//////////////////////////////////////////////////////////////Robot-config.h

vex::brain Brain;

vex::controller Controller = vex::controller();

vex::digital\_out Lift = vex::digital\_out(Brain.ThreeWirePort.A);

vex::digital\_out Trans = vex::digital\_out(Brain.ThreeWirePort.B);

vex::digital\_in IR = vex::digital\_in(Brain.ThreeWirePort.C);

vex::digital\_in CapIR = vex::digital\_in(Brain.ThreeWirePort.D);

vex::limit BallLoaded = vex::limit(Brain.ThreeWirePort.E);

vex::line RLine = vex::line(Brain.ThreeWirePort.F);

vex::line LLine = vex::line(Brain.ThreeWirePort.G);

vex::gyro Gyro = vex::gyro(Brain.ThreeWirePort.H);

vex::motor Roll = vex::motor(vex::PORT11,vex::gearSetting::ratio18\_1,true);

vex::motor LF = vex::motor(vex::PORT10,vex::gearSetting::ratio18\_1,true);

vex::motor LM = vex::motor(vex::PORT7,vex::gearSetting::ratio18\_1,true);

vex::motor LB = vex::motor(vex::PORT4,vex::gearSetting::ratio18\_1,false);

vex::motor RF = vex::motor(vex::PORT6,vex::gearSetting::ratio18\_1,false);

vex::motor RM = vex::motor(vex::PORT5,vex::gearSetting::ratio18\_1,false);

vex::motor RB = vex::motor(vex::PORT2,vex::gearSetting::ratio18\_1,true);

vex::motor FlyL = vex::motor(vex::PORT18,vex::gearSetting::ratio18\_1,true);

vex::motor FlyR = vex::motor(vex::PORT13,vex::gearSetting::ratio18\_1,false);

vex::motor CapFlip = vex::motor(vex::PORT19,vex::gearSetting::ratio36\_1,true);

vex::motor BallFeed = vex::motor(vex::PORT15,vex::gearSetting::ratio18\_1,true);

vex::motor Turret = vex::motor(vex::PORT8,vex::gearSetting::ratio6\_1,false);

vex::motor Roller2 = vex::motor(vex::PORT14,vex::gearSetting::ratio18\_1,true);

//#define DEBUG 1

vex::competition Competition;

using namespace std;

char TrackColor = 'R';

char Alliance = 'R';

int AutoNumber = 1;//76 red 106 blue

float RPMGoal = 114,rpm = 0, rpmError = 100.0, TurnDiff = 0, TempHeight = 0, TempWidth = 0, TurnDir = 1, TempXDist = 200, GlobalFlagOffset = 160;

int fly = 4, R = 4, Tran = 0,LiftVar=2, Shoot = 0, SnapToFlag = 0, AutoRunning = 0, FlipCount,TurretRunning=0;//flywheel//Roller//Transmission

int MoveReturn=0;

int T1 = 0, T3 = 0;

float avgSpeed = 0;

float avgError = 0;

float GlobalGyro = 0;

int FlagYDim=0;

double AutoRPM=0;

int Color=1;

float FinalObject=20,TempObject=0;

float GLOBALP=0.7,GLOBALI=0.000001,GLOBALD=4.1;

//float GLOBALP=1.4,GLOBALI=0.0000001,GLOBALD=8.1;

#define bL2 Controller.ButtonL2.pressing()

#define bL1 Controller.ButtonL1.pressing()

#define bA Controller.ButtonA.pressing()

#define bR2 Controller.ButtonR2.pressing()

#define bR1 Controller.ButtonR1.pressing()

#define bB Controller.ButtonB.pressing()

#define bX Controller.ButtonX.pressing()

#define bY Controller.ButtonY.pressing()

#define bLeft Controller.ButtonLeft.pressing()

#define bRight Controller.ButtonRight.pressing()

#define bUp Controller.ButtonUp.pressing()

#define bDown Controller.ButtonDown.pressing()

#define ch3 Controller.Axis3.value()

#define ch4 Controller.Axis4.value()

#define ch1 Controller.Axis1.value()

#define ch2 Controller.Axis2.value()

#define wait vex::task::sleep

#ifndef NERD\_GYRO

#define NERD\_GYRO

float stdDev;

float avg;

float voltsPerDPS;

int gyroFlipped;

//ignore data within n standard deviations of no motion average

#define GYRO\_STD\_DEVS 3

#define GYRO\_OVERSAMPLE 1

//points or time in mSec that the gyro calibrates for

#define GYRO\_CALIBRATION\_POINTS 2000

float calibrationBuffer [GYRO\_CALIBRATION\_POINTS];

float gyroGetRate ();

/\*\*

\* generate calibration data for the gyro by collecting

\* zero movement data for reference when reading data later

\*

\* @param gyro instance of gyro structure

\*/

void gyroCalibrate (){

float rawAverage = 0.0;

float stdDev = 0.0;

//calculate average gyro reading with no motion

Gyro.startCalibration();

//vex::task::sleep(2000);

for(int i = 0; i < GYRO\_CALIBRATION\_POINTS; ++i){

float raw = Gyro.value(vex::rotationUnits::raw);

rawAverage += raw;

calibrationBuffer [i] = raw;

vex::task::sleep(1);

}

rawAverage /= GYRO\_CALIBRATION\_POINTS;

avg = rawAverage;

//calcuate the standard devation, or the average distance

//from the average on the data read

for (int i = 0; i < GYRO\_CALIBRATION\_POINTS; ++i)

stdDev += fabs (rawAverage - calibrationBuffer [i]);

stdDev /= (float) GYRO\_CALIBRATION\_POINTS;

stdDev = stdDev;

/\*

\* Datasheet from VEX indicates that the sensitivity of the gyro is 1.1mV/dps

\* and the cortex ADC for raw analog reads ranges from 0-4095 for 0v-5v

\* readings. The gyro is scaled from the nominal 2.7v-3.6v operating range

\* that the actual chip has to work on the cortex's 5v logic voltage. The scale multiplier

\* value is in the ballpark of 1.515.

