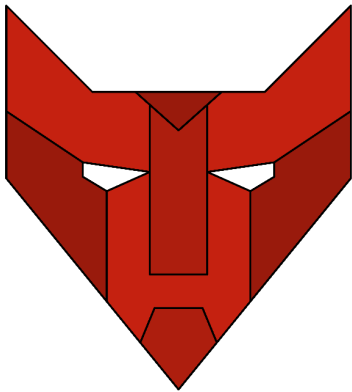

Hyperdrive Manual

By Brach Knutson

V. 0.3



5963 FOXIMUS
PRIME

Table of Contents

Overview 2

How to use Hyperdrive 3

Tips, Tuning, and Troubleshooting 14

PathVisualizer 19

Overview

Hyperdrive is An easy-to-implement autonomous driving library for FRC robots, developed by FRC team 3695, Foximus Prime. It was able to produce a cumulative time of 28.3 seconds on the Auto-Nav challenge and a 9.9 second cumulative time on the Galactic Search challenge in the 2021 FRC season. In its current state, this library only provides support for tank-style robots. Hyperdrive works off of a “record” - “emulate” paradigm; a human driver manually drives the robot through a path, “recording” it, and Hyperdrive can make the robot drive the same path autonomously, “emulating” it. Hyperdrive tracks the robot’s position on the field, and makes its calculations on the spot rather than pre-compiling them.

This manual will provide documentation for the different classes and methods of this library, as well as a guide to getting it to work with your robot. As you read this guide, keep in mind that the project’s test environment (which can be found at github.com/BTK203/Hyperdrive) can be used as a working example of the library in action.

This manual no longer provides the class and method reference for this library. To see the class and method reference, go to <https://btk203.github.io/Hyperdrive/>

How to use Hyperdrive








Putting Hyperdrive into your robot code is easy. To use Hyperdrive on your robot, you will instantiate a `Hyperdrive` object into your drivetrain class, create a “record” and “emulate” command, and tune some PID constants. Before putting Hyperdrive onto your robot however, make sure the robot meets the following requirements:

- The robot must have a gyro. The `TankGyro` class can be used in place of a physical gyro in prototyping or simulation cases, but is **not recommended on competition robots**.
- The robot must have a tank-style drivetrain. Unfortunately, Hyperdrive does not support mecanum or swerve robots.



FRC team 3695's 2019 robot, which will be used to illustrate how to add Hyperdrive to your project.

First, add the Hyperdrive library to your robot project. If you have not done so already, create a new folder called “util” in your project.

	commands	5/29/2021 1:27 PM	File folder	
	subsystems	5/29/2021 1:30 PM	File folder	
<input checked="" type="checkbox"/> 	util	5/29/2021 2:04 PM	File folder	
	Constants.java	5/29/2021 1:37 PM	JAVA File	2 KB
	Main.java	5/29/2021 1:26 PM	JAVA File	1 KB
	Robot.java	5/29/2021 1:26 PM	JAVA File	4 KB
	RobotContainer.java	5/29/2021 1:28 PM	JAVA File	2 KB

Download the Hyperdrive library folder from github.com/BTK203/Hyperdrive/releases/latest (named hyperdrive-vX.zip) and paste that folder into the new util folder.

<input type="checkbox"/>	Name	Date modified	Type	Size
<input checked="" type="checkbox"/>	hyperdrive	5/29/2021 1:43 PM	File folder	

- robot
 - commands
 - subsystems
 - util
 - hyperdrive
 - emulation
 - pathvisualizer
 - recording
 - util

Next, we need to measure two values that will be vital to Hyperdrive’s performance. The first value will convert motor positions to linear positions. In this example, it will be called the *motor units scalar*. To measure it, lay a tape measure on the ground. Position the robot at the very beginning of the tape measure. Then, drive the robot forward a known distance (like 10 feet) and record the change in motor positions.



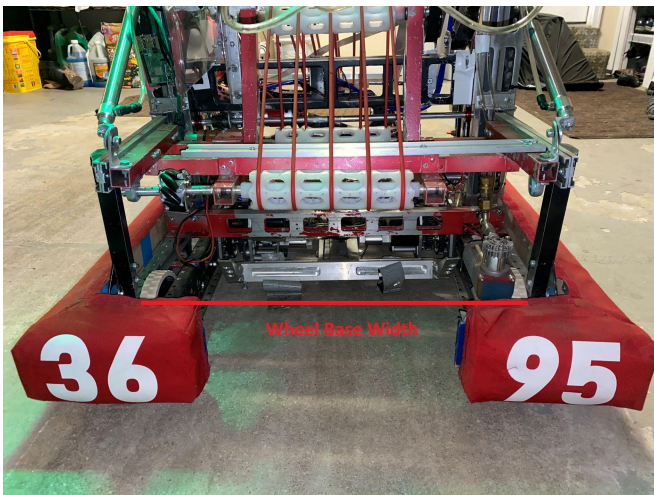
Now, our motor units scalar can be calculated as such:

$$\text{motor units scalar} = \frac{\text{final motor position} - \text{initial motor position}}{\text{distance driven}}$$

