Abstract

Process optimization in machining is still in its infancy, with massive room for growth. In order to address the issue of tool selection within a fragmented industry, a software package is needed to streamline the process of filtering cutting tools from the large catalogues offered by tool manufacturers. An application was developed on the Android platform to offer a simplified interface that provides an optimal cutting tool recommendation for a desired machining task. The industry-standard Android Studio development environment was used to build, test and debug the application. The developed application was able to filter a database of tools, then use a Multiple-Criteria Decision Making (MCDM) algorithm to find the best tool for a specified machining application. The optimal cutting tool is selected based on obtaining favourable performance with regards to the power consumption, material removal rate, shear plane deformation, tool life and surface roughness.

Contents

List of figures

1. Introduction

1.1. Context

Within the manufacturing industry, process optimization is necessary to maximise the benefits and sustainability of Computer Numeric Control (CNC) machining.

With the amount of engineering materials that have are available increasing, the cutting tools that can machine those materials have correspondingly increased in variety. In modern machining environments, there is an extensive range of tooling available. On the Techspex machine tool search engine, over 7500 tools can be searched [1]. Choosing the right tool for the job becomes a difficult task. Traditionally, the selection of the optimal tool for a machine task was determined by the knowledge and expertise of the machine operator. In machine shops with CNC machines, this method cannot apply any more. This is due to the large complexity associated with determining the correct tool in the modern environment with a large number and greater variety of available tools. If the operator is relied on for tool selection, machine stoppages become inevitable due to incorrect tool use and unavailability of tools.

Even if operators rely on the tool manufacturers' recommendations, there is usually only one source of information for the following reason: more experience operators will stick to the same suppliers, while newer operators will usually depend on the first supplier that has the required tools. To allow the selection of optimal tools, there needs to be the development of programs that are supported by extensive manufacturing theory and tool data, and utilise this information to provide reliable recommendations for tool selection.

1.2. Problem Definition

The aim of this project is to develop an application that can recommend the best tool for a machining task. The application will need to filter tools from multiple manufacturers, then an optimisation method must be used to find the best tool, based on criteria such as throughput, surface finish, power consumption and tool life.

1.3. Proposed Solution

An application will be developed on the Android platform. Native SQL support in java will allow a reliable database connection, while increasingly faster mobile processors are able to perform the computations needed to optimise the tool selection.

1.3.1. Scope

Given the time limitations for this course, and the vast amount of tools available, the tool selection in the application will be limited to only solid carbide end mills. Additionally, the application will be developed on the Android platform only.

2. Background

2.1. Literature Review

Machining and cutting tool database systems have been developed as an important component in computer aided process planning (CAPP). The objective of the cutting databases is to provide for a certain machining task, either a list of usable cutting tools, or suitable cutting parameters for a tool. The output of the machining system can also be optimised, so that the most suitable tool is provided, or the best cutting parameters, based on desirable criteria in metal cutting. The literature reviewed involves cutting database systems, as well as the important components that make up cutting databases: expert/knowledge-based systems, cutting tool selection and related machining science. For the literature on cutting database systems, the authors did not provide explanations for many of the decisions related to the design of the system, or methodologies involved.

2.1.1. Metal cutting database systems

Peng et al. [2] created a cutting database system in order to provide the best alternative of cutting parameters based on machining features. The machining features are organised by the material and geometric information, part feature, machine and tool data. The system is built using a browser-server structure, and cutting parameter alternatives are optimised using TOPSIS (Technique for Order of Preference based on Similarity to Ideal Solution). The CATA system was developed by Ribeiro et al. [3] to optimise the selection of cutting tools and cutting conditions, based on maximum productivity in an industrial environment. A set of procedures to make comparative test between cutting tools and optimise the results was used.

A knowledge-based system was developed by Arezoo et al. [4] to select tools and cutting parameters for turning applications. This system, known as EXCATS was designed for indexable tools, and is able to analyse and optimise the cutting insert, toolholder and cutting conditions such as the speed, feed and cutting depth. The tool supplier or user is also able to modify the system to suit individual needs. The system is constructed with the programming language Prolog, and contains a knowledge base, inference engine, explanation facility and model to optimise cutting conditions and tools. Zhao et al [5] integrated CAD functionality and EXCATS for cutting tool selection. This system is able to process a part's CAD data and generate a representation model that can describe elaborate turned components. Representation files for roughing and finishing operations are created in the industry-neutral format, IGES. The system, CADEXCATS, processes the IGES files to produce the data that can be directly imported in EXCATS for optimal cutting tool and condition selection. Another knowledge-based system. COROSolve, was developed by Cakir and Kavdar [6] to investigate metal cutting problems in milling, turning and drilling. The programming language Delphi was used, and the system has three main functions: recommendations for cutting parameters, analyse metal cutting problems and evaluating input cutting parameters. Knowledge-based system utilise the knowledge held by experts in the machining field to provide optimal results.

2.1.2. Expert/knowledge-based systems

Arezoo et al. [4] developed the EXCATS expert system in a modular basis . Different tool manufacturers and machining operations are stored as modules, which allows the system to be easily developed and updated, increasing efficiency. The knowledge base is divided into two sections: data files of material properties, toolholders and inserts, and data rules about tool selection, toolholder capabilities and other factual and heuristic knowledge. In a similar

fashion, the rules for COROSolve's knowledge base is comprised of a mixture of factual statements, 'If-Then' clauses, procedures and cases [6]. The variety of sources that constitutes the knowledge base, helps improve the efficacy of the inference engine. Using literary sources such as machining and production handbooks, and resources from manufactures such as Sandvik AB, Edalew et al. [7] created a knowledge-based system to aid in an intelligent system for automatic tool selection. Additional expertise was obtained from academic research groups and industry experts. The knowledge base is stored in a hierarchal manner, with information related to the part features, cutting tools and cutting process, being represented as different high-level objects. Each of these objects consists of many attributes. The engineering knowledge that is utilised by the inference engine is represented as logical clauses. By representing the knowledge this way, the rules are shown in a natural and simple format.

A hybrid approach was used by Maropoulos [8] to develop of a knowledge-based system for aiding an intelligent tool selection system. The knowledge base is comprised of data and rules that have been derived theoretically, as well as from analysing feedback from the machine shop floor. The feedback, mainly tool life and performance data, is used to update the knowledge base so that future predictions and recommendations increase in accuracy.

2.1.3. Optimising cutting tool selection

Peng et al. [2] utilised TOPSIS to optimise their cutting parameter selection. TOPSIS is a well-known Multiple-Criteria Decision Making (MCDM) method that was introduced by Hwang and Yoon [9], and can be adapted for optimising tool selection. Response surface methodology (RSM) is another method that involves multiple criteria and alternatives. Reddy and Rao [10] based their selection of optimal tool geometry and cutting conditions on a prediction model for surface roughness, and used RSM to develop first and second order mathematical models for this prediction model. Experimental studies were done to observe the effect of end mill geometry such nose radius and radial rake angle and cutting conditions such as speed and feed rate. The adaptive heuristics method, genetic algorithms, were used to optimise the prediction model, so that the minimum surface roughness and the respective optimal conditions are provided. Genetic algorithms were also used by Mizugaki et al. [11] to optimise tool selection based on minimising machining time and uncut area. In a different manner, curvature matching in five-axis milling was used as the basis for an automatic tool selection system by Jensen et al. [12]. The main constraint for tool selection was preventing any interference between the tool and changes in the surface curvature, and the tools are optimised to maximise the material removal rate and minimise any machining errors.

Rho et al. [13] presented a method for finding the optimum solution for the sequence of machining operations and cutting tool for each operation. The tool selection module tries to combine common attributes between operations so that the number of tools required for a part to be machined is reduced while the most suitable tool is selected for an operation. Oral and Cakir [14] also optimised the tool selection and operation sequence based on minimum tool changes. The tool selection module of the system considers the geometric parameter, grade and function of indexable inserts.

Carpenter and Maropoulos [15] described a procedure to select efficient tools for roughing and finishing operations. Initial constraints such as tool type, part geometry, grade, material and the machine capability determine the set of feasible tools. Aggressive cutting parameters are generated for the feasible tools, then this cutting data is refined in terms of constraints such as

tool life, machine power and surface finish. The tool set is then optimised with regards to minimum cost, maximum tool life or maximum throughput. An additional cutting criteria known as harshness, is used, and allows the user to influence the chip thickness that is produced.

2.1.4. Machining sciences

Cutting fluid generally provides many benefits in machining processes, for example better surface finish, cutting speed, longer tool life, etc. Cakir et al. [16] found that the selection of a suitable cutting fluid is dependent on three main factors: the type of machining process, the workpiece material and the cutting tool material. Onouha et al. [17] performed experimentation on the effect of cutting fluids on surface roughness, and found that feed rate has the greatest effect on surface roughness. The cutting fluid was second in influence, and these results agrees to the study conducted by Noordin et al [18] .

Astakhov [19] investigated the validity of tool wear criterion, and found that the minimum tool wear occurred at an optimal cutting speed for the machining application. Kilicap et al. [20] also determined that cutting speed had the largest influence on tool wear, followed by feed rate. The effect of the depth of cut was found to be minimal. The surface roughness was found to be largely affected by the cutting speed and feed rate. A high cutting speed and low feed rate produced a better surface quality. Makadia and Nanvati [21] used RSM to model the effects of tool geometry and cutting parameters on surface roughness, and also found the feed rate to be a main factor, along with the nose radius of the tool. The effect of the endmill length-to-diameter ratio on surface roughness was investigated by Hadi [22]. Longer tools exhibited greater chatter, but a 1:2 ratio between tool length and immersion was found to produce the best surface finish.

The above research can be summarised as follows:

- The various factors that influence the metal cutting process must be taken into account when designed tool selection software.
- Real world expert knowledge produces good results when creating selection models.
- There is no software available commercially that optimises cutting tool selection specifically.
- Many nuances exist that affect cutting tool performance.

2.2. Existing Software Implementations

This section explores software packages that have been created by academic teams and tool manufacturers. Table 1 shows a summary of some of the cutting tool software that is available.

Table 1: Cutting database evaluation

Cutting database	Good aspects	Bad aspects
C. Peng et al. cutting database [2]	-TOPSIS, a well-known MCDM technique, is used to optimise the cutting parameters. -Data has been obtained from many sources: machining data handbooks, software simulation, laboratory experiments and shop experience -SQL used as database platform, web application developed with C#, interfaces with MATLAB. -Data can be inquired, added, modified, and deleted from one interface. -Dynamic drop-down menus employed. -Dedicated tab to manage the machine information.	-Basic web interface -Optimal cutting parameters are given for only the selected tool
NOVO by Kennametal [23]	-Windows-based app, requires registration to access tool advisor. -Works for milling, turning and holemaking -Has 7 general milling profiles (pocket, shoulder, slot, surface, 2D profile, chamfer, 3D profile). -Sub-profiles exist for each general profile (under 'Pocket' for example, is closed pocket, open pocket and cylindrical pocket). -Search can be arranged by any of the ~10 tool properties (e.g. feed rate, power, depth of cut, and others).	-Search displays many (~100) results, but does not allow multiple search filters. -Search does not have optimisation to display best results. -No app or web-based interface.

