//non robot includes

#include <time.h>

#include <iostream>

#include <memory>

#include <string>

#include <cmath>

//robot related includes

#include <IterativeRobot.h>

#include <LiveWindow/LiveWindow.h>

//#include <SmartDashboard/SendableChooser.h>

//#include <SmartDashboard/SmartDashboard.h>

#include "WPILib.h"

#include <PowerDistributionPanel.h>

#include "ctre/Phoenix.h"

#include "time.h"

class Robot: public frc::IterativeRobot {

WPI\_TalonSRX \*l1 = new WPI\_TalonSRX(1);

WPI\_TalonSRX \*l2 = new WPI\_TalonSRX(2);

WPI\_TalonSRX \*r1 = new WPI\_TalonSRX(3);

WPI\_TalonSRX \*r2 = new WPI\_TalonSRX(4);

//bag is the one with the encoder on it

WPI\_TalonSRX \*encoder = new WPI\_TalonSRX(5);

ADXRS450\_Gyro gyro;

BuiltInAccelerometer accelerometer;

public:

Joystick \*joy;

PowerDistributionPanel \*pdp;

//static const float kP = 0.03;

std::shared\_ptr<NetworkTable> roboRealm;

static constexpr int kUltrasonicPort = 1;

static constexpr int kValueToInches = .125;

cs::UsbCamera camera;

bool seen = false;

bool turned = false;

double xAcceleration; //acceleration on the x-axis

double yAcceleration; //acceleration on the y-axis

double zAcceleration; //acceleration on the z-axis

double previousX = 0; //Previous recursive average on x-axis

double previousY = 0; //Previous recursive average on y-axis

double previousZ = 1; //Previous recursive average on z-axis

const double kUpdatePeriod = 0.005; // 5milliseconds / 0.005 seconds.

Robot()

{

joy = new Joystick(1);

pdp = new PowerDistributionPanel();

}

void RobotInit()

{

cs::UsbCamera camera = CameraServer::GetInstance()->StartAutomaticCapture();

encoder->SetSelectedSensorPosition(0,0,0);

SmartDashboard::PutString("DB/String 0", "--");

SmartDashboard::PutString("DB/String 1", "--");

SmartDashboard::PutString("DB/String 2", "--");

SmartDashboard::PutString("DB/String 3", "--");

SmartDashboard::PutString("DB/String 4", "--");

SmartDashboard::PutString("DB/String 5", "--");

SmartDashboard::PutString("DB/String 6", "--");

}

void AutonomousInit() {

//clears dashboard

SmartDashboard::PutString("DB/String 0", "--");

SmartDashboard::PutString("DB/String 1", "--");

SmartDashboard::PutString("DB/String 2", "--");

SmartDashboard::PutString("DB/String 3", "--");

SmartDashboard::PutString("DB/String 4", "--");

SmartDashboard::PutString("DB/String 5", "--");

SmartDashboard::PutString("DB/String 6", "--");

//grabs value of COG\_X from roborealm

double xPosition = SmartDashboard::GetNumber("COG\_X", -1);

gyro.Reset();

printf("Angle: %f", gyro.GetAngle());

static const float kP = 0.03;

TankDrive(.5,.5);

Wait(1);

TankDrive(0,0);

//VisionTracking();

}

void AutonomousPeriodic(){

printf("Angle: %f", gyro.GetAngle());

if(std::abs(gyro.GetAngle()) < 90)

{

TankDrive(-.5,.5);

SmartDashboard::PutString("DB/String 6", std::to\_string(gyro.GetAngle()));

}

else if(std::abs(gyro.GetAngle()) > 90)

{

TankDrive(0,0);

}

}

void TeleopInit(){

encoder->SetSelectedSensorPosition(0,0,0);

}

void TeleopPeriodic() {

// cs::UsbCamera camera = CameraServer::GetInstance()->StartAutomaticCapture();

if(joy->GetRawButton(3)) {

xAcceleration = accelerometer.GetX(); //returns x-axis accel

yAcceleration = accelerometer.GetY(); //returns y-axis accel

zAcceleration = accelerometer.GetZ(); //returns z-axis accel

SmartDashboard::PutString("DB/String 7:", std::to\_string(xAcceleration));

SmartDashboard::PutString("DB/String 8:", std::to\_string(yAcceleration));

SmartDashboard::PutString("DB/String 7:", std::to\_string(zAcceleration));

SmartDashboard::PutNumber("Recursive X-Axis Average:", ((xAcceleration\*0.1) + (0.9\*previousX)));

//returns a recursive average for the x-axis

SmartDashboard::PutNumber("Recursive Y-Axis Average:", ((yAcceleration\*0.1) + (0.9\*previousY)));

//returns a recursive average for the y-axis

SmartDashboard::PutNumber("Recursive Z-Axis Average:", ((zAcceleration\*0.1) + (0.9\*previousZ)));

//returns a recursive average for the z-axis

previousX = (xAcceleration\*0.1) + (0.9\*previousX);

previousY = (yAcceleration\*0.1) + (0.9\*previousY);

previousZ = (zAcceleration\*0.1) + (0.9\*previousZ);

printf("A: %f B: %f C: %f\n" ,xAcceleration,yAcceleration,zAcceleration);

