//non robot includes

#include <iostream>

#include <memory>

#include <string>

#include <cmath>

#include <cstring>

//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 "Drive/DifferentialDrive.h"

#include "DriverStation.h"

Compressor \*compressor;

BuiltInAccelerometer accelerometer;

ADXRS450\_Gyro gyro;

class Robot: public frc::IterativeRobot {

public:

//Drive Train

WPI\_TalonSRX l1 {32};

WPI\_TalonSRX l2 {38};

WPI\_TalonSRX l3 {39};

WPI\_TalonSRX r1 {33};

WPI\_TalonSRX r2 {34};

WPI\_TalonSRX r3 {35};

WPI\_TalonSRX arm1 {36};

WPI\_TalonSRX arm2 {37};

WPI\_TalonSRX w1 {41};

WPI\_TalonSRX w2 {30};

WPI\_TalonSRX w3 {31};

WPI\_TalonSRX m1 {40};

WPI\_TalonSRX m2 {42};

frc::Joystick joy {0};

frc::Joystick OP {1};

DoubleSolenoid \*left, \*right;

frc::PowerDistributionPanel pdp {0};

std::shared\_ptr<NetworkTable> roboRealm;

Encoder \*enc;

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

static constexpr int kUltrasonicPort = 1;

static constexpr int kValueToInches = .125;

cs::UsbCamera camera;

bool seen = false;

bool turned = false;

int auton = 0;

int autonMax = 2;

bool delay = false;

Robot(){

compressor = new Compressor(0);

right = new DoubleSolenoid(2,3);

left = new DoubleSolenoid(0,1);

enc = new Encoder(0, 1, false, Encoder::EncodingType::k4X);

}

void RobotInit() /\*: accelerometer(Accelerometer::Range::kRange\_8G)\*/{

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

compressor->SetClosedLoopControl(true);

arm1.SetSelectedSensorPosition(0,0,0);

}

void AutonomousInit()

{

enc->SetMaxPeriod(.1);

enc->SetMinRate(10);

enc->SetDistancePerPulse(5);

enc->SetReverseDirection(true);

enc->SetSamplesToAverage(7);

enc->Reset();

std::string gameData;

gameData = frc::DriverStation::GetInstance().GetGameSpecificMessage();

std::string startPos = std::to\_string(gameData[0]);

double distance;

TankDrive(0,0);

Autonomous(startPos);

}

void DisabledInit()

{

arm1.SetSelectedSensorPosition(0,0,0);

delay = false;

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

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

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

}

void DisabledPeriodic()

{

SmartDashboard::PutString("DB/String 0", std::to\_string(arm1.GetSelectedSensorPosition(0)));

if(joy.GetRawButton(1))

{

delay = true;

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

}

if(joy.GetRawButton(6))

{

auton++;

Wait(.2);

if(auton == autonMax + 1)

{

auton = 0;

}

else if(auton == -1)

{

auton = 2;

}

}

else if(joy.GetRawButton(5))

{

auton--;

Wait(.2);

if(auton == autonMax + 1)

{

auton = 0;

}

else if(auton == -1)

{

auton = 2;

}

}

auto str = std::to\_string(auton);

SmartDashboard::PutString("DB/String 5","Auton mode: " + str);

std::string gameData;

gameData = frc::DriverStation::GetInstance().GetGameSpecificMessage();

std::string startPos = std::to\_string(gameData[0]);

if(gameData[0] = 'L')

{

if(auton == 0)

{

SmartDashboard::PutString("DB/String 6","Friendly switch left side");

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

}

else if(auton == 1)

{

SmartDashboard::PutString("DB/String 6","Friendly switch left side");

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

}

else if(auton == 2)

{

SmartDashboard::PutString("DB/String 6","Friendly switch left side");

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

}

}

else if(gameData[0] = 'R')

{

if(auton == 0)

{

SmartDashboard::PutString("DB/String 6","Friendly switch right side");

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

}

else if(auton == 1)

{

SmartDashboard::PutString("DB/String 6","Friendly switch right side");

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

}

else if(auton == 2)

{

SmartDashboard::PutString("DB/String 6","Friendly switch right side");

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

}

}

/\*if(auton == 0)

{

SmartDashboard::PutString("DB/String 6","Friendly switch left side");

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

}

else if(auton == 1)

{

SmartDashboard::PutString("DB/String 6","Friendly switch left side");

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

}

else if(auton == 2)

