FTC Auton with Pedro Pathing

Kanthan Rajendran
with Arisha and Vid

INFINITY TECH
FTC TEAM 13684

Canton FTC Kickoff

September 12, 2026

FIRST TECH CHALLENGE

Agenda

- Auton Basics
- Pedro Pathing Overview
- Localization & Odometry
- Pedro Pathing Setup
- Example Auton

INFINITY TECH
FTC 13684

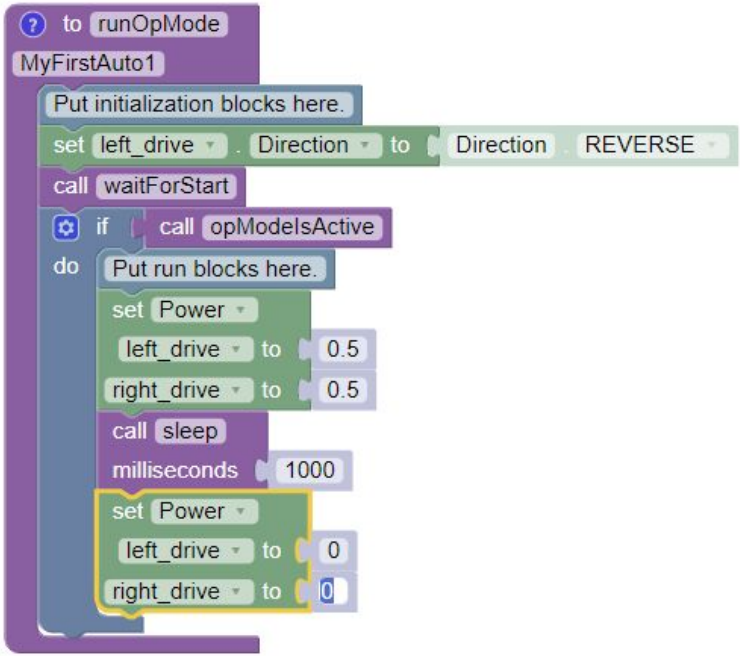
What is Auton?

- The first 30 seconds of an FTC match
- Robots operate with only their pre-programmed instructions
- Drivers may not provide input to their robots
- Importance of Auton
  - More points!
  - Usually there is some bonus for scoring during Auton
  - Makes your team an attractive alliance partner

INFINITY TECH
FTC 13684

How to Auton?

- Pre-programmed code sequence
  - Easier to implement
  - Not accurate
- Path Follower
  - More effort but much more accurate
  - Roadrunner or PedroPathing



INFINITY TECH
FTC 13684

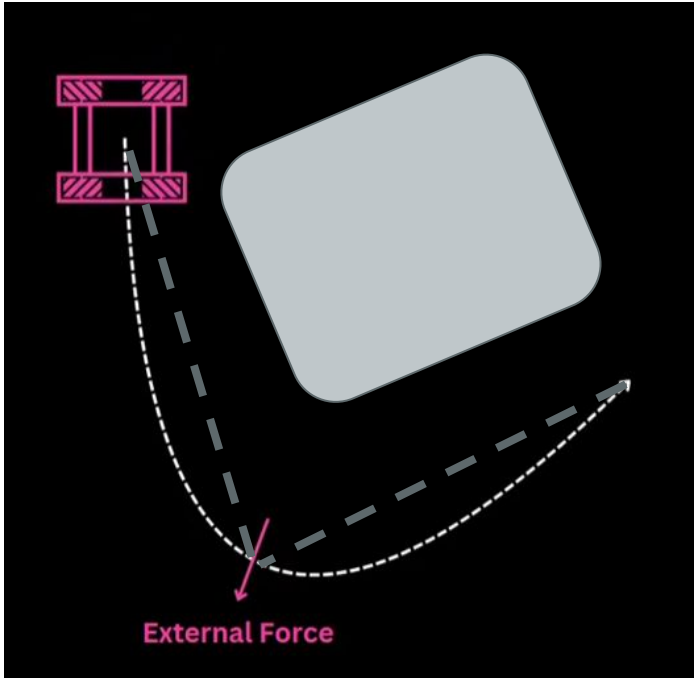
Pedro Pathing Overview

INFINITY TECH
FTC 13684

Pedro Pathing Overview

- Advanced Reactive Vector Follower developed by FTC Team 10158
- Uses Bézier curves to produce smoother, more efficient trajectories
- Dynamic Adjustments: Reacts to obstacles or changes in real-time
- More intuitive to tune and setup than Roadrunner

