University of Derby
School of Computing & Mathematics

A project completed as part of the requirements for the
BSc (Hons) Computer Games Programming

Entitled

Key-stage computing: Evaluating the suitability of Lego Mindstorms NXT 2.0 for use in early computer science education.

By

Christopher R Fisher
chrisfisher@pro-fish.co.uk

In the years 2013 – 2014
# Table of Contents

1. Introduction .................................................................................................................. 5

2. Literature Review ........................................................................................................... 6
   2.1 Key-stage One and Visual Programming Languages .............................................. 6
   2.2 Key-stage Two and RobotC ..................................................................................... 7
   2.3 Suitability of RobotC as an Educational Language .................................................. 9
       2.3.1 Criticisms ......................................................................................................... 9
       2.3.2 Advantages ..................................................................................................... 10
   2.4 Alternatives to RobotC ............................................................................................ 10
       2.4.1 C# .................................................................................................................. 10
       2.4.2 Python ........................................................................................................... 11
   2.5 Outcome .................................................................................................................. 12

3. Methodology .................................................................................................................. 13
   3.1 Robot Configuration ............................................................................................... 13
   3.2 Boilerplate Code ..................................................................................................... 15
   3.3 Test Outlines .......................................................................................................... 15
       3.3.1 Action 1: Output ............................................................................................. 15
       3.3.2 Action 2: Input-Output .................................................................................. 16
       3.3.3 Action 3: Conditional Input ........................................................................... 16
       3.3.4 Action 4: Input Tracking .............................................................................. 16
       3.3.5 Action 5: Simple Movement ......................................................................... 16
       3.3.6 Action 6: Measured Movement ..................................................................... 16
       3.3.7 Action 7: Timed Movement .......................................................................... 16
       3.3.8 Action 8: Combined Movement ................................................................... 16
       3.3.9 Action 9: Move until Sensor ......................................................................... 16
       3.3.10 Action 10: Rover ......................................................................................... 16
   3.4 Complexity Metric .................................................................................................. 17

4. Results & Analysis ........................................................................................................ 19
   4.1 Developer Notes ...................................................................................................... 19
   4.2 Data ......................................................................................................................... 20
       4.2.1 Precise Movement ......................................................................................... 23
       4.2.2 Algorithmic Interaction ................................................................................. 23
       4.2.3 Basic Interaction ............................................................................................ 24
       4.2.4 Basic Movement ............................................................................................. 26
       4.2.5 Algorithmic Movement .................................................................................. 27
   4.3 Summary .................................................................................................................. 28

5. Conclusion ..................................................................................................................... 30
   5.1 Suitability of Natural Language for Education ....................................................... 30
7.3.31 Snippet_Move_Time_Forward_1Sec ................................................................. 62
7.3.32 Snippet_MoveForwardForRotations ................................................................. 62
7.3.33 Snippet_MoveStraightForRotations ................................................................. 63
7.3.34 Snippet_Output.Counter .................................................................................. 63
7.3.35 Snippet_RobotC_Branching ............................................................................. 63
7.3.36 Snippet_SingleInput_response .......................................................................... 64
7.3.37 Snippet_Sound_NoVolume .............................................................................. 64
7.3.38 Snippet_Varibale_Assignment ......................................................................... 64
7.3.39 Snippet_Variable_Tracking .............................................................................. 64
7.3.40 Snippet_Variable_Tracking_Output ................................................................. 64
7.4 Halstead Parser Source Code .............................................................................. 65
  7.4.1 Halstead Parser ............................................................................................... 65
  7.4.2 Program Source ............................................................................................... 70
1. Introduction

“…if the UK is to retain its global strengths in the high-tech creative and digital industries more generally it must urgently address the need for more rigorous teaching of computing in schools” (Livingstone & Hope, 2011, p. 3)

Computer science and in particular computer programming has become a much discussed topic in recent years due largely to the results of the NextGen report, which suggests addressing a lack of essential computer science training in schools and further education (Livingstone & Hope, 2011). The report highlights that computer science is often perceived as a dry subject that could benefit from the inclusion of more engaging technologies, such as those provided by the video games and visual effects industries, which would benefit greatly from a better trained populace.

The desire to cover necessary principles in an enjoyable context is not unique to computer science but a strong correlation between subject interest and subject confidence has been noted in previous works such as the survey performed by the HE Academy of Information and Computer Science which highlighted that: “reasons for choosing to study computing at university are varied, but a previous interest in some aspect of computing is a recognized factor” (McChesney & Alexander, 2007, p. 4).

With computing entering the national curriculum in 2014, the government has outlined a number of attainment targets which are designed to teach children the principles of computer science and theory behind digital systems, with an emphasis on putting ideas into practice through concrete programming skills. Computing will be taught to children of all ages from key-stage one through four\(^1\). For the youngest children focus is placed on more abstract concepts - formation of algorithms, logical reasoning and the role of technology - mixed with some practical applications. Key-stage two places more emphasis on applicable skills and is designed to teach children how to develop simple programs for specific tasks, such as manipulating physical systems (Department of Education, 2013).

Lego has previously attempted to address these issues through the use of robotics with its Mindstorms NXT product range which includes the “Education Base Set” aimed at bringing the Technics based kit into schools. Lego supplies a visual programming language for use with the NXT built upon an open source firmware. The ability to view and modify the firmware has encouraged the development of custom languages often based upon languages used in computing industries including; C, Java, Perl, Ruby, C#, Python, LUA and Assembly (Hassenplug, 2008).

Lego Mindstorms might provide a fun and engaging platform on which to teach the new computer science curriculum for early key-stage attainment targets, however the current language support may not be sufficient. In particular NXT-G and the widely used RobotC are not suitable languages for teaching computer science to younger children. This paper aims to evaluate if a suitable computer science learning path can be created using existing technologies or if a new language is needed that is more tailored to educational requirements.

\(^1\) key-stage one = 5-7; key-stage two= 7-11; 
key-stage three = 11-14; key-stage four = 14-16.
2. Literature Review
This section begins by exploring the suitability of NXT-G for key-stage one attainment targets and comparing its interface to that of the more general visual languages Scratch and Blockly. Next the paper introduces RobotC, a cross-platform robotics programming language that maps closely to C. Existing materials do not actively evaluate the effectiveness of RobotC as an educational tool, so comparisons are instead made against the parent language, with special attention paid to what separates the two and how those differences affect its educational suitability. Finally some alternatives are presented and their suitability compared directly against that of RobotC.

This paper has chosen to concentrate on language choice as a tool for teaching computer science and seeing if a simple learning path can be outlined. In the interests of narrowing the scope of discussion this excludes exploring what makes a good learning path and how this is affected by differences in learning style. Learning is a multifaceted subject and this paper does not attempt to cover all aspects of it, but rather tries to cover the area most directly related to computer science.

2.1 Key-stage One and Visual Programming Languages
The NXT comes bundled with a visual programming language called NXT-G. NXT-G is designed as a building block based, drag & drop language and provides a similar environment to the one offered in other educational visual languages such as Scratch and Google’s Blockly. Visual programming languages are highly suited to achieving the curriculum key-stage one attainment target, by teaching children the basics of algorithms, how they are implemented and their execution via “unambiguous instructions” (Department of Education, 2013). Languages like Blockly and Scratch achieve this using a mixture of words/numbers and graphical jigsaw-pieces that snap together to create logical blocks of coherent, code-like instructions. The resulting algorithms can be read back easily in a form that lies somewhere between code and human language but crucially presents the data involved very clearly:

```
set item to 0
repeat 10 times
  do
    set item to item + 1
  end
print item
```

NXT-G attempts a similar approach but relies more heavily on iconography that obfuscates the data from the visual flow until a block is selected, resulting in a more condensed, less code like structure:
This approach makes the resulting algorithm more visual, but also much less descriptive and crucially, less suited to conveying “unambiguous instructions” making it a somewhat poor choice for attaining key-stage one targets. A disjointed relationship to both working code and human language also make transitioning to key-stage two targets a more difficult task. In this regard, Blockly holds a significant educational advantage over both NXT-G and Scratch, with its ability to be translated directly into a number of real-world languages, including the popular scripting languages python and JavaScript (Fraser, 2014). The ability to see how a visual algorithm translates directly into working code could prove an invaluable insight when moving onto key-stage two and could also be used as a form of prototyping.

Due to the open source nature of the NXT a number of alternative languages are available for it. One such implementation, Enchanting, attempts to create a visual language based upon Scratch that runs on the NXT (RobotClub, 2014). This language uses the same syntax and visualisation methods as Scratch but with custom blocks especially designed for Mindstorms.

This approach has the benefit of using an established language that has been designed specifically for educational purposes. Whilst Enchanting does not enable transitioning to a real-world language in the same way as Blockly, its close ties to the readily available Scratch allow for more opportunities to continue learning outside of the classroom. This is something which NXT-G cannot offer, due to its tight coupling to the platform.

2.2 Key-stage Two and RobotC
RobotC is a non-visual programming language built as a non-standards-compliant subset of the C programming language that comes with a full featured IDE. It is designed to work on a multitude of robotic hardware including NXT, Arduino, TETIRX and VEX, as well as virtual implementations of these platforms (RoboMatter, 2014). RobotC has been created by RoboMatter who have worked closely with Carnegie Mellon University Robotic Academy to
produce a wealth of videos, tutorials and documentation (much of which has been collated into a curriculum) to make teaching Science, Technology, Engineering and Maths (STEM) through robotics a more accessible area (RoboMatter, 2014).

An affinity with a widespread, real-world language creates a lot of opportunity for extracurricular applications and can teach skills that are directly transferable to the job market. Conversely, C is considered to be a very low level language and so adds a layer of complexity to the learning curve that is not present in more high level languages. C also lacks the object orientated paradigms that are commonly found in modern applications/languages, which may prove to be a barrier when transitioning to them.

RoboMatter have gone some way to address RobotC’s syntactical issues by creating an API called ‘Natural Language’ which aims to reduce complexity by providing higher level functions for more menial Mindstorm tasks (Carnegie Mellon Robotics Academy, 2010). Natural Language makes program flow more obvious and helps to introduce the concept of functions/methods whilst still allowing for the use of RobotC.

```
// Plain RobotC
void main()
{
    while (SensorValue(sonar) > 25)
    {
        motor[motorC] = 50;
        motor[motorB] = 50;
    }
    motor[motorC] = 0;
    motor[motorB] = 0;
}
```

```
// RobotC Natural Language
void main()
{
    forward(50);
    untilSonarLessThan(25, sonar);
    stop();
}
```

‘Plain’ RobotC vs Natural Language

Taking this idea one step further, RoboMatter have an upcoming graphical language that combines the simplicity and verboseness of Scratch with the abstracted nature of the Natural Language library. Graphical Natural Language (GNL) is a drag and drop, building block language that translates directly into RobotC using a simple IDE that features support for robot setup and deployment (Friez, 2013).
GNL combines a number of benefits into a single implementation to create a product which would work very well for key-stage one attainment targets and would enable a smooth transition into key-stage two, where a direct mapping to a C based language opens up further opportunities to fulfil curriculum criteria and leads naturally into more advanced computer science topics such as memory management. GNL has not yet been released for the NXT.

2.3 Suitability of RobotC as an Educational Language

2.3.1 Criticisms

C is a language that matches quite succinctly to machine code and is viewed as a fairly low level language when compared to more modern derivatives such as C# or Python. In production code this is often seen as a positive as it affords the developer a large degree of implicit control. C’s suitability as a systems programming language has been proven repeatedly but it also presents a number of ideas and constructs that are seen as difficult or unwieldy. In “A Critique of C++” (Joyner, 1996) a number of arguments are put forward that highlight some of the trapping and pitfalls of the C language; most of which would appear to make it an unsuitable language for beginners. However RobotC is strictly a subset of C (with some language extensions) that has been specifically designed for educational use and so deliberately takes steps to avoid some of these issues (Swan, 2007):

- Primary amongst these concerns is dynamic memory management techniques and array handling. Low-level functions such as free and malloc are not included in the RobotC subset in order to make the language more accessible and safer for beginners. Similarly, pointers are available but with limited functionality and can be avoided for the vast majority of actions (Swan, 2007).
- Joyner criticises Cs implementation of arrays due to their fixed size, static allocation and lack of bounds checking which is at contrast to many modern languages that prefer Lists and dynamic arrays which incorporate checking features. RobotC makes
use of a motor array for accessing the motor ports but Natural Language abstracts away arrays and individual motor manipulation in favour of action based functions.