Wait(kUpdatePeriod); // Wait a short bit before updating again

}

int pulseWidthPos = encoder->GetSelectedSensorPosition(0);

encoder->SetSelectedSensorPosition(0,0,0);

SmartDashboard::PutString("DB/String 0", std::to\_string(encoder->GetSelectedSensorPosition(0)));

printf("PWP: %i\n", pulseWidthPos);

if(joy->GetRawButton(4))

{

encoder->Set(.1);

}

if ( joy->GetRawButton(1)){

while (encoder->GetSelectedSensorPosition(0) < 2133)

{

encoder->Set(.1);

SmartDashboard::PutString("DB/String 5", std::to\_string(encoder->GetSelectedSensorPosition(0)));

}

encoder->Set(0);

encoder->SetSelectedSensorPosition(0,0,0);

}

if ( joy->GetRawButton(2)){

while (encoder->GetSelectedSensorPosition(0) > -2133)

{

encoder->Set(-.1);

SmartDashboard::PutString("DB/String 6", std::to\_string(encoder->GetSelectedSensorPosition(0)));

}

encoder->Set(0);

encoder->SetSelectedSensorPosition(0,0,0);

}

if(std::abs(joy->GetRawAxis(5) > .2 && std::abs(joy->GetRawAxis(1) < .2)))

{

TankDrive(0 ,joy->GetRawAxis(5));

//Reads the input voltage of the PDP

double pdpVin = pdp->GetVoltage();

printf("input voltage: %f \n", pdpVin);

//Reads the input amperage of the talons

double t1 = pdp->GetCurrent(12);

double t2 = pdp->GetCurrent(13);

double t3 = pdp->GetCurrent(14);

double t4 = pdp->GetCurrent(15);

printf("t1: %f; t2: %f; t3: %f, t4: %f \n", t1, t2, t3, t4);

}

else if(std::abs(joy->GetRawAxis(5)) < .2 && std::abs(joy->GetRawAxis(1)) > .2)

{

TankDrive(joy->GetRawAxis(1),0);

//Reads the input voltage of the PDP

double pdpVin = pdp->GetVoltage();

printf("input voltage: %f \n", pdpVin);

//Reads the input amperage of the talons

double t1 = pdp->GetCurrent(12);

double t2 = pdp->GetCurrent(13);

double t3 = pdp->GetCurrent(14);

double t4 = pdp->GetCurrent(15);

// printf("t1: %f; t2: %f; t3: %f, t4: %f \n", t1, t2, t3, t4);

}

else if(std::abs(joy->GetRawAxis(5)) > .2 && std::abs(joy->GetRawAxis(1)) > .2)

{

TankDrive( joy->GetRawAxis(1),joy->GetRawAxis(5));

//Reads the input voltage of the PDP

double pdpVin = pdp->GetVoltage();

printf("input voltage: %f \n", pdpVin);

//Reads the input amperage of the talons

double t1 = pdp->GetCurrent(12);

double t2 = pdp->GetCurrent(13);

double t3 = pdp->GetCurrent(14);

double t4 = pdp->GetCurrent(15);

// printf("t1: %f; t2: %f; t3: %f, t4: %f \n", t1, t2, t3, t4);

}

else

{

TankDrive(0,0);

}/\*

if(joy->GetRawButton(5))

{

bag->Set(1);

}

else if(joy->GetRawButton(6))

{

bag->Set(-1);

}

else

{

bag->Set(0);

}

\*/

}

void TankDrive(double left, double right)

{

l1->Set(-left);

l2->Set(-left);

r1->Set(right);

r2->Set(right);

}

void VisionTracking()

{

printf("Inside vision tracking\n");

frc::AnalogInput ultrasonic {kUltrasonicPort};

double currentDistance = ultrasonic.GetValue() \* .125;

double maxDistance = 6.75+3;

double minDistance = 6+3;

camera.SetResolution(640, 480);

double xPosition;

//reads the values if COG\_X and COG\_Y throught SmartDashboard and assings them to xPosition and yPosition

xPosition = SmartDashboard::GetNumber("COG\_X", -1);

//rotate based on reading of xPosition

printf("currentD: %f", currentDistance);

if(currentDistance > maxDistance)

{

TankDrive(.2,.2);

}

//move backwards if the range finder reads less than one foot

else if (currentDistance < minDistance)

{

TankDrive(-.2,-.2);

}

else

{

TankDrive(0,0);

}

if(xPosition == 0 && seen == false)

{

if(currentDistance > maxDistance)

{

TankDrive(.2,.2);

}

//move backwards if the range finder reads less than one foot

else if (currentDistance < minDistance)

{

TankDrive(-.2,-.2);

}

else

{

TankDrive(0,0);

}

}

else if(xPosition != 0)

{

seen = true;

if(xPosition > 90)

{

TankDrive(-.3,.3);

}

else if(xPosition < 75 && xPosition > 0)

{

TankDrive(.3,-.3);

}

else

{

TankDrive(0,0);

if(currentDistance > maxDistance)

{

TankDrive(.2,.2);

}

//move backwards if the range finder reads less than one foot

else if (currentDistance < minDistance)

{

TankDrive(-.2,-.2);

}

else

{

TankDrive(0,0);

}

}

}

}

};

START\_ROBOT\_CLASS(Robot)