EXCATS [4]	-Uses the popular logic programming language Prolog -developed in a modular basis to accommodate different machining operations and tool manufacturers -User can define a database of tools that are suitable for the company's needs. -EXCATS recommendations can be overridden by the user, and repeatedly consulted. -Can advise user on coolant selection. -Has a knowledge base for troubleshooting	-Text based interface is not appealing and difficult to use. -The selection criteria is based on the suitability of the cutting tool according to the manufacturer, and not on the mechanical properties. -Knowledge base only catered for turning at the time. -Optimises cutting parameters only according to the maximum production rate and the minimum cost per unit. -Programming language is not easy to follow.
Iscar Tool Advisor [23]	-Caters for milling, turning, groove & parting and holemaking. -The milling section has an extensive amount of profiles to choose from (e.g. shouldering, facing, pocketing, slotting, profiling, plunging, undercutting, thread milling and others). -Has a web, Windows and Android client. -Input data is comprised of application, tool, stability and machine details. -All material standards available for material selection.	-Does not optimise the tool selection. -Only provides tool recommendations from the Iscar Metalworking catalogues. -Android app has limited output results.

After evaluating these cutting databases, many beneficial features can be noted: large profile selection, MCDM optimising algorithm, wide knowledge base, multiple database access levels, appealing, intuitive and fast user interface, input and output data categorising, tool and machine profiles and dynamic filtering. These features, and more, will be considered when developing the database and web interface. Overall, Iscar Tool Advisor is the most extensive in terms of input data. Output data consists of tool entries in the database, so digitising different manufacturers catalogues will provide a wide tool database to recommend from. Peng et al. [2] had used a good algorithm to optimise results and prominent languages to code the application. NOVO has a good interface and highlights the need for result filtering.

2.2. Android Fundamentals

Android is the most widely used operating system for mobile phones in world, with xxx handsets running Android, out of xxx smartphones worldwide. The popularity of Android can be largely attributed to its open source nature. The target user group: CNC technicians and operators, are likely to have a cheaper, durable phone as opposed to an expensive, fragile phone. Also, the only other device in the machine shop is like to be the CNC management computer, which may not have a reliable internet connection, reducing the application's functionality with a cloud-based database/server. The cheapest smartphones are Android devices, and phones have wireless connectivity. These reasons: the popularity of android, and the variety of devices that run it, result in it being a good platform for this Machining Tool Adviser application to be developed on. This section covers some of the fundamental aspects of creating an Android application which are necessary to understand the discussion of the application development in section 3.

2.2.1. Mobile design principles

In order to develop an application for a mobile device, the differences between mobile devices and traditional computers need to be considered [24]:

- Phones have much smaller screens. Multiple windows cannot be supported due to the screen size, hence only relatively small layered data is able to be displayed, such as dialog boxes, popup windows, and dropdown menus. Also, large amounts of text can be difficult to read on small screens because the context of the screen can be easily lost if the user moves or gets distracted. Since only one screen is able to be displayed at a time, the application must provide the desired functionality fast.
- Phones have a limited power supply, a battery. All the components within the phone, including the processor, network connectivity, screen and speaker, use power from the battery. If any phone resources are used, the application must be designed to minimise the power used.
- The network connection is purely wireless, relying on a cellular or Wi-Fi network. This can result in inconsistent connectivity to the internet. The application should be designed to function as much as possible despite an inconsistent connection.
- Text entry is more difficult on a mobile device. Shorter texts are preferred by users when typing on phones, therefore other input methods should also be used when developing an application, such as auto-completion, seek bars and buttons.

2.2.2. Architecture

The Android operating system has been created as a stack of many different software that lay at different levels to the underlying hardware. These levels group the software within, as shown in Figure 1, and are organised as follows [25]:

- All System and third-party applications are built on the Application Framework through the use of the same API libraries.
- The Binder Inter-Process Communication mechanism is used to allow the Application Framework to call into the system services in Android. At the Application Framework level, this communication is not visible to the developer.
- The system services are modular components with a focused purpose. The system services allow the underlying hardware to be accessed by applications on the Application Framework.

- A Hardware Abstraction Layer (HAL) is a standard interface that exists for hardware suppliers to implement, resulting in Android supporting many hardware devices.
- Android utilises a version of the Linux Kernel for device driver implementations, as well as for additional system functionality, such as Low Memory Killer.

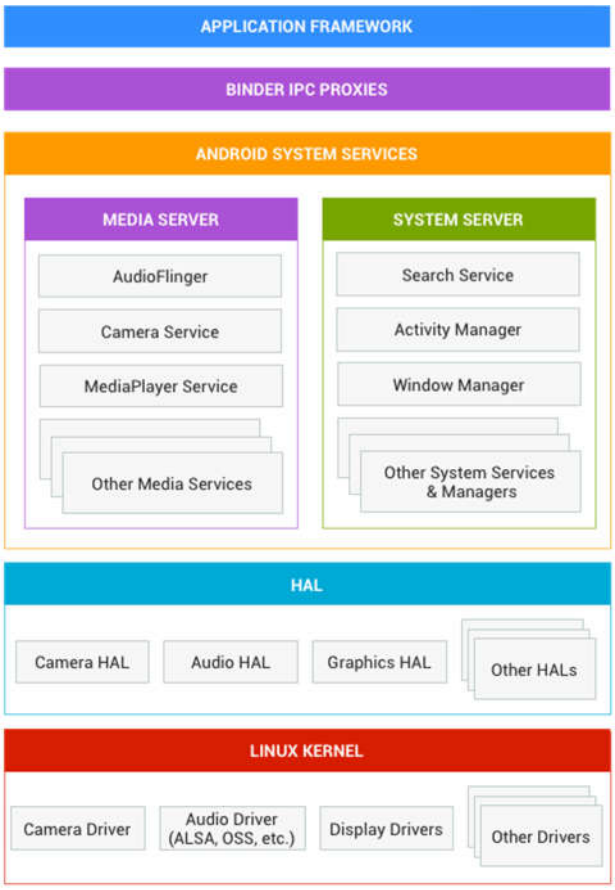


Figure 1:Android system architecture

2.2.2. Development Environment

The most popular Android integrated development environment (IDE) software is Android Studio, by Google. Android Studio provides the necessary Application Programming Interface (APIs) and developer tools that allow Android applications to be built, tested and debugged. Other features include:

- A styled layout editor that allows User Interface (UI) components to be visually added to the application layout.
- Android Virtual Device (AVD) emulation to allow applications to be tested within Android Studio.
- Gradle is utilised as the build automation tool, which involves compiling and packaging the binary code for the application.
- Code refactoring and quick is available for Android, allowing the code structure to be changed without changing the functionality of the application.

2.2.3. Database management

There are two main types of databases: relational and non-relational databases. Relational databases are structured in the form of tables. These tables organise structured data fields into defined columns. A non-relational database does not use a table structure, instead, data is stored as individual files. The advantage of a non-relational database system, such as MongoDB or Oracle NoSQL, is that large volumes of information can be stored easily when the data is largely unstructured. Also, rapid development of a non-relational database is easily achieved, as no preparation of the database structure is needed.

A relational database is written in Structured Query Language (SQL). SQL queries can be used to retrieve data and update, delete or create records in a database [26]. A particular advantage of SQL databases is the JOIN clause, which allows data from multiple tables to be retrieved with a single command. SQL databases, such as MySQL and Microsoft SQL Server, are well suited for structured, static data, as well as database integrity.

For the structured nature of the various data used in the application, such as tool data and recommended cutting conditions, as well as the retrieval flexibility offered by SQL, a SQL based database is chosen to manage the application data. The implementation of the SQL database is discussed in section 3.3.2.

2.2.4. Computation

The main data types used in computations in java are `float` and `double`, representing 32-bit and 64-bit floating point values respectively. The `double` data type has a range of approximately $\pm 2^{971}$ [27].

By placing an array of double values (`double []`) into another array (`double [] []`), two-dimensional matrix calculations can be performed.

The `java.lang.Math` class is built into the java platform, and allows elementary numeric operations to be performed, such as the exponential, logarithm, square root and trigonometric functions.

2.3. Machining Science Fundamentals

In this section, the machining science required to accomplish the application's required functionality is investigated.

The recommended cutting parameters provided by the manufacturers catalogues are: cut depth/diameter ratio, feed/tooth and surface cutting speed. In effect, the depth of cut per pass, width of cut per pass, feed per tooth and cutting speed parameters are the input cutting conditions. These values must used to calculate the parameters by which tools are compared to each other in this application.

2.3.1. Material removal rate

The material removal rate determines the overall throughput of a machining operation. In mass production environments, maximising the rate at which operations are performed results in increased productivity. An expression for the material removal rate, Q, is shown in equation 1 below [28]:

$$Q = D_c \times W_c \times V_f$$

Equation 1: Material removal rate

Where D_c = depth of cut

W_c = width of cut

V_f = feed rate

The depth and cut and width of cut are inputs to an intended machining task. The feed rate, which is unknown, can be related to the feed per tooth as shown in Equation 2 below [29]:

$$V_f = n \times f_z \times Z$$

Equation 2: Feed rate

Where n = spindle speed

F_z = feed per tooth

Z = number of cutting teeth

With the feed per tooth being an input parameter, and the number of cutting teeth generally known for a cutting tool, the spindle speed *n* can be related to the surface cutting speed as shown in Equation 3 below [29]:

$$n = V_c \div \pi \div D \times 1000$$

Equation 3: Spindle speed

Where V_c = surface cutting speed

D = tool diameter

With the spindle speed calculated, the feed rate can be determined hence the material removal rate can be calculated.

2.3.2. Cutting power

The cutting power P_c, can be expressed in terms of the material removal rate as shown in Equation 4 below [29]:

$$P_c = Q \times k_c$$

Equation 4: Cutting power

Where Q = material removal rate

K_c = specific cutting energy (energy needed to remove 1 mm^3 of the workpiece material)

2.3.3. Shear plane deformation

The shear strain that a workpiece material experience influences the quality of the machining process. An expression for shear strain in a metal cutting process is shown in Equation 5.

$$\varepsilon = \frac{cos\gamma_0}{sin\phi - \cos(\phi - \gamma_0)}$$

Equation 5: Shear strain

The shear angle, ϕ, can be computed by utilising the approximate relationship between chip compression ratio and cutting ratio, as shown in Equation 6.

$$r \approx \frac{UTS}{Yield} \approx \xi = \frac{\cos(\phi - \gamma_0)}{sin\phi}$$

Equation 6: Cutting ratio

The friction coefficient that the tool experiences at the tool-workpiece interface can be determined from Equation 7.

$$\phi = \frac{\pi}{4} - (\beta - \gamma_0)$$

Equation 7: Shear angle

2.3.4. Tool life

The life of a cutting is influenced by many factors. One formulation for tool wear is given in Equation 8 below [30].

$$V_{ab} = n^2 \frac{P_y E W^{3/2}}{{k_c}^2 H^{3/2}} L$$

Equation 8: Tool wear

Where n = work hardening factor

P_y = yield strength

E = modulus of elasticity

W = normal load

k_c = fracture toughness

H = hardness of workpiece

L = sliding distance

All of the factors are material properties, except for the W (normal load) and L (sliding distance) factors. The normal load can be found from the Merchant Circle diagram, shown in Figure 2 below [31].

