# COSC 343 Assignment 1: Group 14 report

**Members:**

- Kimberlene Sharma
- Rik Karmakar
- Namarta Jaswal
- Joe Marshall

When we started this assignment, we didn't actually have a specific architecture in mind. Ours was entirely an approach of test an assumption, alter our approach based on the results; repeat.

Naturally, the resulting approach does however exhibit some common embodied A.I architectures. Our approach can be described as a behavior based hardwired reflex agent. Our agent uses Condition-Action rules to gradually correct its actions based on the information it gathers via its sensors.

# The algorithms we use:

Our approach is implemented via four distinct algorithms. The first is simply the main routine of the script. Here's some pseudo code:

*Move forward a small amount*
*Turn to the right*
*Reverse slightly*
*While our black tile count is less than 15:*
    *Move forward a specified distance*
    *Execute the check function*
    *When the check function returns true:*
        *Add to our count variable*
        *Beep to indicate a black tile was found*
    *Else:*
        *We have gone off course, execute the adjust function*
*Turn to the right 90 degrees*
*Move forward a predetermined distance*
*Execute the ultracheck function with a large correction value*
*Move forward a predetermined distance*
*Execute the ultracheck function with a medium correction value*
*Move forward a predetermined distance*
*Execute the ultracheck function with a small correction value*
*Move forward a predetermined distance*
*Report task completion*

The reason we chose to move a set distance each time in the first phase as opposed to moving continuously is because it removed so many possible mistakes the robot could make. By moving a set distance at a time, there are exactly three possibilities should our robot go off course. We will explain these cases in the next section.

The first three commands simply move the agent from the start tile to the first black tile, facing in the appropriate direction. The robot then travels a distance of 15 black tiles in a line, counting each one and beeping when it finds it. Should it ever go off course, the robot will not detect black after moving and will execute the adjust function to realign itself.

Once the agent has found the fifteenth black tile, it makes a 90-degree turn to the right, and then moves forward before taking ultrasonic measurements via the ultracheck function. It repeats these steps twice more, each time moving smaller amounts, as it gets closer to its target.

Finally the agent drives at full speed towards the closest object, and then verbosely reports that it has completed its task.

The main routine makes calls to the check function each time it moves forward in the while loop. Here's pseudo code for the check function:

*If the color sensor detects black or is the reflected light intensity value is under 25:*
    *Return True*
*Else:*
    *Execute the adjust function*
    *Execute the check function*

The check function checks either that the color sensor is detecting black, or that the reflected light intensity value is under 25. The reason we did this is because the color sensor is usually reliable enough to detect the black tiles, meaning we don't worry about light ambiguity. Failing that, only then do we detect the reflected light value. This combination worked excellently, we very rarely get incorrect readings in regards to color since making that change. Sometimes it appears that the agent is incorrectly recording black when it is almost off the black tile, but so long as the agent can see some of the black tile, it counts it correctly.

This approach also helped avoid our agent recording its own shadow as a black tile, which it did when we were using only the color sensor. When only using light reflection, we tended to get incorrect readings for black tiles when we went off course. Using the two in conjunction made our agent much more accurate.

This brings us to the adjust function. Pseudo code:

*Move forward a small amount in case we have fallen short*
*If we have found a black tile:*
    *Return control to the main routine*
*Else:*
    *Reverse*
    *Turn slightly left*
    *Move forward again*
*If we have found a black tile:*
    *Return control to the main routine*
*Else:*
    *Reverse*
    *Turn slightly right compensating for previous turn*
    *Move forward again*
*If we have found a black tile:*
    *Return control to the main routine*
*Else:*
    *Reverse*
    *Turn to the left at a greater angle than before*
    *Move forward again*
*If we have found a black tile:*
    *Make small adjustment right as we are now on an angle*
    *Return control to the main routine*
*Else:*
    *Reverse*
    *Turn right at a greater angle than before*
    *Move forward again*
*If we have found a black tile:*
    *Make small adjustment left as we are now on an angle*
    *Return control to the main routine*

This is messy we admit, but we did so much tinkering that by the time we were happy with it we didn't see a need to encapsulate here. We could have, and it would've made the script a little shorter, but probably more difficult to read, and no more efficient. At any stage that the agent finds black (by the same means as in the check function,) it returns control to the check function so that it can call itself again. This isn't necessary, but there's nothing wrong with another quick check.

