#include "WPILib.h"

#include <math.h>

class RobotDemo : public IterativeRobot

{

public:

Victor \*leftRear, \*leftRearCenter, \*leftFrontCenter, \*leftFront, \*rightRear, \*rightRearCenter, \*rightFrontCenter, \*rightFront;

Joystick \*joy, \*controller, \*joy2;

Compressor \*compressor;

DoubleSolenoid \*bertha, \*forks;

Timer \*timer;

DigitalInput \*limitSwitch;

int autoStrat;

RobotDemo()

{

//drivetrain motors

//left back wheel

leftRear = new Victor(0); //red

leftRearCenter = new Victor(1); //green

//left front wheel

leftFrontCenter = new Victor(2); //blue

leftFront = new Victor(3); //yellow

//right rear wheel

rightRear = new Victor(4); //black

rightRearCenter = new Victor(5); //white

//right front wheel

rightFrontCenter = new Victor(6); //blue-red

rightFront = new Victor(7); //black-red

//driver and operator controls

joy = new Joystick(0);

joy2 = new Joystick(1);

controller = new Joystick(2);

//compressor

compressor = new Compressor(0);

//pneumatics

bertha = new DoubleSolenoid(0, 1);

forks = new DoubleSolenoid(2, 3);

//misc.

limitSwitch = new DigitalInput(2);

//variables

timer = new Timer();

autoStrat = 0;

}

void RobotInit()

{

SmartDashboard::init();

compressor->SetClosedLoopControl(true);

}

void AutonomousInit()

{

switch(autoStrat)

{

case 0:

Auto\_OneTote();

break;

case 1:

Auto\_Strafe\_TwoTotes(true); //strafe right

break;

case 2:

Auto\_Strafe\_TwoTotes(false); //strafe left

break;

case 3:

AutoFunc\_DriveBackLong();

break;

case 4:

Auto\_ThreeTotes(true);

break;

case 5:

Auto\_ThreeTotes(false);

break;

case 6:

Auto\_ThreeTotesCenter(true);

break;

case 7:

Auto\_ThreeTotesCenter(false);

break;

case 8:

Auto\_DoNothing();

break;

}

/\*

\* 6. Three Totes Center Right -> Auto\_ThreeTotesCenter(true);

7. Three Totes Center Left -> Auto\_ThreeTotesCenter(false);

8. Do Nothing -> Auto\_DoNothing();

\*/

}

void TeleopInit()

{

}

void TeleopPeriodic()

{

if (controller->GetRawAxis(1) > 0.25)

{

bertha->Set(DoubleSolenoid::kReverse);

}

else if (controller->GetRawAxis(1) < -0.25)

{

bertha->Set(DoubleSolenoid::kForward);

}

else

{

bertha->Set(DoubleSolenoid::kOff);

}

if (controller->GetRawButton(1))

{

forks->Set(DoubleSolenoid::kForward);

}

else

{

forks->Set(DoubleSolenoid::kReverse);

}

if (joy->GetRawButton(1))

{

//ArcadeShift(joy->GetRawAxis(0) \* 0.5, -joy->GetRawAxis(1) \* 0.5, joy->GetRawAxis(2) \* 0.5);

TankShift(joy->GetRawAxis(0)\*0.5, -joy->GetRawAxis(1)\*0.5, joy2->GetRawAxis(0)\*0.5, -joy2->GetRawAxis(1)\*0.5);

}

else

{

//ArcadeShift(joy->GetRawAxis(0), -joy->GetRawAxis(1), joy->GetRawAxis(2));

TankShift(joy->GetRawAxis(0), -joy->GetRawAxis(1), joy2->GetRawAxis(0), -joy2->GetRawAxis(1));

}

SmartDashboard::PutNumber("Strafe (X)", joy->GetRawAxis(0));

SmartDashboard::PutNumber("F/R (Y)", -joy->GetRawAxis(1));

SmartDashboard::PutNumber("Rotate (Z)", joy->GetRawAxis(2));

SmartDashboard::PutBoolean("Limit Switch", limitSwitch->Get());

}

void TestInit()

{

//\*leftRear, \*leftRearCenter, \*leftFrontCenter, \*leftFront, \*rightRear, \*rightRearCenter, \*rightFrontCenter, \*rightFront;

/\*

leftRear->Set(1);

Wait(3.0);

leftRear->Set(0);

leftRearCenter->Set(1);

Wait(3.0);

leftRearCenter->Set(0);

leftFrontCenter->Set(1);

Wait(3);

leftFrontCenter->Set(0);

leftFront->Set(1);

Wait(3);

leftFront->Set(0);

rightRear->Set(1);

Wait(3);

rightRear->Set(0);

rightRearCenter->Set(1);