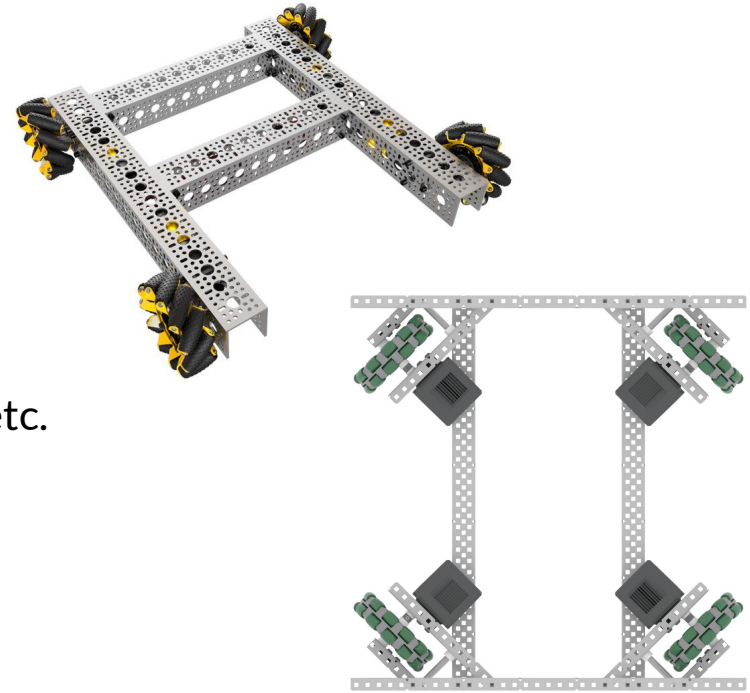
www.pedropathing.com



INFINITY TECH
FTC 13684

Pedro Pathing Prerequisites

- Omni-Directional Drive
  - Mecanum, X-Drive, or Swerve
  - **No** tank drive robots
- Localization
  - Motor encoder, dead-wheel odometry, etc.
- Android Studio
  - **No** Blocks or OnBot Java



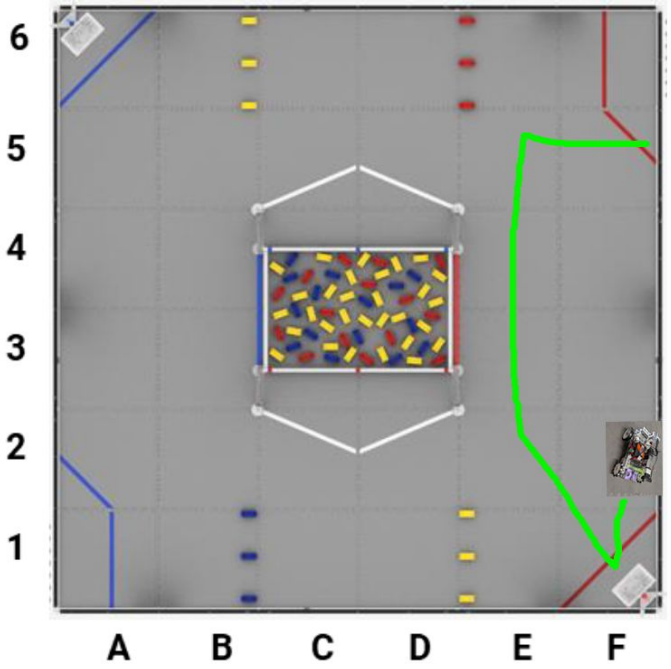
INFINITY TECH
FTC 13684

Localization
&
Odometry

INFINITY TECH
FTC 13684

Localization

- **Localization** is how the robot determines its **position** and **orientation** on the field
- Localization Options
  - **Drive Motor Encoder** ← Not accurate
  - **Dead-Wheel Odometry**
    - Two- or three-wheel pods
    - GoBilda Pinpoint ← PP Default
  - **OTOS**
    - Optical tracking odometry

