/\*----------------------------------------------------------------------------\*/

/\* Copyright (c) FIRST 2008. All Rights Reserved. \*/

/\* Open Source Software - may be modified and shared by FRC teams. The code \*/

/\* must be accompanied by the FIRST BSD license file in the root directory of \*/

/\* the project. \*/

/\*----------------------------------------------------------------------------\*/

//NOTES\\

/\*

- Right Joystick is inverted - STATUS: FIXED

- left encoder + when left wheels forward, right encoder - when right wheels forward - STATUS: FIXED

\*/

package edu.wpi.first.wpilibj.templates;

import edu.wpi.first.wpilibj.CounterBase.EncodingType;

import edu.wpi.first.wpilibj.Encoder;

import edu.wpi.first.wpilibj.IterativeRobot;

import edu.wpi.first.wpilibj.Joystick;

import edu.wpi.first.wpilibj.Timer;

import edu.wpi.first.wpilibj.Victor;

import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;

public class RobotCode extends IterativeRobot {

//drive motors - Digital Sidecar #1

//Victor(int slot, int channel) - slot is DSC #, channel is PWM channel plugged into DSC

Victor motorLeftRear = new Victor(1, 1); //black - CIM #1 - victor 1

Victor motorLeftCenter = new Victor(1, 2); //yellow - CIM #2 - victor 2

Victor motorLeftFront = new Victor(1, 3); //red - CIM #3 - victor 3

Victor motorRightRear = new Victor(1, 4); //green - CIM #4 - victor 4

Victor motorRightCenter = new Victor(1, 5); //blue - CIM #5 - victor 5

Victor motorRightFront = new Victor(1, 6); //white - CIM #6 - victor 6

//misc. motors - Digital Sidecar #2

Victor shooterMotor01 = new Victor(2, 1); //red + black - victor 7

Victor shooterMotor02 = new Victor(2, 2); //green + white - victor 8

Victor intakeDoor = new Victor(2 , 3); //2, 3 ORIGINALLY //blue + yellow - victor 9

Victor intakeWheels = new Victor(2, 5); //green + yellow - victor 11

Victor victor12 = new Victor(2, 6); //black + blue = victor 12

//Compressor compressor = new Compressor(2, 1, 2, 1); //(red-white-blue) compressor(pressSwitchSlot, pressSwitchChannel, compressRelaySlot, compressRelayChannel);

//Relay shooterHoodPiston01 = new Relay(2, 2);

//Relay shooterHoodPiston02 = new Relay(2, 3);

//joysticks and controller

Joystick leftJoy = new Joystick(1); //the left joystick

Joystick rightJoy = new Joystick(2); //the right joystick

Joystick controller = new Joystick(3); //the xbox controller

//encoder for each side of the drivetrain

Encoder leftEncoder = new Encoder(3, 4, true, EncodingType.k4X); //for drivetrain 4-6

Encoder rightEncoder = new Encoder(1, 2, true, EncodingType.k4X); //for drivetrain 1-3

//variables

int autonTurnLoops = 0; //the number of loops auton runs while either the left or right encoders have different values.

int autonNum = 0; //the auton choice that comes from the dashboard

//double autonTimeInit; //the time autonomous starts

boolean autoLooping = true;

boolean turningLeft = false; //whether the robot is turning left or not in auton

boolean turningRight = false; //whether the robot is turning right or not in auton

boolean autonLeftOrRight; //whether the robot needs to turn left or right in autonomous

public void robotInit() {

//compressor.start();

}

public void autonomousInit() {

//resets the encoders

leftEncoder.reset();

rightEncoder.reset();

//encoders can start reading ticks now

leftEncoder.start();

rightEncoder.start();

//sets the distance per pulse of each encoder: the wheel's circumference divided by the number of pulses per rotation

leftEncoder.setDistancePerPulse(0.03490659);

rightEncoder.setDistancePerPulse(0.03490659);

setSafetyEnabled(false);

//autonTimeInit = Timer.getFPGATimestamp();

}

public void autonomousPeriodic() {

if (autoLooping) {

/\*shooterMotor01.set(-0.2);

shooterMotor02.set(-0.2);

intakeDoor.set(0.3);

Timer.delay(1.0);

intakeDoor.set(0.0);

shooterMotor01.set(0.0);

shooterMotor02.set(0.0);

Timer.delay(1.0);

shooterMotor01.set(1.0);

shooterMotor02.set(1.0);\*/

shooterMotor01.set(0.3);

shooterMotor02.set(0.3);

/\*drive(1.0, 0.935); //left is right, right is left

Timer.delay(2.0);

shooterMotor01.set(0.0);

shooterMotor02.set(0.0);

Timer.delay(1.0);

drive(0.0, 0.0);\*/

//

drive(1.0, 0.95); //first is right, second is left

Timer.delay(1.5);

shooterMotor01.set(0.0);

shooterMotor02.set(0.0);

intakeDoor.set(-0.3);

drive(0.0, 0.0);

//

Timer.delay(1.0);

intakeDoor.set(0.0);

shooterMotor01.set(-1.0);

shooterMotor02.set(-1.0);

Timer.delay(4.0);

intakeDoor.set(0.3);

Timer.delay(1.0);

shooterMotor01.set(0.0);

shooterMotor01.set(0.0);

autoLooping = false;

}

//robot drives forward, and shoots the ball into the one point goal

//drive(0.0, 0.0);

//pull shooter door inwards to push ball into the spinning shooter wheels

//intakeDoor.set(-0.3);

//Timer.delay(1.0);

//intakeDoor.set(0.0);