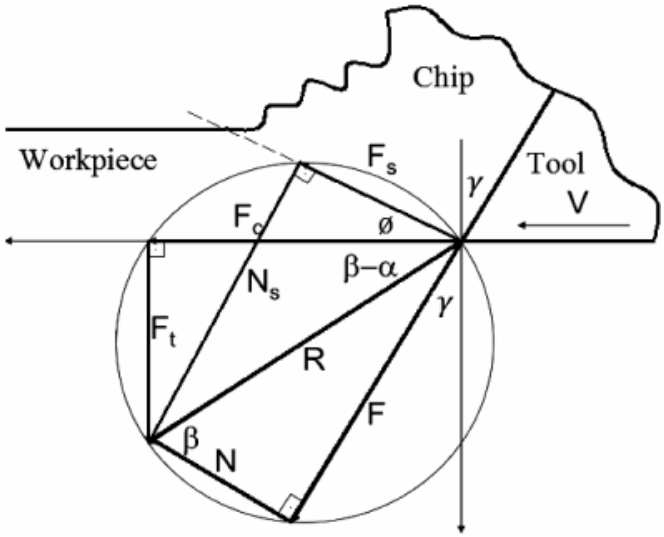


Figure 2: Merchant Circle diagram

The normal force is represented as N in the diagram. The force is expressed analytically in Equation 9 below [31].

$$F_N = F_C sin\phi + F_T cos\phi$$

Equation 9: Normal force

Where ϕ is the shear angle, found in the calculations for shear plane deformation.

The sliding distance L can be found using the expression shown in Equation 10.

$$L = \frac{Chip_{number} \times Contact_{length}}{Teeth_{number}}$$

Equation 10: Sliding distance

The number of chips, Chipnumber can be determined using Equation 11 below.

$$Chip_{number} = \frac{Cut_{Length}}{Feed/tooth}$$

Equation 11: Number of chips

Where Cut_{length} is the length of the part feature (for example, a slot). The contact length of a chip can be found using Equation 12.

$$l = \sqrt{\Delta D}$$

Equation 12: Contact length

Where: Δ is the depth of cut

D is the tool diameter

The contact length, number of chips, and number of flutes are used to calculate the sliding distance. With the normal force calculated as well, a score for the tool life can be found.

2.3.5. Surface roughness

The roughness of a surfaces after a milling operation can be represented as shown in Equation 13 below [32]:

$$R_a = \frac{f^2}{31.2 r_n}$$

Equation 13: Surface roughness

Where f = feed rate

r_n = tool tip radius

2.4. Multi-Criteria Decision Making Fundamentals

The MCDM algorithm TOPSIS is used to identify the best alternative out of many, based on several criteria. TOPSIS, which stands for *Technique for Order of Preference by Similarity to Ideal Solution*, performs a recommendation under the premise the best alternative should have the shortest geometric distance to a theoretically perfect solution, and the longest distance to a theoretically worst solution. This algorithm has a compensatory nature, therefore an alternative with a bad result in one criterion, may have a bad score offset by a good result in another criterion. In machining process optimization, the multitude of factors involved results in all of the factors playing a role in the performance of the process, requiring a compensatory optimisation method to be used.

The algorithm is performed as follows [33]:

1. An evaluation matrix of *m* alternatives and *n* criteria is created, known as $(x_{ij})_{m\times n}$.
2. The evaluation matrix is normalised. The normalised matrix is represented as $R = (r_{ij})_{m\times n}$, where each normalised element $r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} {x_{ij}}^2}}, i,j \in \mathbb{N}$.
3. A weighted normalised decision matrix is created by multiplying the normalised matrix *R* by weightings assigned to the *n* criteria. Each element of the weighted normalised decision matrix $t_{ij} = r_{ij} \cdot w_j$ where w_j is the proportional weighting assigned to that parameter.
4. The best alternative (A_b) and worst alternative (A_w) must be determined. The best alternative consists of the best scores for each criterion for each of the alternatives. For a criterion when maximisation is desired (tool life, for example), the best score will be the highest value, and the worst score will be the lowest value. Conversely, for a criterion when minimisation is desired (power consumption, for example), the best score will be the lowest value, and the worst score will be the highest value.
5. The Euclidean distance between each alternative and the best (t_{bj}) and worst (t_{wj}) conditions are found. The distance from the best condition is $d_{ib} = \sqrt{\sum_{j=1}^{n}(t_{ij} - t_{bj})^2}$, and the distance from the worst condition is $d_{iw} = \sqrt{\sum_{j=1}^{n}(t_{ij} - t_{wj})^2}$.

6. The similarity to the worst condition, or the closeness coefficient for each alternative is $s_{iw} = \frac{d_{iw}}{(d_{iw}+d_{ib})}$.
7. Rank the alternatives according to the s_{iw} score, from highest to lowest.

3. Android Application Development

3.1. Conceptual Design

In order to create a good application, an application should be designed with these features, capabilities and limitations of the platform kept in mind.

From the review of existing tool selector software from tool manufacturers as well as academics, the general approach to providing (at its most basic) tool selection from a catalogue is as follows:

1. The user selects the profile/machining feature to be machined.
2. Input data
3. filter

This functionality is simple and relatively trivial to implement. In order to recommend one cutting tool over another, however, an additional aspect, 'Optimisation', must be implemented on top of the above three aspects.

To achieve the functional, as well as human-interaction requirements, the conceptual solution application consists of three main components: the interface, the database and the knowledge base. The interface serves to accept input data in an easy-to-use fashion. The database stores information about tools, machines, material properties and other static data. The knowledge base will be implemented throughout the application with regards to the engineering sciences and inference rules that will allow the tool selection to be filtered and optimised.

It can be noted that the input data controls the most fundamental functionality of the application: filtering a catalogue of tools for the entries that are usable in the application user's desired machining task. However, the effectiveness of the input data at filtering the tool database, is limited by the information available about each tool (for example, appropriate workpiece materials, or cut profile/machining feature). Therefore, the application must be tailored around the tool information that is available, limited namely to public catalogues from tool manufacturers.

A list of common details that can form the input data is shown below in Figure 3.

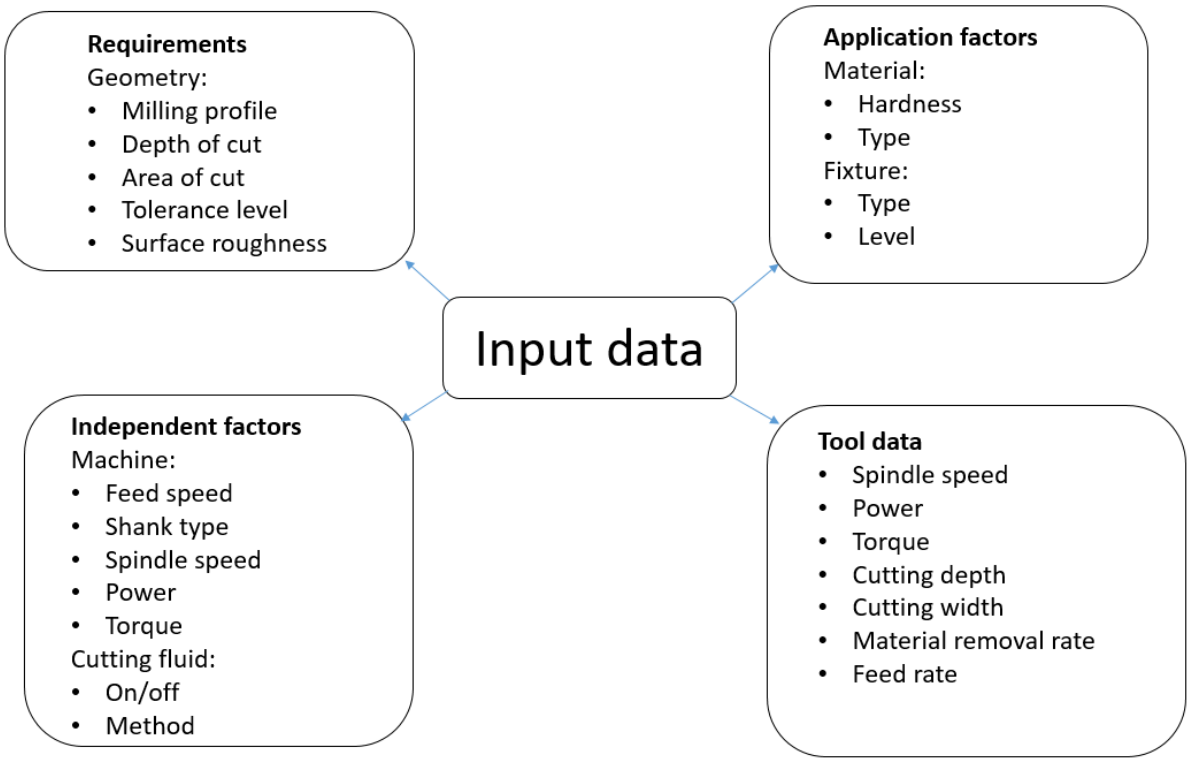


Figure 3: Input data model

These inputs form the bulk of the data used to filter the tool database. The input structure of the application, designed based on the input data shown above, and the tool data available from the Seco Tools Endmill Catalogue (Machining Catalogue), is shown in the form of a flow diagram in Figure 4.

