#include <CameraServer.h>

#include <IterativeRobot.h>

#include <cstdint>

#include "WPILib.h"

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/core/core.hpp>

#include <opencv2/core/types.hpp>

class Robot: public SampleRobot

{

private:

std::shared\_ptr<NetworkTable> roboRealm; //Network table object, communicate with RoboRealm

public:

Robot(); //Constructor

~Robot(); //Destructor

void OperatorControl();

void Autonomous();

void Test();

void Disabled();

};

Robot::Robot():roboRealm(NetworkTable::GetTable("SmartDashboard")) //Construct network object within the scope of Robot

{

printf("File %18s Date %s Time %s Object %p\n",\_\_FILE\_\_,\_\_DATE\_\_, \_\_TIME\_\_, this);

}

Robot::~Robot()

{

}

void Robot::Disabled()

{

printf("\nDisabled\n");

}

void Robot::OperatorControl() //standard driving and shooter control

{

}

void Robot::Test() //tests aligning with vision target

{

double imageWidth = roboRealm->GetNumber("IMAGE\_WIDTH", 320); //get image width

double xPosition, yPosition;

double distFromCenter;

while(IsTest() && IsEnabled())

{

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

// Set the resolution

camera.SetResolution(640, 480);

// Get a CvSink. This will capture Mats from the Camera

cs::CvSink cvSink = CameraServer::GetInstance()->GetVideo();

// Setup a CvSource. This will send images back to the Dashboard

cs::CvSource outputStream = CameraServer::GetInstance()->

PutVideo("Rectangle", 640, 480);

// Mats are very memory expensive. Lets reuse this Mat.

cv::Mat mat;

while (true) {

// Tell the CvSink to grab a frame from the camera and put it

// in the source mat. If there is an error notify the output.

if (cvSink.GrabFrame(mat) == 0) {

// Send the output the error.

outputStream.NotifyError(cvSink.GetError());

// skip the rest of the current iteration

continue;

}

// Put a rectangle on the image

rectangle(mat, cv::Point(100, 100), cv::Point(400, 400),

cv::Scalar(255, 255, 255), 5);

// Give the output stream a new image to display

outputStream.PutFrame(mat);

//xPosition=roboRealm->GetValue("COG\_X");

xPosition = SmartDashboard::GetNumber("COG\_X", 2);

yPosition = SmartDashboard::GetNumber("COG\_Y", 2);

//xPosition = roboRealm->GetNumber("COG\_X", -1.0); //get center of gravity of vision target

distFromCenter = imageWidth/2.0 - xPosition; //positive means object too far right, negative means too far left

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

SmartDashboard::PutNumber("DB/Number 0", xPosition);

if(xPosition != -1) //if set to default value, network table not found

{

if(distFromCenter > 15) //vision target too far right, spin right

{

// SmartDashboard::PutNumber("COG\_X", xPosition);

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

}

else if(distFromCenter < -15) //vision target too far left, spin left

{

// mRobot->ArcadeDrive(0.0,-0.40);

}

else

{

// mRobot->ArcadeDrive(0.0,0.0); //stop, centered within 15 pixels

printf("Centered! "); //lines up with xPosition print above

}

}

else

{

printf("Network table error!!!\n");

}

}

}

}

void Robot::Autonomous() //aligns with vision target then shoots

{

double imageWidth = roboRealm->GetNumber("IMAGE\_WIDTH", 320); //get image width

double xPosition;

double distFromCenter;

while(IsAutonomous() && IsEnabled())

{

xPosition = roboRealm->GetNumber("COG\_X", -1.0); //get center of gravity of vision target

distFromCenter = imageWidth/2.0 - xPosition; //positive means object too far right, negative means too far left

if(xPosition != -1) //if set to default value, network table not found

{

if(distFromCenter > 15) //vision target too far right, spin right

{

xPosition;

// mRobot->ArcadeDrive(0.0,0.2);

}

else if(distFromCenter < -15) //vision target too far left, spin left

{

xPosition;

// mRobot->ArcadeDrive(0.0,-0.2);

}

else

{

// Wait(0.5);

// mShooter->ShootBall(); //centered within 15 pixels, shoot ball

}

}

else

{

printf("Network table error!!!\n");

}

Wait(0.02);

}

}

START\_ROBOT\_CLASS(Robot)