**COSC343 – Assignment 1:**

**Robots on a chessboard**

Task one was broken up into two stages that consisted of several different parts:

**Stage1:**

1. Start at a tile, move towards the black and white tiled lane.
2. Travel in the lane.
3. Count the black tiles, making a sound at each new one.
4. Stop after encountering 15 black tiles and turn right

Stage 1was relatively easy to complete, as the robot relies entirely on the light sensor to determine where to move. The raw code can be found in the appendix, but the basic algorithm is as follows:

* A global variable *totalBlackTile* keeps a count of the number of black tiles passed, and is updated during the course of the program.
* The function *moveForwardWithSpeed()* takes an argument *movingSpeed*, sets both motors to run at that speed, and continues until it detects a colour change from white to black. *totalBlackTile* is incremented and the robot beeps.
* The *task main{}* calls *moveForwardWithSpeed()* to move off the starting tile and onto the line of black and white tiles, then calls the *turnOneMotor()* function to turn the robot 90­o
* While *totalBlackTile* is less than or equal to 15, the *task main{}* repeatedly calls *moveForwardWithSpeed()* followed by the *testBoundary()* function. There is also a global variable *difference*, which stores the difference between two colours detected by the light sensor; if this value is less than 20 (i.e. If the robot is close to the middle of the black tile) it saves time by moving forward two tiles instead of one.

One problem we faced with stage 1 was that the inconstancy of the lighting in the testing area. The weather, time of day and the lights themselves had an effect on the input of the robot’s light sensor. At first we were using the light sensor’s reflected light function, using the built-in *getColorReflected()* function which returns a value from a red light between 0 and 100; taking the difference between our “current” and “previous” variables we could determine when the robot had moved from one tile to the next one of a different colour. We discovered however that it was easier and more accurate to use the *getColourName()* function, which uses a more complex RGB light sensor to return an object representing one of eight colours, since it does a much better job of differentiating between black and white.

Another problem we faced was programming the robot to travel in a straight line. The robot’s wheels tended to get caught in the grooves on each side of the row of tiles, which would cause it to veer off in one direction. At first we changed the code so that the left wheel would just go slightly faster than the right (since the direction was always the same), but clearly this solution wasn’t very robust.

Eventually we came up with a function called *testBoundary()* to solve the problem:

* *testBoundary()* relies on the function *turnOneMotorUntilMeetBoundary()*, which simply rotates the robot by driving a given motor until the sensor detects a colour change and returns the degree the wheel has turned by. *testBoundary()* calls this function once for each wheel and stores the returned value in two variables. The function then takes the absolute value of the difference and uses this to correct the path of the robot, turning a certain distance in the opposite direction.

Hence stage 1 is a looped operation of measuring and correcting the robots path on each black tile. The algorithm is very simple: the robot continues in a straight line until it reaches a black tile, then stops. It checks its position in the black tile relative to the edges using *testBoundary()*, which also corrects the direction in which the robot is travelling if needed.

**Stage2:**

1. The robot should move forward towards the tower placed on the finish tile.
2. It should make contact the target
3. Push tower off the finish tile, making a sound to indicate it has finished.

Stage 2 of the task was harder, as we decided to use the sonar sensor to adjust the path of the robot if needed.

* After stage 1, the robot uses *turnOneMotor()* to turn 90o to the right so that it faces the end tile and moves forward for a few seconds
* While the two bumpers have not hit anything (i.e. while they return a value of 0), the *moveForwardWithSpeed()* function is called followed by the *scan()* function
* *scan()* simply turns the robot in each direction by calling the *turn()* function
* *turn()* takes as parameters a maximum degree by which the wheels should be rotated (*degreeToTurn*) and a turning speed. It rotates the robot in one direction if *degreeToTurn* is positive and the opposite if negative, taking readings from the sonar sensor the whole time; it returns from the function when the distance it reads from the sensor gets smaller, meaning the robot will end up facing the tower on top of the end tile. The actual degree the robot turns is affected by the global variable *targetDistance*, which is set to the current distance to the target each time. This is to avoid accidentally reading a different target (for example a bystander’s leg); the robot will only face towards the object it has detected if the difference between it and the object is less than *targetDistance* – meaning the robot is getting closer. *scan()* uses a boolean value set by *turn()* to determine when to stop rotating – this should result in the robot facing the end tile in all cases as it will keep moving straight forward if no object is detected.