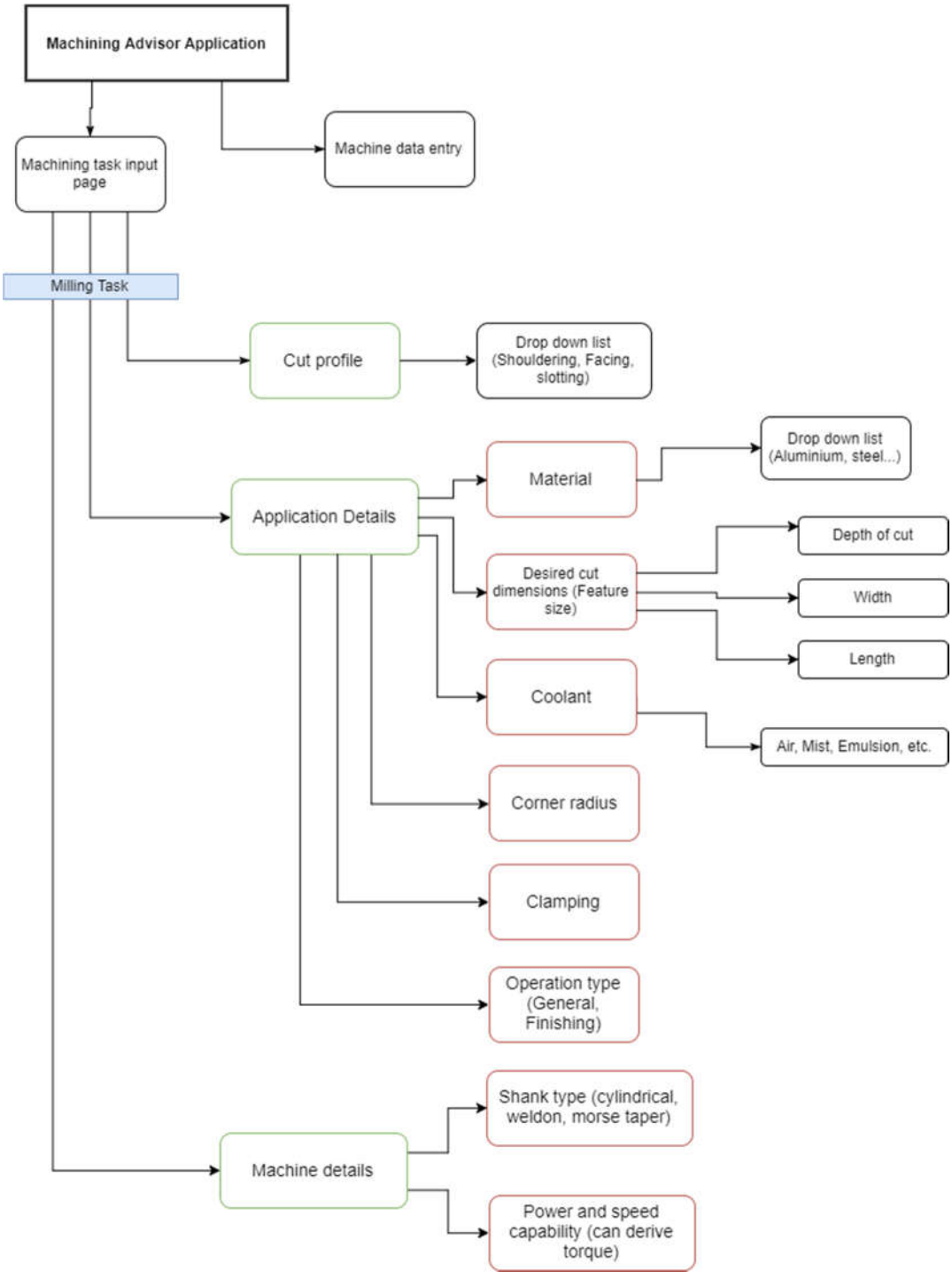


Figure 4: Input structure flow diagram

With the structure of the input known, the layout for visual interface that utilises the inputs can be created. As the visual interface serves the user directly, the limited screen size of a mobile device must be taken into account, with regards to displaying information as well as acquiring information.

A series of images based on the displayed content in an application will be referred to as a screen flow diagram. Screen flow diagrams are used to visually represent the different User Interface (UI) components that are utilised in the application in the various distinct screens available to interact with. Screen flow diagrams are a great tool to use when developing the concept for an application, as the feasibility of the UI elements, as well as that the overall feel of the interface, can be judged from a mobile users perspective, before any software development begins. At a later stage, screen flow diagrams can be used with real-life screen captures from a developed application, to showcase the functionality of the interactive application on a static display (for example, a thesis report). A screen flow diagram for the machining tool advisor concept is shown in Figure 5. Descriptions for each of the screens feature after Figure 5. The free, online flow diagram software, draw.io, was used to create the screen flow diagram in Figure 5.

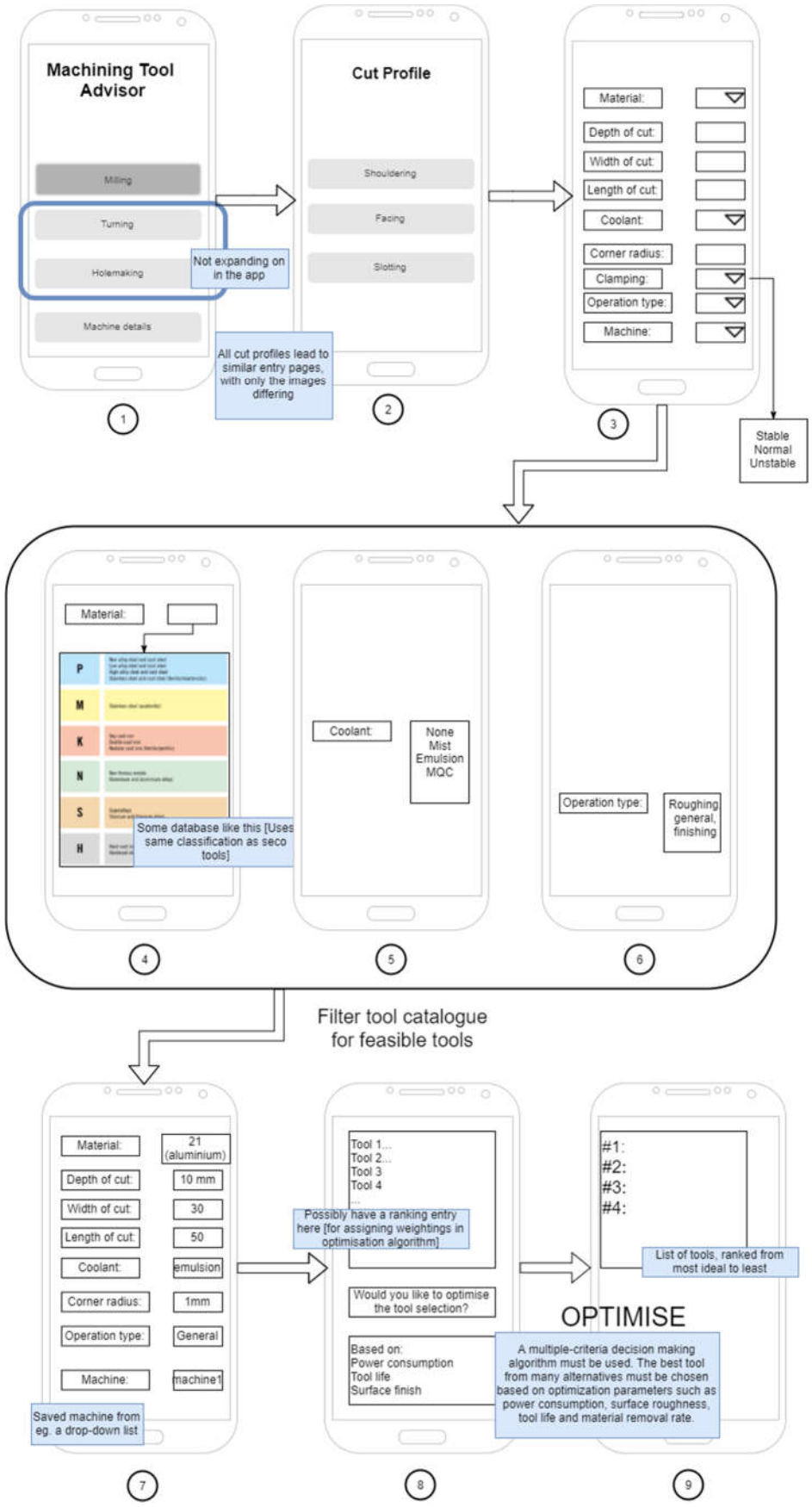


Figure 5: Machining application conceptual flow diagram

As applications involve a human-interaction aspect, a hypothetical use-case is considered. The actions of the hypothetical user of this application will be represented in *italics*. The functions and flow of data as shown in Figure 5, is explained in more detail, based on the numbering of the individual screens:

1. *The user installs and opens the application*. The top level screen for the application is shown. The user is given the option of selecting a desired machining type (Milling, Turning and Holemaking) for which an optimum tool selection will eventually be provided by the application. Also present on this screen, is an option to go to a "Machine details" page, which is necessary for the user to add and edit personal machine details. This data is stored in a local database on the device. Due to the scope of the tool type for this project being limited to solid carbide end mills only, the "Turning" and "Holemaking" options will not be developed further. *The user selects "Milling"*.
2. Depending on the machining type selected in Screen 1, this screen will show a variety of options corresponding to possible cut profiles, also known as part/machining features. In the case of this application, choosing "Milling" in Screen 1 will yield possible milling profiles, such as side milling, face milling and slot milling, in Screen 2. These profiles are dependent on the information available on the cutting tools, as each tool manufacturer specifies the profiles that their tools are suitable to machine. *The user selects "Slotting".*
3. In this screen, the user is required to input details about the intended machining task, for the profile chosen in Screen 2. The user will be required to enter the necessary details to adequately describe the machining task (for example, part requirements, workpiece material, machine, fixture level, etc.). Drop-down lists are used for the inputs that have discrete static options, such as the workpiece material, or the type of coolant available. Possible options for "Clamping" are shown in the box next to Screen 3. The options correspond to the ability offered by clamping technology. For example, the "Stable" option refers to the high level of stability offered by hydraulic clamping. "Normal" refers to clamping technology offering slightly less stability, for example machine clamps.

Screens 4, 5, and 6 show possible options for the dropdown lists for "Material", "Coolant" and "Operation Type" respectively. All of these inputs take place on the same input-page parent activity (Screen 3), and have short descriptions here for clarity.

4. *The user taps on the "Material" dropdown list*. The list of materials is required to be comprehensive enough to cover the most common engineering materials. For this reason, the ISO materials classification is used, with further discretizing available within P (steels), M (stainless steels), K (cast irons), N (non-ferrous metals), S (heat-resistant alloys) and H (hardened steels). Note that the material suitability of a tool is dependent on the manufacturer's specification, therefore the material options presented in the material dropdown list must cover the materials that the tools have been specified to cut. The list of materials, as well as the associated properties for each (for example, ultimate tensile strength and hardness, among others), will be stored in the application database. Picture of ISO material list adapted from [34]. *The user selects "N21 (Aluminium)".*
5. *The user taps on the "Coolant" dropdown list*. The list of coolant types are the most commonly used in CNC machining: no coolant, mist spray, oil-water emulsion and minimum-quantity cooling (MQC). *The user selects "Emulsion".*

6. *The user taps on the "Operation type" dropdown list*. The list of operation types are the most commonly grouped strategies: rough machining, general machining and finishing. *The user selects "General".*
7. *The user completes the rest of the input details.* Other input data such as the part feature dimensions, maximum permitted corner radius and personal machine are needed to filter the tool database, based on the specifications from the manufacturer. The part feature dimensions: the depth, width and length, represent the intended approximate size of the intended machining task. For example, in the case shown in Screen 7, the user desires to machine a slot that has 10 mm depth, 30 mm width and 50 mm length. The maximum corner radius is specified by the user, based on the tolerance needed at the newly-machined corner of the workpiece. The "Machine" dropdown list contains entries that correspond to the machines added in the "Machine Details" activity, the button for which is in Screen 1. *The user clicks the button to filter the tool catalogue.*
8. The page that opens, shows a list of tools that are suitable to use for the user specified machining task. The inputs from the user are used to filter the application's tool database appropriately. In order for the application to provide a recommendation on what is the best tool to use from the list shown, additional user input may be required. Often in multiple-criteria decision making algorithms, weightings are required for the criteria to be optimised, where the weighting reflect the importance level of one criteria over another, for the intended machining task. In this case, the user can choose to make one criterion, such as power consumption, to be more important than the other criteria, namely tool life, surface finish, material removal rate and shear plane deformation. The weightings can be input via text, or through the use of graphical objects. *The user selects preferred weightings for each criterion, and clicks the button to optimise the tool selection.*
9. The MCDM algorithm used to optimise the tool selection in this application is TOPSIS. In order for TOPSIS to be used, comparative scores for the criteria must exist for each tool alternative. Hence, by this stage, computations must be performed to find the criteria scores for each tool alternative, based on the machining task details specified by the user and individual tool information from the database. The optimisation algorithm runs for the list of tools filtered in Screen 8, and will output a list of tools, ranked from most ideal to least ideal, based on the importance weightings specified in Screen 8. Comprehensive details on the best tools shown will be available by clicking the desired tool. *The user selects the first tool item to see additional details. A local supplier can then be contacted to purchase the tool from.*

By visualising the appearance and mechanism of the application with screen flow diagrams, a framework is created around which the application can be developed. The user interface can be constructed based on the screens, and the underlying mechanisms added piece-by-piece to form the desired functionality.

