Easy to Use Graphical User Interface for Robot Programming

Michael Appelstål
Jonatan Michalak
Melker Österberg
Abstract

Easy to Use Graphical User Interface for Robot Programming

Michael Appelstål
Jonatan Michalak
Melker Österberg

The field of robotics is growing rapidly due to the demand of robotic labor, but it is still relatively complicated for someone unexperienced to program a robot. In this project we have developed a web-based graphical user interface and scripting language for programming robot arms. The goal was to make the user interface so easy to use that people with no previous experience can use it to program the robots. The finished product is a web-based drag and drop application which generates robot control code in FeatherScript, a scripting language built by us for this project. The graphical user interface satisfies all technical requirements and from a survey on user experience we can conclude that the GUI is easy to use.
Sammanfattning

# Contents

1 Introduction 1  

2 Background 1  
   2.1 Robot Usage Today 2  
   2.2 Robot Programming 2  
   2.3 Visual Programming 3  
   2.4 Language Recognition 3  
   2.5 External Stakeholder 4  

3 Purpose, Aims and Motivation 5  
   3.1 Purpose 5  
   3.2 Aims 5  
   3.3 Motivation 6  
   3.4 Delimitations 6  

4 Related Work 6  
   4.1 Programming Tools of Robot Manufacturers 7  
      4.1.1 Universal Robots 7  
      4.1.2 ABB 7  
   4.2 Visual Programming Languages 7  
      4.2.1 Scratch 8  
      4.2.2 Lego Mindstorms 8  

5 Method 8  
   5.1 Method for Designing the Graphical User Interface 9
13 Future Work

13.1 Managing Routines ........................................ 24
13.2 Variables .................................................. 24
13.3 Undo actions ................................................ 25
13.4 FeatherScript .............................................. 25
1 Introduction

After the industrial revolution with its beginning in the late 1700s, the industrial robot became common since it could replace human workers and more efficiently produce goods [8]. Approximately 100 years later Joseph Marie Jacquard created what is considered to be the first programmable robot, using a punched-card control mechanism [8]. Today robots are used in all kinds of areas assisting or even replacing humans in numerous tasks [29], and the demand for robots is increasing [16]. For instance, robots can perform tasks faster and more precisely than a human, including hazardous, heavy or in other ways risky work [36]. For these and other reasons industrial robots can save companies a lot of money [36].

Featherway Robotics develops industrial robots and wants to make it easier and cheaper for companies to integrate robots into their production line. According to Niklas Salomonsson, the company’s current CEO, one of the main problems today is that robot programming requires a lot of previous experience and/or education in both robots and programming [37]. Therefore, Featherway Robotics wants an application that allows their customers or employees to program their robots.

To achieve this, we have created a web-based drag-and-drop-application, which allows the user to click and drag around objects on a screen to create programs for the robot. For the application to control the robot we have also created a scripting language called FeatherScript, which communicates with the robot and provides modularity to the entire system. This report will discuss everything from the choice of methods to the details and structure of the application’s different components, to evaluation of the end product, and possible improvements to the end product.

2 Background

The field of robotics is growing rapidly, and the use of robots is becoming more and more common in both workspaces and at home [29]. According to the International Federation of Robotics the growth in the use of industrial robotics increased by 16% in 2016 which was a new peak for the fourth year in a row [16].
2 Background

2.1 Robot Usage Today

Robots are heavily used in factories and industries doing tasks that are unsafe, hazardous, repetitive or unpleasant for humans to perform, like spraying, painting, welding, assembling and more [29]. For example, in Volvo’s factories robots are used to install front windows, side windows and roof hatches in their trucks, these are task that would be both heavy and potentially dangerous for a human worker [42].

Not only are industrial robots a good substitution for humans in for example dangerous and heavy tasks. There are also multiple other benefits in automating work places. Automating industrial processes results in a decrease in both cost and time of production, increase in reliability and accuracy which would lead to less errors than if a human would perform the same task [2]. Moreover, a robot can work indefinitely without any rest, which of course humans need [2].

2.2 Robot Programming

According to Alex Owen-Hill, the first robot programming language was MHI, which was developed in 1960 to control a robotic hand at MIT [7]. However, this language was not general purpose, and it would take another 10 years until these kinds of robot programming languages would start appearing, in an explosion of various robot programming languages [7].

Today, to program robots to perform mentioned tasks robotics companies provide various kinds of programming interfaces to their customers. ABB was the world’s largest robotics company in 2016 in terms of revenue [39]. They provide an offline graphical user interface called RobotStudio [3], which you can read more about in Section 4.1.2. They also provide a scripting language RAPID that can be used to program the robot directly.

According to Jason Ernst one reason why robot programming is difficult today is because there still is a major lack of documentation and programming guides available [11]. Another argument is that robots are not ”plug-and-play”, since there is not yet a standard way of connecting motors, sensors and other components [11]. This requires you to sometimes create your own software and/or hardware solutions for connecting components [11].


