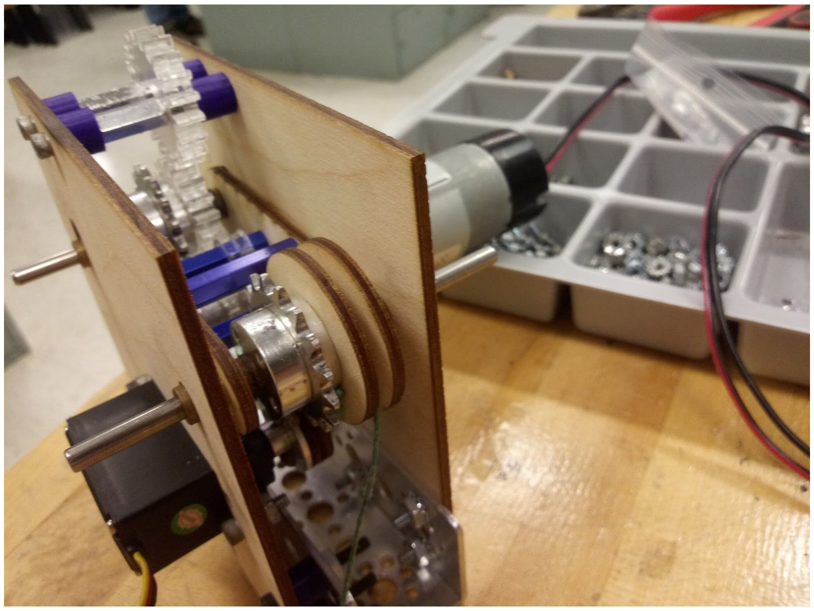


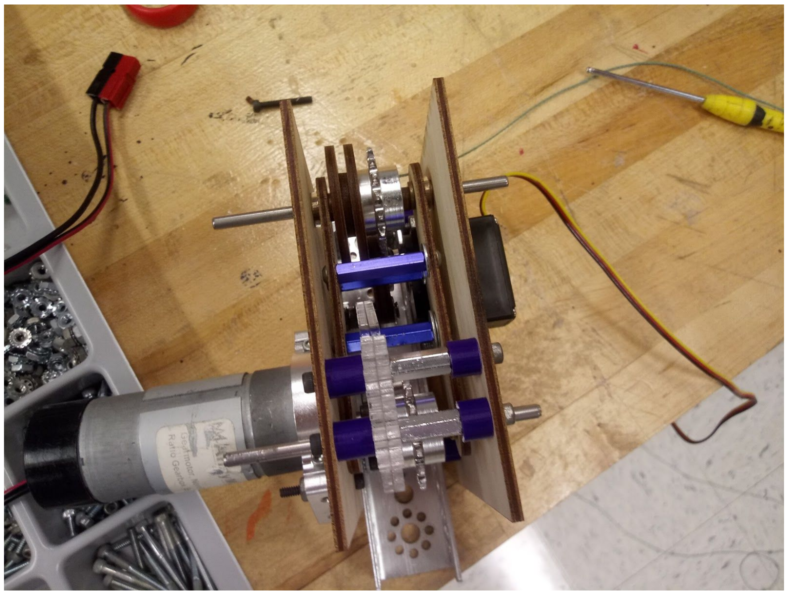
This is the final product. The axle going through the bearings spun very smoothly and was sturdy enough that we think it will be able to support mecanum wheels and the robot itself. We will probably use a similar design for the final drive base. If more stiffness is required, we could extend the distance between bearings.



The last thing we wanted to do is create the PTO test that we had designed. If this works, this will allow us to use an extra motor for the shooter. Even though we plan to eventually run this off of the intake motor, we have a seperate motor to test this, for simplicity of design. This design, while it came together in the end, taught us some about design vs real life. This model had some features that made it hard to assemble, and was a little less rigid than we liked. In the end, we think that this prototype is a success, and have plans to implement it on the test chassis.

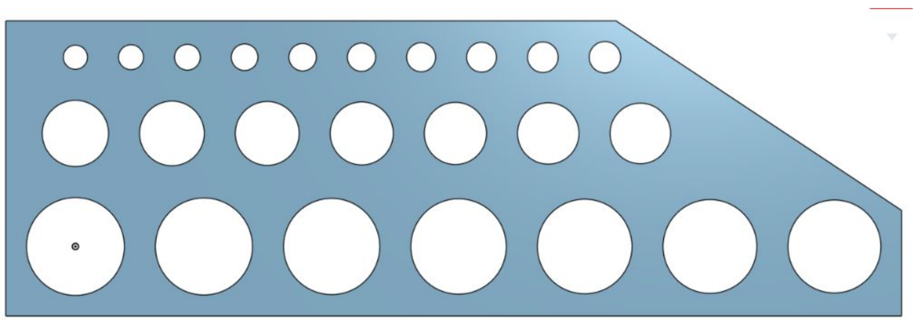


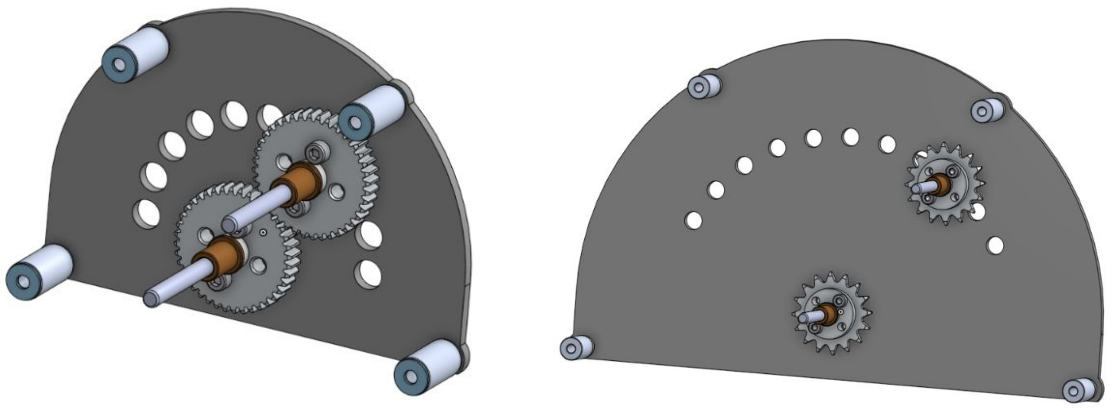




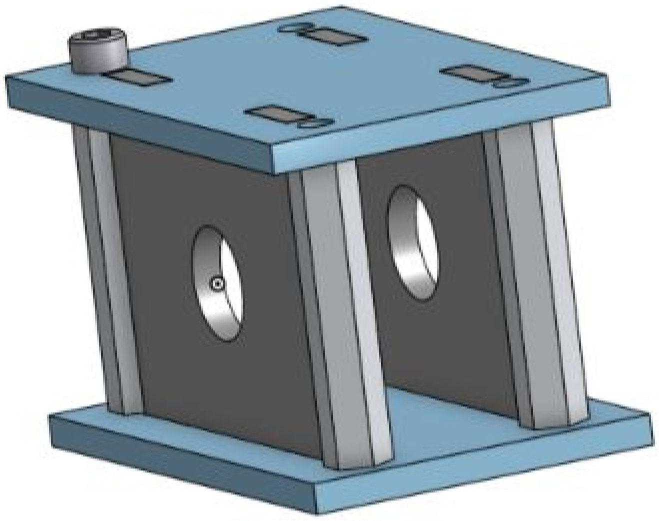
<u>Nov 9</u>

Before we started a lot of design work, we wanted to have a reference to know what size holes worked best with our parts. We also wanted to know how much of a "center add" we would need to add for chain and gears. This is because we noticed that the chain was loose, and gears were slightly tight when we used the exact center-to-center distance. So, we created 3 laser-cutter tests. The first is a hole guide for screws, bushings, and bearings.





