Webb-Swerve-Lib Reference Guide

**Team 1466 Webb School Robotics**

Table of Contents

[[ (p.)]]

[[ (p.)]]

[[ (p.)]]

[[ (p.)]]

[[ (p.)]]

[[ (p.)]]

[[ (p.)]]

1. Swerve Drive Fundamentals

What is Swerve Drive?

Swerve drive is an omnidirectional drive system where each wheel can rotate 360 degrees independently and be driven at variable speeds. This allows for:

* Movement in any direction without changing robot orientation
* Rotation while translating
* Precise position control
* Complex maneuvers that aren't possible with other drive systems

Key Components of Each Module

* **Drive Motor**: Controls wheel speed
* **Steering Motor**: Controls wheel direction
* **Absolute Encoder**: Provides absolute position of steering
* **Module Frame**: Physical structure that holds components

Coordinate Systems

* **Robot-Relative**: X forward, Y left relative to the robot
* **Field-Relative**: X toward opposing alliance, Y toward driver left field boundary
* Commands are typically given in field-relative coordinates for intuitive control

2. Hardware Configuration

Physical Setup

**Module Positioning**: Accurately measure and record the position of each module relative to robot center

**Encoder Mounting**: Ensure absolute encoders are securely attached with no slippage

**Wheel Alignment**: All wheels should point in the same direction when modules are "zeroed"

Encoder Calibration Process

Position all wheels to point in the forward robot direction using a straight edge

Record the raw absolute encoder values for each module

Calculate offsets to make each module read "zero" when pointing forward

Apply these offsets in the ModuleConfig constructor

Example Module Configuration

new ModuleConfig(  
 0, // Module index (FL = 0, FR = 1, BL = 2, BR = 3)  
 new Translation2d(0.3, 0.3), // Position from robot center (meters)  
 1, // Drive motor CAN ID  
 2, // Steer motor CAN ID  
 3, // Absolute encoder CAN ID  
 false, // Drive motor inverted  
 false, // Steer motor inverted  
 Math.toRadians(237.1) // Absolute encoder offset (radians)  
)

3. Software Architecture

Core Components

* **Drive Subsystem**: Central component that coordinates all drive functionality
* **Module**: Represents a single swerve module
* **RobotState**: Tracks robot position and orientation on field
* **DriveTrajectory**: Command for following autonomous paths
* **DefaultDriveCommand**: Command for teleoperated control

Dependency Diagram

Controller Input → DefaultDriveCommand → Drive Subsystem → Module → Motors  
 ↑  
Trajectory → DriveTrajectory ─────────┘  
 ↑  
 RobotState

Data Flow

**Input Sources**: Controllers or autonomous trajectory

**Command Layer**: Processes inputs, generates desired robot motion

**Drive Subsystem**: Converts desired motion to module states

**Modules**: Execute individual wheel movements

**Feedback Loop**: Odometry updates RobotState which influences commands

4. Tuning Process

Recommended Tuning Order

**Module Configuration**: Correct dimensions and offsets

**Drive Motor PID**: Velocity control for wheels

**Steering Motor PID**: Position control for modules

**Kinematics Verification**: Basic movement tests

**Path Following Tuning**: For autonomous trajectory execution

Drive Motor Tuning

|  |  |  |
| --- | --- | --- |
| Parameter | Starting Value | Purpose |
| kP | 0.05 | Response speed |
| kI | 0.0 | Eliminate steady-state error |
| kD | 0.0 | Dampen oscillations |
| kF | 0.046 | Feed-forward for velocity |

Process:

Start with feed-forward (kF)

Add proportional gain (kP)

Add derivative gain (kD) if oscillation occurs

Add minimal integral gain (kI) only if necessary

Steering Motor Tuning

|  |  |  |
| --- | --- | --- |
| Parameter | Starting Value | Purpose |
| kP | 0.2 | Response speed |
| kI | 0.0 | Eliminate steady-state error |
| kD | 4.0 | Dampen oscillations |

Process:

Start with proportional gain (kP)

Add significant derivative gain (kD)

Test with rapid direction changes and fine-tune

Path Following Tuning

|  |  |  |
| --- | --- | --- |
| Parameter | Starting Value | Purpose |
| Translation kP | 1.0 | Position tracking accuracy |
| Rotation kP | 2.0 | Heading tracking accuracy |
| Translation kD | 0.0 | Dampen position oscillations |
| Rotation kD | 0.1 | Dampen heading oscillations |

5. Common Issues and Solutions

Module Issues

|  |  |  |
| --- | --- | --- |
| Problem | Symptoms | Solution |
| Incorrect module offset | Wheels not aligned at startup | Recalibrate absolute encoder offsets |
| Module oscillation | Wheel rapidly changes angle | Reduce kP or increase kD on steering motor |
| Wheel slippage | Robot moves less than commanded | Reduce acceleration or clean wheels |
| One module behaves differently | Inconsistent movement | Check CAN connections and PID values |

Drive System Issues

|  |  |  |
| --- | --- | --- |
| Problem | Symptoms | Solution |
| Robot drifts while driving | Does not move straight | Check gyro calibration and module alignment |
| Robot doesn't respond to commands | No movement when commanded | Check CAN connections and motor configurations |
| Unexpected direction of movement | Robot moves in wrong direction | Check coordinate system conventions |
| Jerky movement | Robot motion not smooth | Adjust slew rate limits or check for loop timing issues |