2.3 Visual Programming

A visual programming language is a way of creating programs and code by manipulating graphical objects. Examples of visual programming languages are Scratch and Lego Mindstorms, which you can read more about in Section 4.2.1 and Section 4.2.2, respectively. These languages are often aimed towards novice programmers or other areas where simplicity and user friendliness are essential, and they have the purpose to simplify programming by adding a layer of abstraction that removes the need to learn syntactics and semantics of the language [34]. Visual programming languages are therefore often easier to learn for unexperienced users, compared to purely text-based programming languages like C or Java.

According to Matthew Revell, the first visual programming tool was created by Ivan Sutherland as part of his PhD thesis in the mid-1960s, and was called Sketchpad [35]. In the system, the user could create two-dimensional graphics like lines and circles using a ”lightpen”, as well as applying operations such as copy to them [9]. In 1975 another major step for visual programming was taken by David Canfield Smith in his PhD ”Pygmalion: A Creative Programming Environment” [9]. For instance, Smith introduced the concept of an icon-based programming paradigm, where each icon represents a certain computation, which is still used by modern visual programming languages [9].

Today, visual programming is used in many different areas, and for various purposes. Scratch uses it to help and teach especially young people, the basics of programming. LabVIEW is a tool for engineers whose work include tests, measurements and/or control [27]. Node-RED is a tool for wiring together hardware [28]. All these three applications use visual programming as their main user interface. As we have seen, visual programming can be used by a broad audience, as engineers, developers and children are some of the target audience.

2.4 Language Recognition

A programming language can be compiled or interpreted. The difference between an interpreted language and compiled language is that an interpreter will execute one line at a time while a compiler will scan the whole program and creating byte code or machine code before it can be executed, making it possible to optimize the program. The benefits of using an interpreted language is that it is easier to implement, and it skips the compiling step, decreasing the time it takes before the program can be executed. On the other hand, the benefit of using a compiled language is that the execution time can be faster. We have created a interpreted language. A common way to build a language is with a process with three steps: lexical analyzing (lexing), parsing and interpreting.
2 Background

Figure 1 An example of an abstract syntax tree (AST) with the input \(1+1\)

The process of lexical analyzing is to turn characters into tokens. Tokens are representing the text that is being input but in a way that is easier to handle than raw characters. For example, writing a 2 can be converted to an `INTEGER` token with the value 2.

A parser will use the tokens generated by a lexical analyzer. Using grammar rules given by the creator of the parser, it will check if the tokens follows the grammar rules. For example, \(2+\) might not be allowed while \(2+2\) might be. The parser will also create an abstract syntax tree (AST) that is used to represent the structure of the program.

An example of how a AST can look like for the input \(1+1\) is shown in Figure 1. Each node in the tree represents an action or a constant.

The interpreter is the last stage in the process of language recognition. The interpreter will use traverse the AST that is generated by the parser and execute the corresponding action. For example, in Figure 1 it will add the two constants and return the number 2. This step is necessary when flow control is introduced to the language. The interpreter will then choose the correct path down the tree depending on the input.

As an alternative to the interpreter, you can use a compiler. The compiler will, instead of walking the AST and executing each node, create byte code or machine code. Then to execute the program, you will have to run that code.

2.5 External Stakeholder

Our external stakeholder is Featherway Robotics AB. Featherway Robotics is a small robotics company started in 2014 by former Uppsala University student Niklas Salomonsson, the company’s current CEO. The company develops lightweight robots aimed to be used in factories and other workplaces. Their aim according to their LinkedIn profile [12] is to introduce a smart, simple and safe “plug and play robot arm”. At the moment Featherway Robotics does not have any complete products available on the market, as their products are still in the development stage.
The company can make their arms significantly lighter compared to competitors’ robot arms by building them in carbon fiber instead of steel or aluminum, which are the typical materials for building robots [31]. According to Niklas Salomonsson the problem with using carbon fiber for robot arms is that its low stiffness makes the arms less accurate [37]. However, by using the company’s patented light sensor technology control system the robot can compensate for any bending in the arms, significantly increasing the robot arm’s accuracy [37]. However, this is not anything a programmer has to think about when programming the robot. Since one of the company’s goals with their product is that it should be simple, they need a way for their customer to easily program the robot.

3 Purpose, Aims and Motivation

3.1 Purpose

The purpose of this project is to simplify robot programming. Our vision is that employees of companies that buy robots from Featherway Robotics using Featherway GUI should be able to program and control robots.