3.2. Screen Flow Demonstration

In the previous section (3.1), the basic appearance, functionality and interface controls of the application was introduced. In order to clearly explain the development of the application, the numerous graphical and programming elements in each screen of the application must be explored in turn. Due to the lengthy nature of such explanations, a clear context of each screen's relation to one another, and to the application as a whole, is not immediately apparent. Hence, to provide the reader with an understanding of the application as a whole, a screen flow diagram of the fully developed application is shown in this section.

Each screen was captured from a test device while running the application. Android Studio was used to save the screenshots and apply a frame to each. Due to the large layout size of the screen flow diagram and physical page limitations, the screen flow diagram will be shown and explained as follows: firstly, the entire screen flow diagram is shown on the next page in Figure 6. The contents of the individual screens will not be clearly visible in this diagram, which serves to provide the overall context of each screen in the application. Following Figure 6, zoomed portions of Figure 6 will be shown sequentially, allowing the contents of a few screens to be distinctly seen at a time. A brief description of the relevant screens is shown after each zoomed-in figure.

As many activities contain too much content to display on the physical screen, the activities are scrollable vertically and horizontally, depending on what is required to fully display the content. For this reason, more than one screenshot is often required to show all of the content available in a particular activity. In this section, the screens are numbered such that scrollable views for a particular activity are grouped together and numbered as one screen. The screenshots that make up a screen group are labelled alphabetically for referencing purposes.

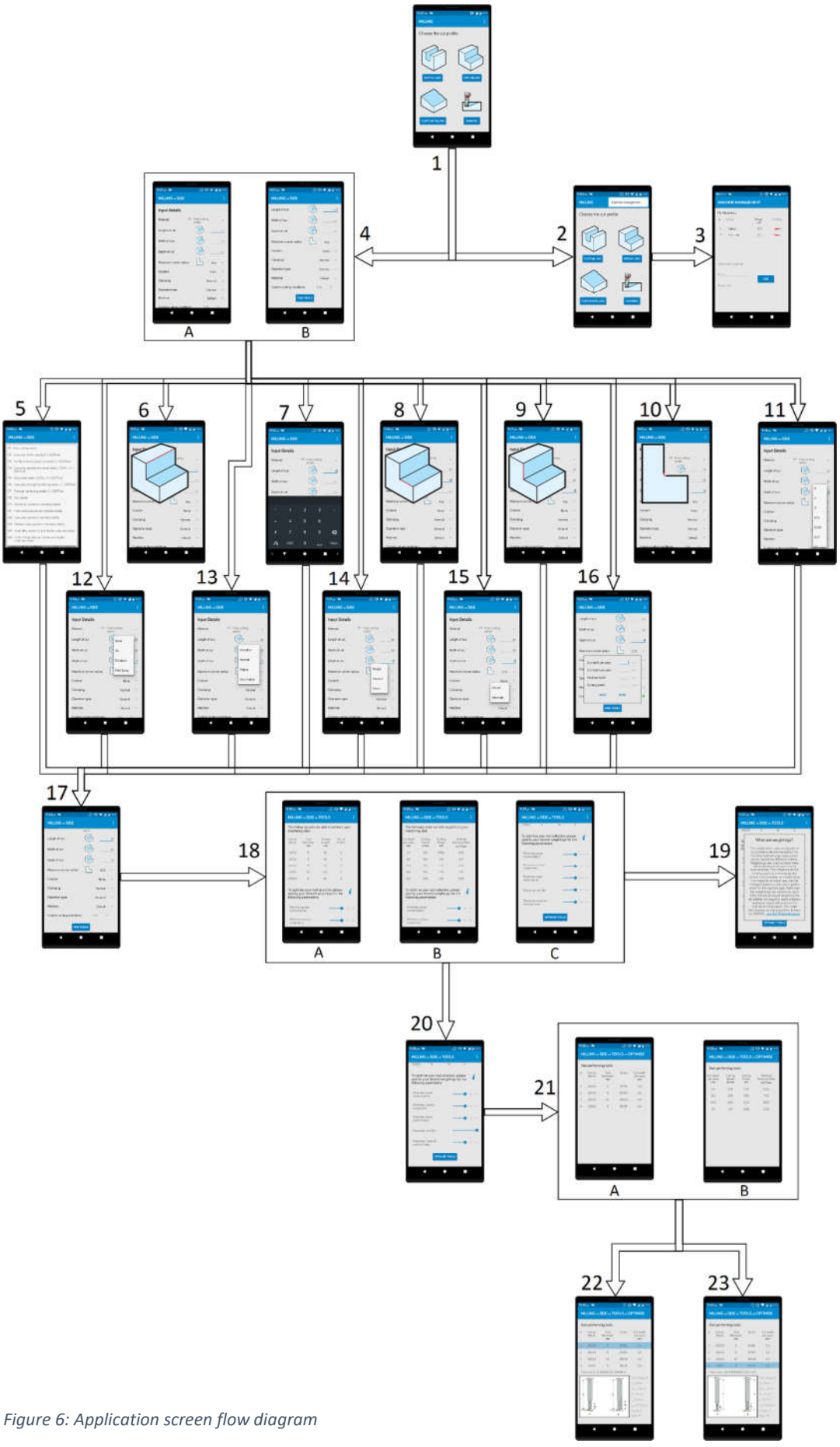


Figure 6: Application screen flow diagram

Similar to Section 3.1, a hypothetical use case is considered, and the choices made by a hypothetical user will be shown in *italics*. In this section however, the completed application is shown, hence the use-case considered, represents the real-life functionality and response of the application. The use case that is considered is a simple side milling operation into P1 steel (free cutting steel).

The first screen that is displayed when the application is opened is shown in Figure 7 below. *The user clicks on the application icon in the application launcher.*

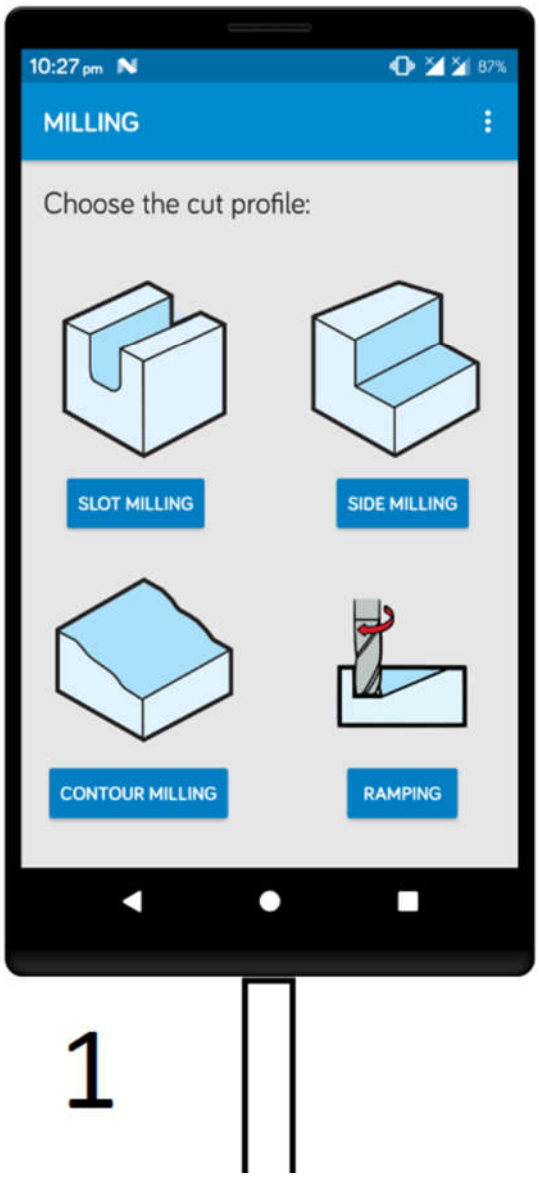


Figure 7: Screen 1

Screen 1: The user is presented with various options for a milling task. Due to the limited scope of tools programmed for (solid carbide end mills), the only type of machining this application can be used for at this point is milling. Also, due to the limited profiles defined by the manufacturer of the tools used to populate the application's tool database, the profile, "Ramping", serves only as a placeholder for future expansion. The other three profiles, namely slot, side and contour milling, have been programmed fully to work, and constitutes the capability of the initial application release. A higher level page than screen 1, as seen in the conceptual screen flow diagram in section 3.1, will serve in the future to display other machining types to the user (for example, turning and holemaking). However, in order to conform to the application's visual theme at this point, images for the different machining types will have to be created from scratch, requiring advanced graphic design skills.