For this example, the robot's drive motors were zeroed and the robot was driven forward 10 feet. After driving 10 feet, the motor position was 49.5 rotations. So, our value was calculated to be

$$\frac{\text{final motor position} - \text{initial motor position}}{\text{distance driven}} = \frac{49.44 \text{ rotations} - 0 \text{ rotations}}{120 \text{ inches}} = 0.412 \text{ rotations per inch}$$

The other value is the wheelbase width of the robot. To measure this, simply measure the length between the left and right wheels of the robot. It is important that this value is measured in whatever "distance unit" the previous value was measured in. Because this example uses inches, the wheelbase width will be measured in inches.



After measuring these two values, they should be put into your `Constants.java` file.

```
public static final double
    MOTOR_ROTATIONS_PER_METER = 16.22, //revolutions per meter
    ROBOT_WHEELBASE_WIDTH = 0.6; //meters
```

Now, add your gyro and a `Hyperdrive` object to your drivetrain class.

```
private AHRS gyro; //navX gyro
private Hyperdrive hyperdrive; // Hyperdrive object

/** Creates a new SubsystemDrive. */
public SubsystemDrive() {
    leftMain = new CANSparkMax(Constants.LEFT_MAIN_ID, MotorType.kBrushless);
    leftFollower = new CANSparkMax(Constants.LEFT_FOLLOWER_ID, MotorType.kBrushless);
    rightMain = new CANSparkMax(Constants.RIGHT_MAIN_ID, MotorType.kBrushless);
    rightFollower = new CANSparkMax(Constants.RIGHT_FOLLOWER_ID, MotorType.kBrushless);

    gyro = new AHRS(Port.kUSB);

    hyperdrive = new Hyperdrive(
        DriveStyle.TANK,           // Drive Style
        Units.LENGTH.METERS,      // Length Unit
        Constants.MOTOR_ROTATIONS_PER_METER, // Motor Units per Unit scalar
        Units.FORCE.POUND,        // Weight Unit
        Constants.ROBOT_WEIGHT     // Robot Weight
    );

    configureMotors();
}
```

In the drivetrain's `periodic()` method, call the `update()` method on your newly-created `Hyperdrive`.

```
@Override
public void periodic() {
    // This method will be called once per scheduler run
    double
        leftPosition = leftMain.getEncoder().getPosition(), //get left position
        rightPosition = rightMain.getEncoder().getPosition(), //get right position
        heading = gyro.getAngle(); //get yaw angle

    hyperdrive.update(leftPosition, rightPosition, heading);
}
```

Then, in the drivetrain, make sure you have a method to set the percent outputs of the drivetrain motors.

```

/**
 * Sets the percent outputs of the motors.
 * @param left New percent output of the left motors
 * @param right New percent output of the right motors
 */
public void setPercentOutputs(double left, double right) {
    leftMain.set(left);
    rightMain.set(right);
}

```

SubsystemDrive.java

You should also have methods that return the left and right motor velocities:

```

/**
 * Returns the velocity of the main left motor.
 * @return Left wheel velocity in meters per second
 */
public double getLeftVelocity() {
    return leftMain.getEncoder().getVelocity();
}

/**
 * Returns the velocity of the main right motor.
 * @return Right wheel velocity in meters per second
 */
public double getRightVelocity() {
    return rightMain.getEncoder().getVelocity();
}

```

Finally, add an accessor to the `SubsystemDrive` that returns the `Hyperdrive` object. You could also make the `Hyperdrive` a `public final` object. It just needs to be accessible by other classes. The `Hyperdrive` will be used later in the example.

```

/**
 * Returns the drivetrain's Hyperdrive.
 * @return Hyperdrive.
 */
public Hyperdrive getHyperdrive() {
    return hyperdrive;
}

```

SubsystemDrive.java

At this point, Hyperdrive is set up. Your robot will now connect to PathVisualizer and you can view your robot's position on the field. However, some commands are needed to make Hyperdrive functional.