3.2 Aims

The goal with this project is to make robot controlling easier by developing an easy to use graphical user interface (GUI) to the robots. By easy to use we mean that the GUI should require very basic previous experience working with robots or programming from the user, and that they should be able to quickly find and use all the GUI’s functionalities. In addition to the GUI our project includes developing a scripting language for Featherway Robotics. This scripting language is called FeatherScript and is used for programming the robot. Our GUI will generate robot control scripts in this language. FeatherScript also adds a layer of abstraction and makes the system more modular. This will be very helpful in the future if Featherway Robotics wants to create a new GUI or build on their existing ones. Because the system is modular they can easily replace one part of the system without having to modify its other parts, making it significantly easier to implement changes and new features to the system. The benefits of creating our own scripting language over an existing language is that we can adapt it to fit our stakeholder’s needs. We can add features that some languages do not have and choose not to implement the features that are not needed.
3.3 Motivation

According to Niklas Salomonsson, current CEO at Featherway Robotics, robots today are relatively difficult to program for a person without previous experience since it requires knowledge and experience in both programming and robotics, and this creates barriers for the creation of working robot programs [37].

Since our GUI will simplify the process of programming the robots, customers that buy these robots will no longer need to also hire staff only for this task. This will make it easier and cheaper primarily for smaller companies to include robots in their business in the future.

3.4 Delimitations

Since the GUI should be available online, security becomes an important aspect. For example, an unauthorized user should for obvious reasons not be able to connect and control a company’s robot. A feature that Featherway Robotics has suggested is that users should be able to securely connect to the Featherway GUI using a web browser from any location, or at least from their internal local area network. A login screen would require login information, including a password, and thereby prohibit unauthorized users. However, this project will not take this into account because of its time limitations.

Featherway Robotics is still developing the robot that is going to use our system. Therefore, the interface between our generated script and the actual robot does not exist yet. So, one delimitation on our project is that we will not be able to test our product on an actual robot.

4 Related Work

In this section we will be examining and discussing work that is related to our project. We will discuss differences and similarities between our project and the related works and why we choose to do things differently or similar. This section will explore different robotics companies to see how other companies has implemented a programming environment for their robots and compare it to our implementation. We are also looking at visual programming languages to compare to our graphical interface.
4 Related Work

4.1 Programming Tools of Robot Manufacturers

The following two companies are robot manufacturers which have developed their own tools for programming their robots. Both companies are manufacturing industrial robots, like Featherway Robotics. We choose to look at ABB and Universal Robots. ABB because they are the biggest manufacturer of industrial robots [39]. We also wanted to investigate a smaller robot developing company that develops robots similar to Featherway and choose to take a closer look at Universal Robots and their systems.

4.1.1 Universal Robots

Universal Robots develops industrial robot arms. Most of their robots are programmable in two ways: either through their GUI called PolyScope [43] or using their scripting language URScript [40]. The URScript is similar to the script we are developing in many ways. However, their GUI differs from ours. One significant difference is that our GUI unlike PolyScope is web-based. There are also big differences in the layout of the two GUIs. Our GUI provides drag and drop functionality and aims to be as simplistic as possible while PolyScope is more static and menu based.

4.1.2 ABB

ABB is one of the biggest producers of industrial robots [39]. Their GUI is called RobotStudio [1] and is, according to their website “The world’s most used offline programming tool for robotics” [3].

Just as with Universal Robots’ PolyScope, the major difference between the Featherway GUI, PolyScope and RobotStudio is that our system is web-based. This opens up to many new possibilities. First of all, a web-based application is available on all platforms, including Linux, MacOS and Windows. Moreover, it could allow the user to program the robots from anywhere over the Internet.

4.2 Visual Programming Languages

The following sections cover two visual programming languages which are focused on simplicity.
4.2.1 Scratch

Scratch is a programming tool developed by MIT Media Lab, used to make games [38]. It uses a simple drag and drop system to create programs and is available for free in both a web-version, and desktop version [38]. Scratch is aimed towards children with no previous programming experience, and therefore has a very childish touch to its GUI and does not give a very industrial or professional impression to the user. Scratch differs from our system by instead of controlling a robot arm it is used to draw and move objects on the screen, therefore it has a lot of extra functionality that our application does not need. We could have made an extension of Scratch, but Scratch is aimed towards children and the GUI we develop is aimed towards industry workers. Because of this reason, and because of its redundant functionalities we decided against developing an extension for Scratch and instead developed our GUI from scratch.

Even though our GUI and Scratch is used to program very different things the end goal of both systems is to simplify programming using a drag and drop interface. Because of this we were inspired by Scratch when making design choices in our GUI. The similarity between our GUI and Scratch is how we visualize if statements and loops.

4.2.2 Lego Mindstorms

LEGO Mindstorms is a product line from LEGO that lets the user to build and program robots using LEGO blocks and a programmable microcomputer [18]. The robot can be programmed using LEGO’s application ”EV3 Programmer” with a ”simple drag-and-drop-interface” [19]. Ev3 Programmer is however not web-based and is aimed at, as the name suggest, programming LEGO toys rather than industrial robots. Lego EV3 is not open source so it is not something that we can use to create our GUI [20]. Even though we could not use EV3 to create the GUI we still took some inspiration on how to create an intuitive GUI. The similarity between EV3 and our GUI is that we both create a horizontal time line for the planned robot actions.

