Robot Samples Documentation

Auto Drive Samples

These samples illustrate different methods for autonomous navigation.

RobotAutoDriveByEncoder_Linear.java

Description:

Demonstrates how to drive a specific path using motor encoders for distance measurement.

Key Features:

- **Encoder Navigation:** Uses `RUN_TO_POSITION` mode to drive precise distances.
- **Path Execution:** Executes a sequence of moves (Forward, Turn, Reverse).
- **Safety:** Includes timeouts for each movement to prevent the robot from getting stuck.

Code Breakdown:

- **Hardware Initialization:**

  Standard setup for a two-motor drive.

```java
leftDrive  = hardwareMap.get(DcMotor.class, "left_drive");
rightDrive = hardwareMap.get(DcMotor.class, "right_drive");
// Reverse one motor so positive power moves both forward
leftDrive.setDirection(DcMotor.Direction.REVERSE);
rightDrive.setDirection(DcMotor.Direction.FORWARD);
// Reset encoders to zero
leftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
rightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
// Set to run using encoders (speed control)
leftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
rightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
```

- **Constants & Math:**

Calculating `COUNTS_PER_INCH` is the most critical step for accuracy.

```java
static final double     COUNTS_PER_MOTOR_REV    = 1440 ;    // eg: TETRIX Motor Encoder
static final double     DRIVE_GEAR_REDUCTION    = 1.0 ;     // No External Gearing.
static final double     WHEEL_DIAMETER_INCHES   = 4.0 ;     // For figuring circumference
static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV * DRIVE_GEAR_REDUCTION) /
                                                  (WHEEL_DIAMETER_INCHES * 3.1415);
```

Formula: `(Encoder Ticks per Rev * Gear Ratio) / (Wheel Diameter * PI)`

- `encoderDrive` **Method - The Core Logic:**

This method performs a blocking move (pauses the main script until finished).

```java
public void encoderDrive(double speed, double leftInches, double rightInches, double timeoutS) {
    // 1. Calculate new targets
    int newLeftTarget = leftDrive.getCurrentPosition() + (int)(leftInches * COUNTS_PER_INCH);
    int newRightTarget = rightDrive.getCurrentPosition() + (int)(rightInches * COUNTS_PER_INCH);
    // 2. Set targets and Mode
    leftDrive.setTargetPosition(newLeftTarget);
    rightDrive.setTargetPosition(newRightTarget);
    leftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    rightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    // 3. Start motion
```

```
        runtime.reset();

        leftDrive.setPower(Math.abs(speed));

        rightDrive.setPower(Math.abs(speed));

        // 4. Wait loop

        // Keeps looping while:

        // - OpMode is active (User hasn't pressed Stop)

        // - Timeout hasn't expired

        // - Both motors are still "busy" (haven't reached target)

        while (opModeIsActive() &&

               (runtime.seconds() < timeoutS) &&

               (leftDrive.isBusy() && rightDrive.isBusy())) {

            telemetry.addData("Running to",  " %7d :%7d", newLeftTarget,  newRightTarget);

            telemetry.addData("Currently at",  " at %7d :%7d",

                                    leftDrive.getCurrentPosition(), rightDrive.getCurrentPosition());

            telemetry.update();

        }

        // 5. Stop and Cleanup

        leftDrive.setPower(0);

        rightDrive.setPower(0);

        leftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER); // Turn off RUN_TO_POSITION

        rightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

    }
```

Crucial Detail: The `isBusy()` check is what makes `RUN_TO_POSITION` work. The motor controller handles the PID loop internally to reach the target position.

RobotAutoDriveByGyro_Linear.java

Description:

A more advanced autonomous driving sample that uses the IMU (Gyro) to maintain a straight heading while driving and to perform accurate turns.

Key Features:

- **Gyro Stabilization:** Uses a Proportional (P) controller to correct the robot's heading while driving straight.
- **Accurate Turning:** Turns to a specific absolute heading using the Gyro.
- **Hold Heading:** actively fights external forces to maintain a heading.