Screens 2 and 3 are shown in Figure 8 below, showcasing the machine management activity. *The user clicks on the three-dot menu button in the upper right of screen 1.*

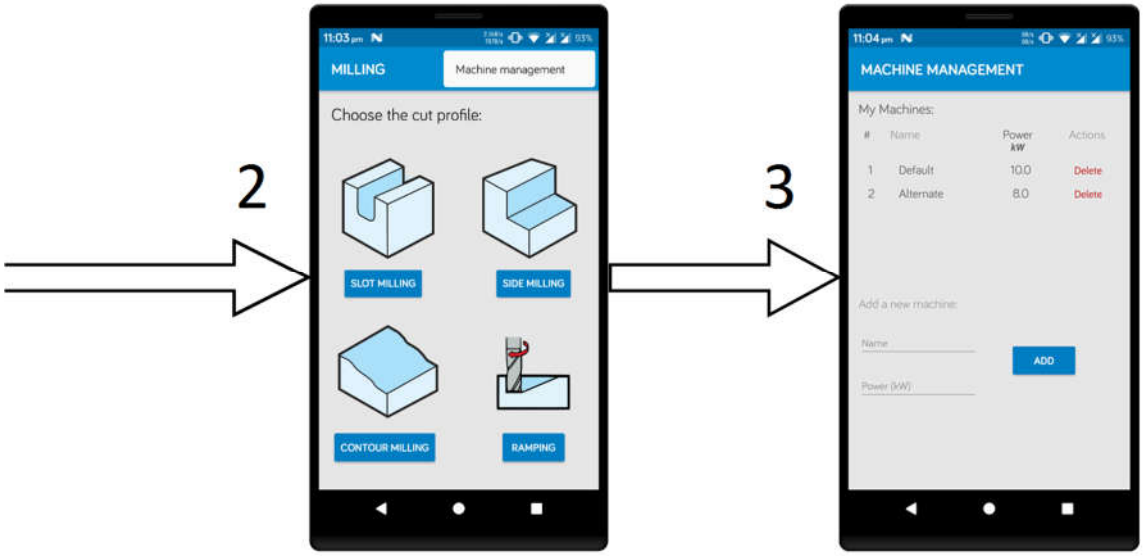


Figure 8: Screens 2 and 3

Screen 2: The menu that expands contains one option: machine management. The menu can be modified to add more items, such "Options" or "Help", as more functionality is added to the application. *The user taps the "Machine management" menu option.*

Screen 3: In this activity, the user is able to view, delete and add personal CNC machines. Currently, the only machine parameter stored is the motor power, however additional machine details, such as stability, spindle type and spindle size, can be captured in this page when the filtering system of the application is refined

further to utilise those parameters. *The user clicks the "Back" button present on the smartphone.*

After the user clicks "Back", the initial page, Screen 1, is displayed again. Screens 2 and 3 were shown for the purposes of this screen flow demonstration. In a normal use scenario the user will select their desired cut profile directly once the application is opened. Screen 4 in Figure 9 below, shows the input page displayed after choosing a cut profile (all input pages are essentially the same, differing only in the profile-specific details). *The user taps on the "Side Milling" button.*

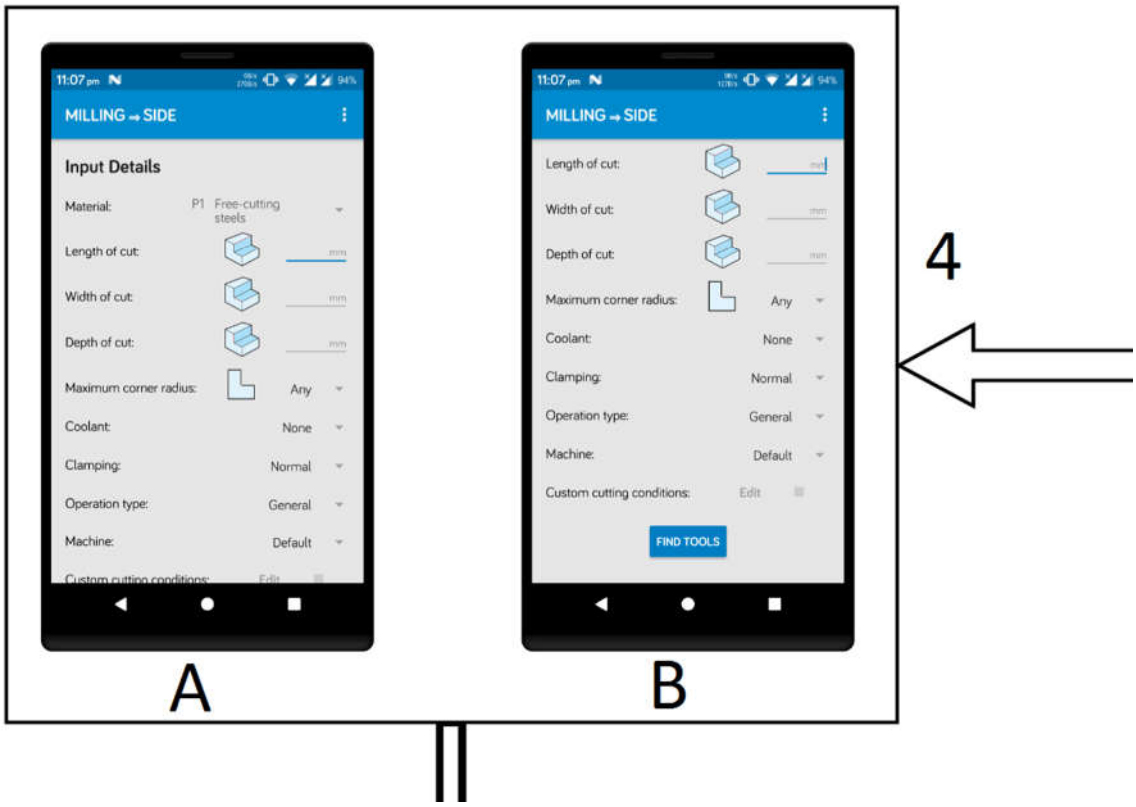


Figure 9: Screen 4

Screen 4: Part A and B seen in the figure together show the complete view of this activity, with Part A being displayed initially, and Part B being shown after the screen is scrolled down. This activity serves to capture details from the user about the intended machining task. The data captured from this page provides the bulk of the information needed to filter the tool database for usable tools, as well as data for optimisation criteria calculations.

Screens 5 – 16 showcases each of the inputs and visual artefacts available to interact with in the input page. Screens 5, 6 and 7 are shown in Figure 10 below.

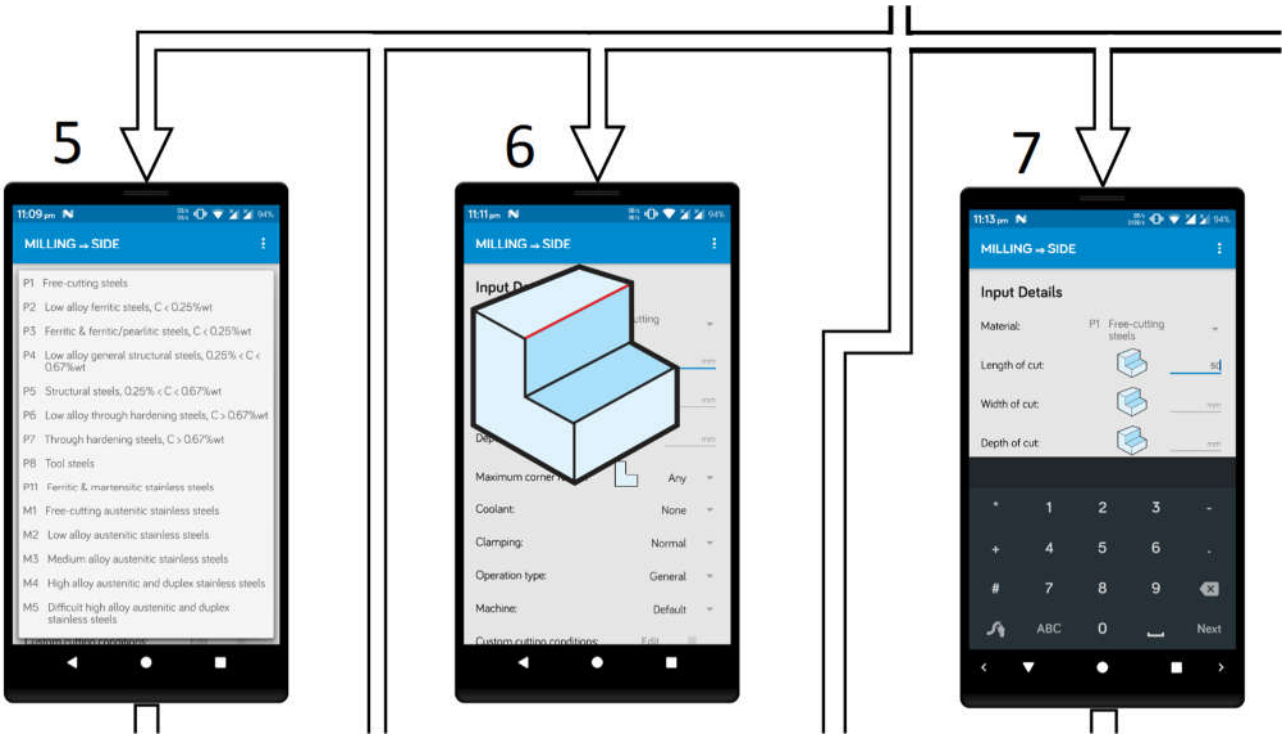


Figure 10: Screens 5, 6 and 7

Screen 5: *The user taps on the spinner item for "Material".* Once the spinner is clicked, a dropdown list of materials is shown. The materials shown are classified according to the ISO engineering materials grouping used by the tool manufacturer (Seco Tools) whose tools populate the application database in this initial release. A total of 36 materials are present in the material database. *The user selects the first entry, "P1: Free-cutting steels".*

Screen 6: After selecting a material, the user is presented with the input page (Screen 4) again. *The user taps on the picture for "Length of cut".* In order for the part feature dimensions to be entered accurately, descriptive pictures are provided. Given the limited screen area available, the images are forced to be small, possibly too small for the distinguishing dimension to be visible. Hence, the images have been developed to be zoomable, so that once clicked, the image will expand into a bigger, much clearer picture. As can be seen from Screen 6, the cut length is highlighted as a red line on the diagram. The same

implementation has been used for the width of cut, depth of cut and corner radius thumbnail images. *The user taps the big image and it disappears.*

Screen 7: *The user taps on the textbox for "Length of cut".* The default system keyboard is opened, in numeric mode if available. This allows quick entering of the desired cut profile dimensions. If the user clicks "Next" on the keyboard (in place of "Enter"), the cursor will jump to the next textbox, in this case, the width of cut. The user can also dismiss the keyboard and choose the next input to fill in. *The user types "50" (mm) in the "Length of cut" textbox, and dismisses the keyboard.*

Screens 8 and 9 in Figure 11 below, depict the main other zoomable images describing the cut profile dimensions.