5 Method

In this section we will describe and motivate our choice of tools, by comparing the different alternatives. We will go through some viable options for designing the GUI and scripting language FeatherScript, and then evaluate what fits our aims the best.
5 Method

5.1 Method for Designing the Graphical User Interface

Featherway Robotics wants the GUI to work on a variety of systems. Therefore, we chose to make it web-based, since that would make it work on any system that has access to a web browser. Building the GUI using regular HTML elements together with CSS, which is the classical approach, does not give us as much flexibility when designing the functionality of the GUI. Since that approach is aimed at creating text websites [25], and our GUI is not a typical website but will have more complicated interactive features, building it using the classical HTML approach is therefore not an optimal approach. By classical HTML approach we refer to building a tree structure by nesting various HTML tags like <div>, <a>, <img>, etc. and styling these with CSS.

Some of the popular tools for rendering interactive graphic in web browsers are HTML5 canvas [23], Adobe Flash Player [5] and WebGL [26]. These allow the programmer to draw shapes, images and text on the web page, as well as to use user input like mouse position and keyboard. Adobe Flash Player applications are programmed using the language ActionScript [4]. WebGL and HTML5 canvases are both controlled using JavaScript [26, 22]. The approach of building a web page using HTML5 canvas should not be confused with building it using a tree structure of regular HTML elements, as mentioned earlier. An HTML5 canvas is an HTML element itself, <canvas>, but on which every graphical component is drawn. It doesn’t have any tree structure but is simply a (at first) blank canvas which can be drawn on using JavaScript. JavaScript is a common language for writing scripts for web pages [24]. We will not be using Adobe Flash Player because Adobe themselves encourage developers to migrate from Flash since it will not be updated or distributed by 2020 [6]. By comparing the web browser compatibilities of HTML5 canvas (seen in Figure 2) with those of WebGL (seen in Figure 3 and Figure 4) we see that HTML5 canvas is supported by a wider range of browser versions than WebGL. Therefore, we will use HTML5 canvas.

5.2 Method for Designing FeatherScript

In general, a scripting language interpreter takes an input and executes actions based on the input. To interpret the input, one needs to find keywords in the input and use them to execute the correct actions. This is done by using a lexical analyzer and a parser, which is explained in Section 6.2.

We could either write our own lexical analyzer and parser or use a lexical analyzer and parser generator. We chose to use generators because of time constraints and because the generator will most likely have more readable and maintainable code than a hand-written parser or lexical analyzer. There are many different generators, and these they
## Method

**Figure 2** Browser compatibility table for HTML5 canvas on both desktop and mobile devices [23]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Desktop</th>
<th>Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic support</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>height</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>opacity</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>width</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Compatible: 
- Chrome 1.5
- Firefox 9
- IE 2

Compatibility unknown: 
- Safari

Non-standard: Expect poor cross-browser support.

*See implementation notes.*

**Figure 3** Browser compatibility table for WebGL on desktop devices [26]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Chrome</th>
<th>Edge</th>
<th>Firefox (Gecko)</th>
<th>Internet Explorer</th>
<th>Opera</th>
<th>Safari</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic support</td>
<td>9</td>
<td>(Yes)</td>
<td>4.0 (2.0)</td>
<td>11</td>
<td>12</td>
<td>5.1</td>
</tr>
<tr>
<td>WebGL</td>
<td>56</td>
<td>No support</td>
<td>51 (51)</td>
<td>No support</td>
<td>43</td>
<td>No support</td>
</tr>
</tbody>
</table>

**Figure 4** Browser compatibility table for WebGL on mobile devices [26]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Chrome for Android</th>
<th>Edge</th>
<th>Firefox Mobile (Gecko)</th>
<th>IE Mobile</th>
<th>Opera Mobile</th>
<th>Safari Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic support</td>
<td>25</td>
<td>(Yes)</td>
<td>4</td>
<td>No support</td>
<td>12</td>
<td>8.1</td>
</tr>
</tbody>
</table>
are presented below.

## 5.2.1 Tools for Text Recognition

Lex [21] is a well-documented tool for generating lexical analyzers. However, today Flex is more commonly used, which is a free and open source implementation of Lex [41]. Lex and Flex are mostly interchangeable; code written in Lex will often work in Flex. The output code from Flex is in C or C++ [41].

Yacc [17] is a tool for generating parsers that works well with Lex. The original Yacc is not available anymore but there exist Yacc-compatible generators, for example GNU Bison [14] which is a parser generator that is free and open source. Just as with Lex and Flex, code written in Yacc will also work with Bison, and output code from Bison is in C or C++ [15].

The Lemon Parser Generator is a parser generator that is similar to Yacc and Bison but not compatible [33]. The advantages that The Lemon Parser Generator has over Bison are that the syntax is less prone to programming errors, is less prone to memory leaks and the parser generated is re-entrant and thread-safe [33].