- Use of the form `function()` leads to ambiguity in function forward-declaration, with the form `function(void)` preferred instead. At an introductory level this quirk is potentially confusing and neither RobotC nor Natural Language does anything to address this issue, so it remains a potential pitfall.
- Defines and other pre-processor instructions are described as being “poorly integrated with the language” due to the large number of errors that can be introduced when using them. RobotC does nothing to address this issue and even generates a number of pragma statements in its boilerplate code.
- Semicolons are seen by Joyner as inconsistent due to their exclusion from control structures (if/else etc) and states that this is bad design. Whilst potentially confusing for beginners GNL - being a visual language - avoids them entirely whilst RobotC and Natural Language do not. Semicolons as a statement terminator are a common feature of many programming languages and as such are largely unavoidable when teaching coding. Languages that do avoid them are much rarer and tend to rely on strict whitespace formats which cause similar issues.
- Case-sensitivity is considered a boon for readability but a hindrance on semantics that introduces unnecessary confusion; however RobotC is case-insensitive and so avoids this issue.

### 2.3.2 Advantages

RobotC has been designed for educational use and is targeted at inexperienced developers, which means great care has been taken to avoid many of the pitfalls mentioned above. Despite this, RobotC remains faithful to its C origins and is syntactically very similar. With C having influenced the design of six out of the ten most popular languages used today (TIOBE Software, 2014), in addition to being a strict subset of C++ (the fourth most popular language), RobotC provides an environment in which to learn highly transferable programming skills.

In addition to being highly influential, C itself is still a highly relevant, active production language that is often the choice for embedded or resource conscious systems. It is also commonly taught in higher education and some grounding in it would prove advantageous to students wishing to expand on their curriculum experience (Cheng, 2011).

### 2.4 Alternatives to RobotC

There are a number of implementations of real-world languages that could be considered for use with key-stage two that avoid the issues related to RobotC’s close ties with C. Higher-level languages such as C#, Python and Java benefit from widespread adoption (TIOBE Software, 2014), more modern approaches to computer science and potential transferability for future education.

#### 2.4.1 C#

One such example is the Microsoft Robotics Developer Studio which uses C# or a graphical language called VPL and benefits from both a virtual testing platform and integration with Visual Studio (Microsoft, n.d.). Like RobotC, MRDS works on a number of robotics platforms in addition to the NXT, however there is no direct mapping from VPL to C#. VPL also suffers from the same issues as NXT-G in that it abstracts the data away from the visual algorithm.
MRDS C# implementation necessitates a complicated setup that requires adding references, subscribing to services and writing subscriptions to services using listeners in order to access sensors. This large amount of overhead and introduction of advanced ideas is less suitable for beginners than RobotC’s minimal boilerplate code.

```csharp
[Partner("Bumper", Contract = bumper.Contract.Identifier,
    CreationPolicy = PartnerCreationPolicy.UseExisting)]
private bumper.ContactSensorArrayOperations _bumperPort = new bumper.ContactSensorArrayOperations();

void SubscribeToBumpers()
{
    // Create the bumper notification port.
    bumper.ContactSensorArrayOperations bumperNotificationPort = new bumper.ContactSensorArrayOperations();

    // Subscribe to the bumper service, receive notifications on the bumperNotificationPort.
    _bumperPort.Subscribe(bumperNotificationPort);

    // Start listening for updates from the bumper service.
    Activate{
        Arbiters.ReceiveBumper.Update =
            [true, bumperNotificationPort, BumperHandler];
    }
}

private void BumperHandler(bumper.Update notification)
{
    if (notification.Body.Pressed)
        LogInfo(LogGroups.Console, "Ouch! - the bumper was pressed.");
}
```

Detecting bumper press in C# (Microsoft, n.d.)

2.4.2 Python

"Why Complicate Things" (Grandell, et al., 2006) singles out Python as a good candidate for teaching programming, making reference to its original design as an educational language. Python is credited as having a large number of positive aspects for beginners including clear and concise syntax (helped in large part due to dynamic typing and whitespace reliance), dynamic lists as a base type and interpreted code. Grendell et al also highlight their own success with Python in trials amongst Finnish high school students who described the language as easier to use and more fun to learn than other languages.

NXT-Python utilises an object orientated approach to NXT programming that allows the inclusion of advanced paradigms including classes and threading (NXT-Python, 2012). Despite this, it is possible to create simple, functional programs with very few lines of code.
For an absolute beginner the procedural programming style of RobotC can be seen as simpler because it requires less understanding of complex object orientated topics. RobotC also simplifies code slightly as it avoids wirelessly locating and connecting to an NXT brick (NXT-Python is deployed via Bluetooth) but at the expense of including a compile stage which adds complexity to the overall deployment process.

As indicated in the examples below, all three implementations use only a few lines of code to perform similar actions. Comparing the three languages highlights the additional overhead of C-style functions and shows how Natural Language can make even simple actions less verbose and requires less understanding of the robot setup. This allows the user to concentrate more on the algorithm itself, which is a clear intention of the curriculum attainment targets.

2.5 Outcome
When compared to other NXT languages RobotC requires less understanding of higher level programming concepts and provides a more accessible API. There is potential for construction of a solid learning path that progresses from a drag and drop graphical language, on to a high-level API, through to lower-level C like code and finally leading to more practical, production quality languages. This progression hinges largely on whether Natural Language succeeds in making NXT programming simple enough for children.

In order to evaluate the suitability of Natural Language it will be necessary to test how difficult it is to implement a number of common programs using the API. The most suitable method with which to achieve this would be to attempt to create these applications in the simplest terms possible and then apply software metrics to the code which will help quantify the complexity of the implementation.
3. Methodology

This methodology first outlines a brief overview of the testing method and what metrics are being employed to discern the complexity of the resulting code. Static aspects of the methodology are presented, including what factors influenced the design of the robot and how it facilitates a fair evaluation, in addition to introducing the template code that forms the base of each test. Finally, a complete list of test algorithms are detailed before presenting an in depth view of the evaluation metrics that will be applied to the resulting code.

The purpose of this evaluation is to test the complexity of code required to implement a range of specified algorithms using RobotC with Natural Language. In order to achieve this a custom robot setup has been designed that leverages the default sensor and motor ports used in the Natural Language API. A number of test algorithms have been included ranging from simple output to basic autonomous behaviour.

Each test represents a specific implementation of an action. Each test is to be completed by a single developer attempting to express the action in the simplest terms possible. Where there is a notable degree of scope for alternative implementations (within the confines of Natural Language), then these variations have been treated as individual tests which contribute to the average complexity of an action. All tests are to be completed without prior knowledge of the evaluation metric in order to reduce bias. Observations of the coding process by the developer that detail issues encountered whilst implementing the algorithms should be recorded and presented as part of the final data.

The following aspects are not being evaluated in this paper although could be factors considered in future studies;

- The development environment: all work will be done in the RobotC IDE,
- Setup of sensors and motors: handled by the boilerplate code,
- Connection setup: applications will be deployed via Bluetooth connection which will be configured inside the RobotC IDE,
- Robot layout: sensor positioning is flexible even within the confines of the Natural Language default port settings. The nature of the Lego Mindstorms kit means that there is potential for a number of designs.

3.1 Robot Configuration

For fairness, every test will use the same robot construction. Natural language uses a selection of default ports, sensors and motors which will be taken advantage of, assigning simple names to each in an effort to increase clarity. The light sensor input method has been set to inactive as it is not being used as a line tracker:

- Motor A: Unused
- Motor B: Right motor     rightWheel
- Motor C: Left motor      leftWheel
- Sensor 1: Touch sensor   touch
- Sensor 2: Sound sensor   sound
- Sensor 3: Light sensor   light
- Sensor 4: Sonar sensor   sonar

For example, the function forward() can be replicated using the array syntax of plain RobotC or by using separate calls to startMotor(). In both of these instances a simpler Natural Language implementation is available so they will not be tested.
Sensors have been located at the front of the robot with the touch and sonar placed centrally to avoid having to take offsets into account:
3.2 Boilerplate Code
Every test contains the same boilerplate code. This code defines sensor and motor setups, as well as the main task definition. This code is not considered as part of the complexity rating due to its inclusion in all tests and the ability to provide this code to students as part of the educational program.

```c
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

//!!Code automatically generated by 'ROBOTIC' configuration wizard

main()
{
    // Test code in here
}
```

3.3 Test Outlines
Below are the outlines and aims of the test programs. In total there are ten actions which represent different tasks that can be completed by the NXT robot. Each action has any number of tests that represent alternative methods of achieving the same or similar results, utilise alternative Natural Language methods, or take advantage of different NXT sensors.

In order to avoid the use of more complicated programming paradigms, tests are to be fulfilled without the use of pre-processor directives or custom functions. In addition, all test code must exist within the main function, as indicated in the boilerplate code.

3.3.1 Action 1: Output
Produce output from the robot; very similar to “Hello World” programs often used as an exercise to introduce a new language. There are a total of 3 tests; one for sound and one for string display, plus an additional string display that takes advantage of a shorter function.

Action1_Sound.c, Action1_Text_DisplayString.c, Action1_Text_ScrollText.c
3.3.2 Action 2: Input-Output
Produce the simplest output as noted during previous test, in response to user interaction. There are 4 tests, one for each possible input sensor.

Action2_Light.c, Action2_Sonar.c, Action2_Sound.c, Action2_Touch.c

3.3.3 Action 3: Conditional Input
Produce an output depending on the type of input provided. Two tests; one using the two simplest sensors and one using the most complex sensors, as determined from results of previous test.

Action3_LightSonar.c, Action3_TouchSensor.c

3.3.4 Action 4: Input Tracking
Report on the total number of times that input is received every time a new input occurs. Perform one test for each sensor.

Action4_Light_SingleInput.c, Action4_Sonar_SingleInput.c, Action4_Sound_SingleInput.c, Action4_Touch_SingleInput.c

3.3.5 Action 5: Simple Movement
Make the robot perform very basic locomotion for any amount of distance. The intent here is to cover the movement types provided by Natural Language. Four tests in total, once each for forward and back, once each for point and swing turns.

Action5_Move_Forward.c, Action5_Move_Backward.c, Action5_Turn_Point.c, Action5_Turn_Swing.c

3.3.6 Action 6: Measured Movement
Make the robot travel precisely 30cm. Two tests, one for forward and one for backward.

Action6_Distance_Forward.c, Action6_Distance_Backward.c

3.3.7 Action 7: Timed Movement
Move the robot for 10 seconds. Test both forward and backward.

Action7_Time_Forward.c, Action7_Time_Backward.c

3.3.8 Action 8: Combined Movement
Combine at least 3 movements to create a pattern using the simplest locomotion’s as determined by previous tests.

Action8_Pattern.c

3.3.9 Action 9: Move until Sensor
Perform the simplest locomotion until input is received from the simplest sensor.

Action9_MoveUntilInput.c

3.3.10 Action 10: Rover
Implement a simple algorithm for constant movement: Move forward until touch sensor is activated. Once sensor is activated, reverse and turn. Repeat.

Action10_Rover.c
3.4 Complexity Metric

In an effort to reduce bias, the metric implementation should remain hidden from the tester.

To evaluate the tests objectively they will be appraised using the approach suggested by Halstead in Elements of Software Science (Halstead, 1977) which uses the number of total and distinct operators and operands to form the basis of a number of calculations, designed to help evaluate program complexity. Equations for these can be found as part of the results tables in the appendix.

Halstead metrics have been chosen to allow the evaluation of very simple applications in an objective and automated fashion. Function and feature point metrics are considered by some to be too open to subjectivity and are more suited to data-processing applications. Conversely, Cyclomatic Complexity is a highly quantitative metric but has a focus on code branching that makes it unsuitable for simplistic programs (Agarwal, et al., 2008).

Using the Halstead metrics a number of properties can be evaluated. Of particular note are:

- **Vocabulary**: Total number of unique elements. Programs with a smaller vocabulary will require the child to understand fewer concepts.
- **Length**: The size of the program. Similar to number of lines of code, but ignores whitespace and takes into account line complexity (GrammerTech, 2014). Smaller programs are quicker and easier to implement.
- **Program Level**: A measure of how abstracted a program is. This is especially useful in determining how successfully the Natural Language API is at reducing complexity.
- **Difficulty**: Represents how difficult the program is to understand. Increases with the amount of unique operators and frequency at which operands occur (GrammerTech, 2014). Less difficult programs require fewer operators and fewer variables.
- **Effort**: Represents the mental effort required to implement or read a program (Halstead, 1977).
- **Intelligence Content**: A language independent measure of how much information a program expresses (Halstead, 1977). Can be seen as measure of the density of an implementation.

There is also a number of Halstead metrics that will not be considered as part of the results:

- **Calculated Length**: Designed to estimate the upper limit of the Length. Flawed due to the assumption that a program is a single ordered set of n elements (Agarwal, et al., 2008)
- **Volume**: The size of the program on disk if each element represents one bit. Included in the results because it forms the basis of Effort, but is not relevant by itself for the purpose of this evaluation.
- **Potential Volume**: The minimum disk size the program can fill. Not relevant to the aims of this evaluation.

To ensure that the metrics are applied fairly and accurately to the test code, a code parser “HalsteadMetric” has been created in C#. The parser first isolates code within task main which excludes the boilerplate pre-processor directives in accordance with the test methodology outlined previously. Next, all comments, string quotes and hidden whitespace characters such as carriage return and new line are stripped from the code, as these constructs do not affect the program implementation directly. All operators are then extracted from the code and recorded before the same process is applied to the remaining operands. Finally the
program name, two sets of raw data and the operator/operand counts are output in Comma Separated Value format.