Wait(3);

rightRearCenter->Set(0);

rightFrontCenter->Set(1);

Wait(3);

rightFrontCenter->Set(0);

rightFront->Set(1);

Wait(3);

rightFront->Set(0);

\*/

}

void TestPeriodic()

{

/\*

leftFront->Set(0.5);

leftFrontCenter->Set(0.5);

leftRear->Set(0.5);

leftRearCenter->Set(0.5);

rightFront->Set(-0.5);

rightFrontCenter->Set(-0.5);

rightRear->Set(-0.5);

rightRearCenter->Set(-0.5);

\*/

}

void DisabledPeriodic()

{

if (controller->GetRawButton(4))

{

Wait(0.4);

if (autoStrat < 8)

{

autoStrat++;

}

else

{

autoStrat = 0;

}

if (autoStrat == 0) { SmartDashboard::PutString("Auto Mode", "1 Tote"); }

else if (autoStrat == 1) { SmartDashboard::PutString("Auto Mode", "2 Totes Right"); }

else if (autoStrat == 2) { SmartDashboard::PutString("Auto Mode", "2 Totes Left"); }

else if (autoStrat == 3) { SmartDashboard::PutString("Auto Mode", "Drive Back Only"); }

else if (autoStrat == 4) { SmartDashboard::PutString("Auto Mode", "3 Totes Right"); }

else if (autoStrat == 5) { SmartDashboard::PutString("Auto Mode", "3 Totes Left"); }

else if (autoStrat == 6) { SmartDashboard::PutString("Auto Mode", "3 Totes Center L"); }

else if (autoStrat == 7) { SmartDashboard::PutString("Auto Mode", "3 Totes Center R"); }

else if (autoStrat == 8) { SmartDashboard::PutString("Auto Mode", "Do Nothing"); }

}

}

void AutoFunc\_GetFirstTote()

{

//open forks

forks->Set(DoubleSolenoid::kReverse);

//drive small amount forward until tote locks in place and then stop moving

while (!limitSwitch->Get())

{

ArcadeShift(0, 0.3, 0);

}

ArcadeShift(0, 0, 0);

//retract forks

forks->Set(DoubleSolenoid::kForward);

}

void AutoFunc\_Strafe(bool direction) //right is true, left is false

{

if (direction)

{

ArcadeShift(0.8, 0, 0);

}

else

{

ArcadeShift(-0.8, 0, 0);

}

Wait(5.0); //CHANGE TO: Wait until vision tracking detects tote in center of screen

ArcadeShift(0, 0, 0);

}

void AutoFunc\_GetTote()

{

//move arm up

bertha->Set(DoubleSolenoid::kReverse);

Wait(0.5);

bertha->Set(DoubleSolenoid::kOff);

//drive small amount forward until tote locks in place and then stop moving

while (!limitSwitch->Get())

{

ArcadeShift(0, 0.3, 0);

}

ArcadeShift(0, 0, 0);

//lower arm to set top tote on bottom tote

bertha->Set(DoubleSolenoid::kForward);

Wait(0.5);

bertha->Set(DoubleSolenoid::kOff);

Wait(0.25);

//open forks to drop top tote onto bottom tote

forks->Set(DoubleSolenoid::kReverse);

Wait(0.25);

forks->Set(DoubleSolenoid::kForward);

//lower arm to grab bottom tote

bertha->Set(DoubleSolenoid::kForward);

Wait(0.25);

bertha->Set(DoubleSolenoid::kOff);

Wait(0.25);

//close forks to grab bottom tote

forks->Set(DoubleSolenoid::kForward);

Wait(0.25);

//lift bottom tote up

bertha->Set(DoubleSolenoid::kReverse);

Wait(0.5);

bertha->Set(DoubleSolenoid::kOff);

}

void AutoFunc\_DriveBackShort()

{

//drive back short amount

ArcadeShift(0, -0.3, 0);

Wait(2);

ArcadeShift(0, 0, 0);

}

void AutoFunc\_DriveBackLong()

{

//drive back long amount

ArcadeShift(0, -1.0, 0);

Wait(5);

ArcadeShift(0, 0, 0);

}

void Auto\_Strafe\_TwoTotes(bool direction)

{

AutoFunc\_GetFirstTote();

AutoFunc\_DriveBackShort();

AutoFunc\_Strafe(direction);

AutoFunc\_GetTote();

AutoFunc\_DriveBackLong();

}

void Auto\_OneTote()

{

AutoFunc\_GetFirstTote();

AutoFunc\_DriveBackLong();

}

void Auto\_ThreeTotes(bool direction)

{

AutoFunc\_GetFirstTote();

AutoFunc\_DriveBackShort();

AutoFunc\_Strafe(direction);

AutoFunc\_GetTote();

