// Are there obstacles around me?

// 1. What is around me?

if (!movementCommenced) // Only update obstacles while we are not in a

// special sequence of movements, to prevent a sequence from being

// prematuraly terminated

{

canDoMovement[0] = !left\_is\_obstacle;

canDoMovement[1] = distance\_value >= distance\_min;

canDoMovement[2] = !right\_is\_obstacle;

left = canDoMovement[0]; // left

straight = canDoMovement[1]; // straight

right = canDoMovement[2]; // right

// Quantify movement options

// one option

if ((!left && !straight && right) ||

(!left && straight && !right) ||

(left && !straight && !right)) {

numOptions = 1;

}

// 2 options

else if ((!left && straight && right) ||

(left && straight && !right) ||

(left && !straight && right)) {

numOptions = 2;

}

// 3 options

else if (left && straight && right) {

numOptions = 3;

}

// no options

else {numOptions = 0;}

}

// 2. If one option, do normal movement

if (numOptions == 1)

{

// normal movement logic

if (left)

{

turnLeftSequence();

}

else if (straight)

{

// go straight

moveForward();

}

else if (right)

{

turnRightSequence();

}

}

// else if more than one option, do intersection logic

else if (numOptions > 1)

{

// intersection logic

// assume we have encountered a new intersection.

if (!hitDeadEnd) {

intersectionsIndex ++; // move to the next item in the intersections array

// Create a new array in our intersections array

intersections[intersectionsIndex][0] = true; // set to active

intersections[intersectionsIndex][1] = left; // record obstacle

intersections[intersectionsIndex][2] = straight; // record obstacle

intersections[intersectionsIndex][3] = right; // record obstacle

// Make a decision on where to go.

int turn = 0;

if (left) {

turnLeftSequence();

turn = 1;

}

else if (straight) {

moveForward();

turn = 2;

}

else if (right) {

turnRightSequence();

turn = 3;

}

lastIntersectionTurnsIndex++; // increment position in array

lastIntersectionTurns[lastIntersectionTurnsIndex] = turn;

}

// We have recently hit a dead end and are returning to an intersection we've

// already seen

else {

// Do not increment the intersectionsIndex, because we assume it's the last

// intersection we've seen.

int lastTurn = lastIntersectionTurns[lastIntersectionTurnsIndex];

lastIntersectionTurns[lastIntersectionTurnsIndex] = 0; // remove last turn

// from the list because it leads to a dead end

lastIntersectionTurnsIndex --;

left = intersections[1];

straight = intersections[2];

right = intersections[3];

// orient ourselves

if (lastTurn == 1) { // if we turned left at the last intersection

left = straight; // our new left is our previous straight

straight = right; // our new straight is our previous right

if (left) {

turnLeftSequence();

hitDeadEnd = false;

}

else if (straight) {

moveForward();

hitDeadEnd = false;

}

else {

turnRightSequence();

// if we return from where we came, we want to keep hitDeadEnd active

if (!movementCommenced) { // we want this to occur only once

for (int i = 0; i < 4; i++) {

intersections[intersectionsIndex][i] = false;

}

intersectionsIndex--;

}

}

}

else if (lastTurn == 2) { // if we went straight at our last intersection

right = left;

// we shouldn't be able to turn left because we just went forward,

// since we always prioritize going left first

if (right) {

turnRightSequence();

hitDeadEnd = false;

}

else { // only return from where we came if it's the last option

moveForward();

// if we return from where we came, we want to keep hitDeadEnd active

if (!movementCommenced) { // we want this to occur only once

for (int i = 0; i < 4; i++) {

intersections[intersectionsIndex][i] = false;

}

intersectionsIndex--;

}

}

}

else if (lastTurn == 3) { // if we went right at our last intersection

// in this case, we know that the only passable direction is to our

// left. we must remove the last intersection off the stack, decrement

// its index, and also remove the lastIntersectionTurns item off the

// stack and decrement its index.

turnLeftSequence(); // return from where we came

// if we return from where we came, we want to keep hitDeadEnd active

if (!movementCommenced) { // we want this to occur only once

for (int i = 0; i < 4; i++) {

intersections[intersectionsIndex][i] = false;

}

intersectionsIndex--;

}

}

}

}

// This means we have encountered a dead-end, and need to turn around.

else if (numOptions < 1) {

// Record that we've hit a dead end

hitDeadEnd = true;

// Turn around

if (movementCommenced == false) {movementCommenced = true;} // tell the

// rest of our program that we are now in a turn sequence which should

// not be interrupted.

// Turn sequence

if (!movementSequenceInitialized) {

movementSequenceStartTime = millis();

movementSequenceInitialized = true;

}

// for the first 3 seconds, turn left

if (millis() - movementSequenceStartTime < 3000)

{

motion\_mode = TURNLEFT;

turn\_count --;

}

// pause for one second

if (millis() - movementSequenceStartTime < 4000) {

motion\_mode = STANDBY;

}

// once the turn sequence is finished, tell the rest of our program to

// resume as normal.

else {

movementSequenceInitialized = false;

movementCommenced = false;

}

}