package org.firstinspires.ftc.teamcode;

import com.qualcomm.hardware.lynx.LynxModule;

import com.qualcomm.hardware.lynx.commands.core.LynxGetBulkInputDataCommand;

import com.qualcomm.hardware.lynx.commands.core.LynxGetBulkInputDataResponse;

import com.qualcomm.robotcore.hardware.DcMotor;

import com.qualcomm.robotcore.hardware.DcMotorSimple;

import com.qualcomm.robotcore.hardware.HardwareMap;

import com.qualcomm.robotcore.hardware.Servo;

import com.qualcomm.robotcore.util.ElapsedTime;

import com.qualcomm.robotcore.util.RobotLog;

import org.firstinspires.ftc.robotcore.external.Telemetry;

import java.util.Arrays;

import java.util.Collections;

public class PsaBot2020 {

private boolean runPosition = false;

LynxModule driveHub = null;

//MOTORS:

public DcMotor frontLeft = null;

public DcMotor frontRight = null;

public DcMotor backRight = null;

public DcMotor backLeft = null;

public DcMotor arm1 = null;

public DcMotor arm2 = null;

//SERVOS:

public Telemetry telemetry = null;

HardwareMap hwMap = null;

private ElapsedTime period = new ElapsedTime();

private static final double TICK\_FACTOR = (35.0 / 45.0) \* (1120 / Math.PI / 4);

public void init(HardwareMap ahwMap, Telemetry telem) {

// Save reference to Hardware map

hwMap = ahwMap;

this.telemetry = telem;

// Define REV Expansion hubs

driveHub = hwMap.get(LynxModule.class, "Expansion Hub 1");

// Define and Initialize Motors

frontLeft = hwMap.get(DcMotor.class, "frontLeft");

frontRight = hwMap.get(DcMotor.class, "frontRight");

backRight = hwMap.get(DcMotor.class, "backRight");

backLeft = hwMap.get(DcMotor.class, "backLeft");

arm1 = hwMap.get(DcMotor.class, "arm");

arm2 = hwMap.get(DcMotor.class, "arm2");

//Define servos

// Set all motors to zero power

backLeft.setPower(0);

backRight.setPower(0);

frontLeft.setPower(0);

frontRight.setPower(0);

arm1.setPower(0);

arm2.setPower(0);

frontLeft.setDirection(DcMotorSimple.Direction.FORWARD);

backLeft.setDirection(DcMotorSimple.Direction.FORWARD);

frontRight.setDirection(DcMotorSimple.Direction.REVERSE);

backRight.setDirection(DcMotorSimple.Direction.REVERSE);

// Set all motors to run without encoders.

// May want to use RUN\_USING\_ENCODERS if encoders are installed.

//runToPosition(false);

// Define and initialize ALL installed servos.

//intakeMechanismLeft = hwMap.get(Servo.class, "intakeMechanismLeft");

//intakeMechanismRight = hwMap.get(Servo.class, "intakeMechanismRight");

// setCapArm(1);

//setIntakeMechanism(false);

}

public void runToPosition(boolean usePos) {

runPosition = usePos;

if (usePos) {

backRight.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

frontRight.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

backLeft.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

frontLeft.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

arm1.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

arm2.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

} else {

backRight.setMode(DcMotor.RunMode.RUN\_USING\_ENCODER);

frontRight.setMode(DcMotor.RunMode.RUN\_USING\_ENCODER);

backLeft.setMode(DcMotor.RunMode.RUN\_USING\_ENCODER);

frontLeft.setMode(DcMotor.RunMode.RUN\_USING\_ENCODER);

arm1.setMode(DcMotor.RunMode.RUN\_USING\_ENCODER);

arm2.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

}

}

public void setDrivePower(double fl, double bl, double fr, double br) {

if (runPosition) {

backRight.setPower(br);

frontRight.setPower(fr);

frontLeft.setPower(fl);

backLeft.setPower(bl);

} else {

backRight.setPower(br);

frontRight.setPower(fr);

frontLeft.setPower(fl);

backLeft.setPower(bl);

}

}

private int move(int fl\_delta, int fr\_delta, int bl\_delta, int br\_delta, double power, double maxTime) {

boolean lastRunPos = runPosition;

ElapsedTime loopTime = new ElapsedTime();

int fl\_p;

int fr\_p;

int bl\_p;

int br\_p;

int busyMotors;

int max = Collections.max(Arrays.asList(Math.abs(fl\_delta), Math.abs(fr\_delta), Math.abs(bl\_delta), Math.abs(br\_delta)));

int[] encoderSpeed = new int[4];

int[] encoderCount = new int[4];

byte digitalInputs = 0;

byte motorStatus = 0;