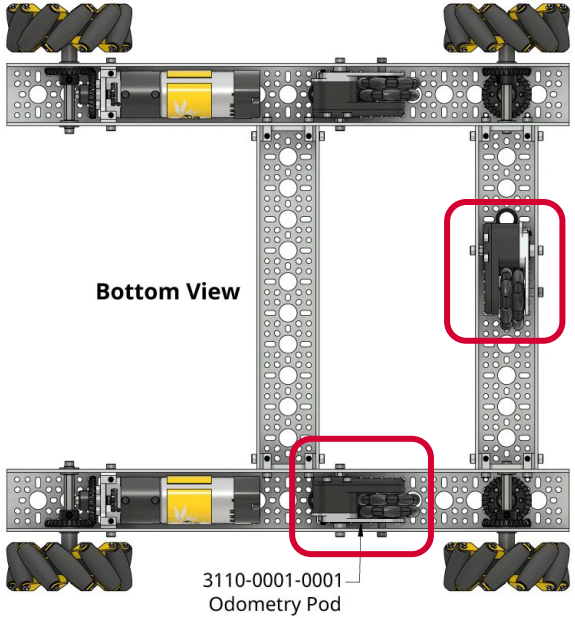


INFINITY TECH
FTC 13684

GoBilda Pinpoint

Pinpoint Odometry Computer

4-Bar Odometry Pod



Pods fit within GoBilda U-channels

Pinpoint set (computer + pods) costs $209.99

INFINITY TECH
FTC 13684

Pedro Pathing
Tuning

INFINITY TECH
FTC 13684

Pedro Pathing Tuning

For Pedro Pathing to work properly with your robot, it must be **tuned** first.

- Constants
- Robot Setup
- Localization Setup
- Auto Tuners
- PID Tuners
- Testing

Step-by-step instructions are on pedropathing.com

INFINITY TECH
FTC 13684

PedroPathing Setup - Constants

Constants file in the pedroPathing package.

1. **Drivetrain constants** - constants specific to your drivetrain type.
2. **Follower constants** - values from the automatic, PID, and centripetal tuners
3. **Localizer constants** - constants specific to your localizer.
4. **Path constraints** - determine under what conditions a path may end.

Follow steps on pedropathing.com and add lines to Constants file for various settings

INFINITY TECH
FTC 13684

PedroPathing Setup - Drivetrain Contants

```
public static MecanumConstants driveConstants = new MecanumConstants()
        .leftFrontMotorName("motorFrontLeft")
        .leftRearMotorName("motorRearLeft")
        .rightFrontMotorName("motorFrontRight")
        .rightRearMotorName("motorRearRight")
        .leftFrontMotorDirection(DcMotorSimple.Direction.REVERSE)
        .leftRearMotorDirection(DcMotorSimple.Direction.REVERSE)
        .rightFrontMotorDirection(DcMotorSimple.Direction.FORWARD)
        .rightRearMotorDirection(DcMotorSimple.Direction.FORWARD)
        .xVelocity(78.2619)
        .yVelocity(61.4945);
```

Specify robot **motors** and **velocity**

INFINITY TECH
FTC 13684

PedroPathing Setup - Follower Constants

```
public static FollowerConstants followerConstants = new
FollowerConstants()
    .mass(16.2)
    .forwardZeroPowerAcceleration(-25.9346)
    .lateralZeroPowerAcceleration(-67.3425)
    .translationalPIDFCoefficients(new PIDFCoefficients(0.1,...))
    .headingPIDFCoefficients(new PIDFCoefficients(0.1,...))
    .drivePIDFCoefficients(new FilteredPIDFCoefficients(0.1,...));
    .centripetalScaling(0.0005);
```