First, you might need to have a command on your dashboard that you can use to zero the robot's drivetrain encoders, gyro, and the robot's position on the field. This command might be called at the start of autonomous or just used to zero the robot's position before testing a path. To start,

define a method in your drivetrain class. In this example, we will call it `zeroDrivetrain()`. This method should zero the drivetrain encoders, the gyro, and Hyperdrive. Note that `Hyperdrive`'s `zeroPositionAndHeading()` method should *only have an argument of `true` if the drivetrain encoder positions are being zeroed as well.*

```
/**
 * Zeros the motors, the gyro, and Hyperdrive.
 */
public void zero() {
    leftMain.getEncoder().setPosition(0);
    rightMain.getEncoder().setPosition(0);
    gyro.zeroYaw();
    hyperdrive.zeroPositionAndHeading(true);
}
```

SubsystemDrive.java

At this point, it might be useful to create an `InstantCommand` that calls this method. That way, you can send it to the dashboard and zero the robot by clicking a button. That is completely up to you.

Now, create a new `Command` that will record paths to be driven later:

- In the command's `initialize()` method, call `Hyperdrive`'s `initializeRecorder()` method to start recording.
- In the command's `end()` method, call `Hyperdrive`'s `stopRecorder()` method to stop recording.

That's right, nothing needs to happen in the `execute()` method. A sample recorder command is provided below.

```
public class CyborgCommandRecordPath extends CommandBase {
    private Hyperdrive hyperdrive;

    /** Creates a new CyborgCommandRecordPath. */
    public CyborgCommandRecordPath(SubsystemDrive drivetrain) {
        hyperdrive = drivetrain.getHyperdrive();
    }

    // Called when the command is initially scheduled.
    @Override
    public void initialize() {
        hyperdrive.initializeRecorder();
    }

    // Called every time the scheduler runs while the command is scheduled.
    @Override
    public void execute() {}

    // Called once the command ends or is interrupted.
    @Override
    public void end(boolean interrupted) {
        hyperdrive.stopRecorder();
    }

    // Returns true when the command should end.
    @Override
    public boolean isFinished() {
        return false;
    }
}
```

Finally, add a `Command` to actually emulate recorded paths. The command will need to have an instance of your drivetrain class, and a `Path` to drive.

```
public class EmulatePath extends CommandBase {
    private SubsystemDrive drivetrain;
    private Path pathToDrive;
```

The rest of the command should be laid out as such:

- In the command's constructor, the member variables must be defined, and your drivetrain needs to be added as a requirement.

```
/** Creates a new EmulatePath. */
public EmulatePath(SubsystemDrive drivetrain, Path pathToDrive) {
    this.drivetrain = drivetrain;
    this.pathToDrive = pathToDrive;

    addRequirements(drivetrain);
}
```

- In the command's `initialize()` method, load the path to drive and perform initial calculations for path driving. In this example, `PreferenceEmulationParams` is used to define drive parameters. This will use the Preferences table on the dashboard to retrieve the parameters, meaning that the robot's path driving can be tuned without even disabling the robot. If you wish to use constants as parameters, use the `ConstantEmulationParams` class.

```
// Called when the command is initially scheduled.
@Override
public void initialize() {
    //create parameters and load path
    IEmulateParams emulateParameters = new PreferenceEmulationParams(drivetrain.getHyperdrive().getLengthUnits());
    drivetrain.getHyperdrive().loadPath(pathToDrive, emulateParameters);

    //perform calculations vital to Hyperdrive's performance
    drivetrain.getHyperdrive().performInitialCalculations();
}
```

- In the command's `execute()` method, calculate the left and right percent outputs needed to keep the robot on its path.

```
// Called every time the scheduler runs while the command is scheduled.
@Override
public void execute() {
    TankTrajectory newTrajectory = drivetrain.getHyperdrive()
        .calculateNextMovements()
        .getTankTrajectory(Constants.ROBOT_WHEELBASE_WIDTH)
        .convertTime(Units.TIME.MINUTES);

    double
        newLeftOutput = newTrajectory.getLeftPercentOutput(drivetrain.getLeftVelocity()),
        newRightOutput = newTrajectory.getRightPercentOutput(drivetrain.getRightVelocity());

    drivetrain.setPercentOutputs(newLeftOutput, newRightOutput);
}
```

It is vital that when providing the robot's current velocities, you provide the angular velocity of the motor as given to you by the motor controller. Do NOT convert your motor velocities to linear velocities or Hyperdrive will give you incorrect outputs.

Now, note the `convertTime()` method. Depending on your motor controller, your argument for that method will change. Use `Units.TIME.DECASECONDS` for CTRE motors (like the Falcon 500) and `Units.TIME.MINUTES` for REV controllers (like the NEO). Incorrect time units will cause Hyperdrive to provide incorrect outputs.

Finally, note the chain of methods used to create `newTrajectory`. This chain may change based on your robot's setup. For example, if your robot drives backwards or turns the wrong way, you may need to chain on the `invertDirection()` or `invertTurn()` methods. Also note that after `getTankTrajectory()`, there is no set order in which methods are chained.