LynxGetBulkInputDataCommand command = new LynxGetBulkInputDataCommand(driveHub);

setDrivePower(0, 0, 0, 0);

runToPosition(true);

backRight.setTargetPosition(backRight.getCurrentPosition() + br\_delta);

frontRight.setTargetPosition(frontRight.getCurrentPosition() + fr\_delta);

frontLeft.setTargetPosition(frontLeft.getCurrentPosition() + fl\_delta);

backLeft.setTargetPosition(backLeft.getCurrentPosition() + bl\_delta);

RobotLog.i("Move");

power /= max; // Scale power (velocity) to match distance to travel

setDrivePower(power \* Math.abs(fl\_delta), power \* Math.abs(fr\_delta),

power \* Math.abs(bl\_delta), power \* Math.abs(br\_delta));

int loopCnt = 0;

loopTime.reset();

do {

loopCnt++;

busyMotors = 0;

try {

LynxGetBulkInputDataResponse response = command.sendReceive();

for (int i = 0; i < 4; i++) {

encoderSpeed[i] = response.getVelocity(i);

encoderCount[i] = response.getEncoder(i);

}

byte[] payload = response.toPayloadByteArray();

digitalInputs = payload[0];

motorStatus = payload[17];

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

} catch (Exception e) {

RobotLog.logStackTrace(e);

}

fl\_p = frontLeft.getTargetPosition() - frontLeft.getCurrentPosition();

fr\_p = frontRight.getTargetPosition() - frontRight.getCurrentPosition();

bl\_p = backLeft.getTargetPosition() - backLeft.getCurrentPosition();

br\_p = backRight.getTargetPosition() - backRight.getCurrentPosition();

if (Math.abs(fl\_p) > 25) busyMotors++;

if (Math.abs(fr\_p) > 25) busyMotors++;

if (Math.abs(bl\_p) > 25) busyMotors++;

if (Math.abs(br\_p) > 25) busyMotors++;

RobotLog.ii("Move", String.format("Time = %6.3f, FL = %6d, FR = %6d, BL = %6d, BR = %6d, BM = %6d", loopTime.seconds(),

fl\_p, fr\_p, bl\_p, br\_p, busyMotors));

telemetry.addLine(String.format("Time = %6.3f, FL = %6d, FR = %6d, BL = %6d, BR = %6d, BM = %6d", loopTime.seconds(),

fl\_p, fr\_p, bl\_p, br\_p, busyMotors));

telemetry.addLine(String.format("Encoders: FL = %6d, FR = %6d, BL = %6d, BR = %6d, BM = %6d",

encoderCount[0], encoderCount[1], encoderCount[2], encoderCount[3], motorStatus));

telemetry.addLine(String.format("Cont/Sec: FL = %6d, FR = %6d, BL = %6d, BR = %6d, BM = %6d",

encoderSpeed[0], encoderSpeed[1], encoderSpeed[2], encoderSpeed[3], digitalInputs));

telemetry.update();

}

while ((busyMotors > 1) && (loopTime.seconds() < maxTime));

setDrivePower(0, 0, 0, 0);

runToPosition(lastRunPos);

return loopCnt;

}

public void move(double fwd, double sideways, double degrees, double power, double maxTime) {

double delta\_f = TICK\_FACTOR \* fwd;

double delta\_s = TICK\_FACTOR \* sideways;

double delta\_t = (TICK\_FACTOR \* Math.PI \* 27.17 \* degrees) / 360.0; // turning radius; distance of opposing wheels

int fl\_delta = (int) (delta\_f + delta\_s + delta\_t);

int fr\_delta = (int) (delta\_f - delta\_s - delta\_t);

int bl\_delta = (int) (delta\_f - delta\_s + delta\_t);

int br\_delta = (int) (delta\_f + delta\_s - delta\_t);

move(fl\_delta, fr\_delta, bl\_delta, br\_delta, power, maxTime);

}

public void forward(double distance\_inInch, double power, double maxTime) {

int delta;

distance\_inInch = TICK\_FACTOR \* distance\_inInch;

delta = (int) distance\_inInch;

setDrivePower(0, 0, 0, 0);

runToPosition(true);

move(delta, delta, delta, delta, power, maxTime);

}

public void turn(double degree, double power, double maxTime) {

double distance\_inInch = Math.PI \* 20.0 \* degree / 360.0; // turning radius; distance of opposing wheels

int delta;

distance\_inInch \*= 360.0 / 265.0;

distance\_inInch = TICK\_FACTOR \* distance\_inInch;

delta = (int) distance\_inInch;

move(delta, -delta, delta, -delta, power, maxTime);

}

public void setArm1(double power){

arm1.setPower(power);

}

public void setArm2(double power){

arm2.setPower(power);

}

/\*public void setIntakeMechanism(boolean deploy) {

// setPlatformGrabberPosition puts in angle below

intakeMechanismRight.setPosition(deploy ? 1.0 : .65);

intakeMechanismLeft.setPosition(deploy ? 0-1.0 :.65);

}

\*/

}