\*/

voltsPerDPS = 0.0011 \* 1.515;

}

/\*\*

\* initialize gyro and run the calibration subroutine

\*

\* @param gyro instance of gyro structure

\* @param portNum the port number of the gyro

\*/

void gyroInit (int gyroFlipped) {

//gyro.portNum = portNum;

gyroFlipped = gyroFlipped;

gyroCalibrate ();

}

/\*\*

\* calculate filtered gyro rate data, ignoring anything within

\* GYRO\_STD\_DEVS standard deviations of the average gyro

\* rate value at zero motion

\*`

\* @param gyro instance of gyro structure

\*

\* @return gyro rate, in degrees per second

\*/

float gyroGetRate (){

float gyroRead = 0.0;

#if defined (GYRO\_OVERSAMPLE)

if (GYRO\_OVERSAMPLE > 0) {

int sampleSum = 0;

int nSamples = pow (4, GYRO\_OVERSAMPLE);

for (int i = 0; i < nSamples; ++i)

sampleSum += Gyro.value(vex::rotationUnits::raw);

gyroRead = (float) sampleSum / (float) nSamples;

}

else

gyroRead = Gyro.value(vex::rotationUnits::raw);

#else

gyroRead = Gyro.value(vex::rotationUnits::raw);

#endif

//Difference from zero-rate value or the average calibration read

float gyroDiff = gyroRead -avg;

//Difference fro zero-rate value, in volts

float gyroVoltage = gyroDiff \* 5.0 / 4095.0;

if (fabs (gyroDiff) > GYRO\_STD\_DEVS \*stdDev)

{

if (gyroFlipped){return -1 \* 1.28\*gyroVoltage / voltsPerDPS;}

else{return 1.28\*gyroVoltage / voltsPerDPS;}

}

return 0;

}

#endif

void brake(vex::motor,char);

int GyroTrack()

{

float GyroAdd = 0;

float GyroTempCheck = 0;

while (1)

{

GyroAdd = gyroGetRate() - GyroTempCheck;

GyroTempCheck = gyroGetRate();

GlobalGyro += GyroAdd;

wait(5);

}

return 0;

}

void run(vex::motor motorname, double speed)

{

if (speed != 0) { motorname.spin(vex::directionType::fwd, speed, vex::velocityUnits::pct); }

else { motorname.stop(vex::brakeType::brake); }

}

void brake(vex::motor motorname, char BRAKE)

{

if (BRAKE == 'c') { motorname.stop(vex::brakeType::coast); }

else if (BRAKE == 'h') { motorname.stop(vex::brakeType::hold); }

else { motorname.stop(vex::brakeType::brake); }

}

void runRPM(vex::motor motorname, double speed)

{

if (speed != 0) { motorname.spin(vex::directionType::fwd, speed, vex::velocityUnits::rpm); }

else { motorname.stop(vex::brakeType::brake); }

}

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

float enc(vex::motor motorname) {return motorname.rotation(vex::rotationUnits::deg);}

#ifndef DEBUG //if DEBUG is defined earlier this section will not compile

void AllianceSelect()

{ //screen 480x272

Brain.Screen.clearScreen(vex::color::black);//show red blue and confirm

int selection = 0;

while (selection < 4)

{

Brain.Screen.setPenWidth(1);

Brain.Screen.setPenColor(vex::color::black);

if (bRight == 1 || (Brain.Screen.xPosition() > 300 && Brain.Screen.xPosition() < 420 && Brain.Screen.yPosition() > 20 && Brain.Screen.yPosition() < 60)) { selection = 1; Controller.Screen.print("Blue"); }//if Touch within blue box:: selection=1

else if (bLeft == 1 || (Brain.Screen.xPosition() > 60 && Brain.Screen.xPosition() < 180 && Brain.Screen.yPosition() > 20 && Brain.Screen.yPosition() < 60)) { selection = 2; Controller.Screen.clearLine(1); Controller.Screen.print("Red"); } //else if Touch within red box:: selection=2

else if (selection == 3 && Brain.Screen.pressing() == 0) { selection = 4; } //else if Touch within select box&& Selection >0 :: selection=3

else if ((bA && selection > 0) || (Brain.Screen.xPosition() > 180 && Brain.Screen.xPosition() < 300 && Brain.Screen.yPosition() > 100 && Brain.Screen.yPosition() < 140 && selection > 0)) { selection = 3; } //else if Touch within select box&& Selection >0 :: selection=3

if (selection == 1) { TrackColor = 'B'; Brain.Screen.drawRectangle(297, 17, 126, 46, vex::color::green); Brain.Screen.drawRectangle(57, 17, 126, 46, vex::color::black); } //if selection==1 draw green box around Blue and Black Box around red

else if (selection == 2) { TrackColor = 'R'; Brain.Screen.drawRectangle(297, 17, 126, 46, vex::color::black); Brain.Screen.drawRectangle(57, 17, 126, 46, vex::color::green); }//else if selection==2 draw green box around red and Black Box around blue

else if (selection == 3) { Brain.Screen.drawRectangle(177, 97, 126, 46, vex::color::green); }//else if selection==3 draw green box around select

