# Honors Program and Design Project: Battle Bot Calculations


By: Alexander Miller, Daniel Blitshtein, Nick Johns

# Goals of The Project

- Gear Ratios
- Weapon Speed
- Mass Moment of Inertia
- Rotational Kinetic Energy

# Why Is This Information Important

- Rotational **Kinetic Energy -** The Energy the weapon delivers
- **Weapon Speed -** How fast the weapon is moving
- **Gear Ratio -** The reduction rate between the motor and pulley of the weapon
- **Mass Moment of Inertia -** Necessary to find Rotational Kinetic Energy

# Flexible Distribution of Labor

Alex: Mass Moment of Inertia and Kinetic Energy Codes, Research, Written Report, debugging

Daniel: Slides Design, Written Report, Research, Gear Ratio Calculator Code, debugging

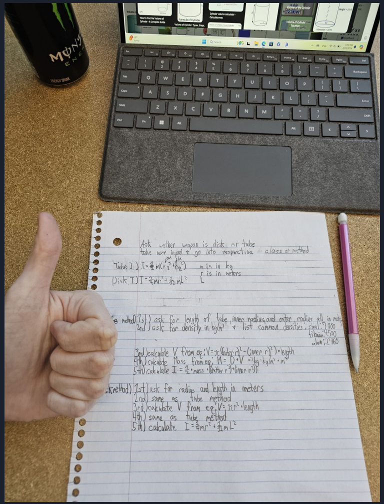
Nick: Research, Weapon Speed Code, Written Report, Slides, editing

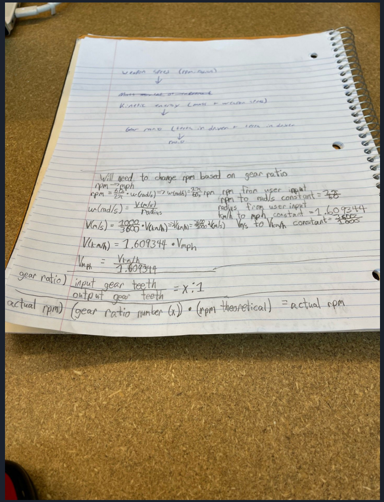
# Our Motivations


- Wanted to do a challenging project that also taught new skills to us beyond class, for robotics and physics
- Our group also had an interest in robotic design in general considering how important robotics is becoming in the research, practical, and entertainment fields
- We also wanted to work on something that we knew was largely achievable in the month we were given

# Methodology

- Began with rough outlining of potential classes and parameters
- Physics equations converted and expanded for easier programming
- Four rudimentary programs completed individually
- Combined to one program utilizing methods under one class
- Further advanced structure, style and cohesion in final program, utilizing classes and subclasses





# Weapon Speed Code

- Constants for kmh to mph and rpm to rads/s
- Program completes calculations under CalculateWeaponSpeed() method
- DisplayInfo() method prints the results to the console

```
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);
    }
```

# Gear Ratio Calculator

- Computed gear ratio by simple division and formatting
- Program completes calculations under CalculateGearRatio() method
- DisplayInfo() method prints the results to the console

```java
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");
    }
}
```

# Kinetic Energy Code

- Uses same constant in speed class for rpm to rad/s conversion
- Program completes calculations under CalculateKineticEnergy() method
- DisplayInfo() method prints the results to the console
- Utilizes result from Moment Inertia class for calculations

```
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;
        double rpmToRadS = (Math.PI * 2) / 60;

        //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);
    }
}
```

# Mass Moment of Inertia Code

- Sorts shape and methods called by user input
- Program completes calculations under three different methods
- No DisplayInfo, instead a getInertia since value only needs to be referenced by other classes

```java
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;
    }
```

# Challenges faced:

- When we attempted to implement the Mass Moment of Inertia program we ran into some challenges with its implementation, which lead to a cost in time
- Time Constraints
- Comprehending and understanding the physics rules and concepts necessary to write the program and check its accuracy
- Learning new programming concepts in relation to the specific classes and libraries learned in class
- Combining our  programming styles

# What We Learned

- Mathematics, physics, robotics, and data analysis concepts
- How to write code with others
- Increased understanding of coding concepts
- How to create a written report

# 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