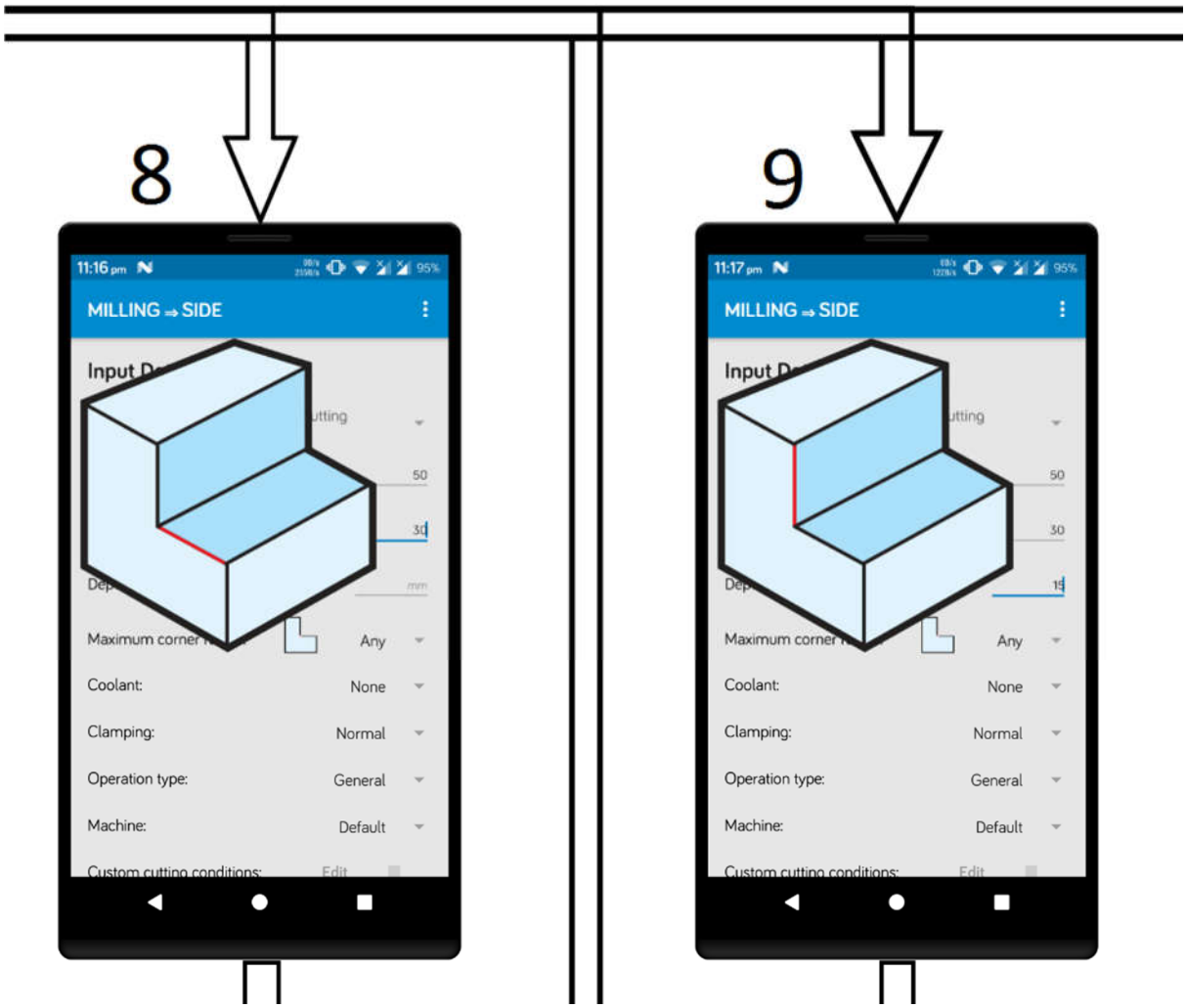


Figure 11: Screens 8 and 9

Screen 8: *The user fills in "30" under the "Width of cut" textbox. The keyboard is dismissed, and the user taps on the respective thumbnail image.* The width of cut for a side milling task is highlighted in the expanded image. *The user dismisses the cut profile image.*

Screen 9: *The user fills in "15" under the "Depth of cut" textbox. The keyboard is dismissed, and the user taps on the respective thumbnail image.* The depth of cut for a side milling task is highlighted in the expanded image. *The user dismisses the cut profile image.*

Screens 10 and 11 in Figure 12 show the interactive elements available for the corner radius input.

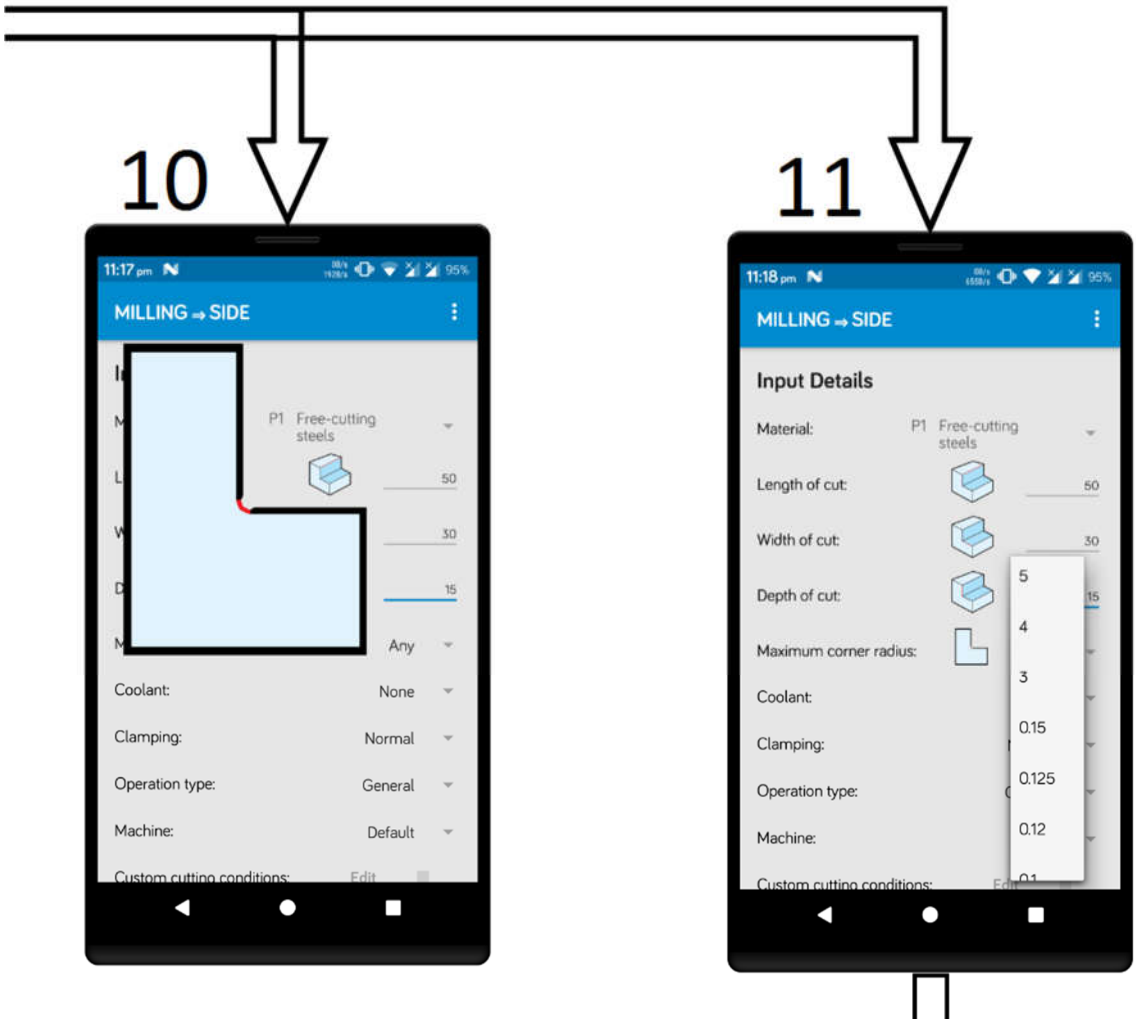


Figure 12: Screens 10 and 11

Screen 10: *The user taps on the thumbnail image for "Maximum corner radius".* This expanded image serves to illustrate which corner radius the application refers to in the input page. *The user dismisses the zoomed image.*

Screen 11: *The user taps on the spinner for "Maximum corner radius".* The values in this dropdown list are populated from the tool database, that is, the unique corner radius values from the tools in the database. The default selection is "Any". *The user selects the first entry, "Any".*

Screens 12, 13 and 14 are shown in Figure 13 below, showcasing other miscellaneous input parameters.

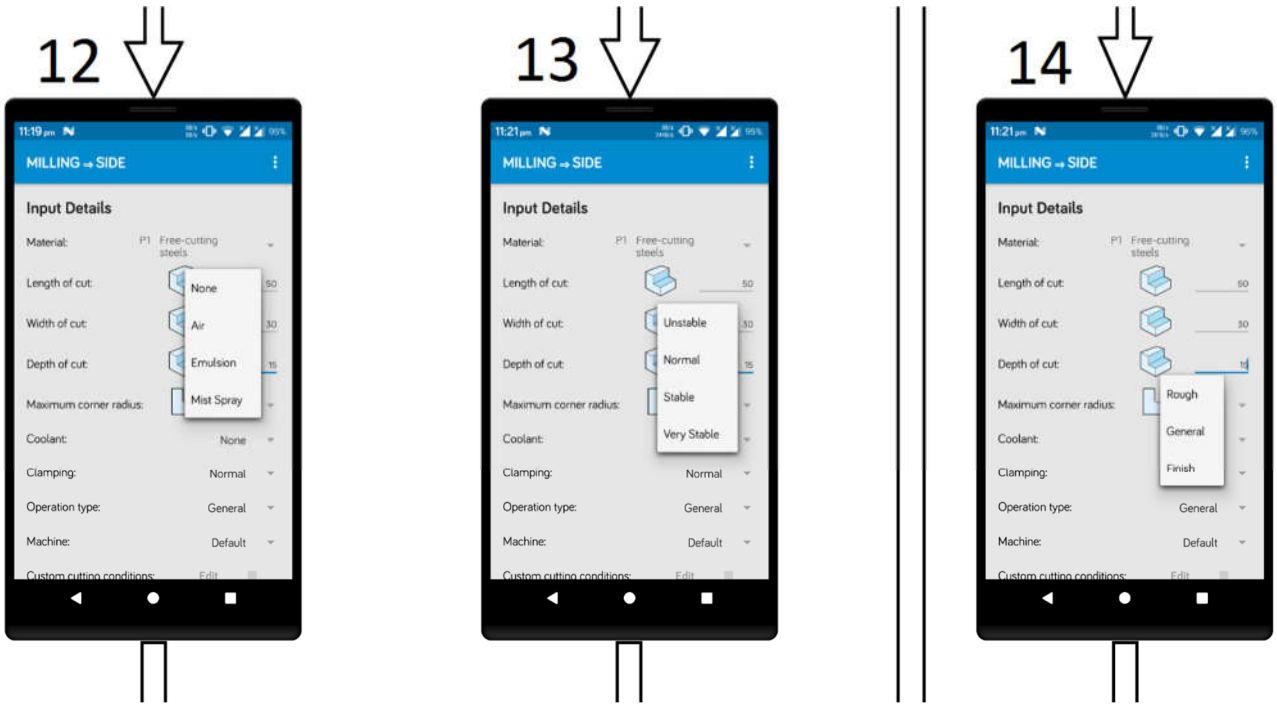


Figure 13: Screens 12, 13 and 14

Screen 12: *The user taps on the spinner for "Coolant".* The options consist of the most common CNC coolant solutions available, for possible use in cutting data recalculations and advanced tool filtering with very large tool databases. The default selection is "None". *The user selects the first entry, "None".*

Screen 13: *The user taps on the spinner for "Clamping".* A four-level scale is provided for the clamping option input. The default selection is "Normal". *The user selects the default entry, "Normal".*

Screen 14: *The user taps on the spinner for "Operation type".* A three-level scale is provided to regulate the aggressiveness of machining. The terms, "Rough" milling and "Finish" milling, are well known by anyone in the machining space, and the option "General", represents a throughput lying between the former two. The default selection is "General". *The user selects the default entry, "General".*

Screens 15 and 16 are shown in Figure 14 below.