Brain.Screen.drawRectangle(60, 20, 120, 40, vex::color::red);

Brain.Screen.drawRectangle(300, 20, 120, 40, vex::color::blue);

Brain.Screen.drawRectangle(180, 100, 120, 40, vex::color::white);

Brain.Screen.setPenWidth(10);

Brain.Screen.setPenColor(vex::color::white);

Brain.Screen.printAt(105, 45, false, "RED");

Brain.Screen.printAt(340, 45, false, "BLUE");

Brain.Screen.setPenColor(vex::color::black);

Brain.Screen.printAt(210, 125, false, "SELECT");

wait(100);

Alliance = TrackColor;

}

if (TrackColor=='B'){

Color=1;

}

else {

Color=2;

}

}

#endif

int PrintScreen()

{

while (1)

{

Brain.Screen.clearScreen(vex::color::black);

Brain.Screen.setPenColor(vex::color::white);

Brain.Screen.printAt(1, 160, "RPM %1.2f", FlyL.velocity(vex::velocityUnits::rpm));

Brain.Screen.printAt(1, 180, "RPM Goal %f", RPMGoal);

Brain.Screen.printAt(1, 60, "%f", Gyro.value(vex::rotationUnits::raw));

//int BallVal=BallLoadSensor.value(vex::analogUnits::pct);

Brain.Screen.printAt(80, 80, "%d",BallLoaded.pressing() );

Brain.Screen.printAt(300, 100, "%d",RLine.value(vex::percentUnits::pct) );

Brain.Screen.printAt(300, 130, "%d",LLine.value(vex::percentUnits::pct) );

Brain.Screen.printAt(100, 20, "P: H:%d Y:%d",Vision.objects[FinalObject].height,Vision.objects[FinalObject].centerY);

Brain.Screen.printAt(340, 100, "P: %1.8f",GLOBALP);

Brain.Screen.printAt(340, 120, "I: %1.8f",GLOBALI);

Brain.Screen.printAt(340, 140, "D: %1.8f",GLOBALD);

Brain.Screen.printAt(180, 80, "%d",CapIR.value() );

if (rpmError < 2.0) { Brain.Screen.setPenColor(vex::color::green); if (Shoot == 1) { Controller.rumble("."); } }

else { Brain.Screen.setPenColor(vex::color::white); }

Brain.Screen.printAt(1, 120, "RPM error %1.2f", rpmError);

wait(75);

}

return 0;

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

int RPMCalc()

{int Counter=0;

int FlagSum=0;

while (1)

{int TempRPM = RPMGoal;

if (fly == 2 || fly == 3)

{//TempRPM = RPMGoal;

//FlyR.spin(vex::directionType::fwd, RPMGoal, vex::velocityUnits::rpm);

//FlyL.spin(vex::directionType::fwd, RPMGoal, vex::velocityUnits::rpm);

while ((fly == 2 || fly == 3) )//&& RPMGoal == TempRPM)

{rpmError = fabs(RPMGoal - (FlyL.velocity(vex::velocityUnits::rpm)));

float objheight=Vision.objects[FinalObject].height;

if (SnapToFlag==1&&Vision.objects[FinalObject].exists)

{ if (Vision.objects[FinalObject].centerY<130 &&objheight<35)//high

{ FlagSum+=2.75\*pow(objheight,2)-109.5\*objheight+1194;}

else if (objheight<28)//Mid far away

{FlagSum+=-0.0625\*pow(objheight,2)+1.4191\*objheight+86.324;}

else{FlagSum+=0.0144\*pow(objheight,3)-1.3286\*pow(objheight,2)+40.392\*objheight-325.79;}

Counter++;

}

if(AutoRPM==1&&Counter%100==0)//if the flagsum has been added 50 times

{RPMGoal=FlagSum/100;}

else if (AutoRPM==0){RPMGoal=116;}

if (Counter==100){Counter=0;FlagSum=0;}

wait(10);

FlyR.spin(vex::directionType::fwd, RPMGoal, vex::velocityUnits::rpm);

FlyL.spin(vex::directionType::fwd, RPMGoal, vex::velocityUnits::rpm);

}

}

else

{FlyL.stop(vex::brakeType::coast); FlyR.stop(vex::brakeType::coast);

while ((fly == 4 || fly == 1) && RPMGoal == TempRPM) { wait(100); }

}

wait(10);

}

return 0;

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int ShootBall()

{

while (1)

{

if (Shoot == 1) //when button is pressed

{ fly = 2;

while(BallLoaded.pressing()==0&&BallFeed.rotation(vex::rotationUnits::deg)<30&&Shoot==1)

{

run(Roller2, -40); //run until ball in

//R=2;

wait(10);

}

if (Shoot==1)

{

//run(Roller2, 0); //stop rolling

R=4;

}

while(Shoot==1)

{

if (rpmError<1.5||abs(enc(BallFeed))>110)

{

run(Roller2,-100);

run(BallFeed,100);

}

else{run(Roller2,0);

run(BallFeed,0);

}

if (abs(enc(BallFeed))>170){break;}

wait(100);

}

const int localT3=T3;

while ((T3-localT3)<750&&IR.value()==1&&Shoot==1)//while timer less than .5 sec and no ball seen on ir

