COSC343 Assignment 1 Group Report

Algorithm

The robot begins on the start tile, facing in the direction of the arrow. The optimal starting position is having the robot's colour sensor placed under the arrow's tip.

The robot drives forwards a set amount to reach the black tile that is immediately in front of it. We then rotate the robot a hard-coded amount to face the direction (in this instance, to the right) that will allow the robot to travel horizontally from tile one to ten. This ensures that it does not detect the start point as a black tile, and also gives us a fairly predictable starting direction and location. The reason we start with the colour sensor placed under the arrow's tip is so the hard-coded value is always as correct as possible.

The robot moves two hard-coded value rotations (0.58 of a rotation) each time, which is defined such that it will also touch both a black and a white tile every move. Robot proceeds to scan for colour values between a certain range - i.e., less than a colour value of 15 for a black tile. If it finds a black tile, it stops, beeps, speaks the name of the tile it thinks it is currently on (a value that is being incremented each time it finds a new tile), and then continues driving. This allows us to constantly keep tracking where the robot is located, and was crucial for error checking. We also thought that a beep by itself was not indicative enough of where the robot is located, so we wanted it to speak as well.

If a robot misses a black tile on its second move, the robot will proclaim it is lost, and proceed to scan the area by rotating. If it detects black with its colour sensor, it will be back on track, and after driving for a few moments, will make an attempt to quickly rotate the wheels to correct itself, before continuing to the next black (if the tile that it found is 10, instead of driving to the next tile, it will reverse back to 10). If it does not detect black, it will increase the radius of its turns. If it still cannot find black, it will reverse a certain distance, and search again, rotating in ever larger increments. If it still cannot find black, it will rotate a little bit, changing it from its original course, and then begin driving on in an attempt to find black again.

When the robot successfully reaches tile 10, it will make a rotation downwards towards the tower. It will then begin to travel downwards, attempting to hit every fifteenth tile - i.e., from tile ten, it should hit tile 25, then tile 40, tile 55, and tile 70.

If the robot misses any of those tiles (tile 25, then tile 40, tile 55, and tile 70), then it will begin error correction, calling the down_fix() function. It will exclaim that it has missed a tile, and reverse back a set distance (presumably the position of the last black tile). On the black tile, it will speak the number of that tile to indicate that it knows where it is. From there, it will rotate first to the right (0.03 of a rotation) and drive to see if the next black tile is there. If it is not, it reverses back to the black tile, and then rotates back to its original front-facing position, then

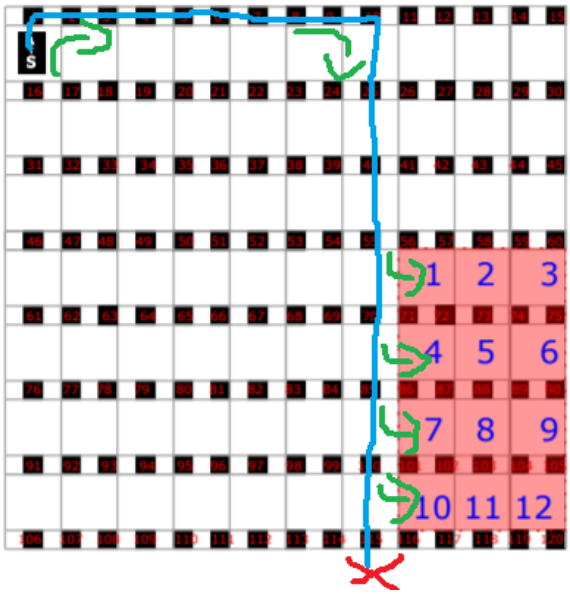
back around to the left, adding another 0.03 of a rotation, and tries to find the tile there. If it still hasn't found the black tile for whatever reason, it continues the process until it does find a black tile, switching between left and right with increased rotations as the algorithm continues. If it drives down a gray line, missing both white and black, it will time out and reverse back to the black tile, and turn again to avoid missing either tile again.

When it reaches tile 55, it will travel onwards for a bit longer, and then proceed to turn down one of the “alleyways”, e.g. tiles 1 to 3. If it finds the tower, we have an equation to guess its location based on how far away the distance detected is and the black tile number it is up to (or the “number of searches”) as it travels downwards. For example, if the robot is at tile 70 (so the number of searches is 2) and the distance is 40-70cm (the second furthest distance in each alleyway, so the offset is 1), it will conclude that the tower is on tile 5 (3 * number of searches - offset). If the distance was greater than 70cm, then the offset would be 0.

