Honors Program and Design Project: Battle Bot Calculations

Introduction & Goals

When deciding what to pick as a project topic we knew that we wanted to do something that was challenging, but also taught us new skills beyond our in class experiences. We decided that we wanted to do something relating to physics and robotics and from there we quickly settled on Battlebots. Once we had decided on a general topic we narrowed it down to the creation of a program that would assist in the creation and function of Battlebots while in combat. Additionally, we also chose this topic because we were confident that it would be achievable in the month time period while also providing a fun and interesting experience at the same time.

Specifically, we aimed to create working gear ratio, weapon speed, rotational kinetic energy, and mass moment of inertia calculators. We chose these values in particular as the weapon speed and kinetic energy values represent the impact a Battlebot's weapon has on the opposing bot while the mass moment of inertia and gear ratio values are necessary to find the two other values. We accomplished this by researching physics concepts as well as Battlebots as a whole. The purpose of this project beyond the application of the calculator itself was to combine and demonstrate our group's knowledge and interests of programming, mathematics, physics, and robotics. By completing this project we have learned more about the world of robotics and its intersection with Computer Science. This is particularly exciting as both of these fields are rapidly developing in the modern day world.

Distribution of Labor

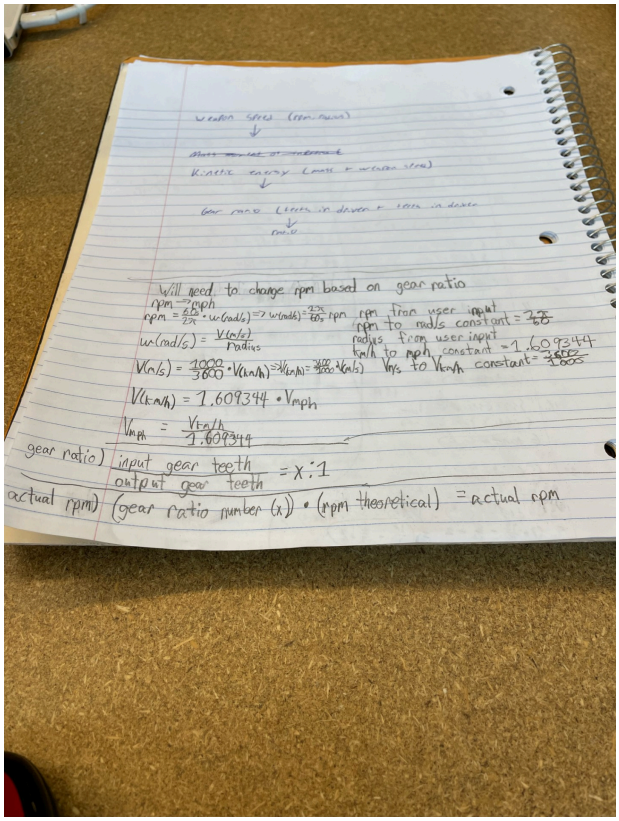
When deciding on how to divide up the labor for the project so that we could both complete it on time and ensure that everyone contributed evenly, we decided to go for a flexible approach. Instead of having each of us strictly focus on one thing we decided to share the workload so that it would be easier to complete, and so that we all benefited from the learning process throughout the project. From actual programming to research, slide design, this very word docs development, and more. We all actively divided up work in multiple areas of the project to achieve what we deemed was the best form of productivity and time management. We would say that this flexible approach was very successful for our first group coding project and will most likely implement it in other class projects and our future careers.