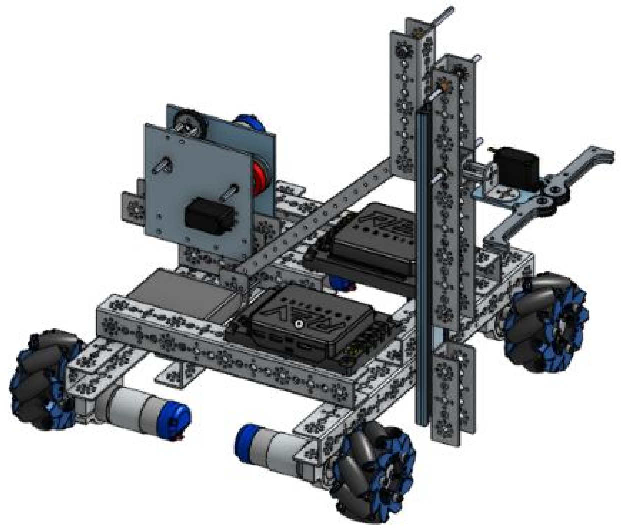
We also created a second bearing block that used tabs to hold a plate perpendicularly in place.



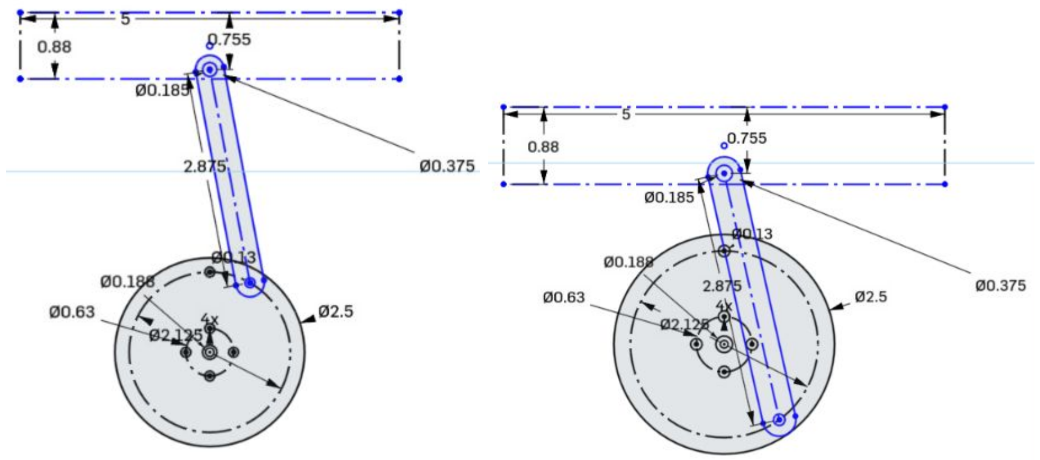
However, once assembled, we found a problem with the gear and sprocket guide. There was so much play in the bushings that all of these combinations worked. Because of this, we will worry less about future designs. For the hole sizes, we found that 3.0mm works for tapping the #6-32 screws, while 3.3mm is a good fit. 7.8 and 11.8mm are the correct sizes for the bushings and bearings. Finally, for the bearing block, there was too much play in the tabs for this design to be successful. So, we will need to adjust this later and re-cut it.

Nov. 11

To maintain its usefulness, we updated the CAD model of the test robot. This will help us when planning to add more mechanisms to the test bed.

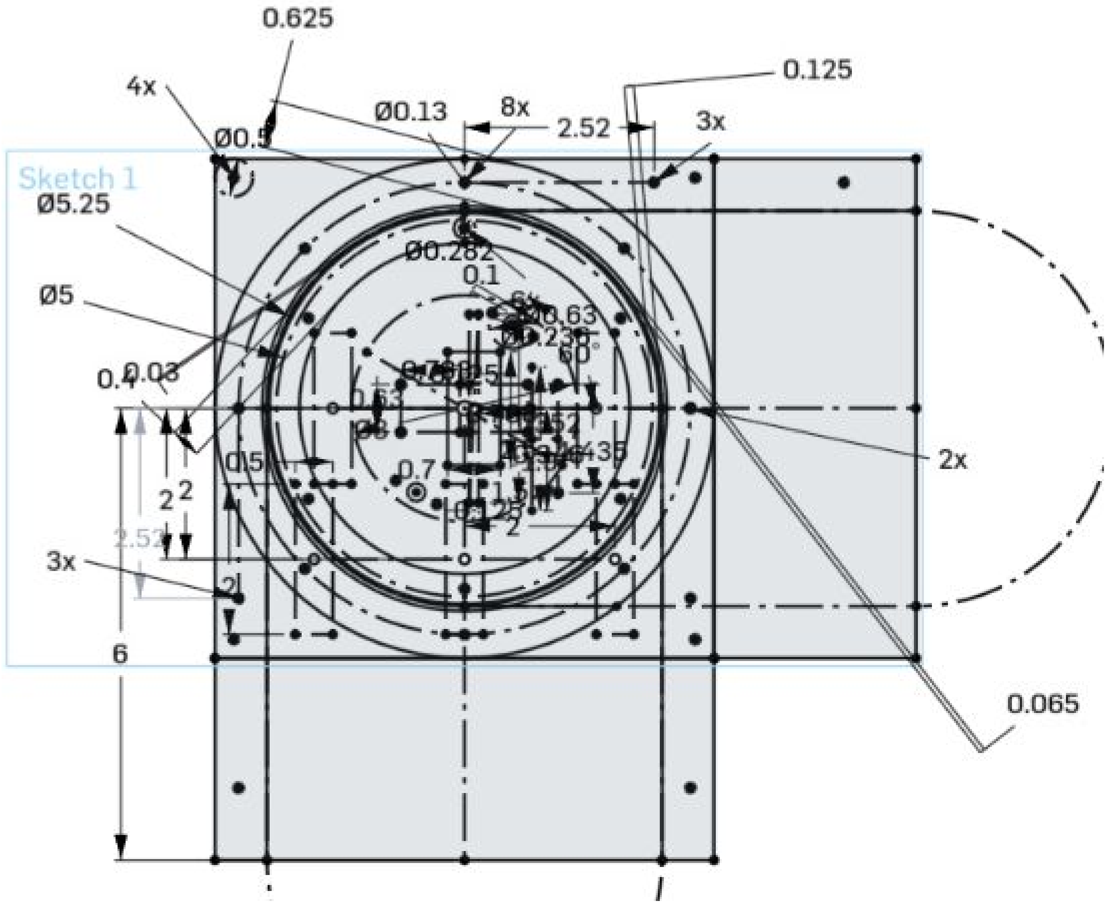


We also wanted to create a second hopper prototype to test Thursday. This will differ completely from the first hopper, as it will push upwards against rollers to dispense. This is an idea that we have been pondering for several reasons. First, we have more control over the discs, and second, we can try to push up as fast as possible to dispense the discs faster. With these characteristics in mind, we decided to create a hopper that resembles a piston

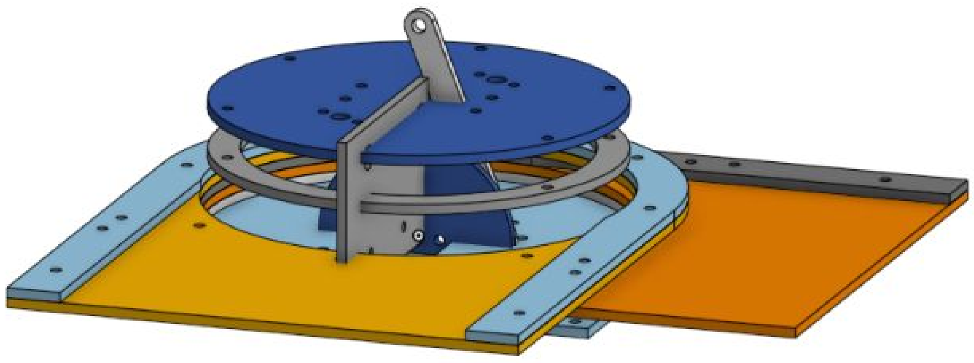


With this basic sketch, we now know what size wheel and lever to use. This is found by ensuring that the hopper will have enough travel for our purpose. We plan to power this with a servo, because it is good at holding a position. But our main concern with this is whether the servo is fast enough for this.