Another Tool for Language Recognition (ANTLR) is another tool for language recognition. ANTLR is equipped with both a lexical analyzer generator and a parser generator. It outputs in various languages e.g. C#, JavaScript, Python, but not C. ANTLR also offers tools for a few IDEs (Integrated Development Environments), e.g. Eclipse or NetBeans, to make ANTLR easier to use [30].

## 5.2.2 Comparison and Choice

The system that will parse our scripting language FeatherScript will be running on a Linux distribution and the external stakeholder prefers that we use C. ANTLR does not offer support for C and will therefore not be an option, although it is a valid choice. We will be using Flex and Bison instead of Lemon Parser Generator mainly because of the amount of documentation and material available. Google Scholar currently grants 6 160 results when searching for "flex AROUND(2) bison”, while searching for ”lemon parser” only grants 1 890 results (The ”AROUND(2)” of the query ensures that the two words ”flex” and ”bison” appear in close proximity to each other in the result page). Moreover, a quick Google search for ”flex AROUND(2) bison” currently grants 3 660 000 results, while querying ”lemon parser” grants only 194 000 results. Since none of us have previously worked with any of these tools, the amount of documentation and material for them available is highly significant.
6 System Structure

The system structure is divided into two main parts, the GUI and the scripting language FeatherScript. The GUI will be web-based and accessible through any modern web browser. In the GUI the user creates robot routines for their robots. The GUI then generates FeatherScript code corresponding to the robot routines that is created. The FeatherScript code is then translated to code that the control system of the robot can read. This process is shown in Figure 5.

6.1 Graphical User Interface Structure

The Featherway GUI is built using two HTML5 canvases, together with JavaScript for its back end. An HTML5 canvas is basically a field on which we can draw any shape, text and image using JavaScript. The canvases will cover the entire screen of the web page and overlap. Most graphical elements will be drawn on the background canvas, and the foreground canvas will be used to draw quickly changing elements. For example, when the user clicks and drags a block across the screen, that object must be drawn, erased and redrawn very rapidly to give the user an illusion of movement. By drawing it on the foreground canvas, nothing in the background must be redrawn. Therefore, by using two overlapping canvases we reduce the number of drawn objects significantly.

The screen is divided into two primary areas; the menu area and the time-line (see Figure 6). The time-line is the chain of blocks representing the program (script) to be run on the robot. In the menu area all function and routine blocks are listed. By dragging these blocks into the time-line the user can create routines for the robot. These routines are then used to generate FeatherScript code that describes what actions the robot should perform.

Some blocks like `if` will create parallel time-lines, as seen in Figure 6. Like in most other programming languages, `if` has a condition attached to it and if that condition is
true, the code nested in the if is run. Otherwise the code nested in else is run. Since an if block can have an else block attached to it, the user should be able to tell the program what to do in both of those cases by dragging blocks in to each of the parallel time-lines.

### 6.2 FeatherScript Structure

Our scripting language FeatherScript uses *lexical analysis* which breaks the text into distinct pieces, such as `while`, `=`, and `end`, usually separated by whitespace or other delimiters. These pieces are called tokens, and they are then parsed into an abstract syntax tree (AST) which in turn is interpreted by an *interpreter*. An AST is a tree structure where the nodes represent operations or constants. An interpreter can then traverse the tree and execute the operations. The tree makes it simple to control the flow of the program. The lexical analyzer is created using Flex, which is a tool for generating lexical analyzers. To parse the tokens, we are using a tool for generating parsers called Bison. The simplified process is shown in Figure 7.

The GUI will generate FeatherScript code that will be read by a parser and send control signals through Featherway Robotics’ interface to their control system.
7 Requirements and Evaluation Methods

In this section we will cover the requirements we have on our system and in what ways we will test whether the system meets these. First, we bring up some general requirements on the GUI and the overall usability of the system. Then we go more into detail on technical requirements on both the GUI and FeatherScript. In the technical requirements we list the required functionality of the system.

7.1 General Requirements on the Graphical User Interface

The Featherway GUI should have a web-based adaptive drag and drop functionality. The user will see numerous icons in a menu, where every icon represents a functionality in the robot of some sort (for example to move the robot’s arm, wait, if-clause, loop, routine, etc.), which then can be dragged into a time-line, telling the robot what actions to take in which order. See Figure 6. Every component representing a functionality that can be dragged by the user into the time-line will be referred to as a block. With adaptive we mean that Featherway Robotics should be able to change the robot’s functionality library in a data file, for example by adding a new routine, and these changes should easily be accessible in the GUI. A routine is a sequence of instructions aimed at performing a specific action, often called multiple times by a program [10]. Functionality library refers to the set of routines available for the user to program the robot with, for instance functions like rotate, move and get position. We will later in this paper refer to functions like these, so keep in mind that these are only examples of routines that Featherway Robotics could choose to add to the GUI and their robot. The company themselves both define and interpret what these function calls should do to the robot. With web-based we mean that the GUI should be accessed through a web browser and be available over an internet connection.

