Morse code decoder robot

**Research:**

**-Compound if**

A compound if was researched as the programming language is not similar to other more ‘traditional’ languages and it was required when processing more than one statement in conjunction with each other.

**-Morse code alphabet**

The Morse code alphabet was researched as each letters dot/dash code needed to be known when making a corresponding 1/0 code for the alphabet within the program.

**-Research by doing**

As we were given the program and learned by testing/researching different methods, the research that was performed within the program should also be included. This is how a final solution was developed through personal research.

Research began with basic stop start methods using button presses on the ev3.

Moved up to using coloured masking tape and the on board colour sensor to measure the reflected light intensity of certain colours.

Combined the stop, start methods with the colour sensor and tape to test if the ev3 could stop when it reached the colour red.

Testing using multiple colour values to see if the ev3 can detect those as it is moving and react differently to each.

Testing viable methods to decode Morse code:

Tried if statements and compare binary values, ultimately settled on comparing strings as it was the simplest and most elegant solution.

Experimenting with different colours to ensure best result.

**Analysis:**

We were told that there would be no ambiguity within variables and sizes, so when designing the solution there was no need to make it dynamic. One starting calibration measurement was all that was needed to calculate the rest of the data sizes.

The ev3 programming language was surprisingly intuitive, it was very easy to understand and test solutions. The ‘block’ style programming was very visual and helped when It came to breaking down problems and building up code. The code blocks feature helped break down the code into more manageable segments. It made for a fun and interesting assignment and I would recommend it to other students.

Some issues that were encountered were with the colour sensor, when the sensor would go over more than one colour at the same time it would not register any of the colours and keep driving until it reaches a solid colour it recognises. Also surfaces can cause issues, if it is a more reflective material it will reflect a higher amount of light than duller materials. This issue can be avoided by printing the track on the same piece of paper, but it is also important to factor in the printer inks reflectivity on paper.

Another issue is that as a track is going to be long it may require taping more than one piece of paper together. The problem with this is that there is a distinct line between sheets of paper and the robot reads this edge as being not white, which stops the program and causes it to try and read red spaces.

**Design:**

‘Functions’ were created using the code blocks features provided by the ev3 software, code blocks allowed for easy managing of the large amount of icons. Within the code blocks are all the smaller code ‘icons’ that make the solution work and run.

Code blocks were created to:

-Measure all colours

-Move until red

-Calibrate dot and dot/dash range

-Measure red space/white space

-Decode binary version of Morse code

Some blocks went through multiple design changes as new issues were discovered in the later development stages. This is present in the screenshots of the code.

To give constant feedback to the user on the sensor values the screen is used, whenever it takes a measurement it will output the value of said measurement and any other relevant information to the screen such as upper and lower boundaries or characters.

**Implementation:**

The code blocks specified above have been implemented as such (all code blocks screenshots are available in the index):

**Measure all colours:**

-ev3 ‘shouts’ the colour it would like to read

-Colour sensor is placed over the specified colour

-Upon pressing the touch sensor on the ev3 it will take a reflected light intensity value of said colour.

-The value is saved to a variable and a range is calculated (+/- n) using the mathematical operators available to have a low and a high range value, these values are stored under new variables.

This process is repeated for all colours required.

**Move until red:**

-The motors are turned on to a moderate speed

-A loop is entered in which the colour sensor is always on and reading all incoming values

-If the colour sensor detects a value between the range of the previous set red low and high values it breaks the loop and turns the motors off

**Calibrate dot/dash range:**

-The motor rotation value is reset to zero

-A loop is entered in which the colour sensor is on and looking for any value within the white high/low range

-Once it has detected white it turns off the motors

-It takes the degrees of rotation value which is equal to the length of the red dot and saves it to a dot variable

-The dot is then used to calculate the values for dot high/low range in the same fashion as the colour calculation.

-A dash is 3 times the length of a dot so mathematical operators are used to multiply the dot low/high values by 3 and then saving them

**Measure red space:**

-Motor rotation is reset to zero

-A loop is entered which moves the motors until it reaches not red

-The motors are stopped

-A measurement is taken in degrees of the current motor rotation

-It is checked if it is a dot using logical operators on the motor rotation value comparing it to the length of a dots high/low value. If it fits within this range it is dot, if not it is a dash

-A dot writes a 0 to a string variable names ‘Morse code letter’ and a dash writes a 1

-On the screen the measured red space is output, and the dot range for testing and comparison purposes

**Measure white space:**

-Motor rotation is set to zero, a loop is entered until the motors reach not white

-The motors are stopped

-A measurement is taken in degrees of the current motor rotation

-It uses switch statements to deal with the different white space options, it will:

-Check if it is a dot using logical operators on the motor rotation value comparing it to the length of a dots high/low value. If it fits within this range it will continue to process the word and therefore do nothing

-If it is not within dot range it checks if it is within the dash range, if this is the case it processes the binary string into a letter using the ‘Morse code’ code block

-If it is not in the dash range it checks if it within the range of a 7 white space amount, if it is it will process the letter and will finish the word by adding a space on the end of the string

-Finally, if it is greater than a 10 white space size it will process the word and finish the program

-A dot writes a 0 to a string variable names ‘Morse code letter’ and a dash writes a 1

-On the screen the measured red space is output, and the dot range for testing and comparison purposes

**Process Morse code:**

-The Morse code letter string of 1’s and 0’s is passed into a switch statement that contains all letters of the alphabet in the form of a string of numbers

-As previously mentioned, a dot is saved as 0 and dash as a 1, that format is followed giving a unique string code for each letter

-The switch statement checks the incoming string and compares it against all letters of the alphabet, if it matches any of them then the corresponding letter is written to the ‘Morse code sentence’ string

-The ‘Morse code sentence’ variable is written to the screen

Ultrasonic sensor usage was also included, it is angled towards the driving surface and when it detects the surface disappears it stops the program. This is run in parallel to the main code, it stops the robot falling off a desk. It could also easily be reversed to stop when an object is detected as being within a certain range. This could be used as an emergency brake or stop by putting a hand in front of the robot.