In theory the *scan()* and *turn()* functions are not actually needed, as the robot could simply drive forward until its bumpers detect a collision, but given our experience with the robot accidentally changing course in stage 1 we decided it was better not to take chances.

In terms of the execution of the program, we tried to use ROBOTC’s multitasking since it seemed like a good way to get the robot moving and correcting its path simultaneously. However as we added more functions and the program got more complex, this method turned out to be error-prone and hard to debug. In the end we decided to encapsulate most of the main algorithms in functions (such as *turn()* and *moveForwardWithSpeed()*) to make everything easier.

Possibly the biggest problem we faced was debugging our program, especially the functions dealing with identifying black and white tiles, partly due to the fact that there is only a small amount of space on the LED screen to display lines of text. Essentially this meant it was only possible to debug very small parts of code at a time, which resulted in hours spent on testing the robot’s performance.

|  |
| --- |
| #pragma config(Sensor, S1, leftBumper, sensorEV3\_Touch) |
| #pragma config(Sensor, S2, rightBumper, sensorEV3\_Touch) |
| #pragma config(Sensor, S3, colorSensor, sensorEV3\_Color, modeEV3Color\_Color) |
| #pragma config(Sensor, S4, sonarSensor, sensorEV3\_Ultrasonic) |
| #pragma config(Motor, motorB, leftMotor, tmotorEV3\_Large, PIDControl, driveLeft, encoder) |
| #pragma config(Motor, motorC, rightMotor, tmotorEV3\_Large, PIDControl, driveRight, encoder) |
| //\*!!Code automatically generated by 'ROBOTC' configuration wizard !!\*// |
|  |
| //--------------------- GOLOBAL VARIABLES------------------------- |
| int totalBlackTile = 0; |
| int MOVINGSPEED = 30; |
| float targetDistance = 255.0; |
|  |
| // variables used in scan() |
| bool RIGHT = true; |
| bool LEFT = false; |
| bool approachingTarget = false; |
| int TURNINGSPEED = 10; |
|  |
| // variables used in testBoundary() |
| int MAX\_BOUNDARY\_EXPECTED = 4\*100; |
| int rightBoundaryDegree = 0; |
| int leftBoundaryDegree = 0; |
| int differences = 30;  //------------------------------------------------------------------ |
|  |
|  |
| /\*\*DEFINED FUNCTION \*/ |
|  |
| /\*\*  \* This function has two uses. |
| \* The first is for turning and recording the distance it gets from the sonar sensor. |
| \* It achieves this by rotating on its own axis with one wheel rotating forward and the  \* other backwards |
| \*  \* The second is during scanning it continues to get the distance directly in front of.  \* When the distance decreases and then begins to increase again the robot will stop  \* turning, and targetDistance is updated.  \*  \* ----Parameters----  \* int degreeToTurn 🡪 how much to turn the robot by.  \* int TURNINGSPEED 🡪 the motor speed for turning  \*/ |
|  |
|  |
|  |
| void turn(int degreeToTurn, int TURNINGSPEED){ |
| float previousDistance = getUSDistance(sonarSensor); |
| float currentDistance = getUSDistance(sonarSensor); |
|  |
| degreeToTurn = degreeToTurn \* 2; |
| resetMotorEncoder(leftMotor); |
| resetMotorEncoder(rightMotor); |
