There are still bugs (in regards to creating more than one robot and class) but I added the code for testing (which is unable to run at the moment)

#include "Header.hpp"

Robot\* robot = new Robot();

int timeStep = robot->getBasicTimeStep();

Motor\* leftMotor, \* rightMotor;

Camera\* colorSensor = robot->getCamera("colourSensor");

Camera\* rCam = robot->getCamera("RCam");

Camera\* lCam = robot->getCamera("LCam");

DistanceSensor\* distSensorRight = robot->getDistanceSensor("distanceSensorRight");

DistanceSensor\* distSensorLeft = robot->getDistanceSensor("distanceSensorLeft");

DistanceSensor\* distSensorFront = robot->getDistanceSensor("distanceSensorFront");

DistanceSensor\* distSensorBack = robot->getDistanceSensor("distanceSensorBack");

InertialUnit\* inertialSensor = robot->getInertialUnit("inertialSensor");

//Lidar\* lidarSensor = robot->getLidar("lidarSensor");

GPS\* gps = robot->getGPS("gps");

//Gyro\* gyro = robot->getGyro("gyro");

void delay(int ms)

{

// get the start time

float init = robot->getTime();

while (robot->step(timeStep) != -1)

{

if ((robot->getTime() - init) \* 1000 > ms)

break;

}

}

//gyro functions start

//#define PI 2\*asin(1)

//

//double angle = 0;

//

//void update\_gyro()

//{

// angle += (timeStep / 1000.0) \* (gyro->getValues())[1];

//}

//

//void set\_speed\_and\_delay(float LM\_speed, float RM\_speed, int delay\_time)

//{

// leftMotor->setVelocity(LM\_speed);

// rightMotor->setVelocity(RM\_speed);

// delay(delay\_time);

// if (delay\_time > 0)

// {

// leftMotor->setVelocity(0.0);

// rightMotor->setVelocity(0.0);

// }

//}

//

//void turn\_right\_gyro(float speed, float target\_degrees)

//{

// angle = 0;

// while ((angle \* 180 / PI) < target\_degrees)

// {

// set\_speed\_and\_delay(speed, -speed, 0);

// update\_gyro();

// }

//}

//

//void turn\_left\_gyro(float speed, float target\_degrees)

//{

// angle = 0;

// while ((angle \* 180 / PI) > (-1 \* target\_degrees))

// {

// set\_speed\_and\_delay(-speed, speed, 0);

// update\_gyro();

// }

//}

//gyro functions end

/\*Color getColor(int r, int g, int b)

{

if ((r > 150) && (g > 150) && (b > 150))

{

return White;

}

else if ((r < 40) && (g < 40) && (b < 40))

{

return Black;

}

else if ((r > 230) && (g < 50) && (b < 50))

{

return Red;

}

else if ((r < 50) && (g < 50) && (b > 230))

{

return Blue;

}

else if ((r > 100) && (g < 50) && (b > 180))

{

return Purple;

}

else if ((r > 80) && (g > 80) && (b > 80))

{

return Gray;

}

else if ((r > 150) && (g < 140) && (b < 80))

{

return Sand;

}

else

{

return Unknown;

}

}

\*/

/\*char getLetter(Mat roi)

{

int height = roi.rows;

int width = roi.cols;

Rect toprect(0, 0, width, height / 3);

Rect midrect(0, height / 3, width, height / 3);

Rect botrect(0, 2 \* height / 3, width, height / 3);

Mat topRoi(roi, toprect);

Mat midRoi(roi, midrect);

Mat botRoi(roi, botrect);

vector<vector<Point>> subContours;

vector<Vec4i>heirarchy;

findContours(topRoi, subContours, heirarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE);

int numTopContours = subContours.size();

subContours.clear();

findContours(midRoi, subContours, heirarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE);

int numMidContours = subContours.size();

subContours.clear();

findContours(botRoi, subContours, heirarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE);

int numBotContours = subContours.size();

if ((numTopContours == 2) && (numMidContours == 2) && (numBotContours == 1))

return 'U';

else if ((numTopContours == 1) && (numMidContours == 1) && (numBotContours == 1))

return 'S';

else if ((numTopContours == 2) && (numMidContours == 1) && (numBotContours == 2))

return 'H';

else

return '0';

