Odometry Basics

Explanation:

Odometry is the tracking of the robot across the playing field using passive wheels. Passive omni wheels are attached to encoders which measure in ticks (similar to degrees/radians). Using the circumference of the omni wheels, this can be converted into linear distance. Using at least 2 perpendicular wheels and trigonometry, an X and Y distance can be created. Relating these distances to the robot's starting location allows for tracking where the robot moves to at all times.

Purpose:

Odometry is important for the autonomous period. It allows the robot capabilities it would not otherwise have. With odometry, the robot can travel to any specific point on the field with accuracy within an inch. Timing robot movements creates inaccuracy as the robot travels different distances every attempt and often doesn't complete a maneuver due to battery voltage changes. A robot will only perform well in autonomous if it has sensors to monitor all of the movements it can do. Driving is the most important function of a robot in autonomous and odometry is the drivetrain's sensor.

Programming

- Constructor
- Variables
- Trigonometry
- Communication

Mechanical

- Mechanisms

Constructor

>>The absolute position of the robot on the field is stored in tick values in X and Y directions. >>An offset for the starting angle of the robot allows for different starting orientations. This causes errors that are accounted for later. >>The Odometry constructor simply sets the starting orientation of the robot: including the X and Y position in tick values, and the angle it starts at from-180° to 180°.

```java
public class Odometry
{
    // Inches / Ticks    ( Circumference of wheel / Ticks per revolution ( 2048 ) )
    private static final double TICK_CONVERT = 0.00185;

    private double angleOffset;
    private double trueAngle;

    // X and Y position stored in Ticks
    private double x;
    private double y;

    // Previous values of encoders
    private double lastFR;
    private double lastBR;
    private double lastFL;
    private double lastBL;

    // Change in value of encoders
    private double FR; // Front Right
    private double BR; // Back Right
    private double FL; // Front Left
    private double BL; // Back Left

    /**
     * Initializes the odometry data and positioning
     *
     * @param startX X location on field robot initializes at
     * @param startY Y location on field robot intializes at
     * @param startAngleOffset Angle relative to driver robot initializes at
     */
    public Odometry( int startX, int startY, int startAngleOffset )
    {
        x = startX;
        y = startY;
        angleOffset = startAngleOffset;
    }
```

Variables

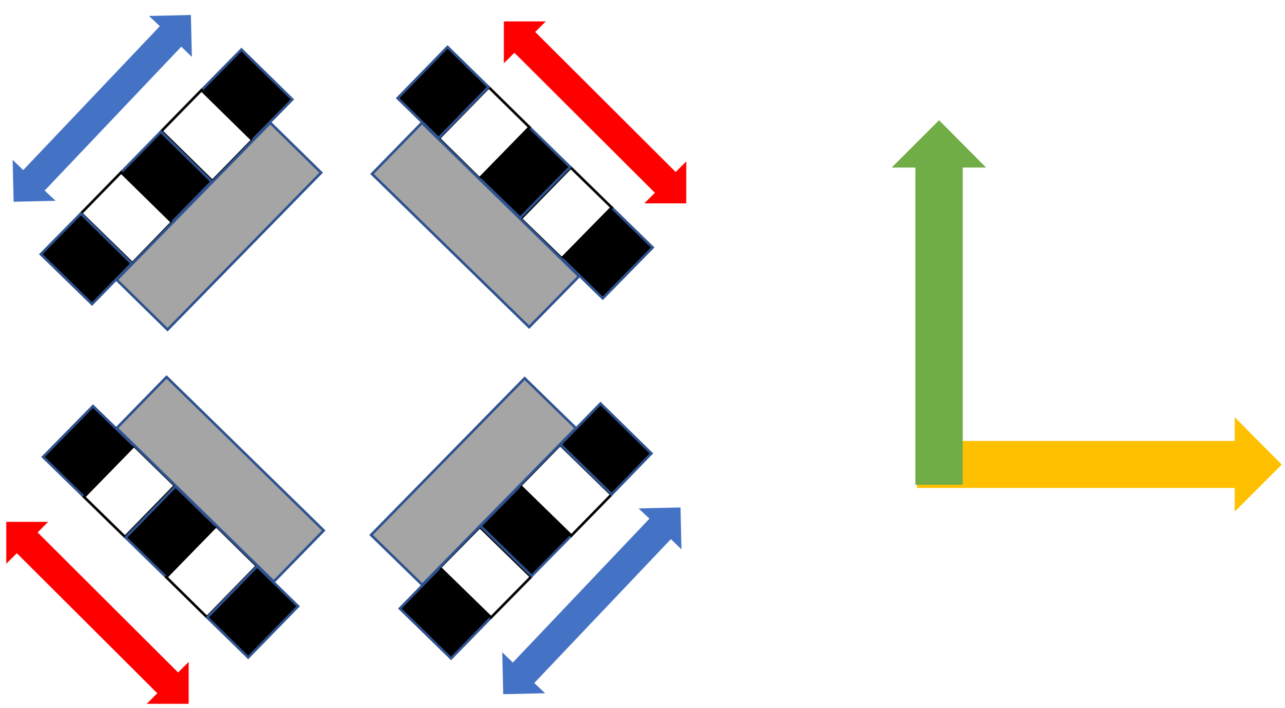
>>The update method is called repeatedly during auto. The encoder readings increase as the robot moves and their values are never reset. After reversed encoders are accounted for, the difference between the previous values and current values are taken to find the distance each encoder moved. >>In order to do this, variables are needed to store the previous values.

```java
// Previous values of encoders
private double lastFR;
private double lastBR;
private double lastFL;
private double lastBL;

// Change in value of encoders
private double FR; // Front Right
private double BR; // Back Right
private double FL; // Front Left
private double BL; // Back Left

/**
 * Initializes the odometry data and positioning
 *
 * @param startX X location on field robot initializes at
 * @param startY Y location on field robot intializes at
 * @param startAngleOffset Angle relative to driver robot initializes at
 */
public Odometry( int startX, int startY, int startAngleOffset )
{
    x = startX;
    y = startY;
    angleOffset = startAngleOffset;
}

/**
 * Updates the odometry data and positioning using current and previous values of the encoders
 *
 * @param newFR Current value of the Front Right encoder
 * @param newBR Current value of the Back Right encoder
 * @param newFL Current value of the Front Left encoder
 * @param newBL Current value of the Back Left encoder
 * @param angle Current value of the IMU angle
 */
public void update( double newFR, double newBR, double newFL, double newBL, double angle )
{
    newFR = -newFR;
    newBL = -newBL;

    // Finds change in value
    FR = newFR - lastFR;
    BR = newBR - lastBR;
    FL = newFL - lastFL;
    BL = newBL - lastBL;
```

