/\*

\* Robotics Navigation 2018

\*/

#define tooClose 1

#define tooFar 2

#define good 0

#define tooCounter 1

#define tooClock 2

#define tooLeft 1

#define tooRight 2

int phase = 1;

double sensors[14];

int treasureMap[] = {-1, -1, -1};

void setup() {

// setup code here; runs once

}

void loop() {

if(treasureMap[0] == -1) {

// Riley

treasureMap = readIRSensor();

return;

}

// perform every 50 ms

if(!is50ms()) {

return;

}

// Cailey

sensors = JSON.sensors;

if(phase == 1) {

toDestinationA();

} else if(phase == 2) {

centerOnRamp();

} else if(phase == 3) {

downRamp();

} else if(phase == 4) {

toDestinationBWall();

} else if(phase == 5) {

// driving over destination B in the process

toFlagWall();

} else if(phase == 6) {

centerOnFlag();

} else if(phase == 7) {

turnAround();

} else if(phase == 8) {

centerOnChest();

} else if(phase == 9) {

atopChest();

} else if(phase == 10) {

pickUpChest();

} else if(phase == 11) {

centerOnRamp2();

} else if(phase == 12) {

upRamp();

} else if(phase == 13) {

toDestinationA2();

} else {

// localization backup routine?

// just do not be here

}

}

void toDestinationA() {

// currently most numbers are arbitrary / filler values

// ID: back close L, desired: 40 mm, threshold: 8 mm

int spacing = hasSpacing(4, 40, 8);

// ID1: back close L, ID2: back close R, threshold: 8 mm

int parallel = isParallel(4, 5, 8);

if(spacing == tooClose) {

moveForward();

return;

}

if(spacing == tooFar) {

moveBack();

return;

}

// else spacing is good

if(parallel == tooCounter) {

rotateClock();

return;

}

if(parallel == tooClock) {

rotateCounter();

return;

}

// Reid's

if(isButtonPressed()) {

phase++;

return;

}

// else parallel and well-spaced to back wall but button not hit

if(treasureMap[0] == 0) {

moveLeft();

} else {

moveRight();

}

}

void centerOnRamp() {

// ID: back close, desired: 40 mm, threshold: 8 mm

int spacing = hasSpacing(4, 40, 8);

// ID1: back close L, ID2: back close R, threshold: 8 mm

int parallel = isParallel(4, 5, 8);

if(spacing == tooClose) {

moveForward();

return;

}

if(spacing == tooFar) {

moveBack();

return;

}

// else well-spaced from back wall

if(parallel == tooCounter) {

rotateClock();

return;

}

if(parallel == tooClock) {

rotateCounter();

return;

}

// else parallel to back wall

// ID1: left far F, ID2: right far F, threshold: 50 mm

spacing = isCentered(8, 10, 50);

if(spacing == tooLeft) {

moveRight();

return;

}

if(spacing == tooRight) {

moveLeft();

return;

}

// else centered and ready to approach ramp

phase++;

}

void downRamp() {

// ID1: back close L, ID2: back close R, threshold: 8 mm

int parallel = isParallel(4, 5, 8);

// still close to back wall?

if(sensors[4] < 200 && sensors[5] < 200) {

if(parallel == tooCounter) {

rotateClock();

return;

}

if(parallel == tooClock) {

rotateCounter();

return;

}

// else parallel too back wall

moveForward();

return;

}

// else really close to ramp or on ramp

// cannot check back wall due to angle of robot

// now read floor sensors

// ID1: floor L, ID2: floor R, threshold: 8 mm

parallel = isParallel(0, 1, 8);

// ID: floor L, desired: 100 mm, threshold: 25 mm

int spacingL = hasSpacing(0, 100, 25);

if(spacingL == tooFar) {

rotateClock();

return;

}

// ID: floor R, desired: 100 mm, threshold: 25 mm

int spacingR = hasSpacing(1, 100, 25);

if(spacingR == tooFar) {

rotateCounter();

return;

}

// else on solid ground

// are we close to chest?

// ID: front close, desired: 80 mm, threshold: 30 mm

int spacingF = hasSpacing(7, 80, 30);

if(spacingF == tooClose) {

phase++;

return;

}

// else on solid ground but not close enough to chest

moveForward();

}

// INSERT: other phase functions

int hasSpacing(int ID, float desired, float threshold) {

float diff = sensors[ID] - desired;

if(diff > threshold) {

return tooFar;

}

if(diff < -threshold) {

return tooClose;

}

return good;

}

int isParallel(int ID1, int ID2, float threshold) {

float diff = sensors[ID1] - sensors[ID2];

if(diff > threshold) {

return tooCounter;

}

if(diff < -threshold) {

return tooClock;

}

return good;

}

int isCentered(int ID1, int ID2, float threshold) {

// same as isParallel

// renamed to improve high level readability

float diff = sensors[ID1] - sensors[ID2];

if(diff > threshold) {

return tooLeft;

}

if(diff < -threshold) {

return tooRight;

}

return good;

}