//non robot includes

#include <iostream>

#include <memory>

#include <string>

//robot related includes

#include <IterativeRobot.h>

#include <LiveWindow/LiveWindow.h>

#include <SmartDashboard/SendableChooser.h>

#include <SmartDashboard/SmartDashboard.h>

#include "WPILib.h"

#include <CANTalon.h>

#include <PowerDistributionPanel.h>

class Robot: public frc::IterativeRobot

{

public:

//Drive train right side

CANTalon \*DTR1, \*DTR2, \*DTR3;

//Drive train left side

CANTalon \*DTL1, \*DTL2, \*DTL3;

//S is shooter I is intake

CANTalon \*S1, \*S2, \*I;

//Conveyer

CANTalon \*Con1;

//Climber

CANTalon \*Climb1, \*Climb2, \*Climb3;

//Driver controller

Joystick \*driver;

//Operator controller

Joystick \*op;

//Power Distributuon panel

PowerDistributionPanel \*pdp;

//connects networktables and roborealm

std::shared\_ptr<NetworkTable> roboRealm;

//declare for the camera as a USB

cs::UsbCamera camera;

//used for stopping some things from repeating in auton

bool seen = false;

bool turned = false;

//used for selectable auton

int auton = 1;

int autonMax = 5;

//values for range finder math

static constexpr int kUltrasonicPort = 1;

static constexpr int kValueToInches = .125;

Robot():roboRealm(NetworkTable::GetTable("SmartDashboard"))//puts the constructor into the scope of roborealm

{

//Sets the values in the daisy chain of the talons for the right side

DTR1 = new CANTalon(1);

DTR2 = new CANTalon(2);

DTR3 = new CANTalon(3);

//Sets the values in the daisy chain of the talons for the left side

DTL1 = new CANTalon(9);

DTL2 = new CANTalon(10);

DTL3 = new CANTalon(11);

//Sets the values in daisy chain of the talons in the shooter

S1 = new CANTalon(4);

S2 = new CANTalon(5);

//Sets the values in the daisy chain of the talons in the conveyer

Con1 = new CANTalon(6);

//Daisy chain for intake

I = new CANTalon(7);

//Daisy chain for climber

Climb1 = new CANTalon(8);

Climb2 = new CANTalon(12);

Climb3 = new CANTalon(13);

//Sets the port for the Driver controller to USB port 0

driver = new Joystick(1);

//Sets the port for the operator controller to USB port 1

op = new Joystick(2);

//Reader for amperage into the talons

pdp = new PowerDistributionPanel();

}

void RobotInit()

{

}

void TeleopPeriodic()

{

//talon number reading amperage from PDP port int ()

double t1 = pdp->GetCurrent(15);

double t2 = pdp->GetCurrent(14);

double t3 = pdp->GetCurrent(13);

double t4 = pdp->GetCurrent(12);

double t5 = pdp->GetCurrent(11);

double t6 = pdp->GetCurrent(10);

double t7 = pdp->GetCurrent(8);

double t8 = pdp->GetCurrent(7);

double t9 = pdp->GetCurrent(0);

double t10 = pdp->GetCurrent(1);

double t11 = pdp->GetCurrent(2);

double pdpVin = pdp->GetVoltage();

//parcing double values into strings for smartdashboard

auto str1 = std::to\_string(t1);

auto str2 = std::to\_string(t2);

auto str3 = std::to\_string(t3);

auto str4 = std::to\_string(t9);

auto str5 = std::to\_string(t10);

auto str6 = std::to\_string(t11);

auto str7 = std::to\_string(pdpVin);

//Sends the values of the y axis's of the tankDrive function

tankDrive(-driver->GetRawAxis(5),-driver->GetRawAxis(1));

//Y button to control intake

if(op->GetRawButton(4))

{

I->Set(-1);

}

else

{

I->Set(0);

}

//left stick y axis for floor intake

if(op->GetRawAxis(1) < -.2)

{

Con1->Set(1);

}

else if(op->GetRawAxis(1) > .2)

{

Con1->Set(-1);

}

else

{

Con1->Set(0);

}

//A button for value readings of talons 4-9

if(op->GetRawButton(1))

{

printf("t4: %f, t5: %f, t6: %f \n", t4, t5, t6);

printf("t7: %f, t8: %f \n", t7, t8);

}

//left trigger for climber

if(op->GetRawButton(5))

{

Climb1->Set(1);

Climb2->Set(1);