Trigonometry

This year's robot uses 4 odometry modules in a square pattern. >>The parallel modules' distances are averaged together. >>These values are then added to the X and Y location through the sine and cosine functions of the robot's current angle. >>The angle is also checked to ensure the value lies between-180° and 180° as the angle offset can push the value outside. This will cause errors with the odometry if not accounted for.

```java
// Averages parallel encoders
double leftDiag = ( FR + BL ) / 2;
double rightDiag = ( BR + FL ) / 2;
angle += angleOffset;

// Angle offset combined with -180 -> 180 range creates blindspots
// This checks for and removes the blindspots
if( angle < 181 && angle > -181 )
    trueAngle = angle;
if( angle > 180 )
    trueAngle = -180 + angle % 180;
else if( angle < -180 )
    trueAngle = 180 + angle % 180;

// Adds the sine and cosine values to the current position
x += ( Math.cos( Math.toRadians( angle + 45 ) ) * leftDiag +
       Math.cos( Math.toRadians( angle - 45 ) ) * rightDiag ) / 2;
y += ( Math.sin( Math.toRadians( angle + 45 ) ) * leftDiag +
       Math.sin( Math.toRadians( angle - 45 ) ) * rightDiag ) / 2;
```



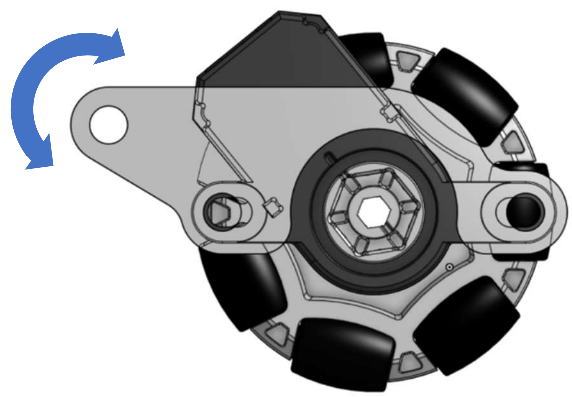
Communication

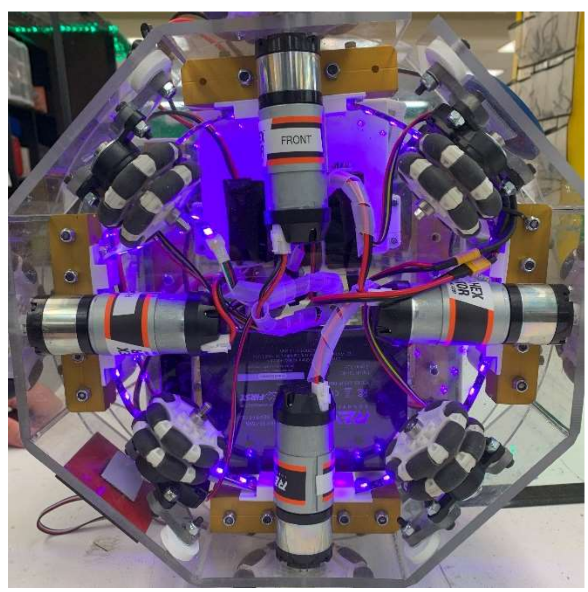
>>Lastly, the variables storing previous values are updated to the current values before the method is called again. >>There are also methods to return the X and Y location of the robot in inches as well as the correct angle of the robot. These enable the odometry data to be used during auto. They should be called in the drive methods of the robot.

```
    lastFR = newFR;
    lastBR = newBR;
    lastFL = newFL;
    lastBL = newBL;
}

/**
 * Returns the X location of the robot in Inches
 * @return X Coord of robot in Inches
 */
public double getX()
{
    return x * TICK_CONVERT;
}

/**
 * Returns the Y location of the robot in Inches
 * @return Y Coord of robot in Inches
 */
public double getY()
{
    return y * TICK_CONVERT;
}

/**
 * Returns the oriented angle of the robot ( with starting offset included )
 * @return Field oriented angle of the robot
 */
public double getAngle()
{
    return trueAngle;
}
}
```