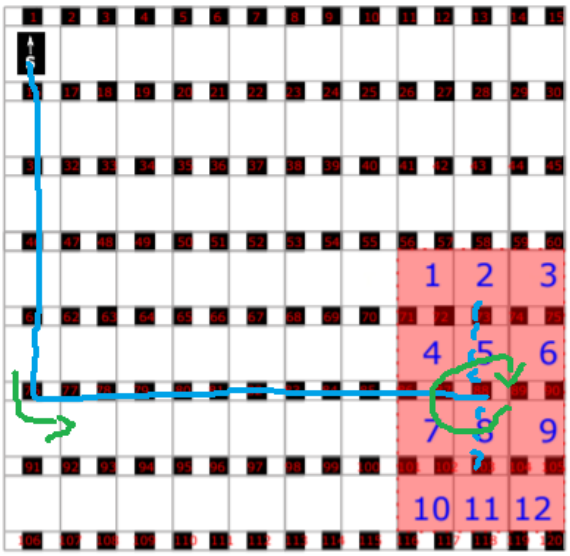
This equation allows us to avoid relying on the robot’s turning, movement and touch sensors. We discovered that the turning especially was a difficult part to get right, so we wanted to avoid movement after a turn if at all possible. We can instead have the robot accurately report the location from a distance. This part of the algorithm also has a maximum range, which means that the robot cannot accidentally detect the wall or any other object that might be considered incorrect, and give us an incorrect final guess.

If it does not detect the tower in the first alley of tiles, the robot remains in a while loop. It will do four separate searches as described above, each time reporting whether or not it has found the tower, and if it did, where the tower is. If the robot has reached the end of the searches and not found the tower, the algorithm will have failed.

Paths



The pathway shown by the left image allows us to demonstrate our program and the error checking our robot can do. We constructed it around the idea that it should be fully autonomous when trying to find the milk bottle. After tile 55, it would sit in between the black tiles and scan down the line of the possible area where the milk bottle could be. From there, if the robot found the milk bottle, depending on how far away from the robot it is, the robot would figure out which tile the milk bottle is on and report back. If it didn’t find the milk bottle it would continue on checking each possible tile until it had either found the bottle or checked all possible tiles. As stated in the algorithm section, if it reaches the end of this sequence, it has failed, and aborts the program.



The diagram on the left shows another path we decided to look at, which would take us directly to the milk bottle area, from there the robot would sit on Tile 88 and do 90 degree rotations. While in those rotations it would search for the bottle, if the robot found the tower, it would stop turning and begin to work its way towards the bottle by simply accelerating towards the point the tower was detected. This idea was eventually decided against because it would require a set of very accurate turns in a very repetitive fashion. In addition, issues with the speed of rotation and where it stops (in other words, we asked the robot to stop at a certain point but it takes a little longer for certain actions to be processed by the robot), have been prevalent in other areas of the program, making it a very risky potential move for the robot.

Problems

We supplied dates with the problems to indicate when we first discovered it, and the progress we made with overcoming these problems.

Key:
Green = resolved
Gold = partially resolved
Red = unresolved

- One of the issues that we had as a fairly recurring problem was that the ambient light values were giving us occasionally odd ranges, meaning we had to clearly define the color value that we were checking black, white and grey tiles for.
    - 18/3: Checking cl values around the time of the testing to update and check values according to light levels
    - 25/3: robot continually detected wrong color value when attempting to reverse during error correction, when travelling vertically "down" towards the tower.
    - 28/3: A time out ability has been added to the follow_path_down() and down_fix() functions, so even if the robot encounters a colour value between white and black, it will eventually stop after a generous amount of time (8 seconds if not correcting, 7 seconds if correction was required) and reverse back a set distance to realign itself.
- Miscounting tiles