Climb3->Set(1);

}

else

{

Climb1->Set(0);

Climb2->Set(0);

Climb3->Set(0);

}

//right trigger for max speed shooter&intake, b for partial shooter, right stick for variable

if(op->GetRawButton(6))

{

S1->Set(-1);

S2->Set(1);

}

else if(op->GetRawButton(2))

{

S1->Set(-.45);

S2->Set(.45);

}

else if(op->GetRawAxis(6) < -.1)

{

S1->Set(op->GetRawAxis(6));

S1->Set(-op->GetRawAxis(6));

}

else

{

S1->Set(0);

S2->Set(0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Writes amperage to talons to Dashboard\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

SmartDashboard::PutString("DB/String 5", "Right drive train");

SmartDashboard::PutString("DB/String 0", "Left drive train");

SmartDashboard::PutString("DB/String 6",str4);

SmartDashboard::PutString("DB/String 7",str5);

SmartDashboard::PutString("DB/String 8",str6);

SmartDashboard::PutString("DB/String 1",str1);

SmartDashboard::PutString("DB/String 2",str2);

SmartDashboard::PutString("DB/String 3",str3);

SmartDashboard::PutString("DB/String 9",str7);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

}

void DisabledPeriodic()

{

//bumpers for cycling auton mode

if(op->GetRawButton(5))

{

auton--;

Wait(.2);

}

else if(op->GetRawButton(6))

{

auton++;

Wait(.2);

}

//reset auton number if out of range

if(auton == 7)

{

auton = 1;

}

else if(auton == 0)

{

auton = 6;

}

//display auton mode to SMB

if(auton == 1)

{

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

}

else if(auton == 2)

{

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

}

else if(auton == 3)

{

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

}

else if(auton == 4)

{

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

}

else if(auton == 5)

{

SmartDashboard::PutString("DB/String 0", "right shooter that is unlikely to work");

}

else if(auton == 6)

{

SmartDashboard::PutString("DB/String 0", "left shooter that is unlikely to work");

}

turned = false;

}

void AutonomousPeriodic()

{

//clears dashboard

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

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

//grabs value of COG\_X from roborealm

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

if(auton == 1)

{

//runs the vision tracking program

printf("Inside Auton Middle run. xPos: %f\n", xPosition);

//executes the VisionTracking function

VisionTracking();

}

else if(auton == 2)

{

//used to make sure this section executes once

if(turned == false)

{

//go forward and turn right

printf("Inside Auton Left run. xPos: %f\n", xPosition);

tankDrive(.35,.35);

Wait(.2);

printf("xPos: %f\n",xPosition);

tankDrive(-.4,.4);

Wait(.5);

}

turned = true;

VisionTracking();

}

else if (auton == 3)

{

if(turned == false)

{

printf("Inside Auton Right run. xPos: %f\n", xPosition);

tankDrive(.35,.35);

Wait(.2);

printf("xPos: %f\n",xPosition);

tankDrive(.4,-.4);

Wait(.3);

}

turned = true;

VisionTracking();

}

else if(auton == 4)

{

if(turned == false)

{

tankDrive(.4,.4);

Wait(3);

tankDrive(0,0);

}

turned = true;

}

else if(auton == 5)

{

if(turned == true)

{

tankDrive(.4,.4);

Wait(.5);

tankDrive(.4,-.4);

Wait(.5);

tankDrive(.4,.4);

Wait(3);

tankDrive(0,0);

S1->Set(-.65);

S2->Set(.65);

Con1->Set(1);

Wait(3);

S1->Set(0);

S2->Set(0);

Con1->Set(0);

}

turned = true;

}

else if(auton == 6)

{

if(turned == true)

{

tankDrive(.4,.4);

Wait(.5);

tankDrive(-.4,.4);

Wait(.5);

tankDrive(.4,.4);

Wait(3);

tankDrive(0,0);

S1->Set(-.65);

S2->Set(.65);

Con1->Set(1);

Wait(3);

S1->Set(0);

S2->Set(0);

Con1->Set(0);

}

turned = true;

}

}

void tankDrive(double left, double right)

{

//Assigns left side of the drive train to the y axis of the left joy stick of the X-box controller

DTL1->Set(-left);

DTL2->Set(-left);

DTL3->Set(-left);

//Assigns right side of the drive train to the y axis of the right joy stick of the X-box controller

DTR1->Set(right);

DTR2->Set(right);

DTR3->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)