| if (degreeToTurn >= 0) { |
| setMotorTarget(leftMotor, degreeToTurn, TURNINGSPEED); |
| setMotorTarget(rightMotor, -degreeToTurn, TURNINGSPEED); |
| while(getMotorEncoder(leftMotor)!=getMotorTarget(leftMotor) && getTouchValue(leftBumper)!=1 && getTouchValue(rightBumper)!=1) { |
| setMotorSync(leftMotor, rightMotor, 100, TURNINGSPEED); |
| currentDistance = getUSDistance(sonarSensor); |
| if (currentDistance<previousDistance) { |
| previousDistance = currentDistance; // it means it is turning toward to target; |
| approachingTarget = true; |
| } else if (approachingTarget && currentDistance > previousDistance && previousDistance < targetDistance) { |
| targetDistance = previousDistance; |
| return; |
| } |
| } |
| } else if (degreeToTurn <= 0) { |
| degreeToTurn = -1 \* degreeToTurn; |
| setMotorTarget(leftMotor, -degreeToTurn, TURNINGSPEED); |
| setMotorTarget(rightMotor, degreeToTurn, TURNINGSPEED); |
| while(getMotorEncoder(rightMotor)!=getMotorTarget(rightMotor)&& getTouchValue(leftBumper)!=1 && getTouchValue(rightBumper)!=1) { |
| setMotorSync(leftMotor, rightMotor, -100, TURNINGSPEED); |
| currentDistance = getUSDistance(sonarSensor); |
| if (currentDistance<previousDistance) { |
| previousDistance = currentDistance; // it means it is turning toward to target; |
| approachingTarget = true; |
| } else if (approachingTarget && currentDistance > previousDistance && previousDistance < targetDistance) { |
| targetDistance = previousDistance; |
| return; |
| } |
| } |
| } |
| }//end turn function |
|  |
|  |
| /\*\* |
| \* This function first rotates 45 degree to its left then 90 degree to its right and  \* then 45 degrees to left once more.  \* It does this by calling the turn() function and sending it a turn degree and speed. |
|  |
|  |
| \* If at any point during these three rotations targetDistance in the turn() function is  \* updated, turning stops and the scan() function is exited. |
| \*/ |
| void scan() { |
| approachingTarget = false; |
| int scanDegree = 45; |
| if(getTouchValue(leftBumper)==1 || getTouchValue(rightBumper)==1){ |
| return; |
| } else { |
| turn(-scanDegree, TURNINGSPEED); |
| if (approachingTarget) {return;} |
| turn(scanDegree\*2,TURNINGSPEED); |
| if (approachingTarget) {return;} |
| turn(-scanDegree, TURNINGSPEED); |
| } |
|  |
| }//end scan function |
|  |
| /\*\* |
| \* This function is a helper function used for turning the robot.  \* A value of true is the rightMotor and false is leftMotor for leftMotorOrRightMotor  \* value, and a negative motorEncoder value results in the motor moving backwards.  \*  \*-----Parameters----- |
| \* bool leftMotorOrRightMotor 🡪 indicates which wheel you want to turn. |
| \* int moterEncoder 🡪 is the encoder value you want to turn. |
| \*/ |
| void turnOneMotor(bool leftMotorOrRightMotor, int motorEncoder) |
| { |
|  |
|  |
|  |
| resetMotorEncoder(leftMotor); |
| resetMotorEncoder(rightMotor); |
|  |
| if(leftMotorOrRightMotor) { // RightMotor move while LeftMotor stop |
| if (motorEncoder > 0) { // move forward |
| setMotorTarget(rightMotor, motorEncoder, TURNINGSPEED); |
| while(getMotorEncoder(rightMotor)!=getMotorTarget(rightMotor)) { |
| setMotorSync(leftMotor, rightMotor, -50, TURNINGSPEED); |