Results for each test have then collated into a single table with labelled rows. Further metrics have been derived using formulas in Excel. In addition, average values have been computed for each action by taking the median values of their associated tests.

In order to break down the results further, several code snippets have been evaluated by the parser and added to the results table. These include small sections of larger algorithms such as variable tracking logic and wheel circumference to motor rotation conversion, as well as two simple Hello World applications written in pure C.
4. Results & Analysis
Note that the following list of programs lack Operands which causes a divide by zero error when calculating \( [\text{ND2} / \text{NT2}] \) as part of the Program Level. In these instances \( [0 / 0] \) has been defined as 1 which reduces Program Level to \( [2 / \text{ND1}] \): Action8_MoveUntilSensor, Action5_Move_Backwards, Action5_Move_Foward, Action5_Turn_Point, Action5_Turn_Swing.

4.1 Developer Notes
Full notes are available in the test source code which can be found in the appendix. Here an overview of the most pertinent observations is presented:

1. Lots of alternative functions exist for displaying text that all take the same format of `function(n, string)`.
2. a. Default values of 30 were sufficient for using input functions.
   b. Some of the input functions were not applicable to this action.
   c. untilBump could have been used in touch sensor test with no impact on code size.
3. There are no Natural Language functions to aid in Boolean comparisons which increases the complexity of these tests.
4. a. A couple of alternative ways exist to increment the counter variable. The more basic `+` operator has been preferred over `++` or `+=`.
   b. Outputting the counter adds a layer of complexity.
   c. Not restricting the counter to a single input makes the logic shorter, but means we have to use an unsigned long to avoid overflow.
   d. Light and Sonar functions required passing variables but Sound and Touch do not.
5. These require less code than displaying output.
6. a. Traveling specific distances require the tester to know the circumference of the wheels.
   b. These are also the only tests that require any maths.
   c. `Move_Distance_Backwards` is supported by Natural Language and requires complex RobotC code.
7. Natural Language makes simple movement very easy.
8. Sound more complicated than Sonar as default volume picks up the sound of the motors.
9. Had to provide a non-integer value for point Turn as the two combined turn the robot 360 degrees. Possibly due to the carpet surface the test was performed on.
4.2 Data

G1: Vocabulary and Length

G2: Program Level
4.2.1 Precise Movement

G1 displays the *vocabulary* (number of distinct elements) and *length* (number of total elements) of all tests. In general, alternative implementations of an action are of a similar length and contain almost identical vocabularies to their counterparts. This means that any implementation of a given action requires a similar understanding of fundamental operations and can be expressed with similar clarity.

The biggest exception to this is Move_Distance_Backwards which requires twice the vocabulary of Move_Distance_Forward and is almost double in length. This test is also considerably more *difficult* (has a higher difficulty rating) than any other and requires at least twice as much *effort* (higher effort rating).

In order to fulfil the precision requirement of this test it is necessary to sync the motors and ensure the encoder values are set to 0. The two lines of code needed to do this score a difficulty of 1.5 (Move_Distance_Backward_Setup) which is as difficult, or more so, than 37.5% of (9/24) tests. In addition the stalling loop (Move_Distance_Backward_Loop) scores a difficulty of 2.5 which is higher than 58% (14/24) of all tests.

Overall the *density* of code (intelligence content) for precise movement is higher than that of other actions; this is due in part to the circumference to rotation conversion which is more dense than over half of the tests (13/24). It should be noted that this action assumes that the circumference of the wheels is a known value and so avoids the potential additional overhead of converting wheel diameter to circumference.

It can be seen from these results that Natural Language has the potential to greatly reduce the complexity of RobotC code and that the methods with which the NXT measures distance travelled are less suited to introductory programming exercises.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Output.Counter</th>
<th>Variable_Tracking</th>
<th>SingleInput_Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct Operators</td>
<td>2.000</td>
<td>3.000</td>
<td>4.000</td>
</tr>
<tr>
<td>Distinct Operands</td>
<td>3.000</td>
<td>4.000</td>
<td>2.000</td>
</tr>
<tr>
<td>Total Operators</td>
<td>2.000</td>
<td>5.000</td>
<td>5.000</td>
</tr>
<tr>
<td>Total Operands</td>
<td>3.000</td>
<td>6.000</td>
<td>2.000</td>
</tr>
<tr>
<td><strong>Vocabulary</strong></td>
<td><strong>5.000</strong></td>
<td><strong>7.000</strong></td>
<td><strong>6.000</strong></td>
</tr>
<tr>
<td><strong>Length</strong></td>
<td><strong>5.000</strong></td>
<td><strong>11.000</strong></td>
<td><strong>7.000</strong></td>
</tr>
<tr>
<td>Volume</td>
<td>11.610</td>
<td>30.881</td>
<td>18.095</td>
</tr>
<tr>
<td><strong>Program Level</strong></td>
<td><strong>1.000</strong></td>
<td><strong>0.444</strong></td>
<td><strong>0.500</strong></td>
</tr>
<tr>
<td><strong>Difficulty</strong></td>
<td><strong>1.000</strong></td>
<td><strong>2.250</strong></td>
<td><strong>2.000</strong></td>
</tr>
<tr>
<td><strong>Intelligence Content</strong></td>
<td><strong>11.610</strong></td>
<td><strong>13.725</strong></td>
<td><strong>9.047</strong></td>
</tr>
<tr>
<td><strong>Effort</strong></td>
<td><strong>11.610</strong></td>
<td><strong>69.482</strong></td>
<td><strong>36.189</strong></td>
</tr>
</tbody>
</table>

4.2.2 Algorithmic Interaction

On average, tracking input is considered one of the lowest *level* implementations (program level) of any action and is also one of the densest. In addition, the effort rating for this action is considerably higher than any other, with each of its tests scoring more than all but Move_Distance_Backwards. Overall these tests represent a good mixture of RobotC and Natural Language with a 50% split when comparing significant lines of code (lines not including whitespace).
Dealing with tracking and displaying the variable (Variable_Tracking_Output) is responsible for a considerable amount of this action’s complexity. With an effort rating of 159 the code to do this is almost as difficult to implement as the Rover test at 166. It is also considered more difficult to understand than Pattern movement and twice as difficult as most forms of locomotion.

In the observations, providing output for the counter is noted as adding “a layer of complexity” to the code. However, when comparing the individual components of the test, it can be seen that Variable_Tracking not only scores the highest difficulty, but is also the densest and requires considerably more effort. In fact, Output.Counter is considered very high level and requires little effort compared to the other components. The disparity between observation and result is due to the use of a string replacement token as part of the output. Conceptually, this token requires some knowledge of data types (d vs f vs s) (CPlusPlus.com, 2014) and so might be considered more difficult to understand, however Halstead metrics make no attempt to assign additional weighting to specific operands and so this is not reflected in the results.

Much like input tracking, conditional input is more difficult than most actions and requires more effort. Here it is shown that the lack of Natural Language constructs once again contributes significantly to complexity, with the RobotC.Branching logic scoring similarly to the combined movement action.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>RobotC.Branching</th>
<th>Action 8 Combined Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct Operators</td>
<td>4.000</td>
<td>5</td>
</tr>
<tr>
<td>Distinct Operands</td>
<td>4.000</td>
<td>1</td>
</tr>
<tr>
<td>Total Operators</td>
<td>7.000</td>
<td>10</td>
</tr>
<tr>
<td>Total Operands</td>
<td>4.000</td>
<td>1</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>8.000</td>
<td>6</td>
</tr>
<tr>
<td>Length</td>
<td>11.000</td>
<td>11</td>
</tr>
<tr>
<td>Volume</td>
<td>33.000</td>
<td>28.435</td>
</tr>
<tr>
<td>Program Level</td>
<td>0.500</td>
<td>0.4</td>
</tr>
<tr>
<td>Difficulty</td>
<td>2.000</td>
<td>2.5</td>
</tr>
<tr>
<td>Intelligence Content</td>
<td>16.500</td>
<td>11.374</td>
</tr>
<tr>
<td>Effort</td>
<td>66.000</td>
<td>71.086</td>
</tr>
</tbody>
</table>

Of particular note is the intelligence content score of this snippet which is higher than two thirds (16/24) of all tests. As a result of this, LightSonar and TouchSound are denser than all tests, with the exception of Move_Distance_Backwards.

4.2.3 Basic Interaction
The two actions; output, and input-output, are conceptually very close to Hello World applications often used as introductory exercises when learning new languages.
Output is achieved using either sound or text. Sound is the most difficult and also requires the most effort. This is caused by the need to ensure that sound can be played by assigning a volume to the robot programatically. It is noted by the developer that this can be avoided by using the correct hardware settings and without this assignment the complexity reduces to that of Text_ScrollText.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Sound_NoVolume</th>
<th>Action1_Text_ScrollText</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct Operators</td>
<td>3.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Distinct Operands</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Total Operators</td>
<td>4.000</td>
<td>4.000</td>
</tr>
<tr>
<td>Total Operands</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td><strong>Vocabulary</strong></td>
<td>4.000</td>
<td>4.000</td>
</tr>
<tr>
<td><strong>Length</strong></td>
<td>5.000</td>
<td>5.000</td>
</tr>
<tr>
<td><strong>Volume</strong></td>
<td>10.000</td>
<td>10.000</td>
</tr>
<tr>
<td><strong>Program Level</strong></td>
<td>0.667</td>
<td>0.667</td>
</tr>
<tr>
<td><strong>Difficulty</strong></td>
<td>1.500</td>
<td>1.500</td>
</tr>
<tr>
<td><strong>Intelligence Content</strong></td>
<td>6.667</td>
<td>6.667</td>
</tr>
<tr>
<td><strong>Effort</strong></td>
<td>15.000</td>
<td>15.000</td>
</tr>
</tbody>
</table>

The developer also notes that Text_DisplayString can be implemented using a number of alternative functions that differ only in name. This would have no effect on the Halstead Metrics as the number of operators/operands does not change. The only difference between Text_DisplayString and Text_ScrollText is that the latter does not require a line number parameter.

Combining input and output increases the effort required by approximately 60% when compared with the output median. There is a variance of 0 between input-output tests, which shows that sensor use is homogenous in terms of complexity. It can also be seen that these tests remain fairly high level and are much less dense than algorithmic input tests. When coupled with the high complexity of RobotC_Branching and Variable_Tracking, this indicates that whilst it is relatively easy to ascertain if a sensor is receiving input, it is much more difficult to perform tasks based on their readings.
When compared to Hello World applications written in pure C, it can be shown that basic output maps quite closely, retaining the program level and difficulty at the cost of increased density and effort. Input-output however show dramatic increases, requiring three times the vocabulary and more than three times the length. The average difficulty of the plain C input-output programs is more than twice that of the output only application (3.6 vs 1.5 respectively) and requires an order of magnitude more effort (283.8). Of particular note is that these rudimental applications outweigh the pure RobotC conditional input in all areas:

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Action 3 Conditional Input</th>
<th>Plain C Input - Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct Operators</td>
<td>7.000</td>
<td>5.500</td>
</tr>
<tr>
<td>Distinct Operands</td>
<td>7.000</td>
<td>8.500</td>
</tr>
<tr>
<td>Total Operators</td>
<td>12.000</td>
<td>9.500</td>
</tr>
<tr>
<td>Total Operands</td>
<td>7.000</td>
<td>11.000</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>14.000</td>
<td>14.000</td>
</tr>
<tr>
<td>Length</td>
<td>19.000</td>
<td>20.500</td>
</tr>
<tr>
<td>Volume</td>
<td>72.340</td>
<td>78.233</td>
</tr>
<tr>
<td>Program Level</td>
<td>0.286</td>
<td>0.285</td>
</tr>
<tr>
<td>Difficulty</td>
<td>3.500</td>
<td>3.563</td>
</tr>
<tr>
<td>Intelligence Content</td>
<td>20.668</td>
<td>21.890</td>
</tr>
<tr>
<td>Effort</td>
<td>253.189</td>
<td>283.792</td>
</tr>
</tbody>
</table>

The fact that more difficult actions require less effort in RobotC than in C, suggests that the former is a less complicated language than its predecessor, although without a more direct comparison it is difficult to say this with any certainty. This assessment becomes more difficult as action complexity increases, due to the different feedback mechanisms of the two platforms: one is most often keyboard/mouse based and relies entirely on visual output, whilst robotics leans more on sensing tangible environment changes and motor adjustments.

4.2.4 Basic Movement

Simple movement is on average one of the least complex actions that can be performed on the NXT; showing a lower difficulty than the average output action and requiring very small vocabularies with very little repetition.

The code for simple movement benefits greatly from the Natural Language API taking full advantage of default parameters to allow for imprecise locomotion. Where the maths requirement of distance movement causes an increase in complexity, adding a time dimension causes minimal impact: the difference between Move_Backward and Move_Time_Backward is a single operand which has no bearing on either difficulty or program level and causes only a minimal increase in effort and density.