}

public void teleopInit() {

//resets the encoders

leftEncoder.reset();

rightEncoder.reset();

//encoders can start reading ticks now

leftEncoder.start();

rightEncoder.start();

//sets the distance per pulse of each encoder: the wheel's circumference divided by the number of pulses per rotation

leftEncoder.setDistancePerPulse(0.03490659);

rightEncoder.setDistancePerPulse(0.03490659);

setSafetyEnabled(true);

}

public void teleopPeriodic() {

tankDrive(); //left and right joysticks control the drivetrain

shooterMotor01.set(controller.getRawAxis(3));

shooterMotor02.set(controller.getRawAxis(3));

intakeDoor.set(-controller.getRawAxis(5));

if (controller.getRawButton(1)) {

double time = Timer.getFPGATimestamp();

shooterMotor01.set(1);

shooterMotor02.set(1);

while ((Timer.getFPGATimestamp() - time <= 3) && (controller.getRawButton(1))) {

//do nothing

}

intakeDoor.set(-0.3);

time = Timer.getFPGATimestamp();

while ((Timer.getFPGATimestamp() - time <= 2) && (controller.getRawButton(1))) {

//do nothing

}

intakeDoor.set(0.0);

}

if (controller.getRawButton(2)) {

double time = Timer.getFPGATimestamp();

intakeDoor.set(0.3);

intakeWheels.set(1.0);

while ((Timer.getFPGATimestamp() - time <= 2) && (controller.getRawButton(2))) {

//do nothing

}

intakeDoor.set(0.0);

intakeWheels.set(0.0);

}

/\*if (controller.getRawAxis(2) > 0.25) {

shooterHoodPiston01.set(Relay.Value.kReverse);

}

else if (controller.getRawAxis(2) < -0.25) {

shooterHoodPiston02.set(Relay.Value.kForward);

} else {

shooterHoodPiston01.set(Relay.Value.kOff);

shooterHoodPiston02.set(Relay.Value.kOff);

}\*/

if (controller.getRawButton(5)) {

intakeWheels.set(1.0);

}

else if (controller.getRawButton(6)) {

intakeWheels.set(-1.0);

} else {

intakeWheels.set(0.0);

}

SmartDashboard.putNumber("Left Encoder", leftEncoder.get());

SmartDashboard.putNumber("Right Encoder", -rightEncoder.get());

}

/\*\*

\* This method causes the drivetrain to move forward until the set number of ticks is reached for both encoders.

\*

\* @param ticks The number of ticks the encoders must read until true is returned.

\* @return Whether or not the encoders turned enough.

\*/

public boolean autonMove(int ticks) {

double left = leftEncoder.get();

double right = -rightEncoder.get();

SmartDashboard.putNumber("Left Encoder", left);

SmartDashboard.putNumber("Right Encoder", right);

if ((left > ticks) && (right > ticks)) { //reached destination

drive(0.0, 0.0);

return true;

} else {

if (left > right) { //turning right

if (turningLeft) {

autonTurnLoops = 0;

}

turningRight = true;

turningLeft = false;

autonTurnLoops++;

drive(0.5, 0.5 - autonTurnLoops \* 0.01);

}

else if (right > left) { //turning left

if (turningRight) {

autonTurnLoops = 0;

}

turningRight = false;

turningLeft = true;

drive(0.5 - autonTurnLoops \* 0.01, 0.5);

} else { //driving straight

autonTurnLoops = 0;

drive(0.5, 0.5);

}

return false;

}

}

/\*\*

\* This method allows the robot to turn left or right during autonomous.

\*

\* @param leftRight Turning the robot left is true, right is false.

\* @param angleToMove The angle in degrees the robot needs to turn.

\* @return Whether or not the robot is done turning.

\*/

public boolean rotate(boolean leftRight, int angleToMove) {

int left = leftEncoder.get();

int right = -rightEncoder.get();

int ticks = (int) (3.7 \* 24685.7143 / angleToMove);

if (!leftRight) { //turning right

drive(0.5, -0.5);

return ((left > ticks) && (right < -ticks));

} else { //turning left

drive(-0.5, 0.5);

return ((left < -ticks) && (right > ticks));

}

}

//six motor drive methods

/\*\*

\* This method ensures that the robot is feeding the watchdog when the robot drive/

\*

\* @param safety This is the mode the robot is set for: true for teleop and false for auton.

\*/

public void setSafetyEnabled(boolean safety) {

if (safety) {

getWatchdog().setExpiration(0.1);

}

getWatchdog().setEnabled(safety);

}

/\*\*

\* This method allows the robot to drive during auton.

\*

\* @param speedLeft The desired speed for the left wheels.

\* @param speedRight The desired speed for the right wheels.

\*/

public void drive(double speedLeft, double speedRight) {

motorLeftRear.set(-speedLeft);

motorLeftCenter.set(-speedLeft);

motorLeftFront.set(-speedLeft);

motorRightRear.set(speedRight);

motorRightCenter.set(speedRight);

motorRightFront.set(speedRight);

}

/\*\*

\* The method gives the driver control over the drivetrain.

\*/

public void tankDrive() {

motorLeftRear.set(leftJoy.getY());

motorLeftCenter.set(leftJoy.getY());

motorLeftFront.set(leftJoy.getY());

motorRightRear.set(-rightJoy.getY());

motorRightCenter.set(-rightJoy.getY());

motorRightFront.set(-rightJoy.getY());

SmartDashboard.putNumber("Left Motors", leftJoy.getY());

SmartDashboard.putNumber("Right Motors", rightJoy.getY());

getWatchdog().feed();

}

}