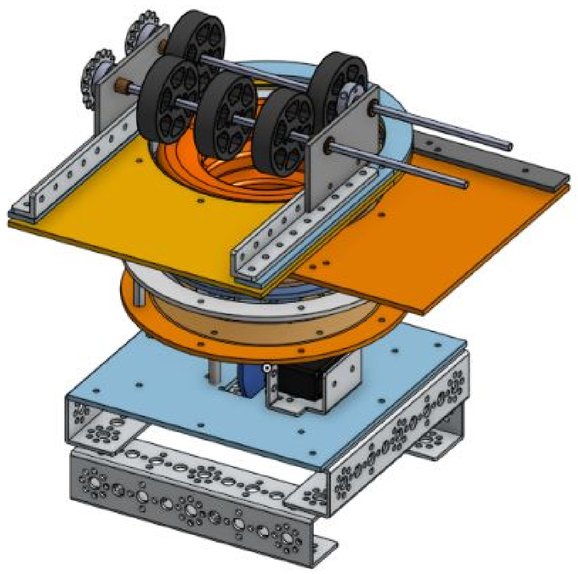
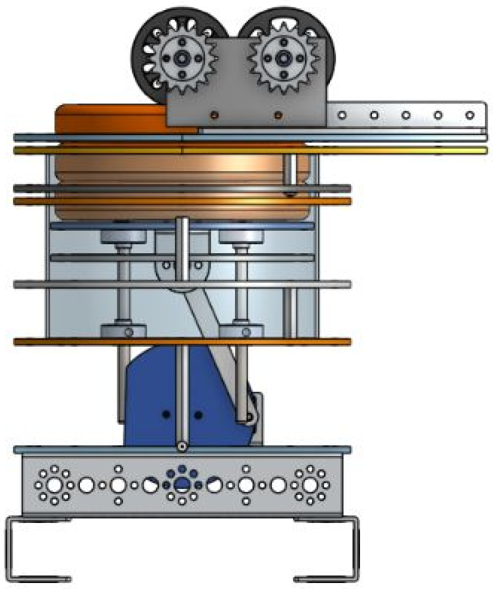
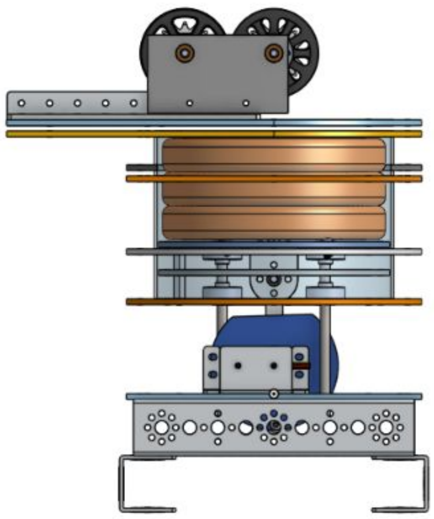
We then go ahead and create a main sketch from the top view. We picked the top view because the main structure will be cut out this way. We know from our previous hopper that a 5.25in ID will work well for holding the discs yet letting them settle properly. We all want to create a layered structure that will help form our cardboard tube. This will contain two layers that include a plate to load from one side, and unload from 90 degrees to the left. Finally, it will have a place to mount a servo that will actuate the base plate. To support this base plate, we plan to use the 6mm rods that we have along with the Tetrix 6mm hubs to "slide along". Here is a sketch of the design once everything has been added:



While it may be confusing for others, this has all our information on one sketch, so any adjustments needed will be easy and reflected on all the parts. We then carefully extrude the pieces we need to make this hopper. We make everything ⅛ in thick, because that is the thickness of our material. We also extrude 2 other sketches. These sketches are different because they are on different planes, and therefore reside elsewhere.



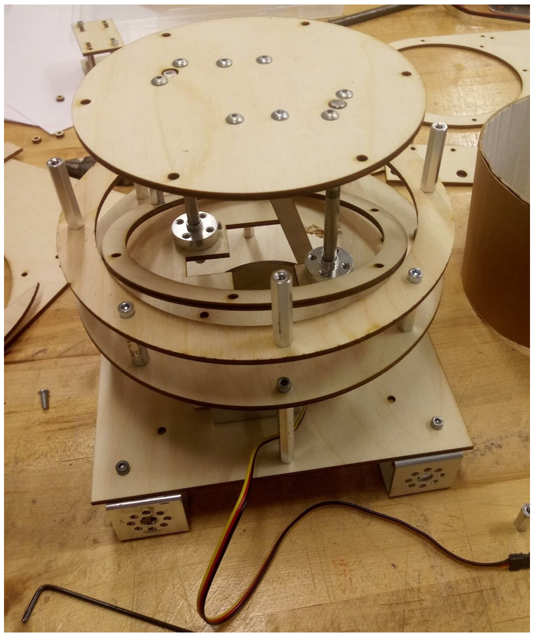
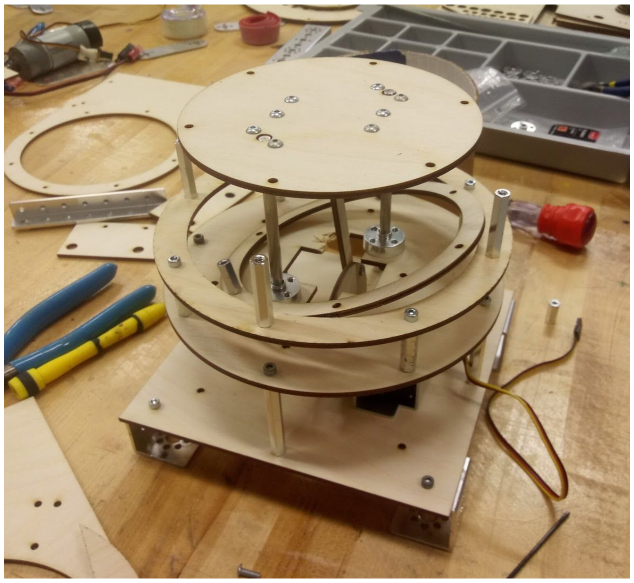
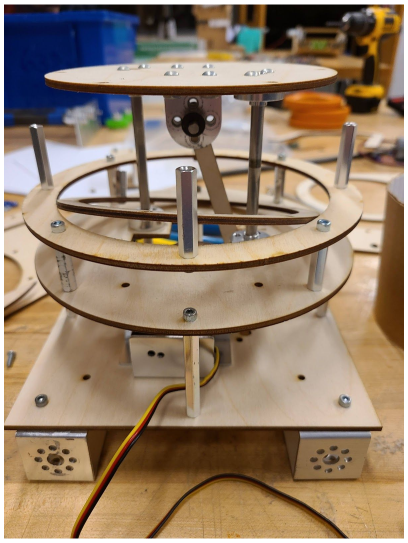
Then, from here we added all the COTS parts and assembled the structure with spaces and shafts. We wanted to try 2in wheels, and the 2 sets will be powered through chains. Here is the assembled version of the hopper:

With this design, we have almost 3 inches of movement, which allows us to load from a lower platform and dispense the rings on the platform upwards. The servo mounted on the bottom will press the rings along the moving rollers and these rollers will move the discs out to where the shooter would be situated.