The aim and purpose of this project is to simplify robot controlling and make it possible for people with no previous experience in the field to use the Featherway Robotics robots. Therefore, one important requirement on the graphical user interface (GUI) is
that it should be easy for the user to use and understand. To test this, we could analyze meta data from users when they use the GUI. A typical test would be conducted by giving out a task for the user to complete. Such tasks could for example be to create a specific sequence of functions and routines in the GUI. We can then study how long it takes for the user to create their robot routine and how many missteps the user took to finish the routine. This gives us useful information on the program’s learning curve and what steps are the toughest for the user to complete. However, this study would probably take a disproportional amount of time. Therefore, another way to test what users think of our program is to have test users use the program and fill in surveys with the following questions regarding their experience, and letting them answer on a scale from 1 to 5 where 1 is ”I disagree completely” and 5 is ”I agree completely”:

1. I have a lot of experience with robot programming.
2. The purpose and functionality of each part of the GUI was clear.
3. The GUI was overall easy to use.

The survey would also have the following questions, in which the user can write an answer in text:

1. Is there anything in the system you would change?
2. Is there anything in the system that you found particularly good/helpful?

### 7.2 Technical Requirements on the Graphical User Interface

As previously mentioned, Featherway Robotics’ interface from FeatherScript to their robot’s control system has not been implemented. Therefore, we are not going to be able to test the code on an actual robot. However, we can test more specific features on the GUI and FeatherScript.

- The user should be able to drag and drop basic blocks such as move, rotate and others into the time-line in any available position. A block is a graphical component representing a functionality that can be inserted into the GUI’s time-line. The block should snap into the right position depending on where the block was dropped. The user should also be able to drag and drop more complex blocks such as if and loop into the time-line. These blocks should have the ability to store alternate time-lines nested inside the block, e.g. multiple if statements nested in each other.
• The user should be able to remove any block, including nested blocks, from the time-line. If the removed block has other blocks nested inside it, those nested blocks should also be removed.

• The user should be able to set appropriate arguments on blocks that are dropped on the time-line. For example, if the `loop` function is dropped the user should be able to set for how many times the nested time-line should loop, or if a `rotate` function is dropped the user should be able to set the number of degrees the robot arm should rotate.

• The GUI should generate FeatherScript code with function call traces following the structure and order of the GUI’s time-line.

7.3 Technical Requirements on FeatherScript

It is important that the language FeatherScript is reliable, it should be working as intended at all times. An error in the language can cause a bug in a system that can be hard to find. The interpreter of FeatherScript is written in C it will be unit tested using CUnit, a unit testing framework for C [13]. Unit testing is a way of breaking down the program in smaller parts that you can test individually. The smaller parts will be tested in a mock environment that simulates the use of the smaller parts of the program. The smaller parts in our case will be every individual function in the interpreter. The lexical analyzer and parser are written in Flex and Bison and there is no unit testing framework for them. But to test the language as a whole we will write test scripts using all features implemented for FeatherScript, and check if the output after running the scripts outputs the correct calls that will be sent to the robot. The results of these test should show that there are no errors in FeatherScript all the unit tests should pass and the test scripts should show correct output. If all tests pass it will show that, to our knowledge, there is no errors in FeatherScript. We have no requirements on the performance of the interpreter because we are focusing on the stability and reliability.

8 Graphical User Interface Data Structure

Since we are going to use the GUI for generating our own script code it is important that we have a good structure for the data in our GUI. A good data structure in our case is one where its data can easily be traversed and modified.

JavaScript supports an object-oriented programming style [24]. According to B. P. Pokkunuri [32], the idea with this programming style is to divide the program into
classes with certain properties (variables) and methods (functions). An instance of a class is called an object. A class $B$ can inherit from another class $A$. An instance of class $B$ then has all methods and properties that class $A$ has.

Every component drawn on the canvas inherits from the `Obj` class (not to be confused with JavaScript’s own `Object` class). Every instance of the `Obj` class has basic properties like position on the canvas, size, etc. and methods that for example draw and move it.

Since blocks like `if` may cause multiple parallel time-lines they should have each time-line, with corresponding `Obj` blocks, nested inside them. Therefore, implementing `loop` and `if` statement blocks proved to be more complicated than we originally thought. The difficult part is that for these kinds of functions the system also needs to keep track of when the nested section of a function block like `loop` ends (i.e., where the last piece of code that is supposed to be looped ends) to generate an end block, as seen in Figure 9, and it also needs to keep track of any recursively nested function blocks they may contain, for example a `loop` inside a `loop` block. We chose to give these kinds of nestable function blocks the class `timelineSplit` which inherits from `Obj` that in addition to the inherited attributes also contains an `innerTimelines` attribute, which is a list containing all timelines. This attribute can, depending on what kind of function it is, contain one or multiple time-lines.

## 9 FeatherScript Usage

In this section we describe what operations and functionalities are supported by FeatherScript. This section also explains how to write in FeatherScript and how the syntax looks like.

