COSC343 Assignment 01 – Robotics

Group 15 – Hayden, Magdeline, Maaha, Jack

Algorithm: *https://github.com/Gamma749/COSC343-EV3-Bot*

* Make robot go
* Mhhhmnmm….……….. **warm**
* We created a *Robot* class that abstract-ed a lot of the details/parameters for the robot. This allows us to program an abstract method of finding the tower, without having to constantly add checks for events like crossing black squares.
* We created *BlackSquareSensor* class, allowing the robot to continually sense it’s environment on a separate thread, allowing parallel processes, such as “driving” and “checking” occur simultaneously. This is divided into the following attributes:

### Table 1: List of *BlackSquareSensor* attributes

|  |  |
| --- | --- |
| *VALUE\_LIST* | The list of values that have been read |
| *VALUE\_LIST\_LOC* | A *Lock* object from the *Threading* class, so we can be sure we don't run into race conditions when writing/taking averages |
| *CONSTANT\_READ* | A Boolean to check if we are to be constantly reading (*Thread* termination condition) |
| *CURRENT\_INDEX* | The current index into the *VALUE\_LIST* array that we are writing to |
| *THRESHOLD* | The threshold that must be surpassed to be on a white square |
| *SENSOR* | The sensor to read from |
| *THREAD* | A reference to the thread |

* We have also decided that to simplify the representation of the environment into cartesian coordinates which represent the position of the robot, and black tiles, with integer values, based on the grid space of a 15x8 layout. For example, when the light sensor is exactly over black square 1 we are at coordinate *(0,0).* At black square 34 we are at *(3,2)*, given we are on the second tile/column of the third row.
  + This also allows us to very easily translate our 2 dimensional location back into a 1 dimensional location on the list [1:120], through the translation .
* We decided to restrict the basic movement options to a) moving forward; *robot.move* and b) turning only at right angles; *robot.rotate*. This reduced the possible points of failure, that can be induced by more complex logic. While saying this, we do have more complex methods that allow for correction, however these are siloed into separate callable functions, limiting them to only coming into play when needed.
* The threading is implemented as follows:

|  |
| --- |
| main thread  |  |  Create robot  | Sensor Thread  |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  | |  | |  Robot Moves, Constantly Read from light sensor  Rotates on main and update the value array  |  Read from value array  |  Take average of the array |

* *robot.move* was implemented to move the robot forward like a tank (dual wheel drive) until it hits a black square, at which point, update the position of the robot, using the cartesian state *[x,y]*. This is implemented in the following way:
  + While the reflectivity array’s average is above our *threshold* for “black”, drive is continued.
  + Once the average value drops below *threshold*, the movement is halted, and the robot report’s it’s position.
  + A correction is made to the angle of exit, to account for our angle of arrival.
* **CORRECTION METHOD EXPLAINED HERE**

Problems

* Wheels being consistent with each other;
  + This can lead to the robot turning a wide arc when it thinks its driving a straight line, as one wheel rotates faster/has more grip than the other.
    - To correct for this, we added a correction that allows us to evaluate and adjust for the angle which we may be off from, when leaving a black tile. Rather than adjust to come out of the black tile at a perpendicular angle, we decided to do reflective angles, which progressively decrease, such that our deviation asymptotically approaches 0°. Without doing this, we could eventually drift off the side of the tile grid, into the void.
  + This also affects spot turns, leading to non-90° turns
* Grey lines having a reflectivity close to that of the black tiles. This meant we couldn’t use instantaneous readings, as the grey lines would terminate robot drive methods. To address this we created an array of readings, to which reflectivity reading were stored, the average of which was returned, and used to determine if we were over a black square.
  + This approach also made our sensor/check method robust to noise from tile texture and shadows on the tiles, as this noise would not draw the average down low enough to effect robot termination.
* Touch sensors being non-effective for this problem. As we were not allowed to push the tower out of the square, the travel velocity of the robot, the elasticity of both the tower and the “spring” loaded touch sensor, meant that the touch sensor would either not fire long enough to trigger, or bounce the tower away from us.
  + As such, we changed our detection method to be using the sonar sensor, rather than using the touch sensor. To avoid too much excess noise from interfering with the sonar sensor, we kept the distance threshold fairly low, which also allows for higher accuracy.