{

run(Roller2,100);

run(BallFeed,-80);

wait(5);

}

while ((T3-localT3)<750)

{

run(BallFeed,-80);

wait(5);

}

run(Roller2, 0); //stop rolling

run(BallFeed,0);

R=2;

wait(200);

Shoot = 0;

BallFeed.resetRotation();

wait(50);

}

else {}

wait(20);

}

return 0;

}

///////////////MAIN.CPP

#include "robot-config.h"

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int TurnToFlag()

{

while (1)

{

while(SnapToFlag == 1)

{if (bX==1){SnapToFlag = 0;}

Vision.takeSnapshot(Color, 7);

//Brain.Screen.setPenColor(vex::color::red);

TempXDist = 200;

for (int i = 0; Vision.objects[i].exists == 1; i++)

{

if (fabs(GlobalFlagOffset - Vision.objects[i].centerX) < fabs(TempXDist)) { TempXDist = GlobalFlagOffset - Vision.objects[i].centerX; }

else {}

}

TempWidth = 0;

TempHeight = 400;

for (int i = 0; Vision.objects[i].exists == 1; i++)

{

//if (Vision.objects[i].width > 8 && Vision.objects[i].height > 8) { Brain.Screen.printAt(230, 20 + i \* 20, "X:%d,Y:%d W:%d H:%d D:%1.2f", Vision.objects[i].centerX, Vision.objects[i].centerY, Vision.objects[i].width, Vision.objects[i].height, 0.0157\*pow(Vision.objects[i].height, 2) - 1.0842\*Vision.objects[i].height + 19.93); }

if ((Vision.objects[i].width > 8 && Vision.objects[i].height > 8) && (Vision.objects[i].centerX > (GlobalFlagOffset - 10 - TempXDist) && Vision.objects[i].centerX < (GlobalFlagOffset + 10 - TempXDist)) && Vision.objects[i].centerY < TempHeight) { TempHeight = Vision.objects[i].centerY; TempWidth = GlobalFlagOffset - Vision.objects[i].centerX; FlagYDim=Vision.objects[i].height; TempObject=i; }

else {}

}

FinalObject=TempObject;

TurnDiff = TempWidth;

if (TurnDiff > 0) { TurnDir = 1; }

else { TurnDir = -1; }

if(SnapToFlag == 1 && fabs(TurnDiff) > 4)

{

double speed;

speed=pow(fabs(TurnDiff),1.1)\*TurnDir\*.5;

Turret.spin(vex::directionType::fwd, -speed, vex::velocityUnits::pct);

}

else{Turret.stop(vex::brakeType::brake);}

wait(5);

}

//wait(750);

//SnapToFlag = 0;

wait(10);

}

return 0;

}

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

int TIMER2()

{while (1)

{wait(1);

T3 += 1;}

return 0;}

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

void StopDrive()

{

brake(RF, 'b');

brake(RM, 'b');

brake(RB, 'b');

brake(LF, 'b');

brake(LM, 'b');

brake(LB, 'b');

}

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

void rightDrive(int power)

{ run(RF, power); run(RM, power); run(RB, power);}

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

void leftDrive(int power)

{run(LF, power);run(LM, power); run(LB, power);

}

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

int Driver()

{BallFeed.setTimeout(3,vex::timeUnits::sec);

while(1)

{ if (Shoot==0&&bY==1){run(Roller2, 90);}

else if(Shoot==0&&(BallLoaded.pressing()==0&&abs(BallFeed.rotation(vex::rotationUnits::deg))<60)){run(Roller2, -90);}//if no ball seen on top(limit or encoder) {run Roller in}

else if(Shoot==0&&abs(BallFeed.rotation(vex::rotationUnits::deg))<70){run(Roller2, -90);BallFeed.startRotateTo(85,vex::rotationUnits::deg,20,vex::velocityUnits::pct);}//encoder less than a value {start running up to value run roller in}

else if(Shoot==0&&BallFeed.isSpinning()){run(Roller2, 0);}

else if(Shoot==0&&IR.value()==1){run(Roller2, 20);}//else if ir doesnt see ball run out

else if(Shoot==0){run(Roller2, 0);}// else if stop

else{}

//if (Shoot==0&&(R==3 || R==4)&&bL1==0){}

if(bL1==1){ run(Roll, -100); R = 5;}//if button pressed {roll out R=5}

else if(Shoot==0&&IR.value()==0&&(BallLoaded.pressing()==1||abs(BallFeed.rotation(vex::rotationUnits::deg))>10)){R=4;run(Roll, 0);}//if ball on top and bottom {stop R=4}

else if(Shoot==0&&IR.value()==0&&(BallLoaded.pressing()==0&&abs(BallFeed.rotation(vex::rotationUnits::deg))<10)){R=2;}

else if (Shoot==0&&R==5){R=2;}

else{}

if (R==4){run(Roll,0);}

if (R==2){run(Roll,80);}

wait(10);

}

return 0;}

////////////////////////////////////////////////////////////////////////////

void Turn(double turningDegree, int turningspeed)

{

int dir;

if (turningDegree>0){dir =1;}

else{dir=-1;}

LF.resetRotation();

RF.resetRotation();

int Loffset,Roffset;

double rotation=3\*turningDegree;

while (abs(enc(RF))<fabs(rotation)||abs(enc(LF))<fabs(rotation))