Mechanisms

The odometry modules consist of an encoder and an omni wheel, along with a polycarbonate piece to attach them to the robot. The current design allows for the module to rotate so that gravity always pulls it onto the ground. Many other teams use gravity to improve accuracy, but some also use springs. Using springs would create better contact but likely not improve accuracy much compared to the increase in complexion and parts that could break. A potential future improvement is to add a bearing to the mounting bolt so the module can be more sturdy and move more freely.





Complete Odometry Class

Used on Noodlenose 2022-2023

To see implementation of the data:

Complete robot code can be found on the schools GitHub page

https://github.com/NicholsSchool/2023-FTC-Noodlenose

```java
/**
 * This class controls the trigonometry and data of the odometry system
 * - This keeps track of position by subtracting the previous encoder values from the current encoder values
 * - This finds the change in values without requiring the STOP_AND_RESET_ENCODERS method call
 * - That call causes motors to constantly stop, creating slow, noisy, and shaky driving
 *
 * @author Zach Boeck
 * @date April 11th, 2023
 */
public class Odometry
{
    // Inches / Ticks    ( Circumference of wheel / Ticks per revolution ( 2048 ) )
    private static final double TICK_CONVERT = 0.00185;

    private double angleOffset;
    private double trueAngle;

    // X and Y position stored in Ticks
    private double x;
    private double y;

    // Previous values of encoders
    private double lastFR;
    private double lastBR;
    private double lastFL;
    private double lastBL;

    // Change in value of encoders
    private double FR; // Front Right
    private double BR; // Back Right
    private double FL; // Front Left
    private double BL; // Back Left

    /**
     * Initializes the odometry data and positioning
     *
     * @param startX X location on field robot initializes at
     * @param startY Y location on field robot intializes at
     * @param startAngleOffset Angle relative to driver robot initializes at
     */
    public Odometry( int startX, int startY, int startAngleOffset )
    {
        x = startX;
        y = startY;
        angleOffset = startAngleOffset;
    }

    /**
     * Updates the odometry data and positioning using current and previous values of the encoders
     *
     * @param newFR Current value of the Front Right encoder
     * @param newBR Current value of the Back Right encoder
     * @param newFL Current value of the Front Left encoder
     * @param newBL Current value of the Back Left encoder
     * @param angle Current value of the IMU angle
     */
    public void update( double newFR, double newBR, double newFL, double newBL, double angle )
    {
        newFR = -newFR;
        newBL = -newBL;

        // Finds change in value
        FR = newFR - lastFR;
        BR = newBR - lastBR;
        FL = newFL - lastFL;
        BL = newBL - lastBL;

        // Averages parallel encoders
        double leftDiag = ( FR + BL ) / 2;
        double rightDiag = ( BR + FL ) / 2;
        angle += angleOffset;

        // Angle offset combined with -180 -> 180 range creates blindspots
        // This checks for and removes the blindspots
        if( angle < 181 && angle > -181 )
            trueAngle = angle;
        if( angle > 180 )
            trueAngle = -180 + angle % 180;
        else if( angle < -180 )
            trueAngle = 180 + angle % 180;

        // Adds the sine and cosine values to the current position
        x += ( Math.cos( Math.toRadians( angle + 45 ) ) * leftDiag +
               Math.cos( Math.toRadians( angle - 45 ) ) * rightDiag ) / 2;
        y += ( Math.sin( Math.toRadians( angle + 45 ) ) * leftDiag +
               Math.sin( Math.toRadians( angle - 45 ) ) * rightDiag ) / 2;

        lastFR = newFR;
        lastBR = newBR;
        lastFL = newFL;
        lastBL = newBL;
    }

    /**
     * Returns the X location of the robot in Inches
     * @return X Coord of robot in Inches
     */
    public double getX()
    {
        return x * TICK_CONVERT;
    }

    /**
     * Returns the Y location of the robot in Inches
     * @return Y Coord of robot in Inches
     */
    public double getY()
    {
        return y * TICK_CONVERT;
    }

    /**
     * Returns the oriented angle of the robot ( with starting offset included )
     * @return Field oriented angle of the robot
     */
    public double getAngle()
    {
        return trueAngle;
    }
```