Move_Time_Foward is the least complex of all actions tested, requiring only a single function call passing one parameter. This action is almost universally the simplest with the shortest vocabulary and length, the highest program level, the lowest difficulty, and the lowest effort. The code needed is slightly denser than the simple movement algorithms due to length and vocabulary equality. The constraints of the test required that the robot move for 10
seconds, however if this had been reduced to 1 second than passing a parameter to the moveStraightForTime could have been avoided, resulting in the absolute lowest possible score for any application that performs an action (one function call + semi-colon, which is less than a single Variable_Assignment).

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Snippet_Move_Time_Forward_1Sec</th>
<th>Variable_Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct Operators</td>
<td>2.000</td>
<td>2.000</td>
</tr>
<tr>
<td>Distinct Operands</td>
<td>0.000</td>
<td>2.000</td>
</tr>
<tr>
<td>Total Operators</td>
<td>2.000</td>
<td>2.000</td>
</tr>
<tr>
<td>Total Operands</td>
<td>0.000</td>
<td>2.000</td>
</tr>
<tr>
<td><strong>Vocabulary</strong></td>
<td><strong>2.000</strong></td>
<td><strong>4.000</strong></td>
</tr>
<tr>
<td><strong>Length</strong></td>
<td><strong>2.000</strong></td>
<td><strong>4.000</strong></td>
</tr>
<tr>
<td><strong>Program Level</strong></td>
<td><strong>1.000</strong></td>
<td><strong>1.000</strong></td>
</tr>
<tr>
<td><strong>Difficulty</strong></td>
<td><strong>1.000</strong></td>
<td><strong>1.000</strong></td>
</tr>
<tr>
<td><strong>Intelligence Content</strong></td>
<td><strong>2.000</strong></td>
<td><strong>8.000</strong></td>
</tr>
<tr>
<td><strong>Effort</strong></td>
<td><strong>2.000</strong></td>
<td><strong>8.000</strong></td>
</tr>
</tbody>
</table>

The difference between Move_Time_Forward and Move_Distance_Backward highlights how a high level API like Natural Language has the potential to greatly reduce the complexity of RobotC and make Mindstorm programming more accessible.

### 4.2.5 Algorithmic Movement

The very low complexity of basic movement translates well to more complicated movement that combine different forms of locomotion. Pattern, Rover and Move_Until_Sensor represent more high-level robotic behaviour than the previously discussed tests, but score much lower than most other forms of movement. In particular Move_Until_Sensor is no more difficult than simple movement despite combining both locomotion and input, and in terms of effort and density is less complex than timed movement.

Pattern combines three different movement types but remains relatively simple. It uses a small vocabulary, requiring only slightly more distinct elements than output, but has a much greater length than many of the more straightforward actions. This in turn causes the difficulty rating to rise beyond that of input-output by 25%. Interestingly this is not well reflected in the effort rating which is more than doubled. It is noted in the observations that Natural Language functions make movement easy to implement and this would appear to be reflected in both simple and combined movement. This ability to scale nicely as code requirements increases is also well represented by the Rover action.

Rover is by far the most demanding of all the actions combining movement, sensor input and repetition to produce very basic autonomous behaviour. Despite this it is still considerably less complex than many tests, requiring a 20% smaller vocabulary than Move_Distance_Forward despite being over twice as long as any of the input-output tests. A difficulty of 3.5 places Rover on par with conditional input, which requires much denser code. Whilst more difficult than Hello_NoPrompt, Rover is less difficult than Hello_Prompt and requires much less than effort than either.
4.3 Summary
Natural Language does an excellent job of masking the complexity of RobotC. By far the most complicated tests are ones in which the developer has been forced to move away from the API and rely on plain RobotC. This is particularly evident when comparing the RobotC_Branching snippet with the combined movement action, in which three types of locomotion score similarly to a simple if statement.

In general, the high level movement abstractions make locomotion a particularly simple task. Arbitrary and time-measured movement tasks are some of the least difficult thanks to Natural Language. The most obvious example of this is the vast difference between the distance movement tests, where lack of support for moving backwards vastly increases both difficulty and effort. Looking at its implementation, the moveStraightForRotations function dwarfs all of the actions tested by quite some margin, requiring more than double the effort of Move_Distance_Backward which is itself very complicated in comparison to other tests.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>MoveStraightForRotations</th>
<th>Action6_Move_Distance_Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct Operators</td>
<td>12.000</td>
<td>8.000</td>
</tr>
<tr>
<td>Distinct Operands</td>
<td>15.000</td>
<td>14.000</td>
</tr>
<tr>
<td>Total Operators</td>
<td>41.000</td>
<td>21.000</td>
</tr>
<tr>
<td>Total Operands</td>
<td>24.000</td>
<td>21.000</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>27.000</td>
<td>22.000</td>
</tr>
<tr>
<td>Length</td>
<td>65.000</td>
<td>42.000</td>
</tr>
<tr>
<td>Volume</td>
<td>309.068</td>
<td>187.296</td>
</tr>
<tr>
<td>Program Level</td>
<td>0.104</td>
<td>0.167</td>
</tr>
<tr>
<td>Difficulty</td>
<td>9.600</td>
<td>6.000</td>
</tr>
<tr>
<td>Intelligence Content</td>
<td>32.195</td>
<td>31.216</td>
</tr>
<tr>
<td>Effort</td>
<td>2,967.050</td>
<td>1,123.777</td>
</tr>
</tbody>
</table>

Despite this, distance movement remains an area where more work could be done to improve the API, perhaps by allowing wheel circumference and distance to be passed as parameters to a function. Doing so would reduce the action to a difficulty of 1 and effort of 8, turning it from one of the most difficult actions into the simplest. In its current state, distance movement is not well suited as an introductory programming exercise.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Action1_Text_DisplayString</th>
<th>Action1_Text_ScrollText</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct Operators</td>
<td>3.000</td>
<td>3.000</td>
</tr>
<tr>
<td>Distinct Operands</td>
<td>2.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Total Operators</td>
<td>4.000</td>
<td>4.000</td>
</tr>
<tr>
<td>Total Operands</td>
<td>2.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>5.000</td>
<td>4.000</td>
</tr>
<tr>
<td>Length</td>
<td>6.000</td>
<td>5.000</td>
</tr>
<tr>
<td>Volume</td>
<td>13.932</td>
<td>10.000</td>
</tr>
<tr>
<td>Program Level</td>
<td>0.667</td>
<td>0.667</td>
</tr>
<tr>
<td>Difficulty</td>
<td>1.500</td>
<td>1.500</td>
</tr>
<tr>
<td>Intelligence Content</td>
<td>9.288</td>
<td>6.667</td>
</tr>
<tr>
<td>Effort</td>
<td>20.897</td>
<td>15.000</td>
</tr>
</tbody>
</table>
Using sensors is a relatively easy task and of particular note is the similarity between their complexity levels. This is a direct result of the “until sensor” functions such as untilTouch which all follow a similar pattern and allow for the use of default parameters. These score well with Halstead metrics because it helps reduce the number of operands quite significantly. The effects of this can be quite clearly seen when comparing Text_DisplayString and Text_ScrollText which differ by a single operand.

Whilst the API helps to ascertain when a sensor receives input, allowing an action to be carried out until it occurs, it does not do much to aid logical comparisons of sensors. This can be seen quite clearly in complexity ratings of the conditional input action, which are forced to rely on plain RobotC methods.
5. Conclusion

5.1 Suitability of Natural Language for Education

In terms of complexity, plain C is much less suited to introductory programming than RobotC where even very simple input-output actions quickly rise in difficulty and effort. Results suggest that RobotC is capable of producing more complex behaviour with less complex code, even when forced to move away from the Natural Language API.

Despite this, areas where pure RobotC have been used are dramatically more complicated than their Natural Language counterparts. Whilst RobotC fairs better than its parent language, the simplicity of Natural Language is far more notable. Analysis of the results indicates that this is partially down to the availability of default parameters which allow the API to score well with Halstead metrics.

The use of default parameters would help enable a progressive learning path by allowing programming concepts to be introduced gradually. Tests such as Pattern and MoveUntilSensor show that it is possible to create locomotion programs that rely solely on function calls. Once students became more confident using these constructs, the parameters could be introduced one by one slowly building on existing knowledge.

A high–level API such as Natural Language has the potential to greatly reduce the complexity of RobotC and makes the Mindstorms platform more accessible by narrowing the breadth of vocabulary necessary to produce working programs. This shows that the initial hypothesis - that RobotC is unsuitable for key-stage attainment targets - is only partly true. Whilst Natural Language caters much better for beginner programmers than either pure RobotC or plain C, it does not provide a rich enough selection of functions to adequately address all the development areas tested.

Where Natural Language does provide a good selection of functions is time based movement. The results in this area are very promising and the Rover test shows that the API allows for the creation of simple autonomous behaviour with only a few lines of code. Conversely, distance movement functions require a considerable amount of overhead and are not particularly well supported by Natural Language.

5.2 Issues with Halstead Metrics

Halstead metrics have been designed as a way to quantify code complexity in a general way that does not take the underlying concept into account. For instance, the developer noted that providing output for variable tracking added “a layer of complexity”, although the results seem to suggest that this is incorrect. Whilst well suited to expressing code complexity, Halstead metrics do little to evaluate the complexity of the underlying computer science topics.

A more refined approach might be to apply a weighting to code elements based on their conceptual complexity. Extensive testing would be required in order to establish which concepts provide the most difficulty, particularly for young children. This is a somewhat underdeveloped area of research and very little quantifiable data is available beyond that gained through sometimes subjective questionnaire methods.

Ultimately this may lead to the creation of a more specific and robust metric that would be better tailored to analysing language suitability.
5.3 Issues with Testing
The selection of tests used in this paper help to show how a range of different actions can be achieved using a multitude of different implementations. Due to the homogeneous approach of Natural Language this has created some redundancy, particularly when testing input-output and simple movement. Focus on a broader set of activities would have generated a more diverse set of data.

Further to this, these tests take a fairly academic approach to language suitability by having an experienced developer implement the algorithms. Whilst this was a suitable approach in ascertaining code complexity it does not truly evaluate a child’s experience with the API. Given more time this would be an excellent continuation of this paper.

In addition this paper focuses too much on a single language. A more in-depth approach would be to repeat the same tests across a number of NXT languages and Hello World applications. Data gathered from other languages would help to better contextualise the results from this paper.

5.4 Recommendations
Despite the shortcomings of Natural Language, with careful structuring it could be an effective tool in fulfilling key-stage two attainment targets, such as those highlighted in the introduction: in addition to allowing the manipulation of physical systems, it is well suited to the creation of programs for specific tasks, such as the locomotion expressed in the Pattern and Rover tests.

In order to allow for a gentle introduction, these tasks would need to start as simple as possible, focusing on arbitrary moment for time and possibly including sensor readings. Once confidence has been instilled, lessons would focus on introducing parameters and encouraging more finely grained control over the actions of the robot. This serves as a very natural precursor to the output functions which do not utilise default parameters quite as successfully as locomotion. From here there exist a number of possible routes for moving towards pure RobotC such as introducing functions, variables, control structures or arrays. Ideally this path would pay close attention to the perceived difficulty of the concept; building gradually to an approach that is closer to plain C. Clear progression paths such as is an area that requires extensive further research, with particular attention being paid to how the complexity of a concept affects the learning curve.

It is hoped that Graphical Natural Language and its ability to map directly to Natural Language would facilitate key-stage one targets and serve as an introduction to both the API and programming in general. Unfortunately the NXT implementation of GNL has yet to be released and so is unavailable for testing at this time.