Code Breakdown:

- **IMU Initialization:**

  Correctly defining the Hub's orientation is the #1 reason for gyro issues.

```java
// Define how the Hub is mounted.
// Example: Logo UP, USB FORWARD
RevHubOrientationOnRobot.LogoFacingDirection logoDirection = RevHubOrientationOnRobot.LogoFacingDirection.UP;
RevHubOrientationOnRobot.UsbFacingDirection  usbDirection  = RevHubOrientationOnRobot.UsbFacingDirection.FORWARD;
RevHubOrientationOnRobot orientationOnRobot = new RevHubOrientationOnRobot(logoDirection, usbDirection);
// Initialize the IMU
imu = hardwareMap.get(IMU.class, "imu");
imu.initialize(new IMU.Parameters(orientationOnRobot));
// Reset Yaw (heading) to 0 on initialization
imu.resetYaw();
```

- `driveStraight` **Method (Driving with Heading Correction):**

  This method drives forward while actively steering to keep straight.

```java
```

```
public void driveStraight(double maxDriveSpeed, double distance, double heading) {
    // 1. Setup Encoders for distance
    int moveCounts = (int)(distance * COUNTS_PER_INCH);
    leftTarget = leftDrive.getCurrentPosition() + moveCounts;
    rightTarget = rightDrive.getCurrentPosition() + moveCounts;
    leftDrive.setTargetPosition(leftTarget);
    rightDrive.setTargetPosition(rightTarget);
    leftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    rightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    // 2. Start Moving
    maxDriveSpeed = Math.abs(maxDriveSpeed);
    moveRobot(maxDriveSpeed, 0); // Start with 0 turn
    // 3. Correction Loop
    while (opModeIsActive() && (leftDrive.isBusy() && rightDrive.isBusy())) {
        // Calculate how much to turn to maintain 'heading'
        turnSpeed = getSteeringCorrection(heading, P_DRIVE_GAIN);
        // Reverse steering if driving backwards
        if (distance < 0) turnSpeed *= -1.0;
        // Apply correction
        moveRobot(driveSpeed, turnSpeed);
    }
    // 4. Stop
    moveRobot(0, 0);
    leftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    rightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
}
```

- `turnToHeading` **Method (Pivot Turn):**

Rotates in place to a specific angle.

```java
public void turnToHeading(double maxTurnSpeed, double heading) {
    // Loop until error is small enough (HEADING_THRESHOLD)
    while (opModeIsActive() && (Math.abs(headingError) > HEADING_THRESHOLD)) {
        // Calculate turn power based on error
        turnSpeed = getSteeringCorrection(heading, P_TURN_GAIN);
        // Clip to max speed
        turnSpeed = Range.clip(turnSpeed, -maxTurnSpeed, maxTurnSpeed);
        // Pivot (0 forward speed, only turn)
        moveRobot(0, turnSpeed);
    }
    moveRobot(0, 0);
}
```

- `getSteeringCorrection` **Method (Proportional Controller):**

The math behind the smooth turning.

```java
public double getSteeringCorrection(double desiredHeading, double proportionalGain) {
    // 1. Calculate Error
    targetHeading = desiredHeading;
    headingError = targetHeading - getHeading();
    // 2. Normalize Error to +/- 180 (Shortest path)
    // e.g., if error is 270, it becomes -90 (turn left instead of long way right)
    while (headingError > 180)  headingError -= 360;
    while (headingError <= -180) headingError += 360;
```

```java
    // 3. Calculate Power (P-Controller)

    // Power = Error * Gain

    return Range.clip(headingError * proportionalGain, -1, 1);

}
```

- `moveRobot` **Helper:**

  Combines drive and turn inputs into left/right motor powers.

```java
public void moveRobot(double drive, double turn) {

    double leftSpeed  = drive - turn;

    double rightSpeed = drive + turn;

    // Normalize if > 1.0

    double max = Math.max(Math.abs(leftSpeed), Math.abs(rightSpeed));

    if (max > 1.0) {

        leftSpeed /= max;

        rightSpeed /= max;

    }

    leftDrive.setPower(leftSpeed);

    rightDrive.setPower(rightSpeed);

}
```

[RobotAutoDriveByTime_Linear.java]

Description:

The simplest form of autonomous driving, based solely on time.

Key Features:

- **No Sensors:** Does not require encoders or a gyro.

- **Simplicity:** Very easy to understand but less accurate due to battery voltage and friction variations.