Now we can discuss the three possible violations should the robot go off course. The first is the possibility that we've fallen short, which seems to be a recurring issue for us. Maybe it was our approach, maybe it was the battery levels; in any case, checking

forward a small amount proved useful in our error checking. Hence the first thing the adjust function does is check if the black tile is just ahead of where we are.

This leaves us with our other two possibilities. Either we are to the left of the black tile or we are to the right of the black tile. Therefore, we simply reverse and change our angle, then check again. Failing that, check the other way. Repeat with greater angles of correction. On the second pair of corrections, we make a small adjustment if we find black. This is to mitigate the effect of finding the black tile on an angle, preventing us from immediately going off course again. Almost surprisingly, this does an excellent job of keeping the robot on track, and it does it very quickly.

Lastly we have our ultracheck function, which is used in the second phase to align the robot with the closest object. Here's the pseudo code:

*Turn to the left 125 degrees*
*Sleep for half a second*
*Take ultrasonic measurement, store in variable representing left distance*
*Turn right 125 degrees*
*Sleep for half a second*
*Take ultrasonic measurement, store in variable representing middle distance*
*Turn to the right 125 degrees*
*Sleep for half a second*
*Take ultrasonic measurement, store in variable representing right distance*
*Turn left 125 degrees*
*If the left distance is the smallest:*
    *Turn to the left at the degree specified by our parameter*
*Else if the right distance is the smallest:*
    *Turn to the right at the degree specified by our parameter*
*Else if the middle distance is the closest:*
    *Beep to indicate no turn was required*
*Else if at least two values are the same:*
    *Move forward a small amount*
    *Execute the ultracheck function with a tiny parameter value*

The ultracheck function takes one parameter representing the intensity of the correction to be made. Each time the ultracheck function is called, we call it with a decreasing steer value, as the closer we get the smaller our corrections will need to be.

The ultracheck function is quite simple. Turning in 125-degree increments, take an ultrasonic measurement to the left, to the right, and straight forward. Then simply turn or remain on course depending on which measurement was the smallest, to align the agent with the closest object. The sleeping between turning and taking measurements is to ensure the robot has come to a halt before taking the measurement. Should two of the

values be equal, the agent shall move forward a small amount and execute the ultracheck function again.

# Issues encountered:

While we are very pleased with the outcome of our project it is important to discuss the issues we encountered in development.

As the assignment mentioned the light levels were inconsistent between testing sessions. Eventually, this had an effect on the readings we received from the color sensor. This led to us using the peculiar ‘if black or low light intensity’ conditional. This approach seemed to overcome the problem; we haven’t had any issues with color sensor readings after making that decision.

Another issue was the varying battery levels of the available robots. The robots perform differently depending on how much charge they currently have. To overcome this, we mainly used rotations instead of time to tell our robot how far to drive. This wasn’t completely overcome however, because we still do use time in the ultracheck function. We haven’t had this cause any issues but it is susceptible to them in theory. The reason we used time in this instance is because the method used was the best choice for the action we wanted to perform. It made it easy to pass a single value representing the intensity of the correction to be made allowing us to encapsulate a lot of repeated code.

There is an issue we weren’t able to completely overcome. Our robot is really good at finding the tower, but not so good at moving it off of the finish tile. This is mainly because the angle at which we approach the tower from makes a huge difference. Even if the robot comes at the tower straight on at full speed, sometimes is just won’t move the tower. We have a few theories for this. Firstly, the uneven surface of the ground the tower is on is enough to have an effect on the robot’s traction. Secondly, if the robot hits the tower from the side that the tower’s handle is on, it is possible that the robot will actually lift onto the tower, rearing it’s wheels off of the ground rendering it unable to move at all. We witnessed many groups have similar issues.

Lastly, we have a known bug in our code, although it’s never happened. Should our adjust function fail to find a black tile; it will be repeatedly called by the check function until it does. It will eventually find black, because the initial move forward made by the adjust function is intentionally never corrected, as the robots seemed to fall behind themselves over time in the first phase, despite moving a set distance each time. Therefore, the adjust function will repeatedly search for black, edging slightly forward each time. The only way this could happen is if the robot went off course so many times that the robot eventually overshot a black tile, therefore skipping it and travelling further than intended in the first phase. We believe this bug wouldn’t be severe enough to derail the second phase, but it is nonetheless a known bug in our code even if it hasn’t happened... yet.