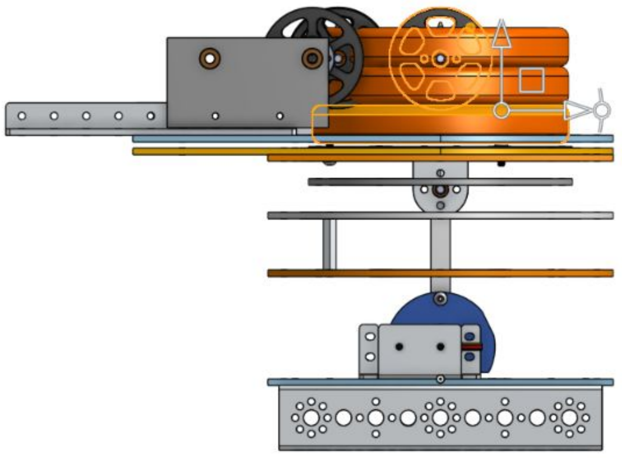
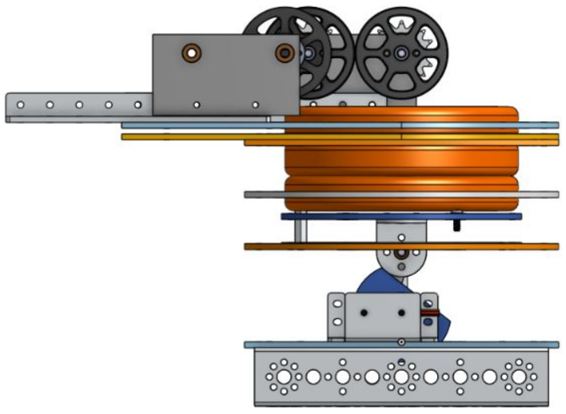
Thursday, Nov 12

We then made the laser-cutting files for this and assembled it in person. Once it was assembled, we noticed that the 6mm rods did not make the platform smooth. They caught up and made it very hard to lift the platform up. The overall design will also be improved, as lifting the discs up that much doesn't seem necessary.

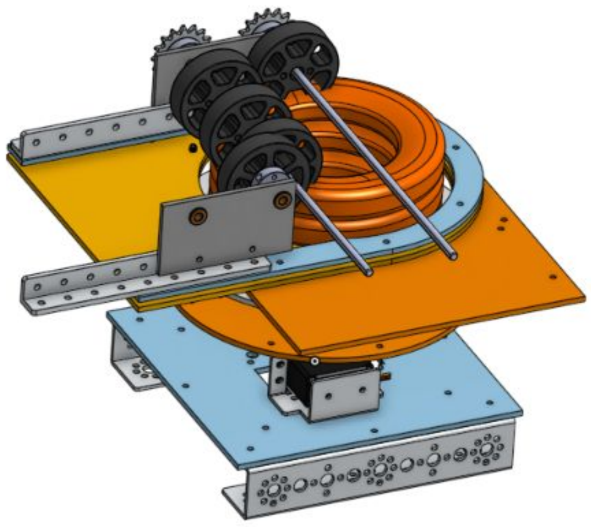




Saturday, Nov 14
In order to fix the hopper, there will be several changes made. First, instead of using the 6mm shafts the guides for the base platform, we will make the platform ride inside the cardboard tube. This should guide the plate straight, yet reduce enough friction. The other change is a bit more complicated. After discussion we envisioned a hopper system with some different properties. Our goal would be that this system allows discs to travel almost effortlessly through it, yet when we acquire more than 1 disc, there is an easy way to store those extra discs, yet still allowing more discs to pass through. With this, we realised that we can make some simple changes to our existing hopper to test this. The first change we make is that the entrance and exit plate should be near the same level. Because of our model, we can easily make changes to the assembly to prepare for Tuesday's meeting. After doing that, we realise that some other

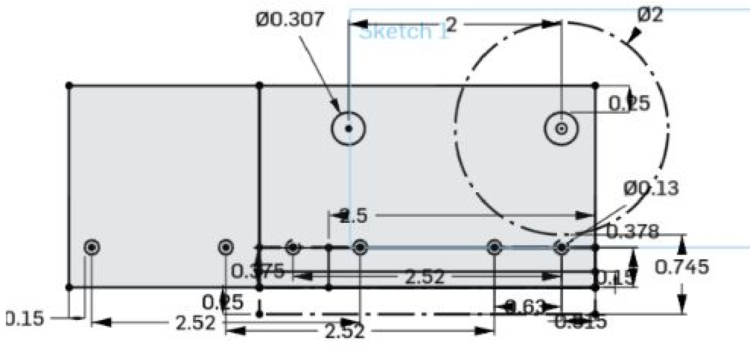
parts get in the way. To fix this problem, we create new plates for the rollers that allow discs to enter the hopper.

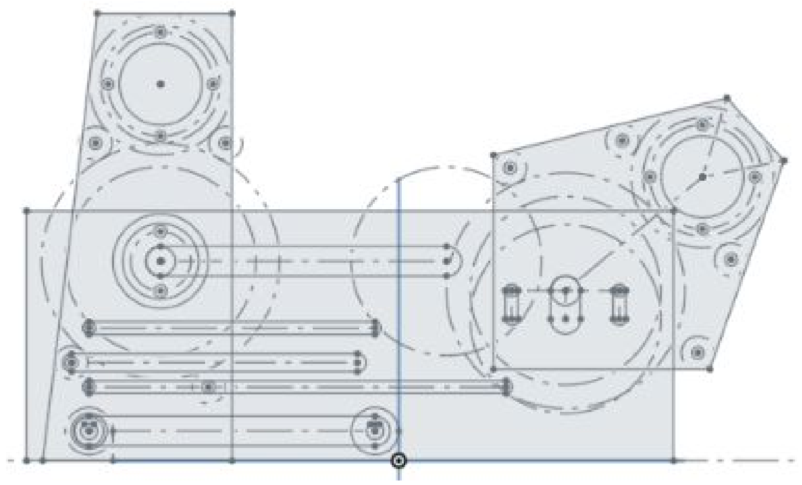
N

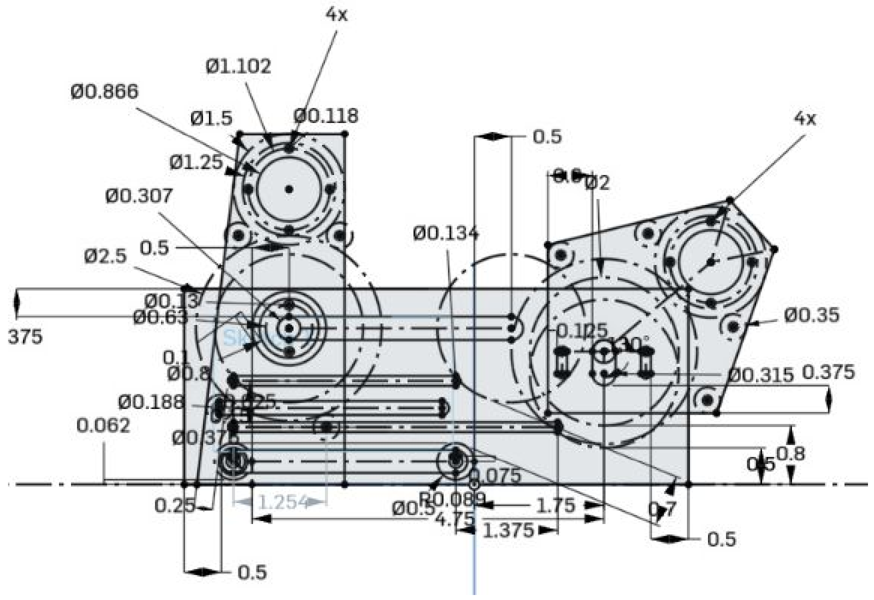


New Plate:

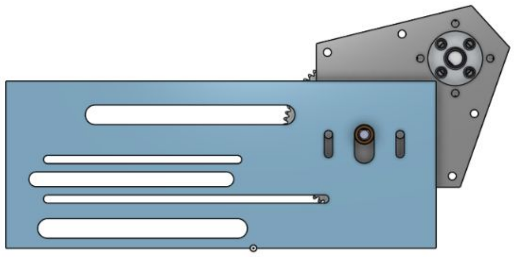


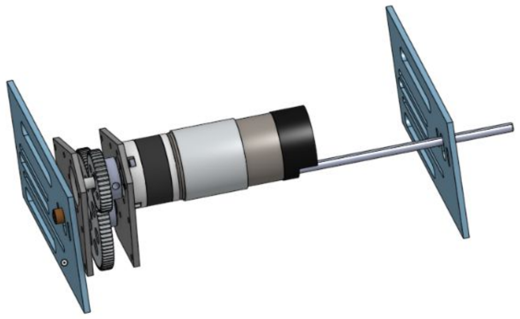
Now that this hopper is ready for Tuesday, we also need to make a new collector prototype to try out a new idea. When we were testing the previous version, we noticed that it only worked when the ramp was flush with the ground. But when we are driving around, we would rather not have the ramp contacting the ground. So one idea we had was to create a small bottom roller that helps the disc lift up and into the collector. But one thing we didn't know is how far back this roller should be from the first roller. So, we decided to create this prototype. Our variable to test will be the compression of the first roller, and the distance to the bottom roller. We set limits for this travel, and create a sketch that includes all of these constraints.

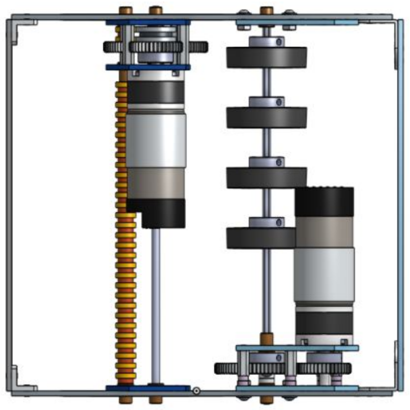


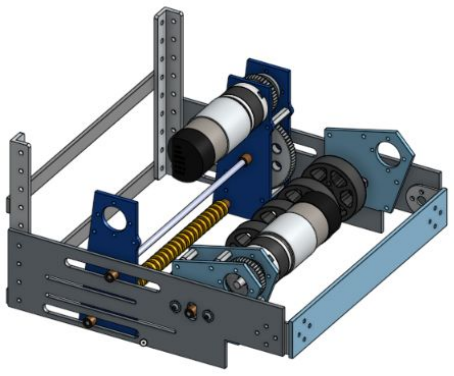


Basically this design contains two small gearboxes that are each allowed to slide along a slot. This allows everything to be rigid, yet also flexible to move around in location. We then extrude this sketch to make all the parts. We then add all the parts in to create an assembly, as usual.

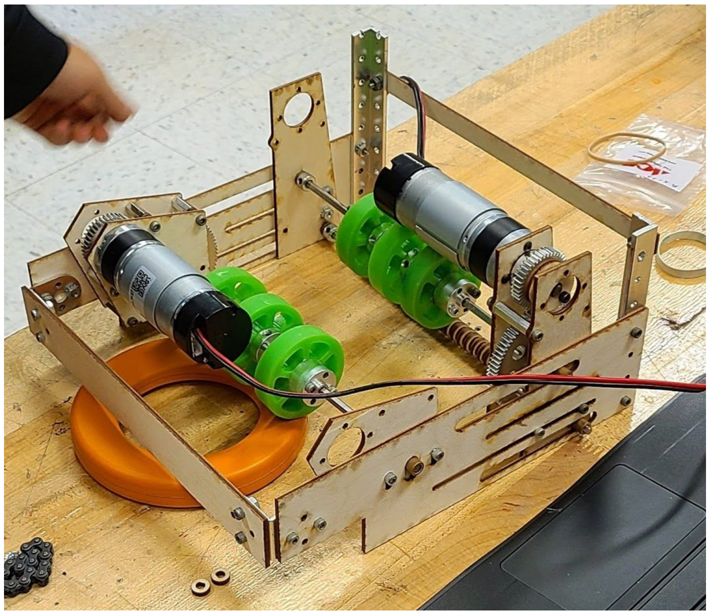
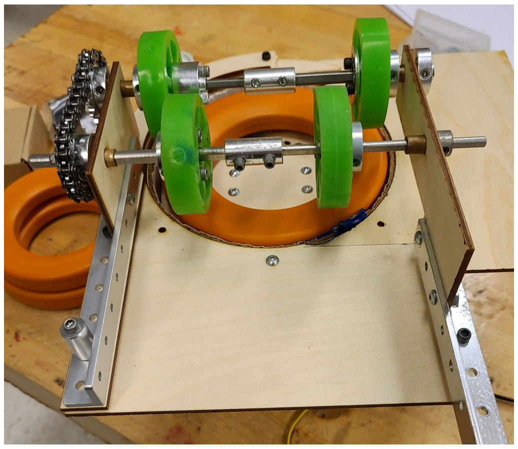
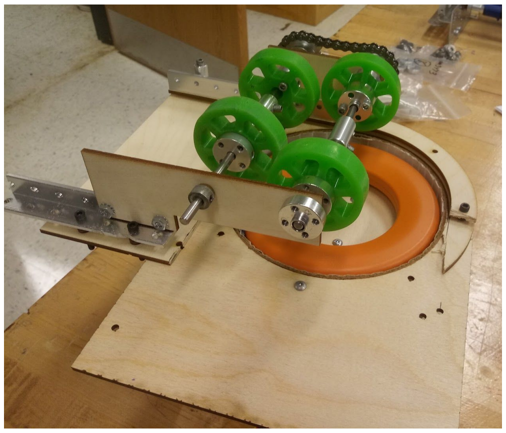
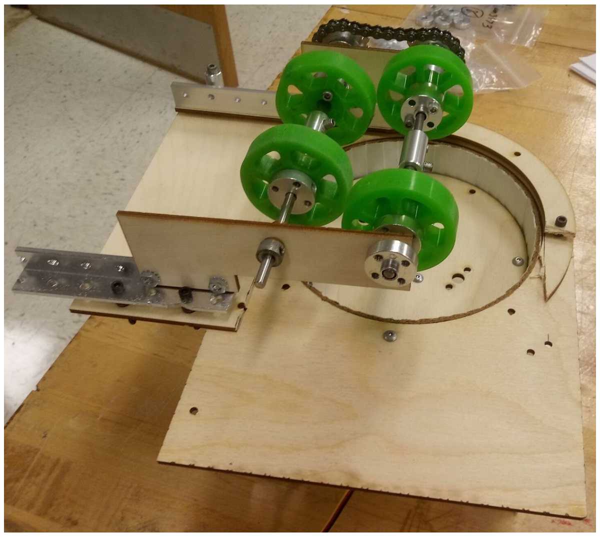