Code Breakdown:

- **Step-by-Step Execution:**

The code executes sequentially. Each block runs a specific movement for a set time.

```java
// Step 1: Drive Forward
leftDrive.setPower(FORWARD_SPEED);
rightDrive.setPower(FORWARD_SPEED);
runtime.reset(); // Start timer
// Wait for 3 seconds
while (opModeIsActive() && (runtime.seconds() < 3.0)) {
    telemetry.addData("Path", "Leg 1: %4.1f S Elapsed", runtime.seconds());
    telemetry.update();
}
// Step 2: Spin Right
leftDrive.setPower(TURN_SPEED);
rightDrive.setPower(-TURN_SPEED); // Opposite powers for spin
runtime.reset();
while (opModeIsActive() && (runtime.seconds() < 1.3)) {
    // ...
}
// Step 4: Stop
leftDrive.setPower(0);
rightDrive.setPower(0);
```

Note: This method is "Open Loop" - the robot has no idea if it actually moved the correct distance. It just applies power for time.

RobotAutoDriveToLine_Linear.java

Description:

Demonstrates how to drive forward until a white line is detected on the floor.

Key Features:

- **Color Sensor:** Uses a `NormalizedColorSensor` to read surface brightness.
- **Thresholding:** Compares the brightness against a `WHITE_THRESHOLD` to detect the line.

Code Breakdown:

- **Sensor Initialization:**

Configuring the sensor correctly is vital for consistent readings.

```java
// Get the sensor
colorSensor = hardwareMap.get(NormalizedColorSensor.class, "sensor_color");
// Turn on the LED so we can see in the dark/shadows
if (colorSensor instanceof SwitchableLight) {
    ((SwitchableLight)colorSensor).enableLight(true);
}
// Set Gain. Higher gain = more sensitive to light.
// You need to tune this so the white line reads clearly different from the tiles.
colorSensor.setGain(15);
```

- **Main Control Loop:**

Drives until the condition `getBrightness() < WHITE_THRESHOLD` becomes FALSE (i.e., brightness >= threshold).

```java
// Start moving
```

```java
leftDrive.setPower(APPROACH_SPEED);

rightDrive.setPower(APPROACH_SPEED);

// Loop while the floor is DARK (less than threshold)

while (opModeIsActive() && (getBrightness() < WHITE_THRESHOLD)) {

    // Sleep to yield processor time

    sleep(5);

}

// Stop immediately when loop exits (Line found)

leftDrive.setPower(0);

rightDrive.setPower(0);
```

- `getBrightness` **Helper:**

  Extracts the brightness value from the sensor data.

```java
double getBrightness() {

    NormalizedRGBA colors = colorSensor.getNormalizedColors();

    // We use the Alpha channel (overall brightness/transparency)

    return colors.alpha;

}
```

Auto Drive to AprilTag Samples

These samples demonstrate using computer vision to navigate to specific targets.

RobotAutoDriveToAprilTagOmni.java

Description:

Designed for Holonomic (Mecanum/X-Drive) robots. It detects an AprilTag and autonomously drives to position the robot directly in front of it at a set distance.

Key Features:

- **3-Axis Control:** Controls Drive (Forward/Back), Strafe (Left/Right), and Turn (Yaw) simultaneously to align with the tag.
- **PID-like Control:** Uses proportional gains (`SPEED_GAIN` , `STRAFE_GAIN` , `TURN_GAIN`) to smooth the approach.

Code Breakdown:

- **AprilTag Initialization:**

Sets up the vision processing pipeline.

```java

// 1. Create Processor

aprilTag = new AprilTagProcessor.Builder().build();

aprilTag.setDecimation(2); // Optimization: Lower res for faster speed

// 2. Create VisionPortal (connects camera to processor)

visionPortal = new VisionPortal.Builder()

        .setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"))

        .addProcessor(aprilTag)

        .build();

```

- **Target Detection Logic:**

Iterates through detections to find the *specific* tag ID we want.

```java

List currentDetections = aprilTag.getDetections();

for (AprilTagDetection detection : currentDetections) {

    if (detection.metadata != null) { // Check if tag is in library

        if ((DESIRED_TAG_ID < 0) || (detection.id == DESIRED_TAG_ID)) {

            targetFound = true;

            desiredTag = detection;

            break; // Found it!
```

        }

    }

}

```

- **Automatic Drive Logic:**

If the button is held and target is found, calculate P-Loop outputs.