{

SmartDashboard::PutString("DB/String 6","Friendly switch left side");

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

}

else if(auton == 3)

{

SmartDashboard::PutString("DB/String 6","Friendly switch right side");

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

}

else if(auton == 4)

{

SmartDashboard::PutString("DB/String 6","Friendly switch right side");

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

}

else if(auton == 5)

{

SmartDashboard::PutString("DB/String 6","Friendly switch right side");

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

}\*/

}

void TeleopInit()

{

arm1.SetSelectedSensorPosition(0,0,0);

}

void TeleopPeriodic() {

SmartDashboard::PutString("DB/String 0", std::to\_string(arm1.GetSelectedSensorPosition(0)));

SmartDashboard::PutString("DB/String 9", std::to\_string(arm1.GetSelectedSensorPosition(0)));

if(OP.GetRawAxis(2) > .2)

{

arm1.SetSelectedSensorPosition(0,0,0);

while(arm1.GetSelectedSensorPosition(0) < 572 & arm1.GetSelectedSensorPosition(0) < 762)

{

arm1.Set(1);

arm2.Set(1);

}

while(arm1.GetSelectedSensorPosition(0) < 762 & arm1.GetSelectedSensorPosition(0) < 1000)

{

arm1.Set(.2);

arm2.Set(.2);

}

}

else if(OP.GetRawAxis(3) > .2)

{

arm1.SetSelectedSensorPosition(0,0,0);

while(arm1.GetSelectedSensorPosition(0) < 572 & arm1.GetSelectedSensorPosition(0) < 762)

{

arm1.Set(-1);

arm2.Set(-1);

}

while(arm1.GetSelectedSensorPosition(0) < 762 & arm1.GetSelectedSensorPosition(0) < 1000)

{

arm1.Set(-.2);

arm2.Set(-.2);

}

}

else

{

arm1.Set(0);

arm2.Set(0);

}

/\*if(OP.GetRawButton(1))

{

w1.Set(.2);

w2.Set(.2);

w3.Set(.2);

}

else if(OP.GetRawButton(2))

{

w1.Set(-.2);

w2.Set(-.2);

w3.Set(-.2);

}

else

{

w1.Set(0);

w2.Set(0);

w3.Set(0);

}\*/

SmartDashboard::PutString("DB/String 0", std::to\_string(pdp.GetCurrent(4)));

SmartDashboard::PutString("DB/String 1", std::to\_string(pdp.GetCurrent(5)));

SmartDashboard::PutString("DB/String 2", std::to\_string(pdp.GetCurrent(6)));

SmartDashboard::PutString("DB/String 3", std::to\_string(pdp.GetCurrent(7)));

SmartDashboard::PutString("DB/String 4", std::to\_string(pdp.GetCurrent(8)));

SmartDashboard::PutString("DB/String 5", std::to\_string(pdp.GetCurrent(9)));

SmartDashboard::PutString("DB/String 6", std::to\_string(pdp.GetCurrent(10)));

SmartDashboard::PutString("DB/String 7", std::to\_string(pdp.GetCurrent(11)));

if(OP.GetRawButton(4))

{

m1.Set(1);

m2.Set(-1);

}

else if(OP.GetRawButton(3))

{

m1.Set(-1);

m2.Set(1);

}

else if(OP.GetRawButton(1))

{

m1.Set(1);

m2.Set(1);

}

else if(OP.GetRawButton(2))

{

m1.Set(-1);

m2.Set(-1);

}

else

{

m1.Set(0);

m2.Set(0);

}

if(OP.GetRawAxis(1) < -.2)

{

right->Set(DoubleSolenoid::kForward);

left->Set(DoubleSolenoid::kForward);

}

else if(OP.GetRawAxis(1) > .2)

{

left->Set(DoubleSolenoid::kReverse);

right->Set(DoubleSolenoid::kReverse);

}

//1,2, 12, 13, 14, 15

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

{

TankDrive(0, joy.GetRawAxis(5));//drive the right side only

//Reads the input voltage of the PDP

/\* double pdpVin = pdp.GetVoltage();

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

\*/ }

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

{

TankDrive(joy.GetRawAxis(1),0);//drive the left side only

//Reads the input voltage of the PDP

/\* double pdpVin = pdp.GetVoltage();

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

\*/

}

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

{

TankDrive(joy.GetRawAxis(1),joy.GetRawAxis(5)); //drive both sides

//Reads the input voltage of the PDP

/\* double pdpVin = pdp.GetVoltage();