{ if (abs(enc(RF))<abs(enc(LF))){Loffset=dir\*10; Roffset=0;}

else if (abs(enc(LF))<abs(enc(RF))){Loffset=0; Roffset=-dir\*10;}

else{Loffset=0; Roffset=0;}

turningspeed\*=dir;

rightDrive(turningspeed+Roffset);

leftDrive(-turningspeed+Loffset);

/\* if ((abs(enc(RF)))<fabs(rotation)){rightDrive(turningspeed\*dir);}

else if ((abs(enc(RF)))>fabs(rotation)+1){rightDrive(-turningspeed\*dir/3);}

else{

brake(RF, 'b');

brake(RM, 'b');

brake(RB, 'b');}

if( abs(enc(LF))<fabs(rotation)){leftDrive(-turningspeed\*dir);}

else if( abs(enc(LF))>fabs(rotation)+1){leftDrive(turningspeed\*dir/3);}

else{brake(LF, 'b');

brake(LM, 'b');

brake(LB, 'b');}

wait(5);

\*/

wait(5);

}

wait(10);

dir\*=-1;

while (abs(enc(RF))>fabs(rotation)||abs(enc(LF))>fabs(rotation))

{

if ((abs(enc(RF)))>fabs(rotation)){rightDrive(turningspeed\*dir/3);}

else{

brake(RF, 'b');

brake(RM, 'b');

brake(RB, 'b');}

if( abs(enc(LF))>fabs(rotation)){leftDrive(-turningspeed\*dir/3);}

else{brake(LF, 'b');

brake(LM, 'b');

brake(LB, 'b');}

wait(5);

}

brake(RF, 'h');

brake(RM, 'h');

brake(RB, 'h');

brake(LF, 'h');

brake(LM, 'h');

brake(LB, 'h');

/\*while (abs(enc(RF))>fabs(rotation)||abs(enc(LF))>fabs(rotation))

{ if (abs(enc(RF))>abs(enc(LF))){Loffset=dir\*5; Roffset=0;}

else if (abs(enc(LF))>abs(enc(RF))){Loffset=0; Roffset=-dir\*5;}

else{Loffset=0; Roffset=0;}

turningspeed\*=dir;

rightDrive(turningspeed/3+Roffset);

leftDrive(-turningspeed/3+Loffset);

wait(5);

}\*/

wait(100);

}

void T(float degree, float vel)

{

T3=0;

LF.resetRotation();

RF.resetRotation();

float x;

float dir,Loffset, Roffset,speeed;

if (degree<0){dir=-1;}

else{dir=1;}//

degree=degree\*10;//Correction factor for this specific gyro

while(fabs(GlobalGyro)>(abs(degree))||fabs(GlobalGyro)<(abs(degree)-10) && T3<2500)

{

if (abs(enc(RF))<abs(enc(LF))){Loffset=dir\*10; Roffset=0;}

else if (abs(enc(LF))<abs(enc(RF))){Loffset=0; Roffset=-dir\*10;}

else{Loffset=0; Roffset=0;}

x=(degree-GlobalGyro)/10;

speeed=vel;//abs(100\*x/(degree\*1.25))+20;/\*pow((x/16),2);\*//\*100\*cos(x/2)/2+(100/2);\*/

speeed=dir\*speeed;

rightDrive(speeed+Roffset);

leftDrive(-speeed+Loffset);

}

wait(30);

dir\*=-1;

while(fabs(GlobalGyro)>(abs(degree)+10)||fabs(GlobalGyro)<(abs(degree)-10) && T3<2500)

{

if (abs(enc(RF))<abs(enc(LF))){Loffset=dir\*5; Roffset=0;}

else if (abs(enc(LF))<abs(enc(RF))){Loffset=0; Roffset=-dir\*5;}

else{Loffset=0; Roffset=0;}

x=(degree-GlobalGyro)/10;

speeed=vel;//abs(100\*x/(degree\*1.25))+20;/\*pow((x/16),2);\*//\*100\*cos(x/2)/2+(100/2);\*/

speeed=dir\*speeed;

rightDrive(speeed/3+Roffset);

leftDrive(-speeed/3+Loffset);

}

rightDrive(0);

leftDrive(0);

}

////////////////////////////////////////////////////////////////////////////

int MoveCounter = 0;

int PID\_MOTOR\_SCALE = 1;

int PID\_MOTOR\_MAX = 100;

int PID\_MOTOR\_MIN = (-100);

int PID\_INTEGRAL\_LIMIT = 50;

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

float pid\_Kp = 0.59;

float pid\_Ki = 0.09;//0.05;

float pid\_Kd = 0.0;//.05;

int pidRunning = 0; //TURN ON OR OFF PID

float RequestedAngle = 0, pidDriveR;

int angleSentinel = 1;

void pidTurn(float globalDegrees, float pid\_Kp, float pid\_Ki, float pid\_Kd, int timeout)

{ int direction = 0;

GlobalGyro = 0;

Brain.resetTimer();

float pidError = 0;

float pidLastError = 0;

float pidIntegral = 0;

float pidDerivative = 0;

float pidDriveR = 0;

T3 = 0;

while (T3 < timeout)

{

//Calculate Error//(Convert Dintance in inches to encoder ticks)

pidError = fabs(globalDegrees \*10) - fabs(GlobalGyro);

// integral - if Ki is not 0(can put threshold)

if (pid\_Ki != 0)

{

// If we are inside controlable window then integrate the error

if (fabs(pidError) < PID\_INTEGRAL\_LIMIT) pidIntegral = pidIntegral + pidError;

else pidIntegral = 0;