Autonomous Issues

|  |  |  |
| --- | --- | --- |
| Problem | Symptoms | Solution |
| Robot doesn't follow path | Deviates from trajectory | Increase path following PID values |
| Oscillation during path | Robot weaves along path | Reduce path following PID values |
| Path too aggressive | Robot tips or slips | Reduce velocity or acceleration constraints |
| Trajectory not found | Error during autonomous | Check trajectory name and generation |

6. Command Reference

Key Commands

DefaultDriveCommand

Purpose: Handles teleoperated driving from controller input

drive.setDefaultCommand(new DefaultDriveCommand(  
 drive,  
 () -> -driverController.getLeftY(), // Forward/backward  
 () -> -driverController.getLeftX(), // Left/right  
 () -> -driverController.getRightX() // Rotation  
));

DriveTrajectory

Purpose: Follows a pre-defined trajectory for autonomous

new DriveTrajectory(drive, new HolonomicTrajectory("trajectoryName"))

SequentialCommandGroup

Purpose: Executes commands in sequence

new SequentialCommandGroup(  
 new InstantCommand(() -> drive.setPose(startPose)),  
 new DriveTrajectory(drive, pickupTrajectory),  
 new IntakeCommand(intake),  
 new DriveTrajectory(drive, scoreTrajectory)  
)

ParallelCommandGroup

Purpose: Executes commands simultaneously

new ParallelCommandGroup(  
 new DriveTrajectory(drive, approachTrajectory),  
 new PrepareArmCommand(arm)  
)

7. Quick Reference Tables

Module Constants (SDS MK4/MK4i)

|  |  |  |  |
| --- | --- | --- | --- |
| Configuration | Drive Ratio | Max Speed | Steering Ratio |
| L1 | 8.14:1 | 14.5 ft/s | ~21.43:1 |
| L2 | 6.75:1 | 17.5 ft/s | ~21.43:1 |
| L3 | 6.12:1 | 19.3 ft/s | ~21.43:1 |
| L4 | 5.14:1 | 22.9 ft/s | ~21.43:1 |

Typical Robot Constraints by Weight

|  |  |  |  |
| --- | --- | --- | --- |
| Robot Weight | Max Velocity | Max Acceleration | Rotation Speed |
| Light (<50 lbs) | 4.5-5.0 m/s | 3.0-4.0 m/s² | 720-900 deg/s |
| Medium (50-100 lbs) | 3.5-4.5 m/s | 2.0-3.0 m/s² | 540-720 deg/s |
| Heavy (100+ lbs) | 2.5-3.5 m/s | 1.5-2.0 m/s² | 360-540 deg/s |

Common PID Values

|  |  |  |  |
| --- | --- | --- | --- |
| Control Type | Typical kP | Typical kI | Typical kD |
| Falcon Drive | 0.05-0.15 | 0.0 | 0.0-1.0 |
| Falcon Steer | 0.2-0.8 | 0.0 | 5.0-15.0 |
| NEO Drive | 0.0001-0.0005 | 0.0 | 0.0-0.001 |
| NEO Steer | 0.0005-0.002 | 0.0 | 0.001-0.01 |
| Path Following Translation | 3.0-5.0 | 0.0 | 0.0-0.5 |
| Path Following Rotation | 3.0-7.0 | 0.0 | 0.1-1.0 |

Troubleshooting Checklist

* Verify all CAN IDs are correct and unique
* Check absolute encoder offsets
* Confirm module positions in code match physical robot
* Verify proper motor and encoder directions
* Check all electrical connections
* Validate gyroscope orientation and calibration
* Verify proper coordinate system conventions

Programming Quick Reference

Essential Imports

// WPILib imports  
import edu.wpi.first.math.geometry.Pose2d;  
import edu.wpi.first.math.geometry.Rotation2d;  
import edu.wpi.first.math.geometry.Translation2d;  
import edu.wpi.first.math.kinematics.ChassisSpeeds;  
import edu.wpi.first.math.kinematics.SwerveDriveKinematics;  
import edu.wpi.first.math.kinematics.SwerveModuleState;  
import edu.wpi.first.wpilibj2.command.Command;  
import edu.wpi.first.wpilibj2.command.CommandBase;  
  
// Webb-Swerve-Lib imports  
import org.webbrobotics.frc2025.subsystems.drive.Drive;  
import org.webbrobotics.frc2025.subsystems.drive.Module;  
import org.webbrobotics.frc2025.RobotState;

Common Code Patterns

**Field-Relative Movement:**

ChassisSpeeds speeds = ChassisSpeeds.fromFieldRelativeSpeeds(  
 xSpeed, ySpeed, rotSpeed, RobotState.getInstance().getHeading()  
);  
drive.runVelocity(speeds);

**Module State Optimization:**

SwerveModuleState state = new SwerveModuleState(wheelSpeed, targetAngle);  
state.optimize(currentAngle);  
module.runSetpoint(state);

**Reset Robot Pose:**

drive.setPose(new Pose2d(1.5, 5.5, Rotation2d.fromDegrees(180)));

**Creating Autonomous Routine:**

public class ExampleAuto extends SequentialCommandGroup {  
 public ExampleAuto(Drive drive, IntakeSubsystem intake) {  
 addCommands(  
 new InstantCommand(() -> drive.setPose(startPose)),  
 new DriveTrajectory(drive, new HolonomicTrajectory("path1")),  
 new IntakeCommand(intake),  
 new DriveTrajectory(drive, new HolonomicTrajectory("path2"))  
 );  
 }  
}