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

\*/

}

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

{

TankDrive(joy.GetRawAxis(1),joy.GetRawAxis(5)); //drive both sides

//Reads the input voltage of the PDP

/\* double pdpVin = pdp.GetVoltage();

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

\*/

}

else

{

TankDrive(0,0);

}

}

void TankDrive(double left, double right)

{

l1.Set(-left);

l2.Set(-left);

l3.Set(-left);

r1.Set(right);

r2.Set(right);

r3.Set(right);

/\*

SmartDashboard::PutString("DB/String 0", std::to\_string(pdp.GetCurrent(2)));//38

//SmartDashboard::PutString("DB/String 1", std::to\_string(pdp.GetCurrent(3)));//39

SmartDashboard::PutString("DB/String 2", std::to\_string(pdp.GetCurrent(12)));//32

SmartDashboard::PutString("DB/String 3", std::to\_string(pdp.GetCurrent(13)));//33

SmartDashboard::PutString("DB/String 4", std::to\_string(pdp.GetCurrent(14)));//34

SmartDashboard::PutString("DB/String 9", std::to\_string(pdp.GetCurrent(15)));//35\*/

}

void Autonomous(std::string lightPos)

{

std::string gameData;

gameData = frc::DriverStation::GetInstance().GetGameSpecificMessage();

std::string startPos = std::to\_string(gameData[0]);

Wait(1);

if(gameData[0] = 'L')

{

if(auton = 0)

{

arm1.SetSelectedSensorPosition(0,0,0);

//while encoder val is less than 8 full rotations move forward

while(enc->GetDistance() < 138)

{

TankDrive(-.5,-.5);

}

while(arm1.GetSelectedSensorPosition(0) < 572/\*test value\*/ & arm1.GetSelectedSensorPosition(0) < 762)

{

arm1.Set(-1);

arm2.Set(-1);

}

while(arm1.GetSelectedSensorPosition(0) < 762 & arm1.GetSelectedSensorPosition(0) < 1000)

{

arm1.Set(-.2);

arm2.Set(-.2);

m1.Set(1);

m2.Set(-1);

}

arm1.Set(.1);

arm2.Set(.1);

Wait(.5);

arm1.Set(0);

arm2.Set(0);

m1.Set(0);

m2.Set(0);

}

else if(auton = 1)

{

while(enc->GetDistance() < 30)

{

TankDrive(-.5,-.5);

}

enc->Reset();

gyro.Reset();

while(gyro.GetAngle() < 90)

{

TankDrive(.5,-.5);

}

TankDrive(0,0);

while(enc->GetDistance() < 24)

{

TankDrive(-.5,-.5);

}

TankDrive(0,0);

enc->Reset();

gyro.Reset();

while(gyro.GetAngle() > -90)

{

TankDrive(-.5,.5);

}

TankDrive(0,0);

enc->Reset();

while(enc->GetDistance() < 120)

{

TankDrive(-.5,-.5);

}

TankDrive(0,0);

while(arm1.GetSelectedSensorPosition(0) < 572/\*test value\*/ & arm1.GetSelectedSensorPosition(0) < 762)

{

arm1.Set(-1);

arm2.Set(-1);

}

while(arm1.GetSelectedSensorPosition(0) < 762 & arm1.GetSelectedSensorPosition(0) < 1000)

{

arm1.Set(-.2);

arm2.Set(-.2);

m1.Set(1);

m2.Set(-1);

}

arm1.Set(.1);

arm2.Set(.1);

Wait(.5);

arm1.Set(0);

arm2.Set(0);

m1.Set(0);

m2.Set(0);

}

else if(auton = 2)

{

SmartDashboard::PutString("DB/String 1", std::to\_string(enc->GetDistance()));

while(enc->GetDistance() < 30)

{

TankDrive(-.5,-.5);

SmartDashboard::PutString("DB/String 1", std::to\_string(enc->GetDistance()));

}

gyro.Reset();

while(gyro.GetAngle() > -90)

{

TankDrive(-.5,.5);

SmartDashboard::PutString("DB/String 0", std::to\_string(gyro.GetAngle()));

}

enc->Reset();

while(enc->GetDistance() < 30)

{

TankDrive(-.5,-.5);

SmartDashboard::PutString("DB/String 0", std::to\_string(gyro.GetAngle()));