```java

if (gamepad1.left_bumper && targetFound) {

    // 1. Calculate Errors

    // Range Error: Difference between current range and desired distance

    double  rangeError      = (desiredTag.ftcPose.range - DESIRED_DISTANCE);

    // Heading Error: Direction the tag is relative to camera center

    double  headingError    = desiredTag.ftcPose.bearing;

    // Yaw Error: Orientation of the tag (is it facing us directly?)

    double  yawError        = desiredTag.ftcPose.yaw;

    // 2. Calculate Power (P-Controller)

    // Note: Gains (SPEED_GAIN, etc.) must be tuned for your robot!

    drive  = Range.clip(rangeError * SPEED_GAIN, -MAX_AUTO_SPEED,
MAX_AUTO_SPEED);

    turn   = Range.clip(headingError * TURN_GAIN, -MAX_AUTO_TURN,
MAX_AUTO_TURN) ;

    strafe = Range.clip(-yawError * STRAFE_GAIN, -MAX_AUTO_STRAFE,
MAX_AUTO_STRAFE);

}

```

- `moveRobot` **(Mecanum Mixing):**

Standard mecanum kinematic formula.

```java

```
public void moveRobot(double x, double y, double yaw) {

    double frontLeftPower   = x - y - yaw;

    double frontRightPower   = x + y + yaw;

    double backLeftPower   = x + y - yaw;

    double backRightPower   = x - y + yaw;

    // ... normalize and set powers ...

}
```

RobotAutoDriveToAprilTagTank.java

Description:

Similar to the Omni version but adapted for Tank Drive (non-holonomic) robots.

Key Features:

- **2-Axis Control:** Can only Drive and Turn. It cannot strafe to align laterally, so it relies on turning to center the tag.
- **Approach:** Aligns the robot's heading with the tag and drives to the target distance.

Code Breakdown:

- **Control Logic (Tank Drive):**

Tank drive is limited compared to Mecanum. We can only move forward/back and turn. We cannot strafe to fix lateral offset.

```java
// 1. Calculate Errors

// We ignore yawError because we can't strafe to fix it.

double  rangeError   = (desiredTag.ftcPose.range - DESIRED_DISTANCE);

double  headingError = desiredTag.ftcPose.bearing;

// 2. Calculate Power

drive = Range.clip(rangeError * SPEED_GAIN, -MAX_AUTO_SPEED,
MAX_AUTO_SPEED);
```

turn = Range.clip(headingError * TURN_GAIN, -MAX_AUTO_TURN, MAX_AUTO_TURN) ;

```

- `moveRobot` **(Tank Mixing):**

Combines drive and turn for left/right sides.

```java
public void moveRobot(double x, double yaw) {
    double leftPower    = x - yaw;
    double rightPower   = x + yaw;
    // Normalize
    double max = Math.max(Math.abs(leftPower), Math.abs(rightPower));
    if (max > 1.0) {
        leftPower /= max;
        rightPower /= max;
    }
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
}
```

TeleOp Samples

These samples demonstrate driver-controlled operation modes.

RobotTeleopMecanumFieldRelativeDrive.java

Description:

Implements "Field Centric" drive for a mecanum robot. Pushing the joystick "forward" always moves the robot away from the driver, regardless of which way the robot is facing.

Key Features:
```

- **Coordinate Transformation:** Uses the IMU to rotate the joystick inputs based on the robot's current heading.
- **Intuitive Control:** Easier for drivers as they don't need to mentally track the robot's orientation.

Code Breakdown:

- **Field Relative Math:**

This transforms the driver's "Forward" command into the robot's "Forward" command based on its current heading.

```java
private void driveFieldRelative(double forward, double right, double rotate) {
    // 1. Convert Joystick (Cartesian) to Polar (Angle & Magnitude)
    double theta = Math.atan2(forward, right);
    double r = Math.hypot(right, forward);
    // 2. Rotate the angle by the Robot's Heading
    // This effectively "cancels out" the robot's rotation
    theta = AngleUnit.normalizeRadians(theta -
            imu.getRobotYawPitchRollAngles().getYaw(AngleUnit.RADIANS));
    // 3. Convert back to Cartesian for the drive method
    double newForward = r * Math.sin(theta);
    double newRight = r * Math.cos(theta);
    // 4. Drive
    drive(newForward, newRight, rotate);
}
```

Example: If the robot is facing Right (90 deg) and you push Forward (0 deg), the code calculates -90 deg. The robot strafes Left (relative to itself), which results in moving Forward (relative to the field).