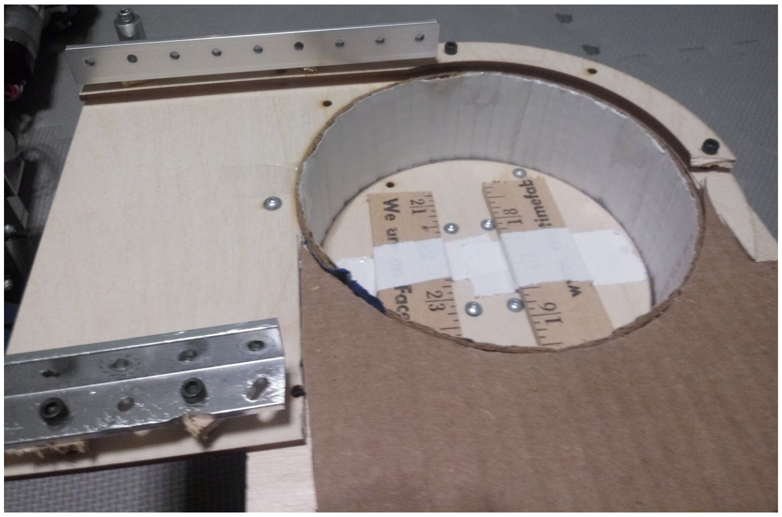
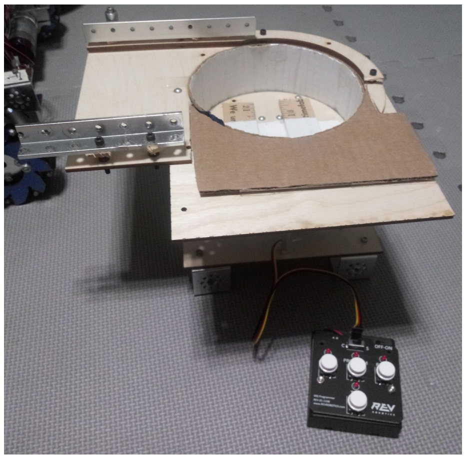
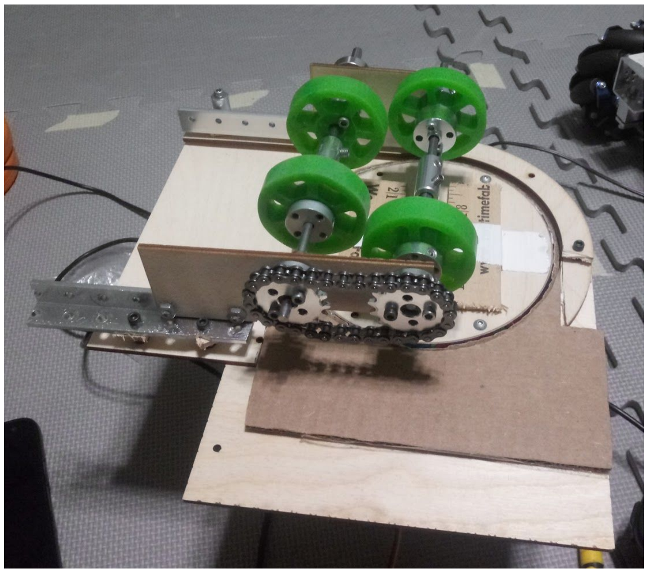




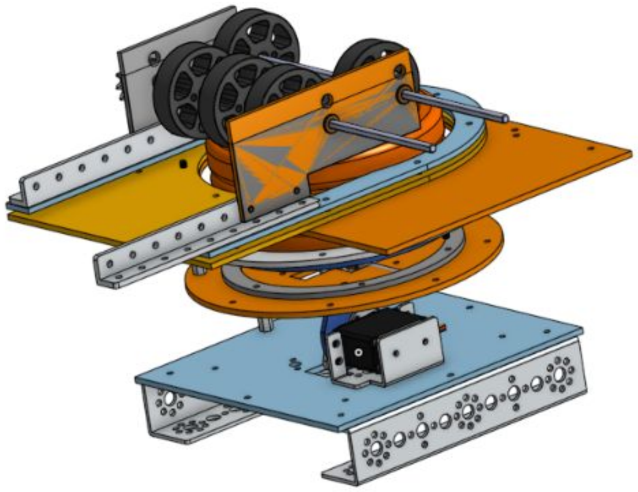
Tuesday, Nov 17
The hopper and collector were assembled and seem to work well. The main problem with the hopper at the moment is that the exit platform is raised up, and that stops the disc from coming onto it. This problem will be fixed by Niranjan, as he is taking it home. Thomas will be taking home the collector to further test it, and find the optimal distances.

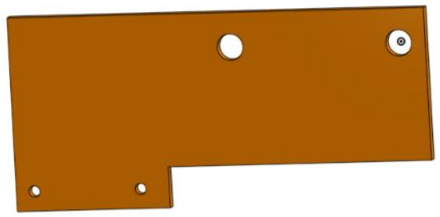





<u>Wednesday, Nov 18.</u>
A quick fix was made to the hopper. By lining the entrance and base plate with cardboard, everything can be made on the same level. To accommodate this, we also raised the top wheels up by 1/8 in as well. We also made the inner moving plate slightly wider (with cardboard) to make the “piston” move a bit smoother. This, along with our newly acquired SRS programmer made it very easy to test the hopper, and we were impressed.
For the collector, we found that the rubber bands were breaking very frequently, so we relocated them to try and solve the problem for now.

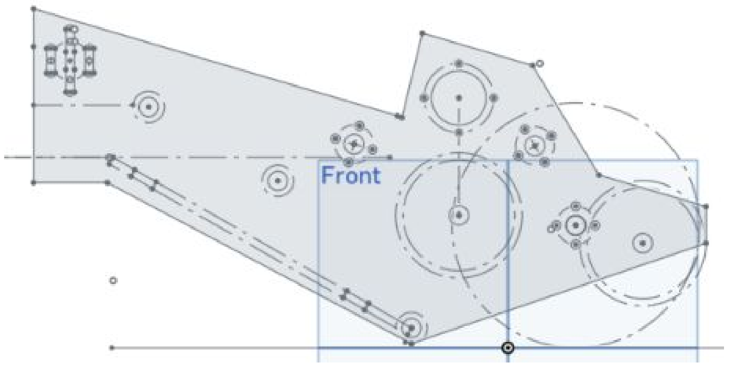




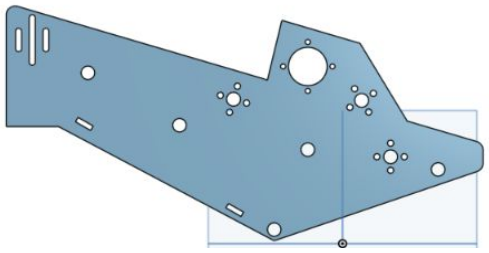
While we were discussing, we realised that the next step would be to create a “final-ish” collector that would fit onto the test robot. For this, we needed all the green wheels that we could get. So, we made some changes to the hopper design to accommodate 3in wheels. This allows both designs to be created, and also gives us the opportunity to try using omni wheels. Because of the rollers on the omni wheels, we thought we may have an easier time loading the discs in perpendicular to the rotation of the wheels.



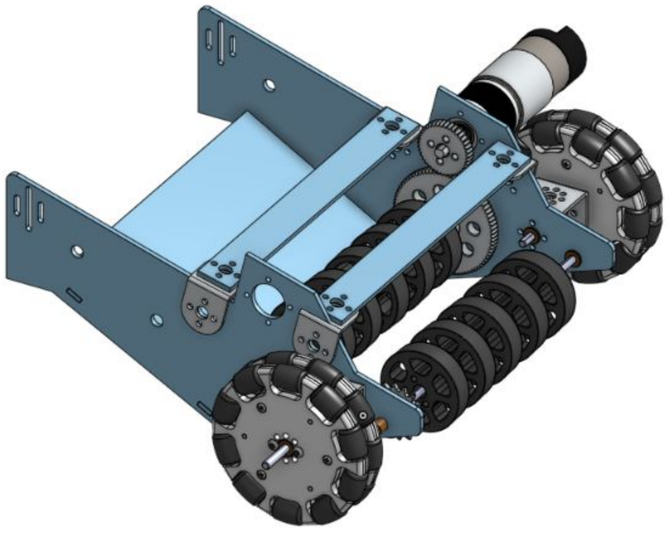


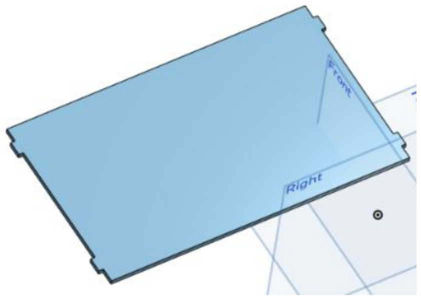
For the collector, we still wanted to have some variability. To allow for this, we wanted a mounting position that was able to change. To reduce complexity, we also were okay with this sticking absurdly out of the test chassis. So, we decided to support the end of the collector with its own omni wheels. Finally, we included the measurements that we collected from the prototype. Since we made the prototype, the dimensions and distances for this design were much easier.