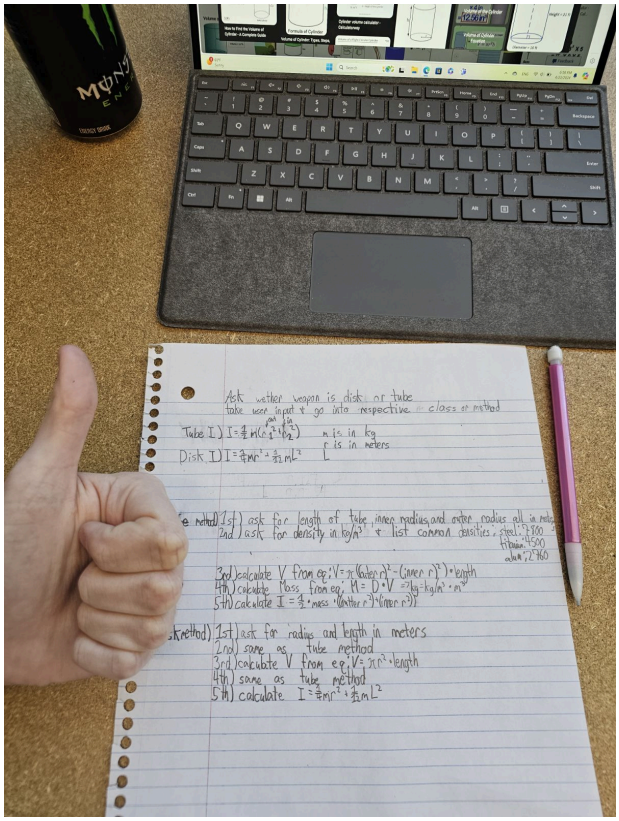
The Rough Flexible Distribution

Alex:, Research, mass moment of inertia, written report, debugging, outline planning, and editing
Daniel: slides design, written report, research, gear ratio calculator code, and debugging
Nick: research, weapon speed code, kinetic energy code, written report, slides, editing, and debugging

Methodology

Our progression for completing this project began with rough outlining and discussing the framework for our final code. This included work on scratch paper outlining roughly what classes we thought would be necessary as well as potential parameters to occupy the methods in each class. We also thought about the outputs we'd be looking for from each class to guide our programming. Additionally, we also completed the math and physics portions of this project, as well as optimizing the equations for easier programming. As seen below, our methods for completing the equations were done so that they could be coded as seamlessly as possible. This includes breaking up otherwise continuous equations into individual mathematical functions. After completing this, we moved into creating our initial programs





From here, we individually completed four different rudimentary programs to create and test the calculations being done by our project, as well as to divide the difficult sections. Each of us separately created the four unique and standalone programs that all complete a specific mathematical function. From here, we hoped to combine them into one cohesive program, and we hoped that this method would increase our efficiency, as all three of us working on one program would be very time consuming and ineffective. Below are our programs, all with our own programming styles and conventions.

Program to Compute a Gear Ratio

```
import java.util.Scanner;

public class GearRatio
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of teeth on the first gear (the driver
gear):");
        int teethDriver = scanner.nextInt();

        System.out.println("Enter the number of teeth on the second gear (the
driven gear):");
        int teethDriven = scanner.nextInt();

        // Calculate the gear ratio
        double gearRatio = (double) teethDriver / teethDriven;

        // Format the gear ratio to truncate after 4 decimal places
        String formattedGearRatio = String.format("%.4f", gearRatio);

        System.out.println("The gear ratio is " + formattedGearRatio + " : 1");
    }
}
```

Program to Compute the Kinetic Energy

```
import java.util.*;
public class KineticEnergy
{
    public static void main(String[] args)
    {
        Scanner doubleScanner = new Scanner(System.in);
        System.out.println("Enter in the rpm of the weapon for the calculation of
the weapon speed");
```

```
        double rpm = doubleScanner.nextDouble();
        System.out.println("Enter in the moment intertia drawn from groupmates
other program");
        double momentIntertia = doubleScanner.nextDouble();

        double kineticEnergy;

        //rotational kintetic energy = 1 / 2 * moment of inertia around the axis of
rotation * angular velocity (rotations per minute)
        kineticEnergy = (1 / 2) * inertia * rpm * rpmToRadS;

        System.out.print(kineticEnergy);
    }
}
```

Program to Compute the Weapon Speed

```
import java.util.*;
public class WeaponSpeed
{
    public static void main(String[] args)
    {
        Scanner doubleScanner = new Scanner(System.in);
        System.out.println("Enter in the rpm of the weapon for the calculation of
the weapon speed");
        double rpm = doubleScanner.nextDouble();
        System.out.println("Enter in the rotational radius for the weapon for
calculation of the weapon speed");
        double weaponRadius = doubleScanner.nextDouble();

        double kmhToMph = 1.609344;
        double rpmToRadS = (Math.PI * 2) / 60;
        double velocityMph, velocityMs, velocityKmh, wRads;

        //Rotational velocity = rpm to radians constant * rpm
        wRads = rpmToRadS * rpm;

        //Velocity meters/second = rotational velocity * radius
        velocityMs = wRads * weaponRadius;

        //velocity kilometers/hour = meters/second to kilometers/hour constant *
velocity meters/second
        velocityKmh = (3600/1000) * velocityMs;

        //Velocity miles/hour = velocity kilometers/hour * kilometers/hour to
miles/hour constant
        velocityMph = (velocityKmh / kmhToMph);
```

```
        System.out.print(velocityMph);
    }
}
```