### 9.1 Types and Operations

A valid mathematical expression is either a number, an operation between two numbers or an operation between two mathematical expressions. In FeatherScript the numbers can either be integers or floats (decimal numbers). The operations that exists are addition, subtraction and multiplication. These are some examples of valid mathematical expressions:

```
6
2+2.5
(1-2)*(3/4)
```
Figure 8 The GUI with clarification of its different sections
1+2/3.5*2.1+(0.1+0.2)

The order of priority is the same as in standard mathematics:

1. Parenthesis
2. Multiplication and Division (from left to right)
3. Addition and Subtraction (from left to right)

In Boolean expressions, 0 is treated as False and any other value is True. A valid Boolean expression is either a mathematical expression or a Boolean operator between two Boolean expressions. The Boolean operators and relational operators are shown in examples below.

0 && 1 || 2
(1<2) && !(1==1)
!((1<=2) || ((2+5)<=2))

Strings are written in quotation marks, "foo". There are no operations working on strings, but they can be used in functions such as print. Quotation marks are not allowed inside strings.

9.2 Variables

The equal sign is used to assign values to variables. If a value is assigned to an existing variable it will overwrite the previous value with the new. Any type of value can be assigned to a variable.

```plaintext
to = 1;
bar = 1.2;
baz = "string";
```

9.3 Flow Control

if-else statements require a Boolean expression for the condition. All if statements need an else statement.
if (1) {
    a = 1;
    b = 2;
} else {
    a = 2;
    b = 1;
}

if ((2<2) || (1<2)) a = 1;
else a = 2;

while loops also expect a numeric value or an expression as the condition.

while (a < 10) {
    a = a + 1;
}

After a while, if and else you can either write a single statement or if you want
multiple statements you can surround the statements with brackets {}.

9.4 Print

The functions that the robot can receive are not defined in FeatherScript. To use the
functions you will use the print function. For example, if the robot has a function
called move(x, y) you can write

print "move(1,2)"

to output the string move(1,2), which can be sent to the robot’s interface, which, as
previously mentioned in 3.4, currently does not yet exist.

10 Evaluation Results

To evaluate the finished system, we will look at its two parts; the GUI and the scripting
language FeatherScript.
10 Evaluation Results

10.1 Evaluation Results of the Graphical User Interface

We asked four classmates from the Master Programme in Computer and Information Engineering at Uppsala University tested the GUI and participated in the survey, since they do have programming experience but not necessarily with robots. The overall results of the survey were positive but there were two complaints, which we will mention in a moment. All users answered 1 to the question "I have a lot of experience with robot programming". The average score for the question "The purpose and functionality of each part of the GUI was clear" was 3.8 and "The GUI was over all easy to use" received an overall score of 4. Since all our users answered 1 to the question "I have a lot of experience with robot programming", their experience with using the GUI is significant since this is our target group for the GUI.

The question whether the Featherway GUI is easy to use for users with no previous robot programming experience is quite subjective in its nature. However, as we can see in Figure 8 the features of the GUI are relatively few, and it is therefore easy for a new user to get an overview of the GUI’s functionalities, which also suggests that the GUI is easy to use.

The system fulfills all technical requirements in Section 7.2. The user can create routines consisting of functions like move, rotate or any other function, by dragging and dropping their blocks into a time line. The GUI also supports adding arguments to blocks like loop and if, if statements and loops can be nested, and the GUI generates correct and working code corresponding to the entire routine. It is also possible to remove any block in the time line.

However, improvements to a system can always be made. Currently all function blocks look very similar at first glance, which was also pointed out by a test user. Therefore, one improvement to the GUI would be to implement the ability to give each function block an icon symbolizing its functionality, making each block more unique. For example, the Rotate block could have a turned arrow icon. Another small improvement to the GUI suggested by a test user was to give each section an explanatory header text. This might improve a user’s first interaction with the GUI since it makes the use of each section even more obvious.

10.2 Evaluation Results of FeatherScript

The unit tests for FeatherScript are written using CUnit. The tests pass with 98,25% coverage and 116 asserts. There are no memory leaks when interpreting a script. We are testing each function of the interpreter separately. The unit tests test the functions both
11 Results and Discussion

In normal cases and edge cases. For example, there is a function \texttt{assign\_variable} that assigns a value to a variable using two arguments, the name of the variable and the value to assign to that variable. A normal case would be to assign the value 2 to the variable \texttt{a}. What is expected to happen is that the value 2 is now stored in \texttt{a}. An edge case can be to try to assign a value to a variable with unusual characters like !#%& to see if it will crash the program or if there is any unintended behavior. An assert is a function in CUnit that if the given argument is true, the assert will pass. An example would be to see if the variable \texttt{a} contains the value 2. With 116 asserts we are covering all the cases that we consider necessary.