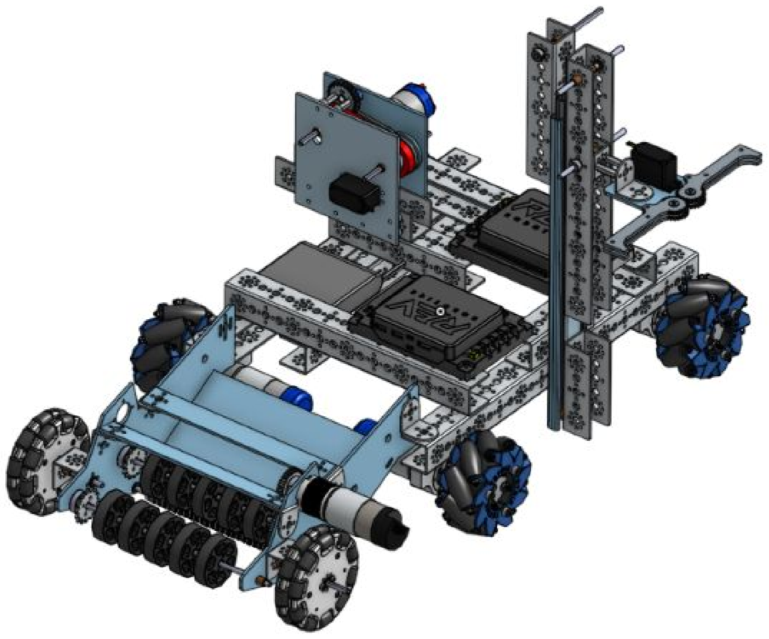


For this design, we used a middle plate with tabs to hold everything together. We then added this bottom plate along with other components to create a CAD assembly.

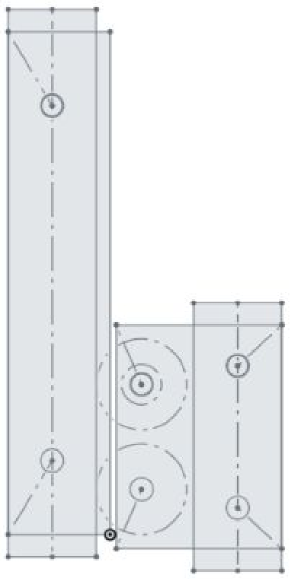
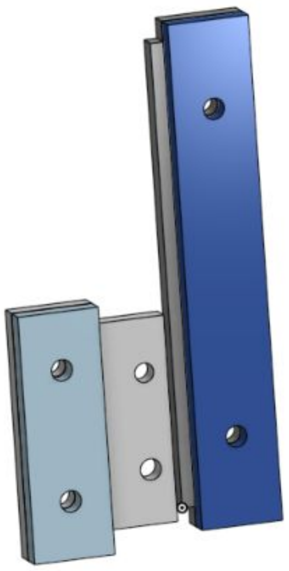
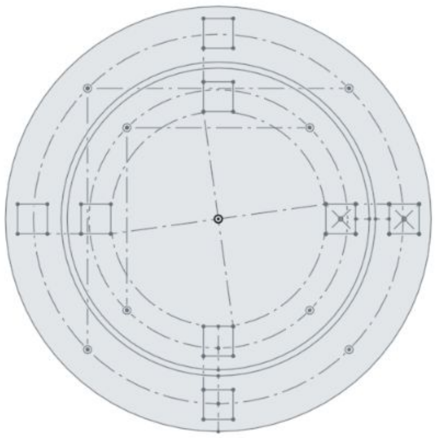
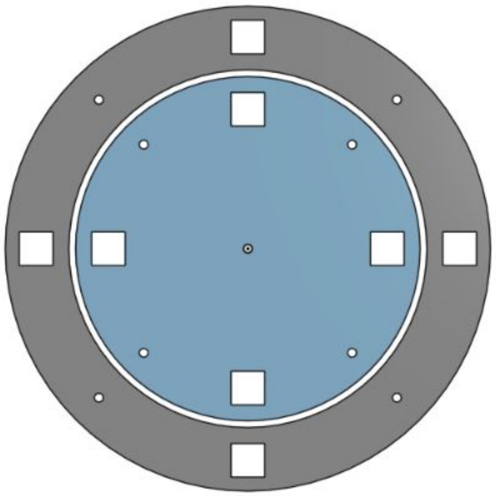
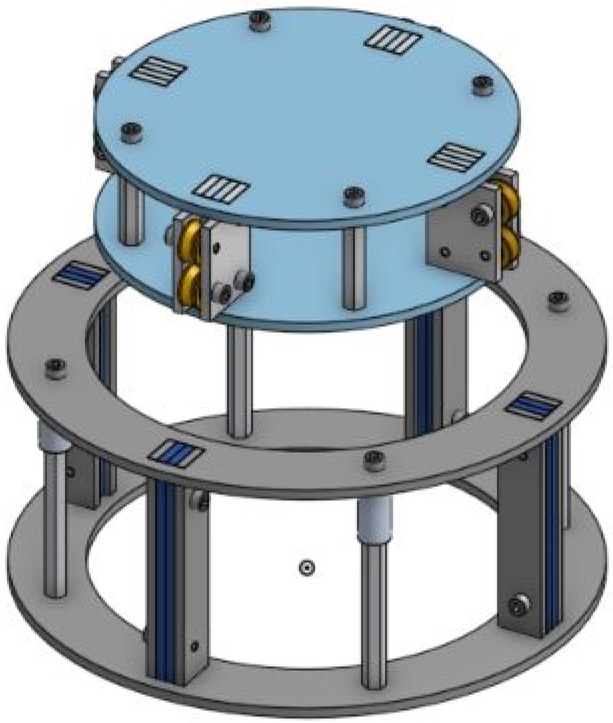




And since we have a model of the test chassis, we were able to make sure this fit on that robot without ever having to create it.

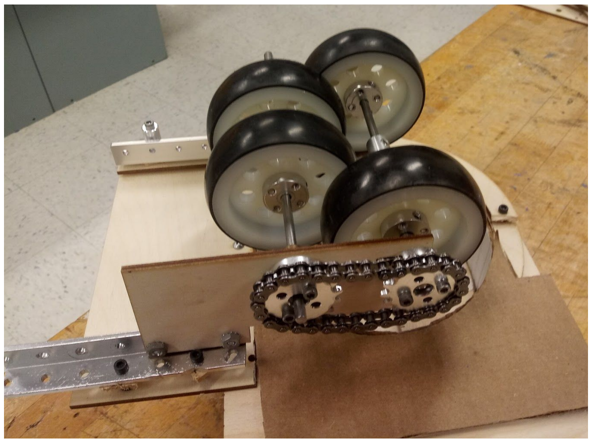


Finally, we wanted to create one last test for Thursday’s meeting. Thinking ahead, we need a more rigid way for the hopper to move up/down, without using cardboard. One idea that we had was to have bearings on the base plate that rolled along tracks. This would, in theory, be a super smooth and rigid structure that took side loads off of the servo.