Future studies should focus on testing the validity of the GNL to Natural Language progression and a more in-depth look at the progression path as suggested above. Other areas of research that fall outside the scope of this paper might include an evaluation of teaching method suitability and a more in-depth review of tool chain complexity, including the pros and cons of using an integrated development environment and the cognitive impact of code compilation.
6. Bibliography


7. Appendix

7.1 Test Results

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Action1_Sound</th>
<th>Action1_Text_DisplayString</th>
<th>Action1_Text_ScrollText</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Operarors:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>; ; ; ; =, PlaySound, wait,</td>
<td>; ; ; nxtDisplayString, wait,</td>
<td>; ; ; nxtScrollText, wait,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operators:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>; , ; , =, PlaySound, wait,</td>
<td>; , nxtDisplayString, wait,</td>
<td>; , nxtScrollText, wait,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operands:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>nVolume, soundLowBuzz, 1,</td>
<td>1, Education,</td>
<td>Education,</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ND1</td>
<td>Distinct Operators</td>
<td>4.000</td>
</tr>
<tr>
<td>ND2</td>
<td>Distinct Operators</td>
<td>3.000</td>
</tr>
<tr>
<td>NT1</td>
<td>Total Operators</td>
<td>6.000</td>
</tr>
<tr>
<td>NT2</td>
<td>Total Operators</td>
<td>3.000</td>
</tr>
<tr>
<td>VOC</td>
<td>ND1 + ND2</td>
<td>Vocabulary</td>
</tr>
<tr>
<td>LTH</td>
<td>NT1 + NT2</td>
<td>Length</td>
</tr>
<tr>
<td>VOL</td>
<td>LTH * log2(VOC)</td>
<td>Volume</td>
</tr>
<tr>
<td>LVL</td>
<td>2/ND1 * ND2/NT2</td>
<td>Program Level</td>
</tr>
<tr>
<td>DIF</td>
<td>1 / LVL</td>
<td>Difficulty</td>
</tr>
<tr>
<td>INT</td>
<td>LVL * VOL</td>
<td>Intelligence Content</td>
</tr>
<tr>
<td>EFF</td>
<td>VOL / LVL</td>
<td>Effort</td>
</tr>
</tbody>
</table>
### Operators:

- Action2_Light: `;; ;;`` n NXTScrollText, untilDark, wait,``
- Action2_Sonar: `;; ;;`` n NXTScrollText, untilSonarLessThan, wait,``
- Action2_Sound: `;; ;;`` n NXTScrollText, untilSoundGreaterThan, wait,``
- Action2_Touch: `;; ;;`` n NXTScrollText, untilTouch, wait,``

### Operands:

- Dark
- Close
- Loud
- Touched

### Metric | Formula | Meaning
--- | --- | ---
ND1 | Distinct Operators | 4.000
ND2 | Distinct Operators | 1.000
NT1 | Total Operators | 6.000
NT2 | Total Operators | 1.000
VOC | ND1 + ND2 | 5.000
LTH | NT1 + NT2 | 7.000
VOL | LTH * log2(VOC) | 16.253
LVL | 2/(ND1 * ND2/NT2) | 0.500
DIF | 1 / LVL | 2.000
INT | LVL * VOL | 8.127
EFF | VOL / LVL | 32.507
<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ND1</td>
<td>Distinct Operators</td>
<td>7.000</td>
</tr>
<tr>
<td>ND2</td>
<td>Distinct Operands</td>
<td>7.000</td>
</tr>
<tr>
<td>NT1</td>
<td>Total Operators</td>
<td>12.000</td>
</tr>
<tr>
<td>NT2</td>
<td>Total Operands</td>
<td>7.000</td>
</tr>
<tr>
<td>VOC</td>
<td>ND1 + ND2</td>
<td>Vocabulary</td>
</tr>
<tr>
<td>LTH</td>
<td>NT1 + NT2</td>
<td>Length</td>
</tr>
<tr>
<td>VOL</td>
<td>LTH * log2(VOC)</td>
<td>Volume</td>
</tr>
<tr>
<td>LVL</td>
<td>2/ND1 * ND2/NT2</td>
<td>Program Level</td>
</tr>
<tr>
<td>DIF</td>
<td>1 / LVL</td>
<td>Difficulty</td>
</tr>
<tr>
<td>INT</td>
<td>LVL * VOL</td>
<td>Intelligence Content</td>
</tr>
<tr>
<td>EFF</td>
<td>VOL / LVL</td>
<td>Effort</td>
</tr>
<tr>
<td>Metric</td>
<td>Formula</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>ND1</td>
<td>Distinct Operators</td>
<td>7.000</td>
</tr>
<tr>
<td>ND2</td>
<td>Distinct Operands</td>
<td>8.000</td>
</tr>
<tr>
<td>NT1</td>
<td>Total Operators</td>
<td>12.000</td>
</tr>
<tr>
<td>NT2</td>
<td>Total Operands</td>
<td>11.000</td>
</tr>
<tr>
<td>VOC</td>
<td>ND1 + ND2</td>
<td>15.000</td>
</tr>
<tr>
<td>LTH</td>
<td>NT1 + NT2</td>
<td>23.000</td>
</tr>
<tr>
<td>VOL</td>
<td>LTH * \log_2(VOC)</td>
<td>89.858</td>
</tr>
<tr>
<td>LVL</td>
<td>2/ND1 * ND2/NT2</td>
<td>0.208</td>
</tr>
<tr>
<td>DIF</td>
<td>1 / LVL</td>
<td>4.813</td>
</tr>
<tr>
<td>INT</td>
<td>LVL * VOL</td>
<td>18.672</td>
</tr>
<tr>
<td>EFF</td>
<td>VOL / LVL</td>
<td>432.444</td>
</tr>
</tbody>
</table>

**Operators:**
- `; ; ; ; ; ; + , =, while, nxtDisplayString, untilDark, untilLight,
- `; ; ; ; ; ; + , =, while, nxtDisplayString, untilSonarGreaterThan, untilSonarLessThan,
- `; ; ; ; ; ; + , =, while, nxtDisplayString, untilSoundGreaterThan, untilSoundLessThan,
- `; ; ; ; ; ; + , =, while, nxtDisplayString, untilBump,

**Operands:**
- `%d, counter, counter, counter, counter, int, true, 0, 1, 1, 40,
- `%d, counter, counter, counter, counter, int, true, 0, 1, 1, 35,
- `%d, counter, counter, counter, counter, int, true, 0, 1, 1,
<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ND1</td>
<td>ND1</td>
<td>3.000</td>
</tr>
<tr>
<td>ND2</td>
<td>ND2</td>
<td>0.000</td>
</tr>
<tr>
<td>NT1</td>
<td>NT1</td>
<td>4.000</td>
</tr>
<tr>
<td>NT2</td>
<td>NT2</td>
<td>0.000</td>
</tr>
<tr>
<td>VOC</td>
<td>VOC</td>
<td>3.000</td>
</tr>
<tr>
<td>LTH</td>
<td>LTH</td>
<td>4.000</td>
</tr>
<tr>
<td>VOL</td>
<td>VOL</td>
<td>6.340</td>
</tr>
<tr>
<td>LVL</td>
<td>LVL</td>
<td>0.667</td>
</tr>
<tr>
<td>DIF</td>
<td>DIF</td>
<td>1.500</td>
</tr>
<tr>
<td>INT</td>
<td>INT</td>
<td>4.227</td>
</tr>
<tr>
<td>EFF</td>
<td>EFF</td>
<td>9.510</td>
</tr>
</tbody>
</table>

**Operators:** Action5_Move_Backward, Action5_Move_Forward, Action5_Turn_Point, Action5_Turn_Swing

**Operands:** ; ; backward, wait, ; ; forward, wait, ; ; pointTurn, wait, ; ; swingTurn, wait.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Meaning</th>
<th>Action6_Move_Distance_Backward</th>
<th>Action6_Move_Distance_Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>ND1</td>
<td>Distinct Operators</td>
<td>8.000</td>
<td>4.000</td>
<td></td>
</tr>
<tr>
<td>ND2</td>
<td>Distinct Operands</td>
<td>14.000</td>
<td>7.000</td>
<td></td>
</tr>
<tr>
<td>NT1</td>
<td>Total Operators</td>
<td>21.000</td>
<td>9.000</td>
<td></td>
</tr>
<tr>
<td>NT2</td>
<td>Total Operands</td>
<td>21.000</td>
<td>13.000</td>
<td></td>
</tr>
<tr>
<td>VOC</td>
<td>( ND1 + ND2 )</td>
<td>Vocabulary</td>
<td>22.000</td>
<td>11.000</td>
</tr>
<tr>
<td>LTH</td>
<td>( NT1 + NT2 )</td>
<td>Length</td>
<td>42.000</td>
<td>22.000</td>
</tr>
<tr>
<td>VOL</td>
<td>( \frac{LTH \times \log_2(VOC)}{2/ND1 \times ND2/NT2} )</td>
<td>Volume</td>
<td>187.296</td>
<td>76.107</td>
</tr>
<tr>
<td>LVL</td>
<td>( \frac{2/ND1 \times ND2/NT2}{2/ND1 \times ND2/NT2} )</td>
<td>Program Level</td>
<td>0.167</td>
<td>0.269</td>
</tr>
<tr>
<td>DIF</td>
<td>( \frac{1}{LVL} )</td>
<td>Difficulty</td>
<td>6.000</td>
<td>3.714</td>
</tr>
<tr>
<td>INT</td>
<td>( LVL \times VOL )</td>
<td>Intelligence Content</td>
<td>31.216</td>
<td>20.490</td>
</tr>
<tr>
<td>EFF</td>
<td>( \frac{VOL}{LVL} )</td>
<td>Effort</td>
<td>1,123.777</td>
<td>282.685</td>
</tr>
</tbody>
</table>

Operators: -, *, /, ;, =, =, =, =, >, while, const, float, float, motorrightWheel, nMotorEncoderrightWheel, nMotorEncoderrightWheel, nSyncedMotors, rotations, synchBC, targetDistance, wheelCircumference, 0, 17.2, 30, 360.

Operands: const, const, float, float, float, motorrightWheel, nMotorEncoderrightWheel, nMotorEncoderrightWheel, nSyncedMotors, rotations, rotations, synchBC, targetDistance, targetDistance, wheelCircumference, wheelCircumference, 0, 17.2, 30, 360.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Meaning</th>
<th>Action7_Move_Time_Backward</th>
<th>Action7_Move_Time_Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>ND1</td>
<td>Distinct Operators</td>
<td>3.000</td>
<td>2.000</td>
<td></td>
</tr>
<tr>
<td>ND2</td>
<td>Distinct Operands</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>NT1</td>
<td>Total Operators</td>
<td>4.000</td>
<td>2.000</td>
<td></td>
</tr>
<tr>
<td>NT2</td>
<td>Total Operands</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>VOC</td>
<td>ND1 + ND2</td>
<td>Vocabulary</td>
<td>4.000</td>
<td>3.000</td>
</tr>
<tr>
<td>LTH</td>
<td>NT1 + NT2</td>
<td>Length</td>
<td>5.000</td>
<td>3.000</td>
</tr>
<tr>
<td>VOL</td>
<td>LTH * log2(VOC)</td>
<td>Volume</td>
<td>10.000</td>
<td>4.755</td>
</tr>
<tr>
<td>LVL</td>
<td>2/ND1 * ND2/NT2</td>
<td>Program Level</td>
<td>0.667</td>
<td>1.000</td>
</tr>
<tr>
<td>DIF</td>
<td>1 / LVL</td>
<td>Difficulty</td>
<td>1.500</td>
<td>1.000</td>
</tr>
<tr>
<td>INT</td>
<td>LVL * VOL</td>
<td>Intelligence Content</td>
<td>6.667</td>
<td>4.755</td>
</tr>
<tr>
<td>EFF</td>
<td>VOL / LVL</td>
<td>Effort</td>
<td>15.000</td>
<td>4.755</td>
</tr>
<tr>
<td>Metric</td>
<td>Formula</td>
<td>Meaning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ND1</td>
<td>Distinct Operators</td>
<td>5.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ND2</td>
<td>Distinct Operands</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NT1</td>
<td>Total Operators</td>
<td>10.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NT2</td>
<td>Total Operands</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOC</td>
<td>ND1 + ND2</td>
<td>Vocabulary</td>
<td>6.000</td>
<td></td>
</tr>
<tr>
<td>LTH</td>
<td>NT1 + NT2</td>
<td>Length</td>
<td>11.000</td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>LTH * log2(VOC)</td>
<td>Volume</td>
<td>28.435</td>
<td></td>
</tr>
<tr>
<td>LVL</td>
<td>2/ND1 * ND2/NT2</td>
<td>Program Level</td>
<td>0.400</td>
<td></td>
</tr>
<tr>
<td>DIF</td>
<td>1 / LVL</td>
<td>Difficulty</td>
<td>2.500</td>
<td></td>
</tr>
<tr>
<td>INT</td>
<td>LVL * VOL</td>
<td>Intelligence Content</td>
<td>11.374</td>
<td></td>
</tr>
<tr>
<td>EFF</td>
<td>VOL / LVL</td>
<td>Effort</td>
<td>71.086</td>
<td></td>
</tr>
<tr>
<td>Metric</td>
<td>Formula</td>
<td>Meaning</td>
<td>C_Hello</td>
<td>C_Hello_NoPrompt</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>-------------</td>
<td>---------</td>
<td>------------------</td>
</tr>
<tr>
<td>ND1</td>
<td>Distinct Operators</td>
<td>3.000</td>
<td>5.000</td>
<td>6.000</td>
</tr>
<tr>
<td>ND2</td>
<td>Distinct Operands</td>
<td>3.000</td>
<td>8.000</td>
<td>9.000</td>
</tr>
<tr>
<td>NT1</td>
<td>Total Operators</td>
<td>4.000</td>
<td>8.000</td>
<td>11.000</td>
</tr>
<tr>
<td>NT2</td>
<td>Total Operands</td>
<td>3.000</td>
<td>10.000</td>
<td>12.000</td>
</tr>
<tr>
<td>VOC</td>
<td>ND1 + ND2</td>
<td>Vocabulary</td>
<td>6.000</td>
<td>13.000</td>
</tr>
<tr>
<td>LTH</td>
<td>NT1 + NT2</td>
<td>Length</td>
<td>7.000</td>
<td>18.000</td>
</tr>
<tr>
<td>VOL</td>
<td>LTH * log2(VOC)</td>
<td>Volume</td>
<td>18.095</td>
<td>66.608</td>
</tr>
<tr>
<td>LVL</td>
<td>2/ND1 * ND2/NT2</td>
<td>Program Level</td>
<td>0.667</td>
<td>0.320</td>
</tr>
<tr>
<td>DIF</td>
<td>1 / LVL</td>
<td>Difficulty</td>
<td>1.500</td>
<td>3.125</td>
</tr>
<tr>
<td>INT</td>
<td>LVL * VOL</td>
<td>Intelligence Content</td>
<td>12.063</td>
<td>21.315</td>
</tr>
<tr>
<td>EFF</td>
<td>VOL / LVL</td>
<td>Effort</td>
<td>27.142</td>
<td>208.150</td>
</tr>
<tr>
<td>Metric</td>
<td>Formula</td>
<td>Meaning</td>
<td>Distance_Movement_Maths</td>
<td>Move_Distance_Backward_Loop</td>
</tr>
<tr>
<td>--------</td>
<td>---------------</td>
<td>-----------------</td>
<td>-------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ND1</td>
<td>ND1 + ND2</td>
<td>Distinct Operators</td>
<td>3.000</td>
<td>5.000</td>
</tr>
<tr>
<td>ND2</td>
<td>ND1 + ND2</td>
<td>Distinct Operands</td>
<td>7.000</td>
<td>3.000</td>
</tr>
<tr>
<td>NT1</td>
<td>NT1 + NT2</td>
<td>Total Operators</td>
<td>7.000</td>
<td>5.000</td>
</tr>
<tr>
<td>NT2</td>
<td>NT1 + NT2</td>
<td>Total Operands</td>
<td>12.000</td>
<td>3.000</td>
</tr>
<tr>
<td>VOC</td>
<td>ND1 + ND2</td>
<td>Vocabulary</td>
<td>10.000</td>
<td>8.000</td>
</tr>
<tr>
<td>LTH</td>
<td>NT1 + NT2</td>
<td>Length</td>
<td>19.000</td>
<td>8.000</td>
</tr>
<tr>
<td>VOL</td>
<td>(\frac{LTH \cdot \log_2(VOC)}{2/ND1 * ND2/NT2})</td>
<td>Volume</td>
<td>63.117</td>
<td>24.000</td>
</tr>
<tr>
<td>LVL</td>
<td>(\frac{2/ND1 * ND2/NT2}{ND1 + ND2})</td>
<td>Program Level</td>
<td>0.389</td>
<td>0.400</td>
</tr>
<tr>
<td>DIF</td>
<td>(\frac{1}{LVL})</td>
<td>Difficulty</td>
<td>2.571</td>
<td>2.500</td>
</tr>
<tr>
<td>INT</td>
<td>LVL * VOL</td>
<td>Intelligence Content</td>
<td>24.545</td>
<td>9.600</td>
</tr>
<tr>
<td>EFF</td>
<td>VOL / LVL</td>
<td>Effort</td>
<td>162.300</td>
<td>60.000</td>
</tr>
</tbody>
</table>
### MoveStraightForRotations