}

else pidIntegral = 0;

///////////////////////////////////////

//CALCULATE DERIVATIVE/////////////////

pidDerivative = pidError - pidLastError;

pidLastError = pidError;

//////////////////////////////////////

//CALCULATE DRIVE/////////////////////

pidDriveR = (pid\_Kp \* pidError) + (pid\_Ki \* pidIntegral) + (pid\_Kd \* pidDerivative);

///////////////////////////////////////

//LIMIT DRIVE//////////////////////////

if (pidDriveR > PID\_MOTOR\_MAX) pidDriveR = PID\_MOTOR\_MAX;

if (pidDriveR < PID\_MOTOR\_MIN) pidDriveR = PID\_MOTOR\_MIN;

///////////////////////////////////////

//SEND POWER TO MOTORS/////////////////

if (globalDegrees > 0) direction = 1;

if (globalDegrees < 0) direction = -1;

//if(gmoveandturn==false)

rightDrive(direction \* pidDriveR \* PID\_MOTOR\_SCALE);

leftDrive(-direction \* pidDriveR \* PID\_MOTOR\_SCALE);

if ((fabs(pidError) < 5) && (fabs(avgSpeed) < 10)) {

StopDrive();

break;

}

if (Brain.timer(vex::timeUnits::msec) < 40) { avgError += pidError; }

else { avgSpeed = avgError / 3; avgError = 0; Brain.resetTimer(); }

//REFRESH RATE 60Hz

wait(16);

}

StopDrive();

}

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

int Move(float speed, float dist) {

float dir;

if (dist < 0) { dir = -1; }

else { dir = 1; }

float tempdir = dir;

leftDrive(dir \* 5);

rightDrive(dir \* 5);

wait(200);

StopDrive();

wait(100);

LF.resetRotation();

RF.resetRotation();

wait(20);

double counter = 0;

while (fabs(enc(LF)) < fabs((dist \* 360.0 / (4.0\*3.14159)))) {

float Roffset = 1.0;

if (fabs(enc(LF)) < fabs((enc(RF)) + 5)) { Roffset = 0.9; }

else if (fabs(enc(LF)) > fabs(enc(RF)) - 5) { Roffset = 1.1; }

else {}

if (counter < 100) { dir = tempdir \* counter\*0.01; counter+=.75; }

else { dir = tempdir; }

run(RF, speed\*Roffset\*dir);

run(LF, speed\*dir);

run(RM, speed\*Roffset\*dir);

run(LM, speed\*dir);

run(RB, speed\*Roffset\*dir);

run(LB, speed\*dir);

wait(12);

}

leftDrive(0);

rightDrive(0);

return 1;

}

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

int Move2(float speed, float dist) {

int status=1;

float dir;

if (dist < 0) { dir = -1; }

else { dir = 1; }

float tempdir = dir;

if (MoveReturn==0){ LF.resetRotation();

RF.resetRotation();MoveReturn=1;}

wait(20);

int counter = 0;

if (fabs(enc(LF)) < fabs((dist \* 360 / (4.0\*3.14159)))){status=1;}

else{status=0;}

float Roffset = 1.0;

if (fabs(enc(LF)) < fabs((enc(RF)) + 5)) { Roffset = 0.9; }

else if (fabs(enc(LF)) > fabs(enc(RF)) - 5) { Roffset = 1.1; }

run(RF, speed\*Roffset\*dir);

run(LF, speed\*dir);

run(RM, speed\*Roffset\*dir);

run(LM, speed\*dir);

run(RB, speed\*Roffset\*dir);

run(LB, speed\*dir);

wait(10);

if (status==0)

{

leftDrive(0);

rightDrive(0);

}

MoveReturn=status;

return status;

}

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

int MoveNoStop(float speed, float dist) {

float dir;

if (dist < 0) { dir = -1; }

else { dir = 1; }

float tempdir = dir;

leftDrive(dir \* 5);

rightDrive(dir \* 5);

wait(200);

StopDrive();

wait(100);

LF.resetRotation();

RF.resetRotation();

wait(20);

int counter = 0;

while (fabs(enc(LF)) < fabs((dist \* 360 / (4.0\*3.14159)))) {

float Roffset = 1.0;

if (fabs(enc(LF)) < fabs((enc(RF)) + 5)) { Roffset = 0.9; }

else if (fabs(enc(LF)) > fabs(enc(RF)) - 5) { Roffset = 1.1; }

else {}

if (counter < 100) { dir = tempdir \* counter\*0.01; counter++;}

else { dir = tempdir; }

run(RF, speed\*Roffset\*dir);

run(LF, speed\*dir);

run(RM, speed\*Roffset\*dir);

run(LM, speed\*dir);

run(RB, speed\*Roffset\*dir);

run(LB, speed\*dir);

wait(10);

}

return 0;

}

void ToWall(double vel)

{T3=0;

float TempLF=0, TempRF=0;

do{

TempLF=enc(LF);

TempRF=enc(RF);

rightDrive(vel);

leftDrive(vel);

wait(100);

}while ((TempLF!=enc(LF)||TempRF!=enc(RF))&&(RLine.value(vex::percentUnits::pct)<68||LLine.value(vex::percentUnits::pct)<68)&&T3<2000);

StopDrive();

}

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

void TurnTurret(double degree)

{ Turret.resetRotation();

int dir;

if (degree>0){dir=-1;}

else{dir=1;}

//degree>0 means left

double count=5;

double gearratio=12.0/175.0;

while(fabs(Turret.rotation(vex::rotationUnits::deg))<fabs((degree/gearratio)))