}

gyro.Reset();

while(gyro.GetAngle() > -90)

{

TankDrive(.5,-.5);

}

while(enc->GetDistance() < 108)

{

TankDrive(-.5,-.5);

}

TankDrive(0,0);

while(arm1.GetSelectedSensorPosition(0) < 572/\*test value\*/ & arm1.GetSelectedSensorPosition(0) < 762)

{

arm1.Set(-1);

arm2.Set(-1);

}

while(arm1.GetSelectedSensorPosition(0) < 762 & arm1.GetSelectedSensorPosition(0) < 1000)

{

arm1.Set(-.2);

arm2.Set(-.2);

m1.Set(1);

m2.Set(-1);

}

arm1.Set(.1);

arm2.Set(.1);

Wait(.5);

arm1.Set(0);

arm2.Set(0);

m1.Set(0);

m2.Set(0);

}

}

else if(gameData[0] = 'R')

{

if(auton = 0)

{

enc->Reset();

while(enc->GetDistance() < 30)

{

TankDrive(-.5,-.5);

}

gyro.Reset();

while(gyro.GetAngle() < 90)

{

TankDrive(.5,-.5);

}

enc->Reset();

while(enc->GetDistance() < 30)

{

TankDrive(-.5,-.5);

}

gyro.Reset();

while(gyro.GetAngle() > 90)

{

TankDrive(-.5,.5);

}

while(enc->GetDistance() < 108)

{

TankDrive(-.5,-.5);

}

TankDrive(0,0);

while(arm1.GetSelectedSensorPosition(0) < 572/\*test value\*/ & arm1.GetSelectedSensorPosition(0) < 762)

{

arm1.Set(-1);

arm2.Set(-1);

}

while(arm1.GetSelectedSensorPosition(0) < 762 & arm1.GetSelectedSensorPosition(0) < 1000)

{

arm1.Set(-.2);

arm2.Set(-.2);

m1.Set(1);

m2.Set(-1);

}

arm1.Set(.1);

arm2.Set(.1);

Wait(.5);

arm1.Set(0);

arm2.Set(0);

m1.Set(0);

m2.Set(0);

}

else if(auton = 1)

{

while(enc->GetDistance() < 30)

{

TankDrive(-.5,-.5);

}

enc->Reset();

gyro.Reset();

while(gyro.GetAngle() > -90)

{

TankDrive(-.5,.5);

}

TankDrive(0,0);

while(enc->GetDistance() < 24)

{

TankDrive(-.5,-.5);

}

TankDrive(0,0);

enc->Reset();

gyro.Reset();

while(gyro.GetAngle() < -90)

{

TankDrive(.5,-.5);

}

TankDrive(0,0);

enc->Reset();

while(enc->GetDistance() < 120)

{

TankDrive(-.5,-.5);

}

TankDrive(0,0);

while(arm1.GetSelectedSensorPosition(0) < 572/\*test value\*/ & arm1.GetSelectedSensorPosition(0) < 762)

{

arm1.Set(-1);

arm2.Set(-1);

}

while(arm1.GetSelectedSensorPosition(0) < 762 & arm1.GetSelectedSensorPosition(0) < 1000)

{

arm1.Set(-.2);

arm2.Set(-.2);

m1.Set(1);

m2.Set(-1);

}

arm1.Set(.1);

arm2.Set(.1);

Wait(.5);

arm1.Set(0);

arm2.Set(0);

m1.Set(0);

m2.Set(0);

}

else if(auton = 2)

{

arm1.SetSelectedSensorPosition(0,0,0);

//while encoder val is less than 8 full rotations move forward

while(enc->GetDistance() < 138)

{

TankDrive(-.5,-.5);

}

while(arm1.GetSelectedSensorPosition(0) < 572/\*test value\*/ & arm1.GetSelectedSensorPosition(0) < 762)

{

arm1.Set(-1);

arm2.Set(-1);

}

while(arm1.GetSelectedSensorPosition(0) < 762 & arm1.GetSelectedSensorPosition(0) < 1000)

{

arm1.Set(-.2);

arm2.Set(-.2);

m1.Set(1);

m2.Set(-1);

}

arm1.Set(.1);

arm2.Set(.1);

Wait(.5);

arm1.Set(0);

arm2.Set(0);

m1.Set(0);

m2.Set(0);

}

}

}

};

START\_ROBOT\_CLASS(Robot)

18