Operators:

- , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , <, =, =, =, =, =, >, else, else, abs, if, if, wait1Msec, while,

- , nxtDisplayString,

- [], [], >, >, else, if, if,

- %d, 1, counter,

- SensorValuesound, SensorValuetouch, 0, 30,

Operands:

- motorleftEncoderPort, motorleftEncoderPort, motorleftEncoderPort, motorrightEncoderPort, motorrightEncoderPort, motorrightEncoderPort, nMotorEncoderleftEncoderPort, nMotorEncoderleftEncoderPort, nMotorEncoderleftEncoderPort, nMotorEncoderrightEncoderPort, nMotorEncoderrightEncoderPort, nMotorEncoderrightEncoderPort, nMotorEncoderrightEncoderPort, rotations, 0, 0, 1, 50, 50, 75, 75, 75, 360,

### ND1

Distinct Operators

- 12.000

### ND2

Distinct Operands

- 15.000

### NT1

Total Operators

- 41.000

### NT2

Total Operands

- 24.000

### VOC

\( ND1 + ND2 \)

Vocabulary

- 27.000

### LTH

\( NT1 + NT2 \)

Length

- 65.000

### VOL

\( LTH \times \log_2(VOC) \)

Volume

- 309.068

### LVL

\( \frac{2}{ND1 \times ND2/NT2} \)

Program Level

- 0.104

### DIF

\( \frac{1}{LVL} \)

Difficulty

- 9.600

### INT

\( LVL \times VOL \)

Intelligence Content

- 32.195

### EFF

\( VOL / LVL \)

Effort

- 2,967.050

### Meanings

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ND1</td>
<td>Distinct Operators</td>
<td>12.000</td>
</tr>
<tr>
<td>ND2</td>
<td>Distinct Operands</td>
<td>15.000</td>
</tr>
<tr>
<td>NT1</td>
<td>Total Operators</td>
<td>41.000</td>
</tr>
<tr>
<td>NT2</td>
<td>Total Operands</td>
<td>24.000</td>
</tr>
<tr>
<td>VOC</td>
<td>( ND1 + ND2 )</td>
<td>27.000</td>
</tr>
<tr>
<td>LTH</td>
<td>( NT1 + NT2 )</td>
<td>65.000</td>
</tr>
<tr>
<td>VOL</td>
<td>( LTH \times \log_2(VOC) )</td>
<td>309.068</td>
</tr>
<tr>
<td>LVL</td>
<td>( \frac{2}{ND1 \times ND2/NT2} )</td>
<td>0.104</td>
</tr>
<tr>
<td>DIF</td>
<td>( \frac{1}{LVL} )</td>
<td>9.600</td>
</tr>
<tr>
<td>INT</td>
<td>( LVL \times VOL )</td>
<td>32.195</td>
</tr>
<tr>
<td>EFF</td>
<td>( VOL / LVL )</td>
<td>2,967.050</td>
</tr>
<tr>
<td>Metric</td>
<td>Formula</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td></td>
<td>SingleInput_Response</td>
<td>Snippet_Move_Time_Forward_1Sec</td>
</tr>
<tr>
<td></td>
<td>Operators:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>; ; untilSonarGreaterThan,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>; untilSonarLessThan,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>while,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operands: true, 35,</td>
<td></td>
</tr>
<tr>
<td>ND1</td>
<td>Distinct Operators</td>
<td>4.000</td>
</tr>
<tr>
<td>ND2</td>
<td>Distinct Operands</td>
<td>2.000</td>
</tr>
<tr>
<td>NT1</td>
<td>Total Operators</td>
<td>5.000</td>
</tr>
<tr>
<td>NT2</td>
<td>Total Operands</td>
<td>2.000</td>
</tr>
<tr>
<td>VOC</td>
<td>ND1 + ND2</td>
<td>Vocabulary</td>
</tr>
<tr>
<td>LTH</td>
<td>NT1 + NT2</td>
<td>Length</td>
</tr>
<tr>
<td>VOL</td>
<td>LTH * log2(VOC)</td>
<td>Volume</td>
</tr>
<tr>
<td>LVL</td>
<td>2/ND1 * ND2/NT2</td>
<td>Program Level</td>
</tr>
<tr>
<td>DIF</td>
<td>1 / LVL</td>
<td>Difficulty</td>
</tr>
<tr>
<td>INT</td>
<td>LVL * VOL</td>
<td>Intelligence Content</td>
</tr>
<tr>
<td>EFF</td>
<td>VOL / LVL</td>
<td>Effort</td>
</tr>
<tr>
<td>Metric</td>
<td>Formula</td>
<td>Variable_Assignment</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>ND1</td>
<td>Distinct Operators</td>
<td>2.000</td>
</tr>
<tr>
<td>ND2</td>
<td>Distinct Operands</td>
<td>2.000</td>
</tr>
<tr>
<td>NT1</td>
<td>Total Operators</td>
<td>2.000</td>
</tr>
<tr>
<td>NT2</td>
<td>Total Operands</td>
<td>2.000</td>
</tr>
<tr>
<td>VOC</td>
<td>ND1 + ND2</td>
<td>Vocabulary</td>
</tr>
<tr>
<td>LTH</td>
<td>NT1 + NT2</td>
<td>Length</td>
</tr>
<tr>
<td>VOL</td>
<td>LTH * log2(VOC)</td>
<td>Volume</td>
</tr>
<tr>
<td>LVL</td>
<td>2/ND1 * ND2/NT2</td>
<td>Program Level</td>
</tr>
<tr>
<td>INT</td>
<td>LVL * VOL</td>
<td>Intelligence Content</td>
</tr>
<tr>
<td>EFF</td>
<td>VOL / LVL</td>
<td>Effort</td>
</tr>
</tbody>
</table>
### 7.2 Average Results

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Meaning</th>
<th>Action 1 Output</th>
<th>Action 2 Input &amp; Output</th>
<th>Action 3 Conditional Input</th>
<th>Action 4 Input Tracking</th>
<th>Action 5 Simple Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>ND1</td>
<td></td>
<td>Distinct Operators</td>
<td>3.000</td>
<td>4.000</td>
<td>7.000</td>
<td>7.000</td>
<td>3.000</td>
</tr>
<tr>
<td>ND2</td>
<td></td>
<td>Distinct Operands</td>
<td>2.000</td>
<td>1.000</td>
<td>7.000</td>
<td>7.500</td>
<td>0.000</td>
</tr>
<tr>
<td>NT1</td>
<td></td>
<td>Total Operators</td>
<td>4.000</td>
<td>6.000</td>
<td>12.000</td>
<td>12.000</td>
<td>4.000</td>
</tr>
<tr>
<td>NT2</td>
<td></td>
<td>Total Operands</td>
<td>2.000</td>
<td>1.000</td>
<td>7.000</td>
<td>10.500</td>
<td>0.000</td>
</tr>
<tr>
<td>VOC</td>
<td>ND1 + ND2</td>
<td>Vocabulary</td>
<td>5.000</td>
<td>5.000</td>
<td>14.000</td>
<td>14.500</td>
<td>3.000</td>
</tr>
<tr>
<td>LTH</td>
<td>NT1 + NT2</td>
<td>Length</td>
<td>6.000</td>
<td>7.000</td>
<td>19.000</td>
<td>22.500</td>
<td>4.000</td>
</tr>
<tr>
<td>LVL</td>
<td>2/ND1 * ND2/NT2</td>
<td>Program Level</td>
<td>0.667</td>
<td>0.500</td>
<td>0.286</td>
<td>0.208</td>
<td>0.667</td>
</tr>
<tr>
<td>DIF</td>
<td>1 / LVL</td>
<td>Difficulty</td>
<td>1.500</td>
<td>2.000</td>
<td>3.500</td>
<td>4.813</td>
<td>1.500</td>
</tr>
<tr>
<td>INT</td>
<td>LVL * VOL</td>
<td>Intelligence Content</td>
<td>9.288</td>
<td>8.127</td>
<td>20.668</td>
<td>17.970</td>
<td>4.227</td>
</tr>
<tr>
<td>EFF</td>
<td>VOL / LVL</td>
<td>Effort</td>
<td>20.897</td>
<td>32.507</td>
<td>253.189</td>
<td>425.626</td>
<td>9.510</td>
</tr>
<tr>
<td>Metric</td>
<td>Formula</td>
<td>Meaning</td>
<td>Action 6 Measured Movement</td>
<td>Action 7 Timed Movement</td>
<td>Action 8 Combined Movement</td>
<td>Action 9 Move Until Sensor</td>
<td>Action 10 Rover</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>-----------------</td>
<td>----------------------------</td>
<td>-------------------------</td>
<td>-----------------------------</td>
<td>---------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>ND1</td>
<td></td>
<td>Distinct Operators</td>
<td>6.000</td>
<td>2.500</td>
<td>5.000</td>
<td>3.000</td>
<td>7.000</td>
</tr>
<tr>
<td>ND2</td>
<td></td>
<td>Distinct Operands</td>
<td>10.500</td>
<td>1.000</td>
<td>1.000</td>
<td>0.000</td>
<td>2.000</td>
</tr>
<tr>
<td>NT1</td>
<td></td>
<td>Total Operators</td>
<td>15.000</td>
<td>3.000</td>
<td>10.000</td>
<td>4.000</td>
<td>13.000</td>
</tr>
<tr>
<td>NT2</td>
<td></td>
<td>Total Operands</td>
<td>17.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.000</td>
<td>2.000</td>
</tr>
<tr>
<td>VOC</td>
<td>ND1 + ND2</td>
<td>Vocabulary</td>
<td>16.500</td>
<td>3.500</td>
<td>6.000</td>
<td>3.000</td>
<td>9.000</td>
</tr>
<tr>
<td>LTH</td>
<td>NT1 + NT2</td>
<td>Length</td>
<td>32.000</td>
<td>4.000</td>
<td>11.000</td>
<td>4.000</td>
<td>15.000</td>
</tr>
<tr>
<td>LVL</td>
<td>2/ND1 * ND2/NT2</td>
<td>Program Level</td>
<td>0.218</td>
<td>0.833</td>
<td>0.400</td>
<td>0.667</td>
<td>0.286</td>
</tr>
<tr>
<td>DIF</td>
<td>1 / LVL</td>
<td>Difficulty</td>
<td>4.857</td>
<td>1.250</td>
<td>2.500</td>
<td>1.500</td>
<td>3.500</td>
</tr>
<tr>
<td>INT</td>
<td>LVL * VOL</td>
<td>Intelligence Content</td>
<td>25.853</td>
<td>5.711</td>
<td>11.374</td>
<td>4.227</td>
<td>13.585</td>
</tr>
<tr>
<td>EFF</td>
<td>VOL / LVL</td>
<td>Effort</td>
<td>703.231</td>
<td>9.877</td>
<td>71.086</td>
<td>9.510</td>
<td>166.421</td>
</tr>
</tbody>
</table>
7.3 Test Code