Specify robot **mass**, **deceleration**

and **PIDF constants** for path correction

INFINITY TECH
FTC 13684

PedroPathing Setup - Localizer Constants

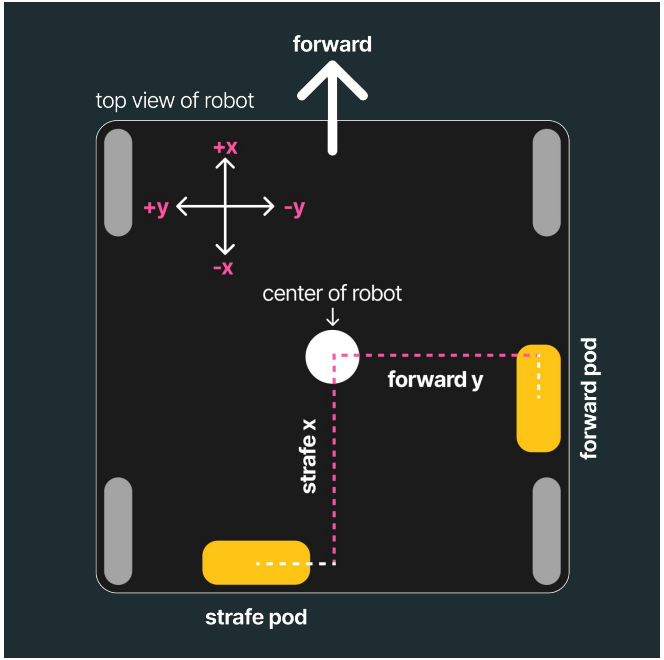
```
public static PinpointConstants localizerConstants = new PinpointConstants()
    .hardwareMapName("pinpoint")
    .distanceUnit(DistanceUnit.INCH)
    .encoderResolution(GoBildaPinpointDriver.GoBildaOdometryPods.
        goBILDA_4_BAR_POD)
    .forwardEncoderDirection(GoBildaPinpointDriver.EncoderDirection.FORWARD)
    .strafeEncoderDirection(GoBildaPinpointDriver.EncoderDirection.REVERSED)
    .forwardPodY(-4.49571)
    .strafePodX(-2.87255);
```

Specify **localizer** settings

INFINITY TECH
FTC 13684

PedroPathing Setup - Localization

- Measure offset of odometry pods from robot's **center of rotation** in inches



Pay attention to
coordinate system

INFINITY TECH
FTC 13684

PedroPathing Setup - Build Follower

```
public static PathConstraints pathConstraints
= new PathConstraints(0.99, 100, 1, 1);
```

Determine when a path is considered complete

```
public static Follower createFollower(HardwareMap hardwareMap) {
   return new FollowerBuilder(followerConstants, hardwareMap)
           .mecanumDrivetrain(driveConstants)
           .pinpointLocalizer(localizerConstants)
           .pathConstraints(pathConstraints)
           .build();
}
```