- In the command's `end()` method, notify Hyperdrive that path driving is finished.

```
// Called once the command ends or is interrupted.
@Override
public void end(boolean interrupted) {
    drivetrain.getHyperdrive().finishPath();
}
```

- The command's `isFinished()` method should return the value of `Hyperdrive.pathFinished()`.

```
// Returns true when the command should end.
@Override
public boolean isFinished() {
    return drivetrain.getHyperdrive().pathFinished();
}
```

The full example command is below.

```

public class EmulatePath extends CommandBase {
    private SubsystemDrive drivetrain;
    private Path pathToDrive;

    /** Creates a new EmulatePath. */
    public EmulatePath(SubsystemDrive drivetrain, Path pathToDrive) {
        this.drivetrain = drivetrain;
        this.pathToDrive = pathToDrive;

        addRequirements(drivetrain);
    }

    // Called when the command is initially scheduled.
    @Override
    public void initialize() {
        //create parameters and load path
        IEmulateParams emulateParameters = new PreferenceEmulationParams(drivetrain.getHyperdrive().getLengthUnits());
        drivetrain.getHyperdrive().loadPath(pathToDrive, emulateParameters);

        //perform calculations vital to Hyperdrive's performance
        drivetrain.getHyperdrive().performInitialCalculations();
    }

    // Called every time the scheduler runs while the command is scheduled.
    @Override
    public void execute() {
        TankTrajectory newTrajectory = drivetrain.getHyperdrive()
            .calculateNextMovements()
            .getTankTrajectory(Constants.ROBOT_WHEELBASE_WIDTH)
            .convertTime(Units.TIME.MINUTES);

        double
            newLeftOutput = newTrajectory.getLeftPercentOutput(drivetrain.getLeftVelocity()),
            newRightOutput = newTrajectory.getRightPercentOutput(drivetrain.getRightVelocity());

        drivetrain.setPercentOutputs(newLeftOutput, newRightOutput);
    }

    // Called once the command ends or is interrupted.
    @Override
    public void end(boolean interrupted) {
        drivetrain.getHyperdrive().finishPath();
    }

    // Returns true when the command should end.
    @Override
    public boolean isFinished() {
        return drivetrain.getHyperdrive().pathFinished();
    }
}

```

Now you can push code to your robot. Before path driving will actually work, Hyperdrive's built-in PID controller must be tuned. To do this, start your record command and drive the robot through a straight path. Stop recording, drive your robot back to the start of the path, and start the emulation command (keeping your finger over the E-Stop button). Nothing should happen because the PID constants default to 0. Now, tune the PID constants until your robot hits the set maximum velocity (in the Preferences table as "Emulate Maximum Velocity" if using `PreferenceEmulationParams`) without overshooting. If you used `PreferenceEmulationParams` when loading the path, the PID constants should be in the Preferences table as "Hyperdrive kP", "Hyperdrive kI " and so forth.

After tuning the velocity PID, you can now use Hyperdrive to drive actual paths. First, zero your robot using the `InstantCommand` you created towards the beginning of the example. Then, activate your `CyborgCommandRecordPath` and use a controller to manually drive the robot through a path. After that, deactivate the command, drive the robot back to its starting position and heading, and zero the robot again. Then, activate your `CyborgCommandEmulatePath` command and watch your robot drive through the path you just made. The robot will probably not drive through the path perfectly, and may have made some major mistakes as it drove through the path that first time. This is why you should use a `PreferenceEmulationParams` to tune your robot's path driving. Use the "Tuning" table in the next chapter to assist you with tuning your parameters for optimal robot function.

Spend some time recording, emulating, and tuning your paths on your robot. At very high speeds, paths may need to have unique parameters because of the increased effect of gyro and positional drift, but at medium and lower speeds, one set of parameters should effectively drive the robot through any path.

The full example project that was just covered can be found at github.com/BTK203/Hyperdrive/tree/main/ExampleProject.

Tips, Tuning, and Troubleshooting

Autonomous driving is complex, so you may come across some trouble while setting up the library. This chapter will cover some tips and troubleshooting strategies that you can use to help you add Hyperdrive to your robot.

Tips

- **Follow the example project outlined in the previous chapter.** The example project closely guides you through all of the components needed for Hyperdrive to fully work on your robot.
- **Start slow.** It is always less scary when the robot veers off course at slow speeds. In addition, Hyperdrive works better at slow speeds. That being said, as you test your robot's first paths, start it with a low minimum and maximum speed and increase it gradually.
- **Use a real gyro.** The `TankGyro` class, while offering potential for future use, simply is not at the stage where it should be used on a competition robot. For best results, use a gyro such as the NavX mini.
- **Do not let your robot drift or slide.** Hyperdrive's robot position tracker relies on the position of the motors accurately depicting their displacement. As soon as the wheels slip, the motor position no longer has that accuracy. When recording and emulating paths, keep robot drift to a minimum.
- **While recording, avoid turning the robot in place.** Hyperdrive is capable of driving backwards and making 3-point turns, but is not programmed to handle turning the robot large amounts in place.
- **Use PathVisualizer.** Especially when tuning the parameters, PathVisualizer is an excellent tool that you can use to compare the desired path to the robot's actual path. This can help you identify problems with your robot's autonomous driving and fix them easily.