7.3.1 Action1_Sound
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

//**!!Code automatically generated by 'ROBOTC' configuration wizard !!**//

task main()
{
    nVolume = 1; // Ensures sound is not muted
    PlaySound(soundLowBuzz);
    wait(); // Give program time to execute
}

// nVolume can be ignored, but can cause issues in testing if not set
// Can be set on the hardware

7.3.2 Action1_Text_Display_String
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

//**!!Code automatically generated by 'ROBOTC' configuration wizard !!**//

task main()
{
    nxtDisplayString(1, "Education");
    wait(); // Give program time to execute
}

// Alternative functions that produce the same result
//nxtDisplayTextLine(1, "Education");
//nxtDisplayBigTextLine(1, "Education");
//nxtDisplayCenteredTextLine(1, "Education");
//nxtDisplayCenteredBigTextLine(1, "Education");
# 7.3.3 Action1_Text_Scroll_Text

```c
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

// **!!Code automatically generated by 'ROBOTC' configuration wizard !!**/

task main()
{
    nxtScrollText("Education");
    wait(); // Give program time to execute
}

// Simplest form. See Test1_Text_DisplayString for alternatives
```

# 7.3.4 Action2_Light

```c
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

// **!!Code automatically generated by 'ROBOTC' configuration wizard !!**/

task main()
{
    // Light sensor covered
    untilDark();
    nxtScrollText("Dark");
    wait();
}

// Works well with default value (30) under sufficient lighting
### 7.3.5 Action2_Sonar

```c
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

// Hand close to sensor untilSonarLessThan();
nxtScrollText("Close");
wait();
```

// Default distance of 30 works fine
// untilSonarGreaterThan is not well suited to this test
// due to its maximum limit causing only a specific distance
// tolerance to register. This is unintuitive

### 7.3.6 Action2_Sound

```c
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

// On clap untilSoundGreaterThan();
nxtScrollText("Loud");
wait();
```

// Works well at default sound value (50)
// untilSoundLessThan() does not work in this context
7.3.7 Action2_Touch
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

//!!Code automatically generated by 'ROBOTC' configuration wizard

task main()
{
    // When pressed
    untilTouch();
    nxtScrollText("Touched");
    wait();
}

// Alternatives
// untilBump();
// untilRelease() not suitable as it fires without interaction

7.3.8 Action3_LightSonar
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

//!!Code automatically generated by 'ROBOTC' configuration wizard

task main()
{
    // Keep looping else program exits
    while(true)
    {
        // Display text based on input
        if(SensorValue[light] < 20)
        {
            nxtScrollText("Dark");
        }
        else if(SensorValue[sonar] < 30)
        {
            nxtScrollText("Close");
        }
    }
}

// Bool logic on until* functions not possible (void return functions)
// No Natural Language sensor functions to help
7.3.9 Action3_TouchSound
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

//!!Code automatically generated by 'ROBOTC' configuration wizard !!*//

task main()
{
    // Keep looping else program exits
    while(true)
    {
        // Display text based on input
        if(SensorValue[touch] > 0)
        {
            nxtScrollText("Touched");
        }
        else if(SensorValue[sound] > 30)
        {
            nxtScrollText("Noise");
        }
    }
}

// Bool logic on unil* functions not possible (void return functions)
// No Natural Language sensor functions to help

7.3.10 Action4_Light_SingleInput
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

//!!Code automatically generated by 'ROBOTC' configuration wizard !!*//

task main()
{
    int counter = 0;

    while(true)
    {
        // Ensure it's light before it goes dark
        // Increment counter
        untilLight(40);
        untilDark();
        counter = counter + 1;
        nxtDisplayString(1, "%d", counter);
    }
}
// Alternatives
// counter++;  
// counter += 1;

// Displaying a numeric variable using %d adds a layer of complexity to this test

// untilLight value must be sufficiently higher than dark (default 30 for both)  
// otherwise counting is very imprecise  
// Room must be sufficiently lit.

7.3.11 Action4_Sonar_SingleInput

Pragma config(Sensor, S1, touch, sensorTouch)
Pragma config(Sensor, S2, sound, sensorSoundDB)
Pragma config(Sensor, S3, light, sensorLightInactive)
Pragma config(Sensor, S4, sonar, sensorSONAR)
Pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
Pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

//!!Code automatically generated by 'ROBOTC' configuration wizard!!*//

Task main()
{
    Int counter = 0;

    While(true)
    {
        // Ensure something approached the sensor
        // Increment counter
        untilSonarGreaterThan(35);
        untilSonarLessThan();
        counter = counter + 1;
        nxtDisplayString(1, "%d", counter);
    }

    // Alternatives
    // counter++;  
    // counter += 1;

    // Displaying a numeric variable using %d adds a layer of complexity to this test

    // untilSonarGreaterThan needs to be above the default values (both 30)
    // otherwise counter goes up erratically.
    // This is due to the use of < and > in place of <= and >= in the
    // Natural Language code

}
7.3.12 Action4_Sound_SingleInput
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)
/*!!!Code automatically generated by 'ROBOTC' configuration wizard !!*/

task main()
{
    int counter = 0;

    while(true)
    {
        // Ensure it's quite, and a sound has been made
        // Increment counter
        untilSoundLessThan();
        untilSoundGreaterThan();
        counter = counter + 1;
        nxtDisplayString(1, "%d", counter);
    }
}

// Alternatives
// counter++;
// counter += 1;

// Displaying a numeric variable using %d adds a layer of complexity to this test

7.3.13 Action4_Touch_SingleInput
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)
/*!!!Code automatically generated by 'ROBOTC' configuration wizard !!*/

task main()
{
    int counter = 0;

    while(true)
    {
        untilBump();
        counter = counter + 1;
        nxtDisplayString(1, "%d", counter);
    }
}
// Alternatives
// counter++;  
// counter += 1;

// Displaying a numeric variable using %d adds a layer of complexity to this test

7.3.14 Action5_Move_Backwards
#pragma config(Sensor, S1,  touch, sensorTouch)
#pragma config(Sensor, S2,  sound, sensorSoundDB)
#pragma config(Sensor, S3,  light, sensorLightInactive)
#pragma config(Sensor, S4,  sonar, sensorSONAR)
#pragma config(Motor,  motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorC, leftWheel, tmotorNXT, PIDControl, encoder)
// ***Code automatically generated by 'ROBOTC' configuration wizard ***

task main()
{
    backward();
    wait();
}

// Very simple

7.3.15 Action5_Move_Forward
#pragma config(Sensor, S1,  touch, sensorTouch)
#pragma config(Sensor, S2,  sound, sensorSoundDB)
#pragma config(Sensor, S3,  light, sensorLightInactive)
#pragma config(Sensor, S4,  sonar, sensorSONAR)
#pragma config(Motor,  motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorC, leftWheel, tmotorNXT, PIDControl, encoder)
// ***Code automatically generated by 'ROBOTC' configuration wizard ***

task main()
{
    forward();
    wait();
}

// Very simple
7.3.16 Action5_Turn_Point

```c
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

//**!!Code automatically generated by 'ROBOTC' configuration wizard!!**//

task main()
{
    pointTurn();
    wait();
}

// Very simple
// Turns 360 degrees
```

7.3.17 Action5_Turn_Swing

```c
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

//**!!Code automatically generated by 'ROBOTC' configuration wizard!!**//

task main()
{
    swingTurn();
    wait();
}

// Very simple
7.3.18 Action6_Move_Distance_Backward

```c
#pragma config(Sensor, S1,     touch,          sensorTouch)
#pragma config(Sensor, S2,     sound,          sensorSoundDB)
#pragma config(Sensor, S3,     light,          sensorLightInactive)
#pragma config(Sensor, S4,     sonar,          sensorSONAR)
#pragma config(Motor,  motorB,          rightWheel,    tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorC,          leftWheel,     tmotorNXT, PIDControl, encoder)

// Code automatically generated by 'ROBOTC' configuration wizard
```

```c
#include <RobotCompliance.h>

const float wheelCircumference = 17.2;
const float targetDistance = 30;
float rotations = targetDistance / wheelCircumference;

// Sync motors to ensure they rotate at the same speed
nSyncedMotors = synchBC;
// Set encoder value and start motor
nMotorEncoder[rightWheel] = 0;
motor[rightWheel] = -30;

// Wait for correct encoder value
while(nMotorEncoder[rightWheel] > (-rotations * 360)){}

// Not well supported by Natural Language.
// Complicated RobotC code but much less than the implementation of
moveStraightForRotations

// Need to convert distance to rotations using wheel circumference
// Assumes circumference is known or has been measured
// First test to need any maths
```

7.3.19 Action6_Move_Distance_Forward

```c
#pragma config(Sensor, S1,     touch,          sensorTouch)
#pragma config(Sensor, S2,     sound,          sensorSoundDB)
#pragma config(Sensor, S3,     light,          sensorLightInactive)
#pragma config(Sensor, S4,     sonar,          sensorSONAR)
#pragma config(Motor,  motorB,          rightWheel,    tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorC,          leftWheel,     tmotorNXT, PIDControl, encoder)

// Code automatically generated by 'ROBOTC' configuration wizard
```

```c
#include <RobotCompliance.h>

const float wheelCircumference = 17.2;
const float targetDistance = 30;
float rotations = targetDistance / wheelCircumference;

moveStraightForRotations(rotations);
```
// Need to convert distance to rotations using wheel circumference
// Assumes circumference is known or has been measured
// First test to need any maths

7.3.20 Action7_Move_Time_Backward
#pragma config(Sensor, S1,     touch,          sensorTouch)
#pragma config(Sensor, S2,     sound,          sensorSoundDB)
#pragma config(Sensor, S3,     light,          sensorLightInactive)
#pragma config(Sensor, S4,     sonar,          sensorSONAR)
#pragma config(Motor,  motorB,          rightWheel,    tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorC,          leftWheel,     tmotorNXT, PIDControl, encoder)

/*/!!Code automatically generated by 'ROBOTC' configuration wizard !!*/

task main()
{
    backward();
    wait(10);
}

// Even simpler if we use default time of 1 sec
// Not as accurate as forward version which uses motor encoding

7.3.21 Action7_Move_Time_Foward
#pragma config(Sensor, S1,     touch,          sensorTouch)
#pragma config(Sensor, S2,     sound,          sensorSoundDB)
#pragma config(Sensor, S3,     light,          sensorLightInactive)
#pragma config(Sensor, S4,     sonar,          sensorSONAR)
#pragma config(Motor,  motorB,          rightWheel,    tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorC,          leftWheel,     tmotorNXT, PIDControl, encoder)

/*/!!Code automatically generated by 'ROBOTC' configuration wizard !!*/

task main()
{
    moveStraightForTime(10);
}

// Even simpler if we use default time of 1 sec
### 7.3.22 Action8_Pattern

```c
#pragma config(Sensor, S1,     touch,          sensorTouch)
#pragma config(Sensor, S2,     sound,          sensorSoundDB)
#pragma config(Sensor, S3,     light,          sensorLightInactive)
#pragma config(Sensor, S4,     sonar,          sensorSONAR)
#pragma config(Motor,  motorB,          rightWheel,    tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorC,          leftWheel,     tmotorNXT, PIDControl, encoder)
```

```c
// Code automatically generated by 'ROBOTC' configuration wizard
```n

```c
task main()
{
    moveStraightForTime(1);

    pointTurn();
    wait();

    backward();
    wait();
}
```

// Natural Language makes simple movement very easy

### 7.3.23 Action9_MoveUntilSensor

```c
#pragma config(Sensor, S1,     touch,          sensorTouch)
#pragma config(Sensor, S2,     sound,          sensorSoundDB)
#pragma config(Sensor, S3,     light,          sensorLightInactive)
#pragma config(Sensor, S4,     sonar,          sensorSONAR)
#pragma config(Motor,  motorB,          rightWheel,    tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorC,          leftWheel,     tmotorNXT, PIDControl, encoder)
```

```c
// Code automatically generated by 'ROBOTC' configuration wizard
```n

```c
task main()
{
    forward();
    untilSonarLessThan();
}
```

// Sound more complicated as default volume picks up the motors
7.3.24 Action10_Rover
#pragma config(Sensor, S1, touch, sensorTouch)
#pragma config(Sensor, S2, sound, sensorSoundDB)
#pragma config(Sensor, S3, light, sensorLightInactive)
#pragma config(Sensor, S4, sonar, sensorSONAR)
#pragma config(Motor, motorB, rightWheel, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftWheel, tmotorNXT, PIDControl, encoder)

//!!Code automatically generated by 'ROBOTC' configuration wizard!!*/

task main()
{
    while(true)
    {
        forward();
        untilTouch();

        backward();
        wait();

        pointTurn();
        wait(0.4);
    }
}

// Had to provide a non integer value for pointTurn as the two combined
turn the robot 360 degrees
// Might be different on other surfaces?