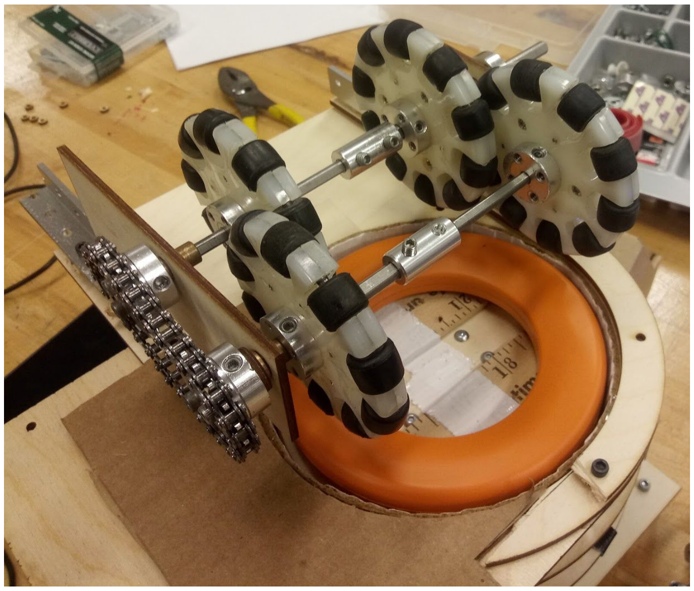






Thursday, Nov 19

Today we assembled and made changes to all of the designs we had created on Wednesday. Starting with the hopper, the 3 in tetrix wheels were too wide, and caused problems rubbing against things.

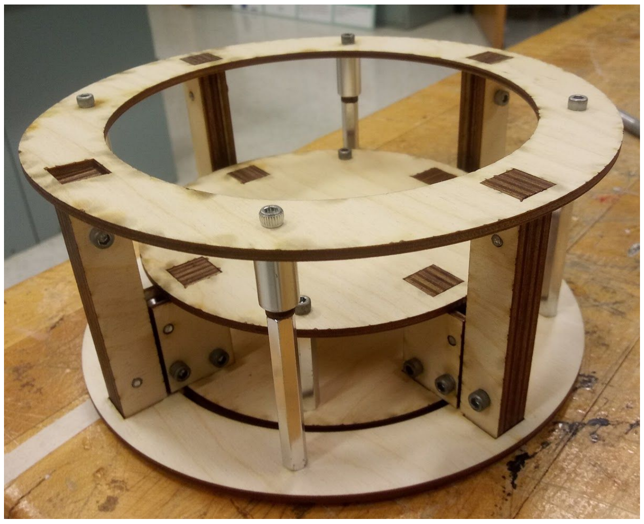


But the omni wheels seem to work okay, as they are thinner. While there are some benefits of the omni wheels (loading the discs is slightly easier), the wheels seem to slip a little. This is detrimental to our design because it relies on the wheels ejecting the disc as fast as possible. Our plan for this is to keep testing this to substantiate this concern.



For the collector we were able to assemble it and attach it to the robot. After some testing, the only concern is one that was already brought up. The design only works when the bottom roller is spinning. This just means that we will need to power it with something better than a rubber band on the final version.

Finally, our hopper structure was a bit of a flop. With the design, the tolerances were too tight, and the bearings were getting pressed against the rails very hard. This made it very tight. In order to fix this, we can make the rails slightly farther apart.



Final Robot Notes:

Collector System CAD Notes:

- Currently set up for a 18 tooth pulley on the motor and a 36 tooth pulley on the third roller. With a 49 tooth belt in between them. I figured the pulleys we are making are slightly smaller than commercially available ones so I used the exact center to center distance (can be changed if we feel we need to). https://www.technobotsonline.com/timing-pulley-distance-between-centres-calculator.html - this is the calculator I used.
- Current idea is that the motor has its own plate that is held with standoffs to the main plate. This motor plate is also attached to both the third and fourth rollers for stability.
- Current idea for the front pivoting roller is four super simple plates. Sets of two held together with a standoff. One set on either side of the front roller. This could be tensioned down with some sort of rubber. (?)
- Gear cover plate could be connected to the main plate using standoffs.
- Gearing setup for the bottom roller is two gears in between the second roller and the bottom roller. An extra gear is needed to reverse the direction. A double check needs to be done on this to make sure there will be enough clearance for the collar that goes on the back of the front wheel axle.

Top Sketch + Side Sketch Notes:

- Another pulley calculator (in inches) https://www.pic-design.com/calculators/belts-and-pulleys/calculator.htm
- Set Bumper thickness to 0.09 in, which is the thickness of the thin acrylic at school. These plates can be pretty thin, but we need to make sure we have enough to make all 4 sides.
- Right now I have set our thin washers to 1 mm thick. The purpose of these washers is just to make sure that hubs and wheels contact the inner race of the bearings.
- The new wheel mounts are slightly larger than the previous one. But, it trades the 5mm it gained to be even more sturdy, by using a collar instead of a retaining ring.
- Right now there is 0.004 in. of added distance to the center-center distance for the chain. This should help a little, but there is no set number that we should use.
- The motors represented in the sketches are slightly larger than the actual motors by very little.
- A cut-out in the base plate for the collector was created. The left side of this will have to be increased to allow for the gear cover plate.
- A hole will be cut-out of the base plate to allow the color sensor to shine below. This will help orient to the midline. To make orienting more efficient, this will need to be moved closer to the back of the robot.

- If the cut-outs weaken the base plate too much, we can add layers of Delrin to the areas that require stiffening.
- The collector ramp in the side sketch was changed so that is was inline with the bottom roller
- The collector ramp is now at 27 degrees. This gives enough space for a hopper, but also isn't too large of an angle.
- As a note, the collector cutout is an approximation, so any mounting should be done separately with holes.
- A cutout in the sides bumper plates will be required to allow the bolt heads to have enough space.
- A cutout in the front/back bumper plates will also be required. This will be a small rectangle to give the wheels enough clearance to spin. This is because the wheels were made tangent with the outside edge of the robot for maximised inner space.
- The base plate for the front wheels is very thin, and therefore may be able to flex around. But, with the current idea for the drive assembly, there should be enough support for the base.
- Gears were added to the bottom collector. This was later fixed so it spins the correct way.
- Mounting plates for front wheels were created. These plates will have tabs that fit together with one another. They will also have standoffs securing everything together. Lastly, there will be brackets that secure this to the bottom plate.
- A "chain" was added to the top sketch for reference. THis will help make sure that parts stay away from this.
- Mounting spaces for the distance sensors will need to be added. These will probably mount to the bumper plates.
- The pivot plate now uses two standoffs to hold itself together. There will be two of these pairs, which should hold the collector parallel
- A Shooter+Hopper sketch was made. These dimensions are not exact and this is just a quick sketch to see where the space is. This also helped create a sense of where interferences may be.
- The Hopper for the sketch was made 5.75in. Wide with a 5.25in. ID. All of these dimensions can easily be changed
- The front edge of the hopper was made in-line with the arc. This was to help prevent mis-alignments
- A problem was found. The 104t belts that were ordered for the shooter are too short. After some calculations, a 112t was found to work better. This new update was added to the sketch.
- Horizontal wheels need to be added to help guide discs into the collector.

- Parts were extruded and made into an assembly to help judge where space is. This assembly is not accurate, and therefore will need to be replaced for the final assembly.
- A quick shooter plate was created, but this also will need to be replaced.

Edits made to collector (12/8-12/)

- Added tabs and mounting brackets
- Made a servo hook (should work but it might be janky, this is something we can change later though if needed)
- Added mounting brackets for top plate
- Added bearing holes for the shooter/hopper pivot points