Program to Compute the Momentary Inertia

```
import java.util.Scanner;

public class MomentInertia {
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);

        // Prompt user to select the shape
        System.out.println("Enter the shape (1 for disk, 2 for tube):");
        int shape = scanner.nextInt();

        if (shape == 1)
        {
            calculateDiskInertia();
        } else if (shape == 2)
        {
            calculateTubeInertia();
        } else
        {
            System.out.println("Invalid shape selection.");
        }
    }

    public static void calculateDiskInertia()
    {
        Scanner scanner = new Scanner(System.in);

        // Ask for dimensions
        System.out.println("Enter radius of the disk (in meters):");
        double radius = scanner.nextDouble();

        System.out.println("Enter length of the disk (in meters):");
        double length = scanner.nextDouble();

        // Ask for density
        System.out.println("Enter density of the material (in kg/m^3):");
        System.out.println("Common densities: Steel (7800), Titanium (4500),
Aluminum (2760)");
        double density = scanner.nextDouble();
```

```
        // Calculate volume
        double volume = Math.PI * Math.pow(radius, 2) * length;

        // Calculate mass
        double mass = density * volume;

        // Calculate inertia
        double inertia = (0.25 * mass * Math.pow(radius, 2)) + (1.0/12 * mass *
Math.pow(length, 2));

        // Print inertia
        System.out.println("The Mass Moment of Inertia of the disk is: " + inertia
+ " kg*m^2");
    }

    public static void calculateTubeInertia()
    {
        Scanner scanner = new Scanner(System.in);

        // Ask for dimensions
        System.out.println("Enter length of the tube (in meters):");
        double length = scanner.nextDouble();

        System.out.println("Enter inner radius of the tube (in meters):");
        double innerRadius = scanner.nextDouble();

        System.out.println("Enter outer radius of the tube (in meters):");
        double outerRadius = scanner.nextDouble();

        // Ask for density
        System.out.println("Enter density of the material (in kg/m^3):");
        System.out.println("Common densities: Steel (7800), Titanium (4500),
Aluminum (2760)");
        double density = scanner.nextDouble();

        // Calculate volume
        double volume = Math.PI * (Math.pow(outerRadius, 2) - Math.pow(innerRadius,
2)) * length;

        // Calculate mass
        double mass = density * volume;

        // Calculate inertia
        double inertia = 0.5 * mass * (Math.pow(outerRadius, 2) +
Math.pow(innerRadius, 2));

        // Print inertia
```

```
        System.out.println("The Mass Moment of Inertia of the tube is: " + inertia
+ " kg*m^2");
    }
}
```

Continuing from this point, we then looked to format and fundamentally change our individual codes into one cohesive rough design. We created a program incorporating numerous methods that represented each individual program. We also looked to merge our individual programming conventions in terms of variable and method names into one product. More work in this would still be required in terms of perfecting this as well as writing style in comments.