7.3.25 C_Hello
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    // Output
    printf("%s", "Hello");
    return 0;
}

7.3.26 C_Hello_NoPrompt
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    char line[156];

    // get input
    fgets(line, sizeof line, stdin);

    // Output
    printf("Hello %s", line);
    return 0;
}

7.3.27 C_Hello_Prompt
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
    char line[156];

    // Prompt and get
    printf("%s", "Name: ");
    fgets(line, sizeof line, stdin);

    // Output
    printf("Hello %s", line);
    return 0;
}

7.3.28 Snippet_Distance_Movement_Maths

    task main()
    {
        // Calculate rotations
        const float wheelCircumference = 17.2;
        const float targetDistance = 30;
        float rotations = targetDistance / wheelCircumference;
    }

7.3.29 Snippet_Move_Distance_Backward_Loop

    task main()
    {
        // Wait for correct encoder value
        while(nMotorEncoder[rightWheel] > (-rotations * 360)){}
    }

7.3.30 Snippet_Move_Distance_Backward_Setup

    task main()
    {
        // Sync motors to ensure they rotate at the same speed
        nSyncedMotors = synchBC;
        // Set encoder value and start motor
        nMotorEncoder[rightWheel] = 0;
        motor[rightWheel] = -30;
    }

7.3.31 Snippet_Move_Time_Foward_1Sec

    task main()
    {
        moveStraightForTime();
    }

7.3.32 Snippet_MoveForwardForRotations

    #pragma config(Sensor, S1,     touch,          sensorTouch)
    #pragma config(Sensor, S2,     sound,          sensorSoundDB)
    #pragma config(Sensor, S3,     light,          sensorLightInactive)
    #pragma config(Sensor, S4,     sonar,          sensorSONAR)
    #pragma config(Motor,  motorB,          rightWheel, tmotorNXT,
                  PIDControl,  encoder)
    #pragma config(Motor,  motorC,          leftWheel,   tmotorNXT,
                  PIDControl,  encoder)
task main()
{
    moveStraightForRotations(17.2, 30);
}

// Need to convert distance to rotations using wheel circumference
// Assumes circumference is known or has been measured
// First test to need any maths

7.3.33 Snippet_MoveStraightForRotations

```cpp
// task main()
{
    nMotorEncoder[rightEncoderPort] = 0;
    nMotorEncoder[leftEncoderPort] = 0;
    while(nMotorEncoder[rightEncoderPort] < (abs(rotations) * 360))
    {
        if(nMotorEncoder[rightEncoderPort] ==
           nMotorEncoder[leftEncoderPort])
        {
            // Move Forward
            // Right Motor is run at power level 75
            // Left Motor is run at power level 75
            motor[rightEncoderPort] = 75;
            motor[leftEncoderPort] = 75;
        }
        else if(nMotorEncoder[rightEncoderPort] >
           nMotorEncoder[leftEncoderPort])
        {
            // Turn slightly right
            // Right Motor is run at power level 50
            // Left Motor is run at power level 75
            motor[rightEncoderPort] = 50;
            motor[leftEncoderPort] = 75;
        }
        else // if leftEncoder has counted more encoder counts
        {
            // Turn slightly left
            // Right Motor is run at power level 75
            // Left Motor is run at power level 50
            motor[rightEncoderPort] = 75;
            motor[leftEncoderPort] = 50;
        }
    }
    wait1Msec(1);
}
```

7.3.34 Snippet_Output.Counter

```cpp
// task main()
{
    nxtDisplayString(1, "%d", counter);
}
```

7.3.35 Snippet_RobotC_Branching
{  
    if(SensorValue[touch] > 0)
    else if(SensorValue[sound] > 30)
}

7.3.36 Snippet_SingleInput_response
task main()
{
    while(true)
    {
        untilSonarGreaterThan(35);
        untilSonarLessThan();
    }
}

7.3.37 Snippet_Sound_NoVolume
task main()
{
    PlaySound(soundLowBuzz);
    wait(); // Give program time to execute
}

7.3.38 Snippet_Variable_Assignment
task main()
{
    nVolume = 1;
}

7.3.39 Snippet_Variable_Tracking
task main()
{
    int counter = 0;
    counter = counter + 1;
}

7.3.40 Snippet_Variable_Tracking_Output
task main()
{
    int counter = 0;
    counter = counter + 1;
    nxtDisplayString(1, "%d", counter);
}
7.4 Halstead Parser Source Code

7.4.1 Halstead Parser

```csharp
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace HalsteadMetric
{
    class HalsteadParser
    {
        const string EXT_CSV = ".csv";

        public HalsteadParser()
        {
            FileName = String.Empty;

            MathOpTors = new List<string>();
            ControlOpTors = new List<string>();
            FunctionOpTors = new List<string>();

            StringOpRands = new List<string>();
            NumericalOpRands = new List<float>();
        }

        public string FileName { get; set; }
        public int TotalOpTors { get; set; }
        public int DistinctOpTors { get; set; }
        public List<string> MathOpTors { get; set; }
        public List<string> FunctionOpTors { get; set; }
        public List<string> ControlOpTors { get; set; }

        public int TotalOpRands { get; set; }
        public int DistinctOpRands { get; set; }
        public List<float> NumericalOpRands { get; set; }
        public List<string> StringOpRands { get; set; }

        /// <summary>
        /// Read code elements into memory
        /// </summary>
        /// <param name="directory">Directory containing the file</param>
        /// <param name="name">Name of the file w/out extension</param>
        public void ParseFile(string directory, string name)
        {
            // Remove extension from name
            FileName = name.Split('.')[0];

            try
            {
                RemainingApplicationCode = File.ReadAllText(directory + name);
            }
            catch (IOException e)
            {
                Console.WriteLine("Error opening file: ", e.Message);
                return;
            }
        }
    }
}
```
// Extract Main function
int first, last;
first = RemainingApplicationCode.IndexOf('{');
last = RemainingApplicationCode.LastIndexOf('}');
RemainingApplicationCode = RemainingApplicationCode.Substring(first, last - first);

// Remove comments, braces, quotes and whitespace characters
RemainingApplicationCode = Regex.Replace(RemainingApplicationCode, @"(//.*)", String.Empty);
RemainingApplicationCode = Regex.Replace(RemainingApplicationCode, @"([{}])", String.Empty);
RemainingApplicationCode = Regex.Replace(RemainingApplicationCode, "\(|\)", String.Empty);
RemainingApplicationCode = Regex.Replace(RemainingApplicationCode, @"\[\r\n\t\]", " ");

// Parse code
ExtractOpTors();
ExtractOpRands();

/// <summary>
/// Output file as .CSV
/// Name generated automatically
/// </summary>
/// <param name="directory">Output directory</param>
public void CreateCSV(string directory)
{
    // Output
    // Format Operators inside " "
    StringBuilder opTorOutput = new StringBuilder("\"");
    // Mathematical
    MathOpTors.Sort();
    foreach (string s in MathOpTors)
    {
        opTorOutput.AppendFormat("{0}, ", s);
    }
    // Controls
    ControlOpTors.Sort();
    foreach (string s in ControlOpTors)
    {
        opTorOutput.AppendFormat("{0}, ", s);
    }
    // Functions
    FunctionOpTors.Sort();
    foreach (string s in FunctionOpTors)
    {
        opTorOutput.AppendFormat("{0}, ", s);
    }
    opTorOutput.Append("\n");

    // Format Operands inside " "
    StringBuilder opRandOutput = new StringBuilder("\"");
    // Strings
    StringOpRands.Sort();
    foreach (string s in StringOpRands)
    {
        opRandOutput.AppendFormat("{0}, ", s);
    }
    // Numbers

NumericalOpRands.Sort();
foreach (float f in NumericalOpRands)
{
    opRandOutput.AppendFormat("(0), " , f);
}
opRandOutput.Append("\n");

try
{
    // Create directory and write to file
    string path = directory + FileName + EXT_CSV;
    Directory.CreateDirectory(directory);

    using (StreamWriter file = new StreamWriter(path))
    {
        // TestName
        file.WriteLine(FileName);
        // Tors \n
        // BlanksLine
        file.WriteLine(opTorOutput);
        file.WriteLine(opRandOutput);
        file.WriteLine();
        // # Distinct Tors
        // # Distinct Ands
        file.WriteLine(DistinctOpTors);
        file.WriteLine(DistinctOpRands);
        // # Total Tors
        // # Total Ands
        file.WriteLine(TotalOpTors);
        file.WriteLine(TotalOpRands);
    }
    catch (IOException e)
    {
        Console.WriteLine("Error saving to disk: {0}", e.Message);
    }
}

#region Private
private string RemainingApplicationCode { get; set; }

/// <summary>
/// Extract Operators from RemainingApplicationCode
/// </summary>
private void ExtractOpTors()
{
    // Extract Mathematical operands
    string mathematicalOperandsExpression = @"(<=|>=|<|>|==|=|+|--|+|-=|/=|;|::|[]|[]|\|\|&&|&|&|");
    MathOpTors = ExtractFromFileString(mathematicalOperandsExpression);
    // [ and ] should be compressed into a single entry
    for (int i = MathOpTors.Count - 1; i >= 0; i--)
    {
        if (MathOpTors[i] == "[")
        {
            MathOpTors[i] = "[]";
        }
        else if (MathOpTors[i] == "]")
        {
{ MathOpTors.RemoveAt(i);
}

// Extract functions
string functionExpression = @"\b\w+(?=\()";
FunctionOpTors = ExtractFromFileString(functionExpression, "");

// Extract control statements
// NOTE: depending on bracket placement these might already
// have been picked up as functions
// as both are operators this is not an issue
string controlsExpression = @"\b(if|else|while|for|do|switch|break|return|case)";
ControlOpTors = ExtractFromFileString(controlsExpression);

// Brackets are part of controls/functions so have been
accounted for
RemainingApplicationCode = Regex.Replace(RemainingApplicationCode, @"[\{\}\]", String.Empty);

// Calculate metrics
DistinctOpTors = 0;
DistinctOpTors += ControlOpTors.Distinct().Count();

/// <summary>
/// Extract Operands from RemainingApplicationCode
/// </summary>
private void ExtractOpRands()
{
    // Remove and parameter separators
    RemainingApplicationCode =
        Regex.Replace(RemainingApplicationCode, @"(,)", " ");

    // Split into strings and numbers so duplicates can be handled
    StringOpRands = RemainingApplicationCode.Split(new char[] { ' ', 
    }, StringSplitOptions.RemoveEmptyEntries).ToList<string>();
    for (int i = StringOpRands.Count - 1; i > 0; i--)
    {
        // If value is number, move to numericalOperators list
        float parseResult = 0.0f;
        if (float.TryParse(StringOpRands[i], out parseResult))
        {
            NumericalOpRands.Add(parseResult);
            StringOpRands.RemoveAt(i);
        }
    }

    // Calculate Metrics
    // Each numerical constant counts as distinct
    DistinctOpRands = 0;
    DistinctOpRands += StringOpRands.Distinct().Count();
}
private List<string> ExtractFromFileString(string pattern, string replacement = "")
{
    // Get matches then remove
    MatchCollection matches =
        Regex.Matches(RemainingApplicationCode, pattern);
    RemainingApplicationCode =
        Regex.Replace(RemainingApplicationCode, pattern, replacement);

    // Convert to string values
    List<string> extractedValues = new List<string>(matches.Count);
    foreach (var match in matches)
    {
        extractedValues.Add(match.ToString());
    }

    return extractedValues;
}
7.4.2 Program Source

```csharp
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace HalsteadMetric
{
    class Program
    {
        static void Main(string[] args)
        {
            const string DIR_CODE = @"Tests\"
            string[] files = Directory.GetFiles(DIR_CODE);
            HalsteadParser parser;
            foreach (string f in files)
            {
                parser = new HalsteadParser();
                parser.ParseFile(DIR_CODE, Path.GetFileName(f));
                parser.CreateCSV(@"ResultsCSV\"");
            }
        }
    }
}
```