AutoFunc\_DriveBackShort();

AutoFunc\_Strafe(direction);

AutoFunc\_GetTote();

AutoFunc\_DriveBackLong();

}

void Auto\_ThreeTotesCenter(bool direction)

{

AutoFunc\_Strafe(direction);

AutoFunc\_GetFirstTote();

AutoFunc\_DriveBackShort();

AutoFunc\_Strafe(!direction);

AutoFunc\_GetTote();

AutoFunc\_DriveBackShort();

AutoFunc\_Strafe(!direction);

AutoFunc\_GetTote();

AutoFunc\_DriveBackLong();

}

void Auto\_DoNothing()

{

//Do Nothing

}

/\*

0. One Tote -> Auto\_OneTote();

1. Two Totes - Strafe Right -> Auto\_Strafe\_TwoTotes(true);

2. Two Totes - Strafe Left -> Auto\_Strafe\_TwoTotes(false);

3. Back Only -> AutoFunc\_DriveBackLong();

4. Three Totes Right -> Auto\_ThreeTotes(true);

5. Three Totes Left -> Auto\_ThreeTotes(false);

6. Three Totes Center Right -> Auto\_ThreeTotesCenter(true);

7. Three Totes Center Left -> Auto\_ThreeTotesCenter(false);

8. Do Nothing -> Auto\_DoNothing();

\*/

void ArcadeShift(double x, double y, double z)

{

double lf = 0, lr = 0, rf = 0, rr = 0; //lf = left front, lr = left rear, rf = right front, rr = right rear

double max = 0;

double deadband = 0.1;

if (fabs(y) >= deadband)

{

lf += y;

lr += y;

rf += y;

rr += y;

}

if (fabs(z) >= deadband)

{

lf += z;

lr += z;

rf -= z;

rr -= z;

}

if (fabs(x) >= deadband)

{

lf += x;

lr -= x;

rf -= x;

rr += x;

}

if (max < fabs(lf))

{

max = fabs(lf);

}

if (max < fabs(lr))

{

max = fabs(lr);

}

if (max < fabs(rf))

{

max = fabs(rf);

}

if (max < fabs(rr))

{

max = fabs(rr);

}

if (max > 1)

{

lf = lf/max;

lr = lr/max;

rf = rf/max;

rr = rr/max;

}

leftFront->Set(lf\*.75);

leftFrontCenter->Set(lf\*.75);

leftRear->Set(lr\*.75);

leftRearCenter->Set(lr\*.75);

rightFront->Set(-rf);

rightFrontCenter->Set(-rf);

rightRear->Set(-rr);

rightRearCenter->Set(-rr);

}

void TankShift(double leftX, double leftY, double rightX, double rightY)

{

//without deadband

//

//lf = ly + x;

//lr = ly - x;

//

//rf = ry - x;

//rr = ry + x;

//with deadband

double lf = 0, lr = 0, rf = 0, rr = 0; //lf = left front, lr = left rear, rf = right front, rr = right rear

double deadband = 0.1;

double max = 0;

if (fabs(leftY) >= deadband)

{

lf += leftY;

lr += leftY;

}

if (fabs(rightY) >= deadband)

{

rf += rightY;

rr += rightY;

}

//double strafeSpeed;

/\*

if (fabs(leftX) > fabs(rightX))

{

strafeSpeed = leftX;

}

else

{

strafeSpeed = rightX;

}

\*/

if (fabs(leftX) >= deadband)

{

lf += leftX;

lr -= leftX;

}

else if (fabs(rightX) >= deadband)

{

rf -= -rightX;

rr += -rightX;

}

/\*

if (fabs(strafeSpeed) >= deadband)

{

lf += strafeSpeed;

lr -= strafeSpeed;

rf -= strafeSpeed;

rr += strafeSpeed;

}

\*/

if (abs(lf) > max) {

max = abs(lf);

}

if (abs(lr) > max) {

max = abs(lr);

}

if (abs(rf) > max) {

max = abs(rf);

}

if (abs(rr) > max) {

max = abs(rr);

}

//normalize the wheel's speed values to fit in range of motor output (-1 to 1) if required

if (max > 1) {

lf = lf / max;

lr = lr / max;

rf = rf / max;

rr = rr / max;

}

leftFront->Set(.775\*lf);

leftFrontCenter->Set(.775\*lf);

leftRear->Set(.775\*lr);

leftRearCenter->Set(.775\*lr);

rightFront->Set(-rf);

rightFrontCenter->Set(-rf);

rightRear->Set(-rr);

rightRearCenter->Set(-rr);

}

void Wait(double time)

{

timer->Reset();

timer->Start();

while (timer->Get() < time) { }

timer->Stop();

}

};

START\_ROBOT\_CLASS(RobotDemo);