Combined Rough Draft Code With Methods

```
import java.util.*;
public class CombinedProject {
    public static void main(String[] args) {
        Scanner doubleScanner = new Scanner(System.in);
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter in the rpm of the weapon for the calculation of
the weapon speed");
        double rpm = doubleScanner.nextDouble();
        System.out.println("Enter in the rotational radius for the weapon for
calculation of the weapon speed");
        double weaponRadius = doubleScanner.nextDouble();
        System.out.println("Enter the number of teeth on the first gear (the driver
gear):");
        int teethDriver = scanner.nextInt();
        System.out.println("Enter the number of teeth on the second gear (the
driven gear):");
        int teethDriven = scanner.nextInt();
        System.out.println("Enter the shape (1 for disk, 2 for tube):");
        int shape = scanner.nextInt();

        if (shape == 1) {
            DiskInertia();
            System.out.println(KineticEnergy(rpm, DiskInertia()));
            System.out.println(WeaponSpeed(rpm, weaponRadius));
        } else if (shape == 2) {
            TubeInertia();
            System.out.println(KineticEnergy(rpm, TubeInertia()));
            System.out.println(WeaponSpeed(rpm, weaponRadius));
        } else {
            System.out.println("Invalid shape selection.");
        }
    }
```

```
    public static double DiskInertia() {
        Scanner scanner = new Scanner(System.in);

        // Ask for dimensions
        System.out.println("Enter radius of the disk (in meters):");
        double radius = scanner.nextDouble();

        System.out.println("Enter length of the disk (in meters):");
        double length = scanner.nextDouble();

        // Ask for density
        System.out.println("Enter density of the material (in kg/m^3):");
        System.out.println("Common densities: Steel (7800), Titanium (4500),
Aluminum (2760)");
        double density = scanner.nextDouble();

        // Calculate volume
        double volume = Math.PI * Math.pow(radius, 2) * length;

        // Calculate mass
        double mass = density * volume;

        // Calculate inertia
        double inertia = (0.25 * mass * Math.pow(radius, 2)) + (1.0 / 12 * mass *
Math.pow(length, 2));

        // Print inertia
        System.out.println("The Mass Moment of Inertia of the disk is: " + inertia
+ " kg*m^2");
        return inertia;
    }

    public static double TubeInertia() {
        Scanner scanner = new Scanner(System.in);

        // Ask for dimensions
        System.out.println("Enter length of the tube (in meters):");
        double length = scanner.nextDouble();

        System.out.println("Enter inner radius of the tube (in meters):");
        double innerRadius = scanner.nextDouble();

        System.out.println("Enter outer radius of the tube (in meters):");
        double outerRadius = scanner.nextDouble();

        // Ask for density
```

```
        System.out.println("Enter density of the material (in kg/m^3):");
        System.out.println("Common densities: Steel (7800), Titanium (4500),
Aluminum (2760)");
        double density = scanner.nextDouble();

        // Calculate volume
        double volume = Math.PI * (Math.pow(outerRadius, 2) - Math.pow(innerRadius,
2)) * length;

        // Calculate mass
        double mass = density * volume;

        // Calculate inertia
        double inertia = 0.5 * mass * (Math.pow(outerRadius, 2) +
Math.pow(innerRadius, 2));

        // Print inertia
        System.out.println("The Mass Moment of Inertia of the tube is: " + inertia
+ " kg*m^2");
        return inertia;
    }

    public static double KineticEnergy(double rpm, double momentInertia) {
        double kineticEnergy;

        //rotational kintetic energy = 1 / 2 * moment of inertia around the axis of
rotation * angular velocity (rotations per minute)
        kineticEnergy = (1 / 2) * momentInertia * rpm * rpmToRadS;

        return kineticEnergy;
    }

    public static double WeaponSpeed(double rpm, double weaponRadius) {
        double kmhToMph = 1.609344;
        double rpmToRadS = (Math.PI * 2) / 60;
        double velocityMph, velocityMs, velocityKmh, wRads;

        //Rotational velocity = rpm to radians constant * rpm
        wRads = rpmToRadS * rpm;

        //Velocity meters/second = rotational velocity * radius
        velocityMs = wRads * weaponRadius;

        //velocity kilometers/hour = meters/second to kilometers/hour constant *
velocity meters/second
        velocityKmh = (3600 / 1000) * velocityMs;
```

```
        //Velocity miles/hour = velocity kilometers/hour * kilometers/hour to
miles/hour constant
        velocityMph = (velocityKmh / kmhToMph);

        System.out.print(velocityMph);
        return velocityMph;
    }

    public static double GearRatio(int teethDriver, int teethDriven)
    {
        // Calculate the gear ratio
        double gearRatio = (double) teethDriver / teethDriven;

        return gearRatio;
    }
}
```

From this point, we produced the final code, with one style and design, as well as utilizing a more effective and advanced structure. We moved away from one class with numerous methods to multiple classes and subclasses with objects to more effectively complete our project. This way is more reminiscent of advanced programming as well as taking advantage of the object-oriented aspect of Java. We also spent time making sure the grammar, syntax and style of variables, declarations, and comments are consistent. The result is a complete and cohesive project that is effective at solving our problem.