Tuning

This table outlines some symptoms of a Hyperdrive that needs to be tuned, and which parameters to change to tune them out:

Symptom	Fix
Robot takes turns too shallowly	Increase the "Emulate Overturn" value.
Robot takes turns too tightly	Decrease the "Emulate Overturn" value.

When the robot gets off course, it takes a long time to get back on course (if it even tries at all)	Increase the “Emulate Positional Correction Inhibitor” value. If you are using big length units such as meters or yards, this value may need to be increased drastically to as high as 1 or 1.5.
<p>The robot works too hard to stay on course, resulting in a wavy path like the one pictured:</p> 	Decrease the “Emulate Positional Correction Inhibitor” value.
The robot takes turns too late and/or the path that it drives is very rough	Increase the “Emulate Immediate Path Size” value.
The robot takes turns too early and cuts corners too much	Decrease the “Emulate Immediate Path Size” value.
The robot’s drive gearboxes make a kind of grinding noise, as if the motors are trying to change directions multiple times per second	Increase your motor controllers’ closed loop ramp rate. This ramp rate should be greater than 0 seconds but less than 0.3 seconds.
The robot does not seem very responsive as it drives the path	Decrease your motor controllers’ closed loop ramp rate. This ramp rate should be greater than 0 seconds but less than 0.3 seconds. If that does not work, decrease the “Emulate Immediate Path Size” value.
The robot takes corners too fast, resulting in a gyro or positional drift	Increase the “Emulate Coefficient of Static Friction” value.
The robot takes corners too slowly	Decrease the “Emulate Coefficient of Static Friction” value.

Troubleshooting

This table outlines some problems that you may experience while using Hyperdrive and some potential fixes for those problems:

Symptom	Fix
Hyperdrive drives unpredictably, specifically upon reaching the first turn of a path.	Ensure that the gyro is connected to the robot and is functioning properly.
Upon starting a path, the robot drives in the wrong direction or displays some other unpredictable behavior.	<p>First, position the robot back at its starting point and make sure that its position is correct. This can be done using PathVisualizer (line the black “Robot Position” dot up to the green dot marking the start of your path). Load the path that you wish to emulate and ensure that the black robot position dot is by the start of the path (which is marked with a green dot).</p> <p>If that does not fix the problem, then the robot’s direction may be inverted. You can fix that problem by calling the <code>invertDirection()</code> method on the <code>TankTrajectory</code> object that you use to set your motor velocities.</p> <p>If inverting the direction does not help, ensure that the correct path is being loaded into Hyperdrive. You may need to un-invert the direction if that was not a problem.</p> <p>If the correct path is loaded and you are using the built-in PID controller, make sure that you are passing the <i>angular velocities</i> of the motors into the percentOutput methods, not the <i>linear velocities</i> of those sides.</p>
The robot turns the wrong direction.	Call the <code>invertTurn()</code> method on the <code>TankTrajectory</code> object that you use to set your motor velocities.
The robot finishes the path, but it is facing the wrong direction.	Re-record the path. While recording, keep in mind that Hyperdrive was not programmed to turn robots in place. Anytime the robot turns, try to make sure that it is moving forwards or backwards as well.
The robot seems to exaggerate a turn that does	Hyperdrive, depending on the configuration of

not exist.	the parameters, will sometimes exaggerate small turns. To fix this, re-record the path and try to minimize the number of turns that the robot makes. If the robot can drive straight through an area, it should.
The robot does not drive smoothly. The drive motors seem to be giving an inconsistent amount of power.	Ensure that the velocity PID that the drivetrain is running is tuned. The robot should be able to speed up to almost any speed without overshooting or oscillating about the target velocity. Most likely, the kP value is too high.
PathVisualizer says that the robot emulated a path correctly, but it didn't	Ensure that the robot started in the correct place and that the wheels did not drift while the robot was driving the path. Also check to see that the gyro heading is correct. It should be consistent with what it was before the path was driven.