{ //if(LeftTurretLimit.pressing()==1){break;}

// else if(RightTurretLimit.pressing()==1){break;}

if ((fabs(Turret.rotation(vex::rotationUnits::deg))>fabs(((degree/gearratio)/2)))&&count>=5){count-=1.5;}

else{count+=1;}

Turret.spin(vex::directionType::fwd,(count\*dir),vex::percentUnits::pct);

wait(10);

Brain.Screen.printAt(230, 20,"%f", fabs(Turret.rotation(vex::rotationUnits::deg)));

Brain.Screen.printAt(230, 50,"%f", ((degree/gearratio)));

}

run(Turret,0);

TurretRunning=0;

}

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

void pre\_auton(void) {

BallFeed.resetRotation();

gyroInit(0);

#ifndef DEBUG //if DEBUG is not defined earlier this section will not compile

AllianceSelect();

#endif

vex::task fourth(PrintScreen);

vex::task second(ShootBall);

vex::task first(RPMCalc);

vex::task fifth(TurnToFlag);

vex::task mid2(TIMER2);

//vex::task last(anglePIDControl);

vex::task third (Driver);

vex::task one(GyroTrack);

//CapFlip.spin(vex::directionType::fwd,10,vex::velocityUnits::rpm);

}

void autonomous(void) {

//CapFlip.spin(vex::directionType::fwd, 15, vex::velocityUnits::rpm);

Trans.set(0);

fly = 2;

run(Roll, 120);

RPMGoal = 110;

//wait(200);

StopDrive();

wait(50);

Move(80, 38.74);//40

wait(200);//200

Move(80, -22.263);//20

Shoot = 1;

Brain.resetTimer();

while (Shoot == 1) { wait(10); if (Brain.timer(vex::timeUnits::msec) > 2000) { break; } }

Shoot = 0;

wait(50);//250

if (Alliance == 'B') { pidTurn(90, 0.09, 0.05, 0.00, 4000); }//Amount initial, secondary

else{ pidTurn(-90, 0.09, 0.05, 0.00, 4000);}

///leftDrive(-2);

///rightDrive(-2);

///wait(300);

R=6;

run(Roll, -100);

TrackColor=Alliance;

//wait(100);

//leftDrive(-50);

//rightDrive(-50);

//wait(1000);

ToWall(-40);

StopDrive();

run(Roll,-50);

while (Move(90, 50.26) != 1) {wait(10); }//Drive forward but if ball is seen by sensor kill flipper

//2400

// while(Vision.objects[0].exists!=1){wait(5);}rightDrive(10);

wait(200);//400

run(Roll, 0);

Move(40, -10);//480

wait(600);//600

if (Alliance == 'B') { pidTurn(52, 0.078, 0.01, 0.01, 4000); }//Amount initial, secondary

else{ pidTurn(-50, 0.07, 0.01, 0.01, 4000);}

leftDrive(2);

rightDrive(2);

wait(300);

run(Roll, 100);

//wait(300);

MoveNoStop(30, 19.6);//890/////GRAB BALL UNDER CAP

wait(100);//700

run(Roll, 0);

Move2(30,3);

wait (100);//300

run(Roll,100);

//wait(300);

//Move(30,-3);

Move(80, -11.25);//203//3 was added

wait(300);//400

if (Alliance == 'B') { pidTurn(30, 0.056, 0.01, 0.01, 4000); }//Amount initial, secondary

else{ pidTurn(-30, 0.05, 0.01, 0.01, 4000);}

wait(300);

/\*Shoot = 1;

Brain.resetTimer();

while (Shoo0t == 1) { wait(10); if (Brain.timer(vex::timeUnits::msec) > 2000) { break; } }

Shoot = 0;\*/

leftDrive(-100);

rightDrive(-100);

wait(900);

ToWall(-40);

StopDrive();

//wait(300);

Move(60,3);

wait(300);

if (Alliance == 'B') { pidTurn(-97, 0.073, 0.01, 0.01, 4000); }//Amount initial, secondary

else{ pidTurn(97, 0.09, 0.05, 0.00, 4000);}

//leftDrive(20);

//rightDrive(20);

//wait(2000);

RPMGoal=140;

ToWall(25);

StopDrive();

if (Alliance == 'B') { TurnTurret(-35);}

else {TurnTurret(35);}

Move(50,-13.18);//270

run(Roll,100);

if (Alliance == 'B') { TurnTurret(20);}

else {TurnTurret(-20);}

Shoot = 1;

Brain.resetTimer();

while (Shoot == 1) { wait(10); if (Brain.timer(vex::timeUnits::msec) > 2000) { break; } }

Shoot = 0;

wait(300);

if (Alliance == 'B') { pidTurn(90, 0.07, 0.01, 0.01, 4000); }//Amount initial, secondary

else{ pidTurn(-90, 0.07, 0.01, 0.01, 4000);}

//wait(300);

//leftDrive(-20);

//rightDrive(-20);

//wait(1500);

ToWall(-20);

StopDrive();

//if (Alliance == 'B') { TurnTurret(-15);}

//TurnTurret(15);

//wait(300);

Move(100,85);

run(Roll, 0);