We have created a GUI with drag and drop functionality. It is possible to drag boxes to a designated area and drop them to build a time line. It supports \texttt{if-else} statements that will make the chain of blocks parallel. Once inside the designated area, a block can be moved, deleted and take multiple inputs, as seen in Figure 9, in the \texttt{Move} block which take an x-, y- and z-coordinate as input. From the chain of blocks, the application can generate corresponding FeatherScript code, as seen in Figure 10, that can be used to control a robot using the functionality library provided by Featherway Robotics, in this case the functions \texttt{rotate, move, getX} and \texttt{getY}. By studying the FeatherScript call trace and comparing it to the time-line of the GUI we can ensure that the GUI is generating FeatherScript code correctly. According to our test survey, test users agree that the system is easy to understand and use.
12 Conclusions

We have also built the scripting language FeatherScript. It can handle mathematical expressions with integers and decimal numbers, Boolean expressions, flow control and variables making FeatherScript a simple but sufficient language to communicate with Featherway Robotics’ robot control system. Since it passes all tests that indicates that the language is reliable, as required in Section 7.3.

The quality of the system is in line with what we expected but we did not have the time to implement all the features that we planned. In the Section 13 we go through some of these features. The application is easy to use; however, this may partly be because the functionality of the GUI is very limited. This is mostly because we didn’t have the time to implement all originally intended features. The application is therefore a good prototype but not a finished product.

One big motivation for this project is that the tools that other robotics companies offer are not easy to use for people with no previous experience and this in turn creates a barrier for creation of working robot programs, as discussed in Section 3.3. The GUI we developed was aimed to be as simplistic as possible and while it cannot perform as complex robot commands as the competitors, it is however simple and easy to understand.

12 Conclusions

We have built an application that allows users with little to no previous experience to program robots. The application consists of two major parts: a graphical user interface (GUI) and a scripting language called FeatherScript. While being very simple, the GUI does allow the user to program the robot through drag and drop, and generates FeatherScript code which, in the future, will control the robot. FeatherScript is a relatively
simple language, however it has several basic features of many languages. The sur-
vey testing the user experience suggests that the GUI is easy to use and understand for
people with little previous experience in robot programming. This application should
ideally make it easier and cheaper for companies that want to implement robots into their
current business, but do not have the resources to hire an experienced robot technician.

13 Future Work

There are various features that can be added to both the GUI and FeatherScript to im-
prove the overall system. In this section we list some features that we consider are worth
implementing.

13.1 Managing Routines

When the user has created a routine, they should be able to save the routine and then
load it in another session. With this implemented the user can build a library of routines
to use whenever needed. An extension to this is to add support to load a routine inside
a routine as a block. This will make the routines modular, easier to implement and
easier to make changes to. Just as in any programming language there exist libraries
that other users have created, it will also be possible to use routines other users have
created and share your own. This could also be beneficial for companies that are going
to use multiple robots for the same tasks, then they can create and modify the routine on
one robot and load it onto all other robots.

13.2 Variables

Even though FeatherScript has full support for variables, the GUI doesn’t yet. In the
future the user should be able to create and use variables in multiple blocks in the GUI.
If for example the robot should move to a certain position multiple times the user should
only have to drag and drop the same position variable with the x, y, z coordinates already
set. In our system right now, the user has to manually insert the values to all blocks.
These variables would make it easier for the user to change the behavior of the robot
since now it is only necessary to change the variable in one place instead of multiple. It
would also minimize mistakes that might occur when the values of multiple blocks have
to be changed.
The Featherway Robot will have several functions that receive a value or a signal. The robot could for example have a function `getPosition` that would return the position of the robot arm as coordinates \((x,y,z)\). If then these values need to be used or stored for later, the GUI would need to support variables.

### 13.3 Undo actions

Being able to undo actions in an application is a very useful feature. If the user accidentally removes a block or for some other reason regrets deleting one, they should be able to undo that action and bring that block back.

### 13.4 FeatherScript

FeatherScript can handle expressions, control flow statements and variables with real numbers and strings. `else if` statements do not exist, nor do `switch` statements. These statements are not possible:

```c
if (a) {
    ...
} else if (b) {
    ...
}
```

Code that will have the same functionality as the code above that works in FeatherScript will look like this:

```c
if (a) {
    ...
} else {
    if (b) {
        ...
    }
}
```

These statements are not essential because `if` and `else` statements can be used where there would be a `switch` statement or an `else if` statement, but the code can become more readable. `for` loops do not exist either. The functionality of a `for` loop can be achieved using `while` loop but, just like a `switch` statement, it can make the code more readable.
There are little to no helpful error messages when compiling a script with invalid syntax. It would be helpful for a developer implementing a new FeatherScript feature if there were more verbose error messages that show where and/or what the error is. Since this would help with debugging FeatherScript, it would ease the development process.

Arrays and custom data types would be a useful addition. They would for instance enable creating a position type that consists of three integers or three floats to represent a position in a three-dimensional space, or a trajectory type that consists of two positions.

References


References