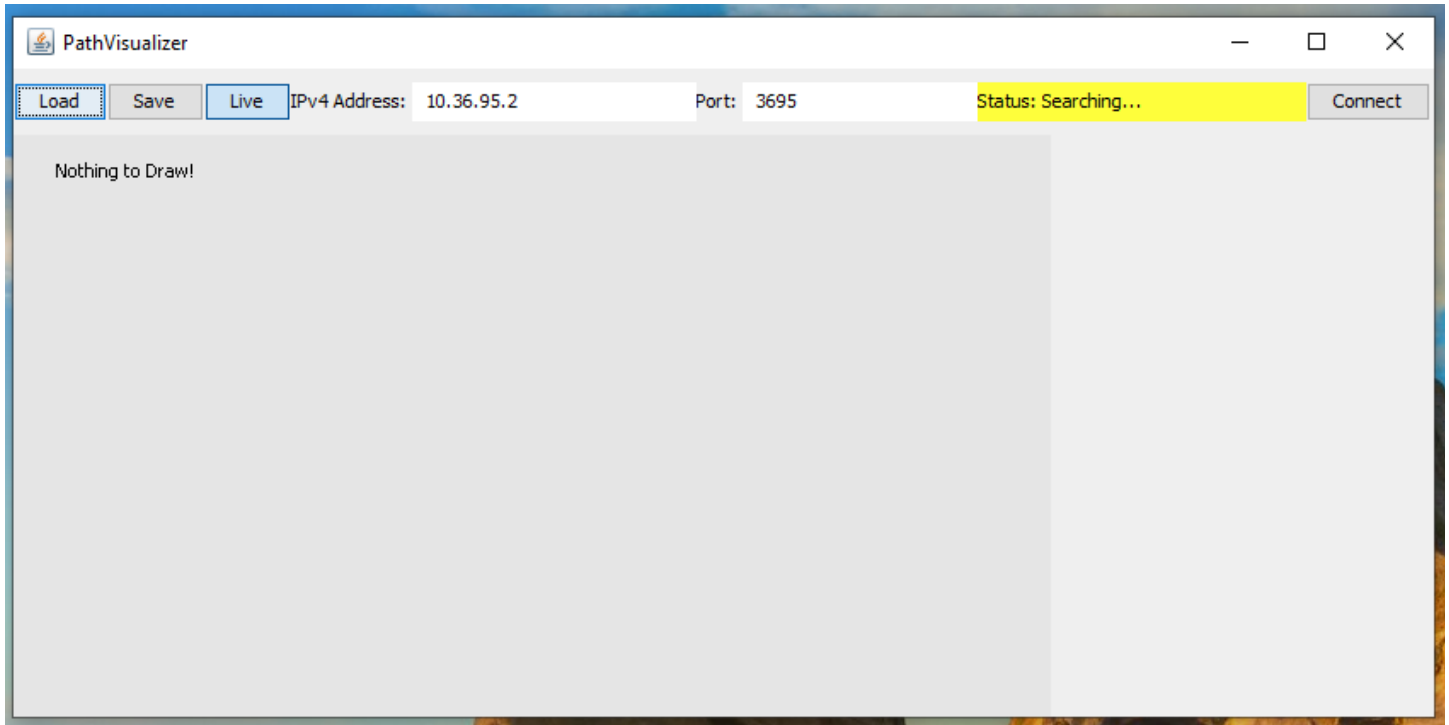
Troubleshooting Strategies

The table above should cover most of the problems that you might come across as you add autonomous driving abilities to your robot. However, there may be more problems that may come up. This list of steps, if followed in order, should help you resolve most of those problems.

- If the problem involves a specific Path, re-record it.
- Slow down the robot by decreasing the minimum and maximum speeds.
- Try starting your robot in a different position or record a different path that achieves the same purpose.
- Make sure that your gyro is plugged in and functioning properly.
- Try recording the path with no changes in direction and minimal turns.
- Make sure that your target velocities are being calculated with the correct time unit (i.e make sure that `convertTime()` is being called on your `TankTrajectory` with the right time unit)
- Make sure that your robot's heading is correctly reported by your gyro.
- Make sure that your robot's position is correctly reported by Hyperdrive.
- Try using the default parameters (`ConstantEmulationParams.getDefault()`)
- Make sure that your drivetrain velocity PID loop is properly tuned.
- Make sure that Hyperdrive is correctly implemented and functioning on your robot. Use the example project to help.