    - 18/3-19/3: As at one stage the robot was going too far/too short from one tile to another, giving us inconsistent tile values (i.e. counting one tile twice, missing a tile it should have counted), we decided to tinker with the exact values of how far it drives.
    - 20/3: Modified the number of rotations to get from one tile to another so that the wheels did 58/100 of a rotation.
    - 21/3: At this stage we still had the problem that the robot can count the same tile twice, which means that we get an incorrect distance travelled, which throws everything off, making entire test runs invalid.
    - 26/3: Resolved by making the robot sleep after reporting a tile
- The robot often did not stop when speaking/stopped, which meant that it ended up doing a lot of things out of order, or lost synchronization.
    - 19/3: Fixed by making the robot turn off its motors every time it needed to speak
- Robot rotations can sometimes be off, causing the robot to go off course / Rotating specific angles e.g. a perfect 90 degrees
    - 19/3: We could either find a way to get more accurate turns or try and get it to fix itself
    - 20/3: Course correction was implemented, where if it doesn't sense another black tile after 2 tries, the robot will rotate left and right to search for a tile. Failing that, it will reverse and try rotating left and right again, and will repeat this until it finds another black tile.
    - 19/3: Testing gyro sensor to make perfect turns, but a gyro sensor is not installed on any robots used in the assignment
    - 19/3: Tried calibrating the robot by having it rotate 360 degrees, using the arrow on the start tile as a reference point. Success was reliant on the robot's initial position and direction.
        - 21/3: Sometimes the robot overshoots rotation, but that was when we forgot to turn off the motor when it had already rotated 360 degrees.
    - 20/3: worked out an equation to try and figure out the exact degree that the wheels need to spin
        - WheelDegrees = (HalfDistanceBetweenWheels * DegreesToRotateBot) / WheelRadius
            - To figure this I [Sam] used the equation:
            - WheelRadius * AngleWheelMoves = HalfDistanceBetweenWheels * Angle robot to turn
            - This is a simplified equation based on me figuring out the circumference of the wheels and the circumference of the robot rotation and seeing how many rotations the wheel had to do to move the distance of the robot circumference. To get degrees I then multiplied it by 360
            - If width radius = 5.3cm circ = 33.3cm
              If tire rad = 2.6cm circ = 16.33628cm
              33.3/16.33528=2.03853255041
              2.03853255041*360 = 733.871718148

- We were looking at the sonar sensor's range for a while, trying to identify where exactly its end point was, and how wide its peripheral vision was. To our surprise, it actually does have considerable peripheral vision, which helped us to decide whether or not we needed to be located on a grey tile or whether it was okay if the robot turned earlier.
    - 22/3: In the end, we found that it was best to go with the safer option of locating the robot on a grey tile.
- Robot spotting object from different row
    - 21/3: The robot was able to detect the object even though it was on a different row from it. For example, the robot detected the object (on big tile 4) even though it was looking at the big tiles 1-3. This was fixed by having the robot sleep for a lengthier period of time before scanning the distance.
    - 22/3: Since the robot assumes that a target was spotted if its distance is less than 255 (the max distance), it's possible that it can spot an object that is 200 or so meters away, which most likely wouldn't be the tower.
    - 26/3: We reduced the distance that the robot reported a tower to 150cm, then 100cm.
- Path to follow for bottle detection
    - 21/3: We managed to get the robot continually (i.e. four times, but this could be increased if there were more possible tower tiles) searching the "alleyways" - a set of three possible tower location tiles. Each time the robot turns down the alleyway it uses its ultrasonic sensor to scan for a requisite distance - one that is less than a maximum range. When it gets a distance that is less than 255 it uses the distance it obtained to make an educated guess about where the tower is located. It then reports that location through speech, celebrates, and exits. If it does not find the tower down an alleyway, it turns and continues to the middle of the next large grey tile, and searches again until the tower is found.
- Error correction pathways
    - 22/3: We managed to get the robot to correct itself enough on the vertical pathway (tiles 1-10) to follow the path, however if the robot wasn't completely centered on the 10th tile it would have executed error correction on the horizontal pathway. When it executed this error correction on the horizontal pathway it wouldn't correct itself enough for the robot to get back on course and therefore would fail to find the bottle.
    - 26/3: Added a special case for course correction on the 10th tile. If the robot had to error correct to find the 10th tile, it will now drive forwards, center itself and then drive backwards the same distance.
- Method of reporting tiles
    - 23/3: We decided to go with the robot beeping, then speaking which tile it was on compared with other options such as just beeps, or displaying it as text, because it would allow us at all times to know what the robot was reporting. Beeps are not distinct from each other and the text is far too small, so you'd have to follow the robot at all times watching carefully for flickers of text.
    - 23/3: After deciding on speech, this made it easier to do error correction (such as if it was counting a tile twice or not at all). It would also allow the robot to give

feedback/updates on where it was or if error correction was needed. We also found it would let us know which part of the code was running, such as if it didn't find a tile, it would alert us to the fact that it had explicitly failed. From that we knew it would be starting to run the error correction code, another example would be that we know when the robot was going to start searching for the bottle. This allowed us to make sure the pathway up to the milk bottles was correct and that everything was being executed correctly. If we had any concerns as to what the robot was doing, the speaking (much like a print line would in the command line) allowed us to error check easily.