| } |
| } else if (motorEncoder<0) { // move backward |
| setMotorTarget(rightMotor, -motorEncoder, TURNINGSPEED); |
| while(getMotorEncoder(rightMotor)!= -1\*getMotorTarget(rightMotor)) { |
| setMotorSync(leftMotor, rightMotor, -50, -TURNINGSPEED); |
| } |
| } |
| } else { // LeftMotor move while RightMotor stop |
| if (motorEncoder > 0) { |
| setMotorTarget(leftMotor, motorEncoder, TURNINGSPEED); |
| while(getMotorEncoder(leftMotor)!=getMotorTarget(leftMotor)) { |
| setMotorSync(leftMotor, rightMotor, 50, TURNINGSPEED); |
| } |
| } else if (motorEncoder < 0) { |
| setMotorTarget(leftMotor, -motorEncoder, TURNINGSPEED); |
| while(getMotorEncoder(leftMotor)!= -1\*getMotorTarget(leftMotor)){ |
| setMotorSync(leftMotor, rightMotor, 50, -TURNINGSPEED); |
| } |
| } |
| } |
| }//end turnOneMotor function |
|  |
| /\*\*  \* This function moves one wheel of the robot forward until the colorSensor detects a  \* colour change, storing the motor encoder value in a variable.  \* It then moves back to its previous position and returns the variable. |
|  |
| \*  \* -----Parameters-----  \* bool leftMotorOrRightMotor 🡪 indicates which wheel you want to turn. |
| \*/ |
| int turnOneMotorUntilMeetBoundary(bool leftOrRightMotor) { |
| resetMotorEncoder(leftMotor); |
| resetMotorEncoder(rightMotor); |
|  |
| int saved\_leftBoundaryDegree = 0; |
| int saved\_rightBoundaryDegree = 0; |
|  |
| if(leftOrRightMotor) { |
| // turn RightMotor |
| setMotorTarget(rightMotor, MAX\_BOUNDARY\_EXPECTED, TURNINGSPEED); |
| while(getMotorEncoder(rightMotor)!=getMotorTarget(rightMotor) && getColorName(colorSensor)==colorBlack) { |
| setMotorSync(leftMotor, rightMotor, -50, TURNINGSPEED); |
| } |
|  |
| saved\_leftBoundaryDegree = getMotorEncoder(rightMotor); |
| turnOneMotor(RIGHT, -1\*saved\_leftBoundaryDegree); |
| return saved\_leftBoundaryDegree; |
|  |
| } else { |
| setMotorTarget(leftMotor, MAX\_BOUNDARY\_EXPECTED, TURNINGSPEED); |
| while(getMotorEncoder(leftMotor)!=getMotorTarget(leftMotor) && getColorName(colorSensor)==colorBlack) { |
| setMotorSync(leftMotor, rightMotor, 50, TURNINGSPEED); |
| } |
|  |
| saved\_rightBoundaryDegree = getMotorEncoder(leftMotor); |
| turnOneMotor(LEFT, -1\*saved\_rightBoundaryDegree); |
| return saved\_rightBoundaryDegree; |
| } |
| }//end turnOneMotorUntilMeetBoundary function |
|  |
| /\*\*  \* This function calls the turnOneMotorUntilMeetBoundary() function for both motors and  \* tests the black tile's boundary. |
| \* The two encoder variables returned from the turnOneMotorUntilMeetBoundary() function are \* compared and the difference used to adjust the robots position on the black tile. |
| \*/ |
| void testBoundary() { |
| leftBoundaryDegree = turnOneMotorUntilMeetBoundary(RIGHT); |
| rightBoundaryDegree = turnOneMotorUntilMeetBoundary(LEFT); |
| differences = fabs(rightBoundaryDegree - leftBoundaryDegree); |
| if(leftBoundaryDegree>rightBoundaryDegree) { |
| turnOneMotor(RIGHT, differences/4); |
| // turn(-differences/20); |
| } else { |
| turnOneMotor(LEFT, differences/4); |
| //turn(differences/20); |
| } |
|  |
| }//end testBoundary function |
|  |