Build the follower

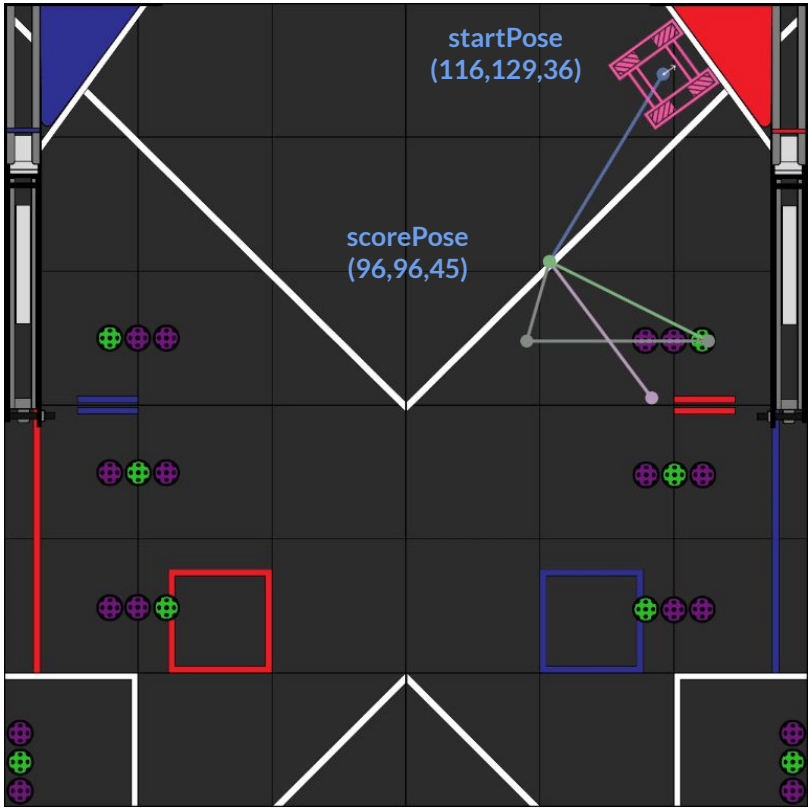
Follower manages robot movement

INFINITY TECH
FTC 13684

Pedro Pathing
Poses and Paths

INFINITY TECH
FTC 13684

PedroPathing Poses



Pose specifies the robot position and heading

Pose(x , y, Θ)

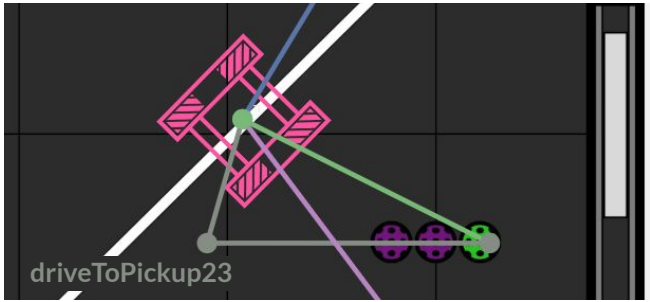
```
startPose = new Pose(116, 129, Math.toRadians(36));
scorePose = new Pose(96, 96, Math.toRadians(45));
```

Note: All headings are in **radians**

Use **visualizer.pedropathing.com**

INFINITY TECH
FTC 13684

PedroPathing Paths



Pathchains are defined routes for the robot to follow

```
driveToPickup23 = follower.pathBuilder()
        .addPath(new BezierLine(scorePose, prePickupPose23))
        .setLinearHeadingInterpolation(scorePose.getHeading(), prePickupPose23.getHeading())
        .addPath(new BezierLine(prePickupPose23, pickupPose23))
        .setLinearHeadingInterpolation(prePickupPose23.getHeading(), pickupPose23.getHeading())
        .build();
```

Note: You can chain as many `addPath` together as you want.

Switch Case Construct

```
public void autonomousPathUpdate() {
    switch (pathState) {
        case 10:
            follower.followPath(driveToGoal);
            setPathState(11);
            break;
        case 11:
            if (!follower.isBusy()) {
                shooter.openServoStop();
                setPathState(12);
            }
            break;
        case 12:
            if (pathTimer.getElapsedTimeSeconds() > 0.6) {
                shooter.closeServoStop();
                setPathState(13);
            }
            break;
```

The pathState variable tracks where the robot is in the routine, and the switch statement transitions between states based on conditions.