Core Code

```
import java.util.*;

class WeaponSpeed
{
    private double rpm, weaponRadius;
    private String formattedVelocityMph;

    //Constructor
    public WeaponSpeed(double rpm, double weaponRadius)
    {
        this.rpm = rpm;
        this.weaponRadius = weaponRadius;
    }

    //Method to calculate weapon speed
    public void CalculateWeaponSpeed()
```

```
    {
        double kmhToMph = 1.609344;
        double rpmToRadS = (Math.PI * 2) / 60;
        double velocityMph, velocityMs, velocityKmh, wRads;

        //Rotational velocity = rpm to radians constant * rpm
        wRads = rpmToRadS * rpm;

        //Velocity meters/second = rotational velocity * radius
        velocityMs = wRads * weaponRadius;

        //velocity kilometers/hour = meters/second to kilometers/hour constant *
velocity meters/second
        velocityKmh = (3600 / 1000) * velocityMs;

        //Velocity miles/hour = velocity kilometers/hour * kilometers/hour to
miles/hour constant
        velocityMph = (velocityKmh / kmhToMph);

        //Formats final velocity in miles/hour as a string to truncate after 4
decimal places
        formattedVelocityMph = String.format("%.4f", velocityMph);
    }

    //Display calculations
    public void DisplayInfo()
    {
        System.out.println("The velocity of the weapon in miles per hour is: " +
formattedVelocityMph);
    }
}

class KineticEnergy
{
    private double rpm, inertia;
    private String formattedKineticEnergy;

    //Constructor
    public KineticEnergy(double rpm, double inertia)
    {
        this.rpm = rpm;
        this.inertia = inertia;
    }

    //Method to calculate kinetic energy of the weapon
    public void CalculateKineticEnergy()
    {
```

```
        double kineticEnergy;

        //rotational kintetic energy = 1 / 2 * moment of inertia around the axis of
rotation * angular velocity (rotations per minute)
        kineticEnergy = (1 / 2) * inertia * rpm * rpmToRadS;

        //Formats final kinetic energy in miles/hour as a string to truncate after
4 decimal places
        formattedKineticEnergy = String.format("%.4f", kineticEnergy);
    }

    //Display calculations
    public void displayInfo()
    {
        System.out.println("The kinetic energy of the weapon in joules is: " +
formattedKineticEnergy);
    }
}

class MomentInertia
{
    private int type;
    private double inertia;

    //Constructor
    public MomentInertia(int type)
    {
        this.type = type;
    }

    //Method to calculate the momentary rotational inertia
    public void CalculateMomentInertia()
    {
        //Sorts calculations done by the given shape
        if (type == 1)
        {
            CalculateDiskMomentInertia();
        } else if (type == 2)
        {
            CalculateTubeMomentInertia();
        } else
        {
            System.out.println("Invalid shape selection.");
        }
    }

    //Method to continue calculations based on the shape of the weapon
```

```
    public void CalculateDiskMomentInertia()
    {
        Scanner scanner = new Scanner(System.in);

        // Ask for dimensions
        System.out.println("Enter radius of the disk (in meters):");
        double radius = scanner.nextDouble();

        System.out.println("Enter length of the disk (in meters):");
        double length = scanner.nextDouble();

        // Ask for density
        System.out.println("Enter density of the material (in kg/m^3):");
        System.out.println("Common densities: Steel (7800), Titanium (4500),
Aluminum (2760)");
        double density = scanner.nextDouble();

        // Calculate volume
        double volume = Math.PI * Math.pow(radius, 2) * length;

        // Calculate mass
        double mass = density * volume;

        // Calculate inertia
        inertia = (0.25 * mass * Math.pow(radius, 2)) + (1.0/12 * mass *
Math.pow(length, 2));
    }

    //Method to continue calculations based on the shape of the weapon
    public void CalculateTubeMomentInertia()
    {
        Scanner scanner = new Scanner(System.in);

        // Ask for dimensions
        System.out.println("Enter length of the tube (in meters):");
        double length = scanner.nextDouble();

        System.out.println("Enter inner radius of the tube (in meters):");
        double innerRadius = scanner.nextDouble();

        System.out.println("Enter outer radius of the tube (in meters):");
        double outerRadius = scanner.nextDouble();

        // Ask for density
        System.out.println("Enter density of the material (in kg/m^3):");
        System.out.println("Common densities: Steel (7800), Titanium (4500),
Aluminum (2760)");
```

```
        double density = scanner.nextDouble();

        // Calculate volume
        double volume = Math.PI * (Math.pow(outerRadius, 2) - Math.pow(innerRadius,
2)) * length;

        // Calculate mass
        double mass = density * volume;

        // Calculate inertia
        inertia = 0.5 * mass * (Math.pow(outerRadius, 2) + Math.pow(innerRadius,
2));
    }

    //Saves inertia to a method getInertia for access in future calculations
    public double getInertia()
    {
        return inertia;
    }
}

class GearRatio
{
    private int teethDriver, teethDriven;
    private String formattedGearRatio;

    //Constructor
    public GearRatio(int teethDriver, int teethDriven)
    {
        this.teethDriver = teethDriver;
        this.teethDriven = teethDriven;
    }

    //Method to calculate the gear ratio of the drivetrain
    public void CalculateGearRatio()
    {
        double gearRatio = (double) teethDriver / teethDriven;
        formattedGearRatio = String.format("%.4f", gearRatio);
    }

    //Display calculations
    public void displayInfo()
    {
        System.out.println("The gear ratio between the two given gears is: " +
formattedGearRatio + " : 1");
    }
}
```

```
public class Main
{
    public static void main(String[] args)
    {
        Scanner doubleScanner = new Scanner(System.in);
        Scanner intScanner = new Scanner(System.in);

        //Gathering of necessary variables for calculation
        System.out.println("Enter in the rpm of the weapon for the calculation of
the weapon speed");
        double rpm = doubleScanner.nextDouble();
        System.out.println("Enter in the rotational radius for the weapon for
calculation of the weapon speed");
        double weaponRadius = doubleScanner.nextDouble();
        System.out.println("Enter the number of teeth on the first gear (the driver
gear):");
        int teethDriver = intScanner.nextInt();
        System.out.println("Enter the number of teeth on the second gear (the
driven gear):");
        int teethDriven = intScanner.nextInt();
        System.out.println("Enter the shape (1 for disk, 2 for tube):");
        int type = intScanner.nextInt();

        //Object instantation for an example weapon to calculate weapon speed
        WeaponSpeed exampleWeapon = new WeaponSpeed(rpm, weaponRadius);

        //Object instantation for an example weapon to calculate momentary
rotational inertia
        MomentInertia exampleWeapon2 = new MomentInertia(type);
        double foundInertia = exampleWeapon2.getInertia();

        //Object instantation for an example weapon to calculate kinetic energy
        KineticEnergy exampleWeapon3 = new KineticEnergy(rpm, foundInertia);

        //Object instantation for an example weapon to calculate gear ratio
        GearRatio exampleWeapon4 = new GearRatio(teethDriver, teethDriven);
    }
}
```