- Try using a length unit of either inches, feet, or meters (those three are confirmed to work with driving the robot around; the other length units can be correctly converted to and from any other supported unit, but have not been used to drive a robot around)

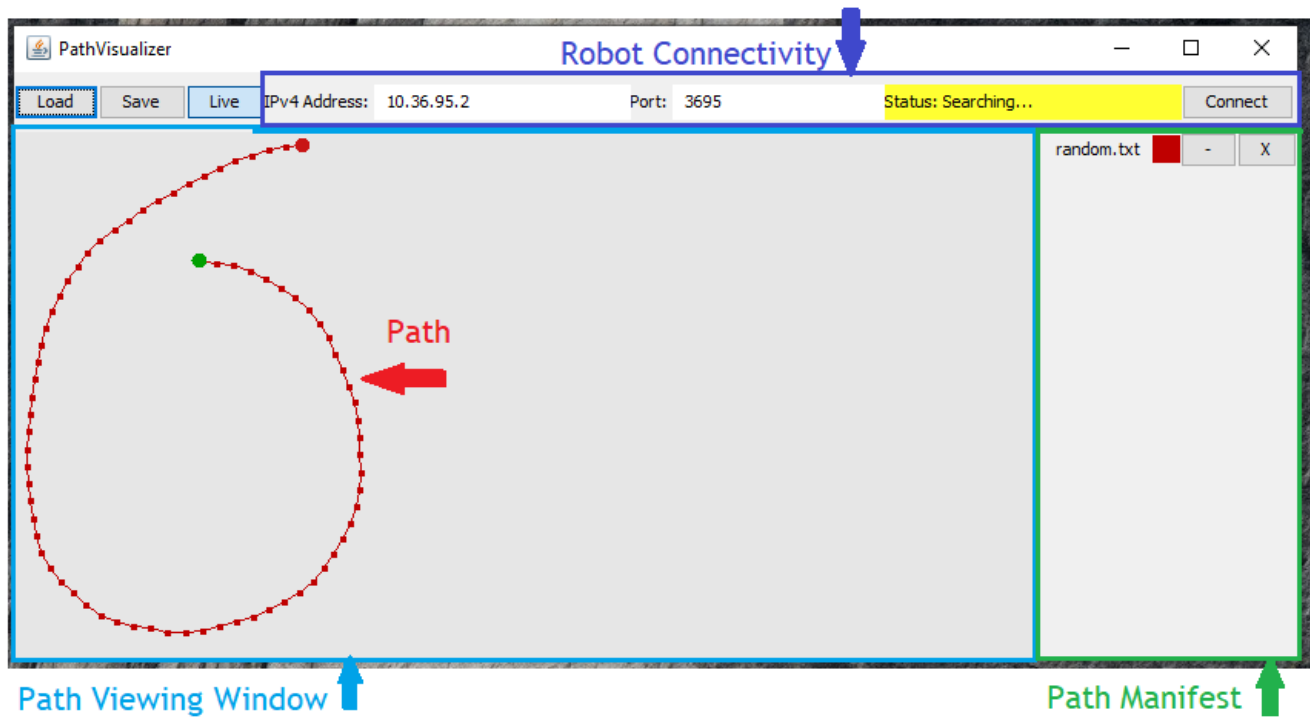
PathVisualizer





This chapter will cover how to use PathVisualizer with your robot. PathVisualizer is a Java application for viewing paths generated by Hyperdrive. With this tool, it is easy to identify and fix problems with your robot's autonomous driving. PathVisualizer can render paths to scale, display the robot's current position relative to those paths, and read paths from and save paths to the computer, as well as the robot's file system.

In order for PathVisualizer to successfully communicate with the robot, an instance of the `Hyperdrive` class must be present in the robot code and its `update()` method must be called in one of the robot's `periodic()` methods. PathVisualizer's robot-related functions will not work on robots where these conditions are not met.

Anatomy

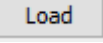


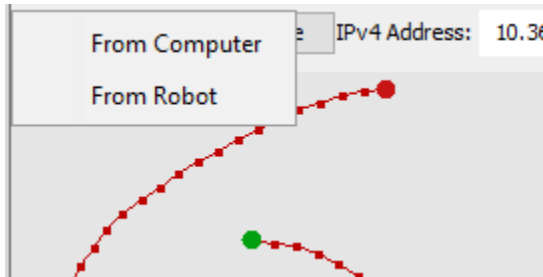
The above image shows the general layout of PathVisualizer. The majority of the window is taken up by the Path Viewing Window, where all paths and the robot's position on the field (if the robot is connected) will be rendered. To the right of the Path Viewing Window is the Path Manifest, where the names of all paths and their respective colors are shown. Here, the user can press the  button to hide the path and the  button to delete the path. Above the manifest is the Robot Connectivity area where the user can specify the IP Address and port to use to connect to the robot. This area also features a connection status indicator, which will read out the status of the connection to the robot. Finally, to the left of that region are the "Load", "Save", and "Live" buttons. Using these buttons, the user can read a path from the computer or robot, save paths to the computer or robot, and toggle on or off the ability of PathVisualizer to receive live path data from the robot.