R = 4;

fly = 4;

while (1) { wait(100); }

}

void usercontrol(void) {

#ifdef DEBUG //if DEBUG is not defined earlier this section will not compile

while(DEBUG){

if (bUp == 1) { GLOBALP += .01; wait(100); }

if (bDown == 1) { GLOBALP -= .01; wait(100); }

if (bX== 1) { GLOBALI += .0000001; wait(100); }

if (bB == 1) {GLOBALI -= .0000001; wait(100); }

if (bR1== 1) { GLOBALD += .05; wait(100); }

if (bR2 == 1) {GLOBALD -= .05; wait(100); }

wait(10);

}

#endif

R=2;

AutoRunning=0;

SnapToFlag = 0;

GlobalFlagOffset=160;

Shoot=0;

//R = 4;

FlipCount = 2;

Tran = 2;

RPMGoal=116;

while (1)

{

if (bL2 == 1) { SnapToFlag = 1; }

//else if ((fabs(ch3) > 2 || fabs(ch4) > 2) && SnapToFlag == 1) { SnapToFlag = 0; }

else if ( AutoRunning == 0 && (abs(ch3) > 2 || abs(ch4) > 2))

{ if(abs(LF.velocity(vex::velocityUnits::pct))>(ch3 + ch4))

{

run(LF, ((ch3 + ch4)+LF.velocity(vex::velocityUnits::pct)+LF.velocity(vex::velocityUnits::pct))/3); //(Axis3+Axis4)/2

run(LM, ((ch3 + ch4)+LM.velocity(vex::velocityUnits::pct)+LM.velocity(vex::velocityUnits::pct))/3); //(Axis3+Axis4)/2

run(LB, ((ch3 + ch4)+LB.velocity(vex::velocityUnits::pct)+LB.velocity(vex::velocityUnits::pct))/3); //(Axis3+Axis4)/2

run(RF, ((ch3 - ch4)+RF.velocity(vex::velocityUnits::pct)+RF.velocity(vex::velocityUnits::pct))/3);//(Axis3-Axis4)/2

run(RM, ((ch3 - ch4)+RM.velocity(vex::velocityUnits::pct)+RM.velocity(vex::velocityUnits::pct))/3);//(Axis3-Axis4)/2

run(RB, ((ch3 - ch4)+RB.velocity(vex::velocityUnits::pct)+RB.velocity(vex::velocityUnits::pct))/3);//(Axis3-Axis4)/2

}

else{

run(LF, (ch3 + ch4)); //(Axis3+Axis4)/2

run(LM, (ch3 + ch4)); //(Axis3+Axis4)/2

run(LB, (ch3 + ch4)); //(Axis3+Axis4)/2

run(RF, (ch3 - ch4));//(Axis3-Axis4)/2

run(RM, (ch3 - ch4));//(Axis3-Axis4)/2

run(RB, (ch3 - ch4));//(Axis3-Axis4)/2

}

}

else if (AutoRunning == 0)

{

StopDrive();

}

else {}

if (fly == 4 && bR1 == 0) { fly = 1; }//Flywheel

else if (fly == 1 && bR1 == 1) { fly = 2; }

else if (fly == 2 && bR1 == 0) { fly = 3; }

else if (fly == 3 && bR1 == 1) { fly = 4; }

else {}

if (R == 4 && bR2 == 0) { run(Roll, 0); R = 1; } //Roller //4 for off

else if (R == 1 && bR2 == 1) { run(Roll, 80); R = 2; }//2 for on

else if (R == 2 && bR2 == 0) { run(Roll, 80); R = 3; }

else if (R == 3 && bR2 == 1) { run(Roll, 0); R = 4; }

else {}

if (Tran == 4 && bB == 0) { Trans.set(true); Tran = 1; } //Trans

else if (Tran == 1 && bB == 1) { Trans.set(false); Tran = 2; }

else if (Tran == 2 && bB == 0) { Trans.set(false); Tran = 3; }

else if (Tran == 3 && bB == 1) { Trans.set(true); Tran = 4; }

else {}

if (LiftVar == 4 && bLeft == 0) { Lift.set(true); LiftVar = 1; } //Trans

else if (LiftVar == 1 && bLeft == 1) { Lift.set(false); LiftVar = 2; }

else if (LiftVar == 2 && bLeft == 0) {Lift.set(false); LiftVar = 3; }

else if (LiftVar == 3 && bLeft == 1) { Lift.set(true); LiftVar = 4; }

else {}

if (bUp==1){AutoRPM=1;}

else if (bDown==1){AutoRPM=0;}

if (bA == 1) { Shoot = 1; } //Shoot ball

else if (bY == 1) { Shoot = 0; } //Stop Mid shot

else {}

//if (bUp == 1 && RPMGoal < 200) { RPMGoal += 2; wait(200); }

//if (bDown == 1 && RPMGoal > 50) { RPMGoal -= 2; wait(200); }

//else {}

if (abs(ch1)>5){run(Turret,ch1);SnapToFlag\*=2;}

else if (bL2==1){SnapToFlag=1;}

else if (SnapToFlag>1){SnapToFlag=0;}

else if (SnapToFlag==0){run(Turret,0);}

wait(20);

}

}

int main() {

AutoRunning = 0;

Lift.set(false);

gyroInit(0);

//Run the pre-autonomous function.

pre\_auton();

//Set up callbacks for autonomous and driver control periods.

Competition.autonomous(autonomous);

Competition.drivercontrol(usercontrol);

//Prevent main from exiting with an infinite loop.

while (1) {

wait(100);//Sleep the task for a short amount of time to prevent wasted resources.

}

while (1) { wait(1000); }

}