Line detection has been experimented with but not implemented. If implemented it would run in parallel with the main code in the same fashion as the ultrasonic sensor. Its purpose would be to detect an edge and turn back onto the track. This is done by using the same colour detection method that is used at the beginning of the main program. It would require two different colour borders either side of the track and then it would turn left/right depending on which colour is read. The motors on either side of the ev3 would be used independently, turning one in a direction and another in parallel to turn it on the spot until the edge colour is not detected anymore plus a few extra degrees for best positioning. The motors would then be run in the same direction with the outer motor (furthest from line) moving faster than the inside motor (~15%). Doing this will make the ev3 turn in more of a curve hopefully placing it straight on the track. The problem with this solution is that there are times where the colour sensor is over 3 different colours at one time, it does not read any value and keeps driving.

**Testing:**

(Must provide evidence to demonstrate the solution satisfies the functionality, not just screenshots).

Consideration must be given to determining how the final solution should be tested and evidence presented. For each test consider:

Specifying the purpose of the test;

The existing pre-test conditions prevailing prior to the test;

The expected post-test conditions or outcomes;

Screen shots to support the test.

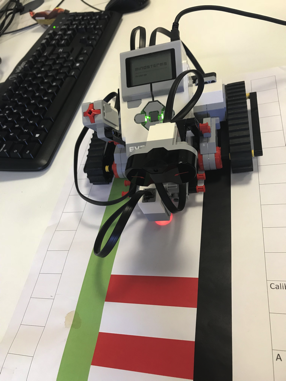
**Test 1(Measure colour):**

Purpose:

Test that the program can accurately read and store colour values.

Pre-condition:

Track with colour, nothing stored on the ev3



Post condition:

On button press the ev3 should take store the reflected light intensity and display it to the screen.



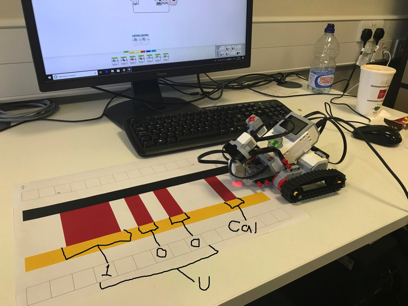
**Test 2(Measure red space):**

Purpose:

To test whether the ev3 can detect red, moving forward until the red space ends and be able to store the measured length of the red space and convert it into the correct value of 1/0.

Pre-condition:

The ev3 should have the colour values stored and will be moving towards a red space, ready to measure it once it hits it.



Post condition:

The ev3 should measure the red space and interpret it as one of two things, a 1(dash) or a 0(dot) and should display its current strings 1’s and 0’s to the screen while ‘shouting’ 1/0 depending on the character read.



**Test 3(Measure white space):**

Purpose:

Test whether the program can take a measurement of white space and perform a specific action depending on the size of the white space.

Pre-condition:

The ev3 should have just measured a red space and be moving into a white space.

**(No screenshots included as the screenshots for Test 4 are adequate for this test)**

Post condition:

Depending on the size of the white space it should either:

-Read it as an inter-character space and keep moving

-Read it as a finished character and display all characters to the screen

-Read it as a space displaying all characters currently stored, with a space added to the end

-Read it as end program outputting the message and ending the program

**(See screenshots for Test 4)**

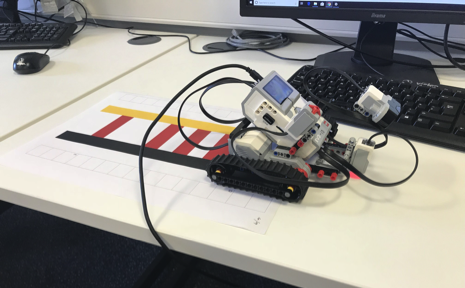
**Test 4(Output message):**

Purpose:

Check the ev3’s ability to understand when a message has finished and output the whole string to the screen.

Pre-condition:

The ev3 should have read all available red spaces, just finishing the last one and in front of it should be all white space.



Post condition:

After the 10-space limit it should stop and output the full decoded message to the screen before finishing the program.



**How should the final solution be tested?**

To prove that the program created has the ability to decode Morse code a test needs to be created that will show to anyone viewing it all aspects work correctly and are fully functional.

A way to do this would have a predetermined ‘track’ of dots and dashes that the ev3 would need to decode, once decoded the message could be displayed on the ev3s built in screen. The program could also display/tell the user if it has just read a dot or a dash in real time as it is moving down the track. Constant live feedback to the user ensures that the program cannot hide or be dishonest with the values/Morse code it is currently decoding as it moves down the track. Once it has reached the end of the track the full message should be decoded correctly and displayed on screen.

The program must also prove that it can calculate and perform the correct action based upon the length of white space after characters.

A whitespace may mean (depending on its size):

-New letter

-New word

-End of message

Each of these possibilities should be tested by adding them to the message ‘track’, for example the encoded message could be ‘I am’. This would test that each whitespace functionality has been implemented correctly.

The track can also have two guidelines either side of the Morse code message, by including this you can stop and turn left or right if the robot is swaying off track.

Use of the ultrasonic sensor can be implemented so that someone may put their hand in front of the robot for a ‘quick stop’, or to see if the table is going to end and stop the robot before it drives itself off a table.