Tip: Add one or two switch cases at a time; test before adding more

INFINITY TECH
FTC 13684

Pedro Pathing
Example Auton

INFINITY TECH
FTC 13684

```java
@Autonomous
public class REDNearKickoffExample extends OpMode {

    ShooterSpinfinityDuo shooter = new ShooterSpinfinityDuo();
    FlywheelSpinfinityDuo flywheel = new FlywheelSpinfinityDuo();
    SpintakeSpinfinity spintake = new SpintakeSpinfinity();

    private Follower follower;
    private Timer pathTimer;
    private int pathState;

    double targetRPM;
    double TARGET_AUTON_RPM = 2400.0;

    // Poses
    private final Pose startPose = new Pose(116, 129, Math.toRadians(36));
    private final Pose scorePose = new Pose(96, 96, Math.toRadians(45));
    private final Pose prePickupPose23 = new Pose(92, 82, Math.toRadians(0));
    private final Pose pickupPose23 = new Pose(124, 82, Math.toRadians(0));
    private final Pose endPose = new Pose(114, 72, Math.toRadians(90));

    private PathChain driveToGoal, driveToPickup23, driveToGoal23, driveToEnd;
```

```
private void buildPaths() {

    driveToGoal = follower.pathBuilder()
            .addPath(new BezierLine(startPose, scorePose))
            .setLinearHeadingInterpolation(startPose.getHeading(), scorePose.getHeading())
            .build();

    driveToPickup23 = follower.pathBuilder()
            .addPath(new BezierLine(scorePose, prePickupPose23))
            .setLinearHeadingInterpolation(scorePose.getHeading(), prePickupPose23.getHeading())
            .addPath(new BezierLine(prePickupPose23, pickupPose23))
            .setLinearHeadingInterpolation(prePickupPose23.getHeading(), pickupPose23.getHeading())
            .build();

    driveToGoal23 = follower.pathBuilder()
            .addPath(new BezierLine(pickupPose23, scorePose))
            .setLinearHeadingInterpolation(pickupPose23.getHeading(), scorePose.getHeading())
            .build();

    driveToEnd = follower.pathBuilder()
            .addPath(new BezierLine(scorePose, endPose))
            .setLinearHeadingInterpolation(scorePose.getHeading(), endPose.getHeading())
            .build();
}
```

```java
@Override
public void init() {
    shooter.init(hardwareMap);
    flywheel.init(hardwareMap);
    spintake.init(hardwareMap);

    follower = Constants.createFollower(hardwareMap);
    buildPaths();
    follower.setStartingPose(startPose);
    pathTimer = new Timer();

    setPathState(10);
}

@Override
public void loop() {
    follower.update();
    autonomousPathUpdate();

    targetRPM = TARGET_AUTON_RPM;
    flywheel.setFlywheelVel(targetRPM);
    shooter.closeServoStop();
    spintake.turnIntakeOn();
}
```

```
private void setPathState(int pState) {
    pathState = pState;
    pathTimer.resetTimer();
}

public void autonomousPathUpdate() {
    switch (pathState) {
        case 10:
            follower.followPath(driveToGoal);
            setPathState(11);
            break;
        case 11:
            if (!follower.isBusy()) {
                shooter.openServoStop();
                setPathState(12);
            }
            break;
        case 12:
            if (pathTimer.getElapsedTimeSeconds() > 0.6) {
                shooter.closeServoStop();
                setPathState(13);
            }
            break;
```

```
case 13:
    follower.followPath(driveToPickup23);
    setPathState(14);
    break;
case 14:
    if (!follower.isBusy()) {
        follower.followPath(driveToGoal23);
        setPathState(15);
    }
    break;
case 15:
    if (!follower.isBusy()) {
        shooter.openServoStop();
        setPathState(16);
    }
    break;
case 16:
    if (pathTimer.getElapsedTimeSeconds() > 0.6) {
        shooter.closeServoStop();
        setPathState(17);
    }
    break;
```

```
            case 17:
                follower.followPath(driveToEnd);
                setPathState(18);
                break;
            case 18: // last state, just stops and waits
                if (!follower.isBusy()) {
                    setPathState(99);
                }
                break;
        }
    }
}
```

Questions?

INFINITY TECH
FTC 13684