Viewing Paths

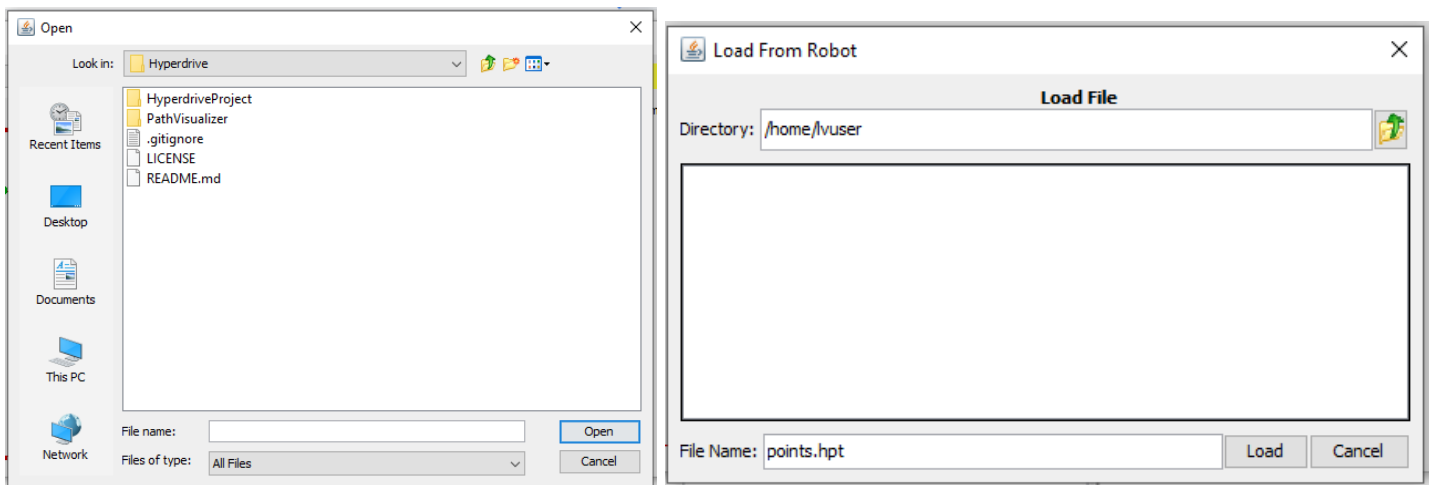
PathVisualizer's main function, which gives it its name, is to view paths that the robot can drive or has already driven. This section will cover a few ways to do that.

- **Using the “Load” button**

PathVisualizer can load paths from either the computer or the robot's file system. To do either of these, click on the  button. Then, use the context menu to select whether or not to load from the computer or from the robot.



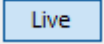
A file dialog should pop up, prompting you to select the file to load.



Computer File Dialog (left), and Robot File Dialog (right)

After selecting the file to load, PathVisualizer should load that file and the path should become visible on the Path Viewing Window.

- **From Live Robot Data**

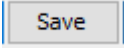
PathVisualizer also has the ability to automatically load paths that the robot sends. However, in order for PathVisualizer to accept the paths being sent, the  option must be enabled. When live data is enabled, PathVisualizer will display the following:

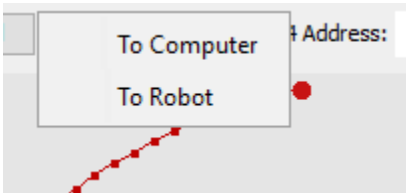
- Newly recorded paths every time the robot finishes recording a new path.

- Newly driven paths every time the robot finishes driving a path.
- Other paths sent with `Hyperdrive`'s `sendPath()` method.

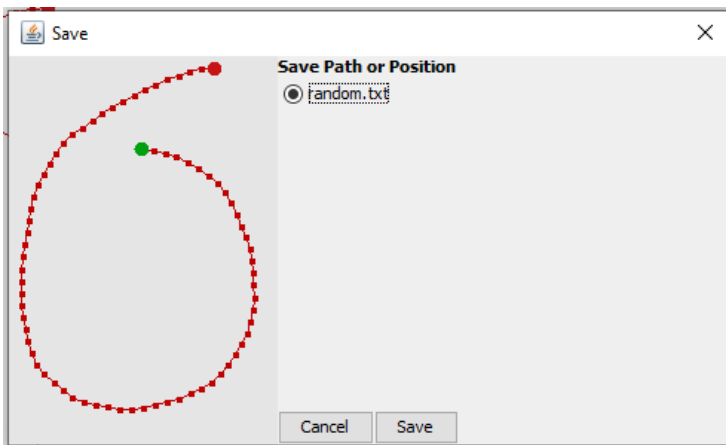
Saving Paths

PathVisualizer also offers an option to save paths to the computer or the robot.

In PathVisualizer, click the  button. Using the context menu, select whether you wish to save a path to the computer or to the robot.



Use the dialog to select which path to save.



Finally, use the file dialogs to choose where to save the path or position.