| /\*\* \* This function moves the robot forward until the colorSensor detects a change in colour  \* indicating the next black tile.  \* -----Parameters-----  \* int movingSpeed 🡪 the motor speed  \*/ |
| void moveForwardWithSpeed(int movingSpeed) { |
| TLegoColors previousColor = getColorName(colorSensor); // it should be black at start |
| TLegoColors currentColor = getColorName(colorSensor); // it should be black at start |
| int count = 0; |
| while(count<1 && getTouchValue(leftBumper)==0 && getTouchValue(rightBumper)==0){ |
| motor[leftMotor] = movingSpeed; |
| motor[rightMotor] = movingSpeed; |
| currentColor = getColorName(colorSensor);  // now robot enters the white tile. |
| if (currentColor == colorWhite) { |
| previousColor = colorWhite; |
| } else if(currentColor == colorBlack && previousColor == colorWhite) { |
| count++; |
| previousColor = colorBlack; |
| totalBlackTile++; |
| playSound(soundUpwardTones); |
| } |
| } |
| motor[leftMotor] = 0; |
| motor[rightMotor] = 0; |
| sleep(500); |
| }//end moveForwardWithSpeed function  /\*\*  \* Function performs the first part of Stage 1, moving the robot to the black and white  \* tile line.  \*/ |
|  |
|  |
| void moveToLine(){ |
| moveForwardWithSpeed(MOVINGSPEED); |
| turnOneMotor(RIGHT, 330); |
| }//end moveToLine function |
| /\*\* \* This function simply uses the setMotorSyncTime() to move the robot.  \* This function is used for moving the robot backwards one tile between the transition from \* to stage 1 and stage 2. It is also used at the end of stage 2, to push over the tower.  \* -----Paramters----- \* int movingSpeed 🡪 the motor speed  \* int movingTime 🡪 how long you want the robot to move for  \* int withRatio 🡪 how much you want the two motors synced  \*/ |
| void moveWithSpeedAndTime(int movingSpeed, int movingTime, int withRatio) |
| { |
| setMotorSyncTime(leftMotor, rightMotor, withRatio, movingTime\*1000, movingSpeed); |
| wait1Msec(movingTime\*1000); |
| }//end moveWithSpeedAndTime function |
|  |
| task main() |
| { |
| //stage 1 |
| moveToLine(); |
| while(totalBlackTile<15) { |
| if (differences<20 && totalBlackTile<14) { |
| moveForwardWithSpeed(MOVINGSPEED); |
| moveForwardWithSpeed(MOVINGSPEED); |
| testBoundary(); |
| } else { |
| moveForwardWithSpeed(MOVINGSPEED); |
| testBoundary(); |
| } |
| } |
|  |
| // End of Stage 1 |
|  |
| /\*\*Start of stage 2:\*/ |
| moveWithSpeedAndTime(-1\*MOVINGSPEED, 1, 0); |
|  |
| turnOneMotor(LEFT, 345); |
| moveWithSpeedAndTime(MOVINGSPEED, 4, 0); |
|  |
|  |
| while (getTouchValue(leftBumper)==0&&getTouchValue(rightBumper)==0){ |
| moveForwardWithSpeed(MOVINGSPEED); |
| scan();  //if robot is close to tower, slow turning speed and therefore scan() to make it more accurate |
| if (targetDistance < 100) { |
| TURNINGSPEED = 7; |
| } |
| } |
|  |
| // After it making contacts with targets |
| moveWithSpeedAndTime(-1\*MOVINGSPEED, 2, 0); // moving back for 2 seconds |
| moveWithSpeedAndTime(3\*MOVINGSPEED, 2, 0); // then full speed and push 2 seconds |
| moveWithSpeedAndTime(-1\*MOVINGSPEED, 1, 0); // moving back for 1 second |
| moveWithSpeedAndTime(MOVINGSPEED, 2, 100); //turning for 2 seconds |
|  |
| playSound(soundUpwardTones); |
| wait1Msec(1000); |
| }//end task main |