- **Reset Yaw:**

Crucial for re-calibrating "Forward" during a match.

```java
if (gamepad1.a) {
    imu.resetYaw();
}
```

- `drive` **Method:**

  Standard mecanum mixing, but with a `maxSpeed` scaler for safety/precision.

  ```java
  public void drive(double forward, double right, double rotate) {
      double frontLeftPower = forward + right + rotate;
      double frontRightPower = forward - right - rotate;
      double backRightPower = forward + right - rotate;
      double backLeftPower = forward - right + rotate;
      // ... normalize and set power ...
  }
  ```

[RobotTeleopPOV_Linear.java]

Description:

A standard "Arcade" or "POV" style control for a tank drive robot. One joystick controls forward/back, the other controls turning.

Key Features:

- **LinearOpMode:** Runs in a sequential loop.
- **Accessories:** Includes simple control for an arm (Buttons Y/A) and a claw (Bumpers).

Code Breakdown:

- **Drive Logic (Arcade/POV):**

  Mixes Y-axis (Drive) and X-axis (Turn) from different sticks (or same stick).

  ```java
  ```

```
// 1. Read Inputs
// Note: Joystick Y is negative when pushed UP, so we invert it.
drive = -gamepad1.left_stick_y;
turn  =  gamepad1.right_stick_x;
// 2. Mix Inputs
left  = drive + turn;
right = drive - turn;
// 3. Normalize (Ensure we don't exceed +/- 1.0)
max = Math.max(Math.abs(left), Math.abs(right));
if (max > 1.0) {
    left /= max;
    right /= max;
}
// 4. Set Power
leftDrive.setPower(left);
rightDrive.setPower(right);
```

- **Servo Control (Incremental):**

Moves the servo slowly while a button is held.

```java
// Check bumpers
if (gamepad1.right_bumper)
    clawOffset += CLAW_SPEED; // Open
else if (gamepad1.left_bumper)
    clawOffset -= CLAW_SPEED; // Close
// Clip range to prevent mechanical binding
clawOffset = Range.clip(clawOffset, -0.5, 0.5);
```

// Set position (assuming mirror image servos)

leftClaw.setPosition(MID_SERVO + clawOffset);

rightClaw.setPosition(MID_SERVO - clawOffset);

```

- **Arm Control (Bang-Bang):**

Simple On/Off control.

```java

if (gamepad1.y)

    leftArm.setPower(ARM_UP_POWER);

else if (gamepad1.a)

    leftArm.setPower(ARM_DOWN_POWER);

else

    leftArm.setPower(0.0); // Stop if no button pressed

```

RobotTeleopTank_Iterative.java

Description:

Implements "Tank" style control where the left stick controls the left wheels and the right stick controls the right wheels.

Key Features:

- **IterativeOpMode:** Uses `init()`, `loop()`, etc., instead of a `while` loop.
- **Direct Control:** Provides independent control of each side of the drivetrain.

Code Breakdown:

- **Drive Logic (Tank):**

Direct 1:1 mapping of joystick Y-axes to motor powers.

```java

// Left stick controls Left Wheel

left = -gamepad1.left_stick_y;

// Right stick controls Right Wheel

right = -gamepad1.right_stick_y;

leftDrive.setPower(left);

rightDrive.setPower(right);

```

This provides very direct control but requires more driver skill to drive straight.

- **Iterative Structure:**

Unlike `LinearOpMode`, this uses `loop()` which runs repeatedly.

```java

@Override

public void loop() {

    // Read Gamepad

    // Calculate Powers

    // Set Powers

    // Update Telemetry

}

```

The loop must execute quickly (non-blocking). You cannot use `sleep()` or `while()` loops inside `loop()`.
```