Explanation of Core Code

Our code is divided into one main class and four subclasses designated with each mathematical function of our project. Each subclass has a constructor method to instantiate the necessary variables, as well as a calculation method in which the functions are computed. In the Moment Inertia class, there are also two different methods for calculations based on the shape of

the object, which is user determined. That class also has a getInertia method so that the value can be accessed in the main class for use in other calculations. The other three classes have a display info method that prints the results as a truncated and formatted string. The main class has a block of inputs for necessary variables entered by the user and ends with the objects being created so that the calculations can run.

A brief explanation of the math is as follows: the gear ratio is found by dividing the teeth of the driving gear by the teeth of the driven gear. The mass moment of inertia is found by first determining the volume of the weapon using the dimensions provided by the user, then finding the mass by multiplying the density by this volume. Finally, the mass moment of inertia is calculated by incorporating the dimensions given by the user and the mass. The speed of the weapon is found by converting rpm to rad/s, then rad/s to m/s, followed by m/s to km/h, and finally km/h to mph. The rotational kinetic energy is found by squaring the rad/s and then multiplying it by one half and the mass moment of inertia.

Challenges Faced

Throughout this project we faced many challenges and hurdles, which needed to be overcome. Some of those challenges were typical group project issues like: sharing ideas with others, time management, or meshing of our respective programming styles. However some of the challenges came as a result of the project itself. For example, things like: the proper implementation of the physics to both write the program and check its accuracy or learning new classes (Scanner, Math, etc.) were problems that we faced. Overall, this project came with plenty of challenges to keep us "on our toes" but we overcame them all.

Conclusion

Over the course of this project, we delved into the interconnected disciplines of mathematics, physics, robotics, and data analysis with the goal of creating a program that would help in the construction of a Battlebot. Beyond even these technical topics we explored how to collaboratively code as this was a first time experience coding with others for all of us. Additionally, we acquired proficiency in creating comprehensive reports and presentations, refining our articulation and presentation abilities. These skills learned throughout this project's course, hold significant value as we continue our future academic and professional pursuits, equipping us to meet challenges with both confidence and adeptness.

Sources

- http://runamok.tech/AskAaron/tools.html
- https://lucidar.me/en/unit-converter/revolution-per-minute-to-miles-per-hour/
- https://www.omnicalculator.com/physics/gear-ratio
- http://runamok.tech/RunAmok/spincalc.html
- http://runamok.tech/squid/newtorquecalc.htm
- https://openstax.org/books/university-physics-volume-1/pages/10-4-moment-of-inertia-and-rotational-kinetic-energy