}\*/

void stop()

{

leftMotor->setVelocity(0);

rightMotor->setVelocity(0);

delay(1);

}

void turn(int deg, Direction dir)

{

int rev = 1;

if (dir == Left)

{

rev = -1;

}

leftMotor->setVelocity(rev \* 5.0);

rightMotor->setVelocity(rev \* -5.0);

delay(5.0 \* deg);

stop();

}

void straight(int tiles, bool forward)

{

int rev = 1;

if (!forward)

rev = -1;

int power = 5.0;

leftMotor->setVelocity(power \* rev);

rightMotor->setVelocity(power \* rev);

delay(1200);

stop();

}

int main(int argc, char\*\* argv) {

RobotSensing robo; //work in progress

leftMotor = robot->getMotor("wheel2 motor");

rightMotor = robot->getMotor("wheel1 motor");

leftMotor->setPosition(INFINITY);

rightMotor->setPosition(INFINITY);

colorSensor->enable(timeStep);

rCam->enable(timeStep);

lCam->enable(timeStep);

distSensorLeft->enable(timeStep);

distSensorRight->enable(timeStep);

distSensorFront->enable(timeStep);

distSensorBack->enable(timeStep);

inertialSensor->enable(timeStep);

//lidarSensor->enable(timeStep);

gps->enable(timeStep);

leftMotor->setVelocity(0);

rightMotor->setVelocity(0);

robot->step(timeStep);

unsigned char r, g, b;

Mat original;

while (robot->step(timeStep) != -1)

{

Coordinate robot\_coords = robo.getCoords();

printf("Coordinates: (%lf, %lf, %lf \n)", robot\_coords.x, robot\_coords.y, robot\_coords.z);

double distance = robo.getDist(Front);

printf("Dist: %lf \n", distance);

//Color color = robot.getColor();

robo.getColor();

robo.getSign(Right);

char letter = robo.getLetter(Left);

//hazard\_detection();

//leftMotor->setVelocity(5);

//rightMotor->setVelocity(5);

//get color values

const unsigned char\* image = colorSensor->getImage();

r = colorSensor->imageGetRed(image, colorSensor->getWidth(), colorSensor->getWidth() / 2, colorSensor->getHeight() / 2);

g = colorSensor->imageGetGreen(image, colorSensor->getWidth(), colorSensor->getWidth() / 2, colorSensor->getHeight() / 2);

b = colorSensor->imageGetBlue(image, colorSensor->getWidth(), colorSensor->getWidth() / 2, colorSensor->getHeight() / 2);

const unsigned char\* img = rCam->getImage();

Mat frame(rCam->getHeight(), rCam->getWidth(), CV\_8UC4, (void\*)rCam->getImage());

original = frame.clone();

Mat hsv\_frame;

cvtColor(original, hsv\_frame, COLOR\_BGR2HSV);

cvtColor(frame, frame, COLOR\_BGR2GRAY);

threshold(frame, frame, 20, 255, THRESH\_BINARY\_INV);

vector<vector<Point>> contours;

vector<Vec4i> heirarchy;

findContours(frame, contours, heirarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE);

cout << "Number of contours: " << contours.size() << endl;

vector<Rect> boundRect(contours.size());

auto redcolor = Scalar(0, 0, 255);

for (int i = 0; i < contours.size(); ++i)

{

boundRect[i] = boundingRect(contours[i]);

rectangle(original, boundRect[i].tl(), boundRect[i].br(), redcolor, 2);

cout << boundRect[i].area() << endl;

if (boundRect[i].area() >= 270)

{

Mat roi(frame, boundRect[i]);

imshow("roi", roi);

//cout << "Letter: " << getLetter(roi) << endl;

}

else

{

cout << "too small\n";

}

}

Mat hsvThresh;

inRange(hsv\_frame, Scalar(0, 0, 60), Scalar(20, 255, 255), hsvThresh);

imshow("hsv", hsv\_frame);

imshow("hsv thresh", hsvThresh);

imshow("orig", original);

imshow("thresh", frame);

waitKey(1);

cout << "Location-xyz: (" << gps->getValues()[0] << ", " << gps->getValues()[1] << ", " << gps->getValues()[2] << ")" << endl;

}

robot;

return 0;

}