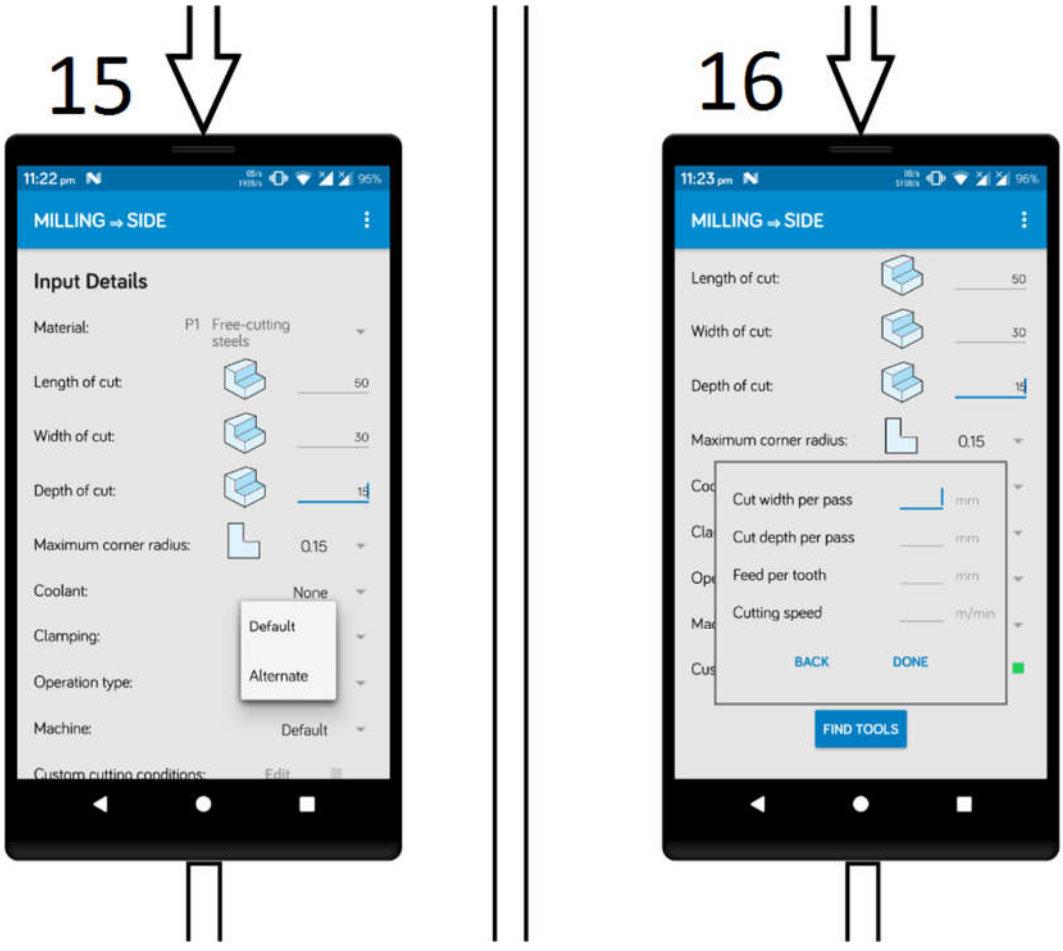


Figure 14: Screens 15 and 16

Screen 15: *The user taps on the spinner for "Machine".* For this input, the user selects one of the personal machines entered in the "Machine management" page, accessible from the three-dot menu in most application pages. A "Default"

machine is present in the application by default, and an "Alternate" machine was added on the test device, the details of which can be seen in Screen 3. *The user selects the default entry, "Default".*

Screen 16: *The user taps on the switch for "Custom cutting conditions".* With the rest of the input data, a cutting tool recommendation can be done, using recommended cutting parameters from the manufacturer, that are stored in the application's database. If the user desires to override the recommended cutting data, the "Custom cutting conditions" popup allows that capability. If the user desires to change the preferred cutting parameters, the "Edit" button from the input page (Screen 4) expands the cutting conditions window. *The user clicks "Back".*

With all of inputs selected, Screen 17 in figure below, shows the final view of the input page in this use case.

Figure 15: Screen 17

Screen 17: With the desired machining task defined by the input parameters, the next step is to filter the tool catalogue for usable tools. *The user taps the "Find Tools" button.*

Screen 18 in Figure 16 below shows the next main activity of the application: the filtered tools page.

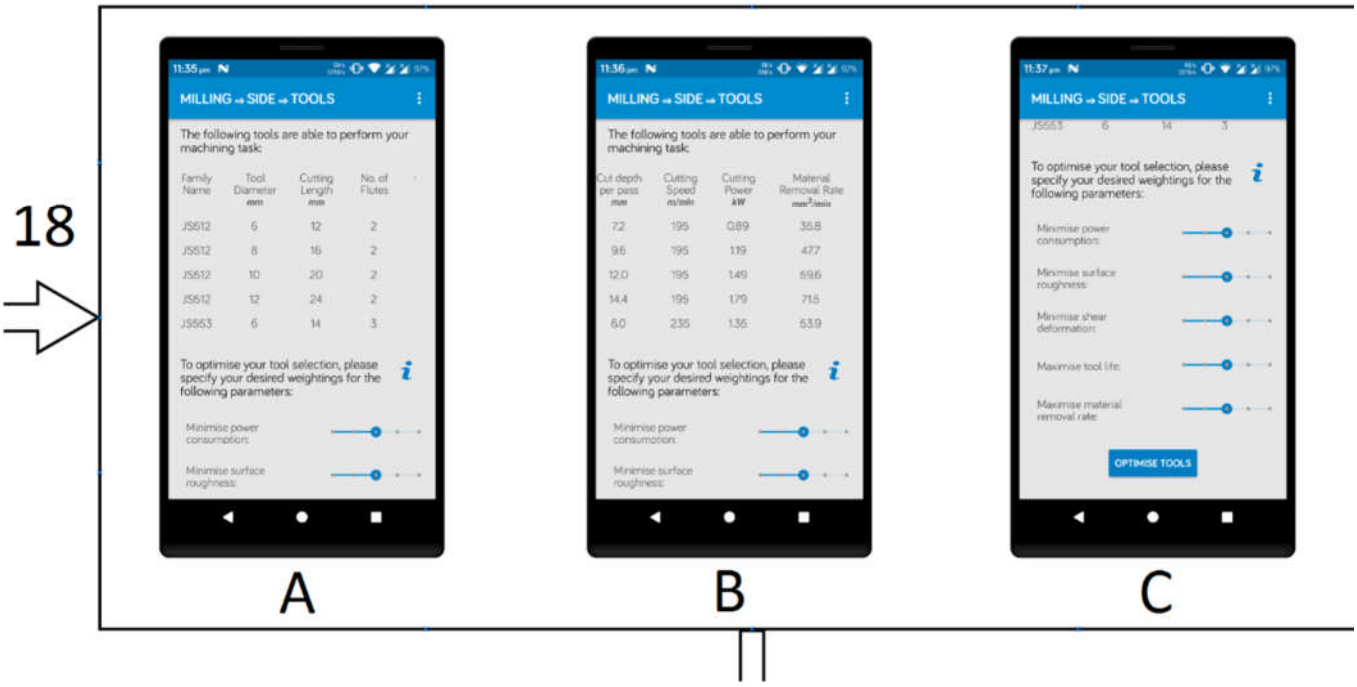


Figure 16: Screen 18

Screen 18: The filtered tools page has two main sections: the list of usable tools, and input sliders for criteria weightings. Part A and B show the tool list clearly, with the table being scrolled to the right in Part B to show additional columns. Part C is shown when the whole page is scrolled down, making visible all of the weighting sliders. For the tool list, a variety of identifying data and cutting parameters are shown, just to provide the basic tool information at a glance. For the TOPSIS MCDM algorithm, criteria weightings are necessary to proceed to optimise the tool selection. Sliders, or seekbars, are used as the input mechanism, due to the ease of use. A range of 0 to 4 is available for each slider, with the default being 2, resulting in each criteria playing and equal role in the decision making process. If the user prefers one criteria to be more emphasised in the tool recommendations, the corresponding slider can be easily increased.

One of the interactive elements present in the filtered tools page (Screen 18), is the blue “i" in the center right of the activity. This represents an information button, and when clicked, Screen 19 in Figure 17 below shows the information window that appears.

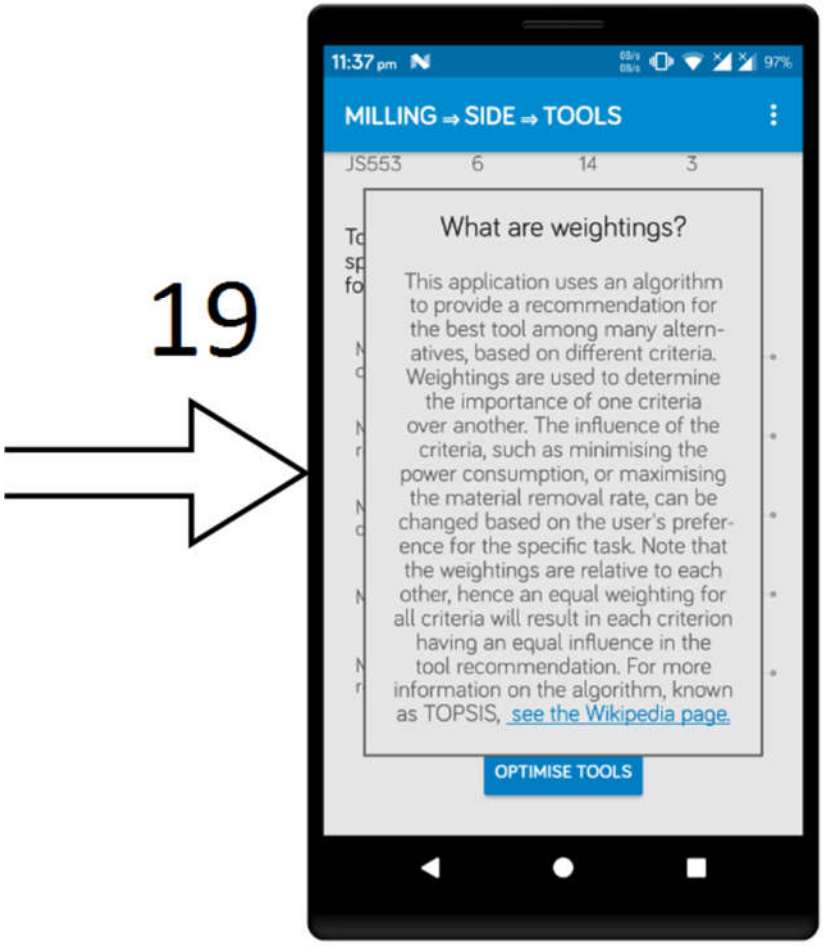


Figure 17: Screen 19

Screen 19: A short description is given to clarify the idea of “weightings” used in the application. If the user is interested in reading about the MCDM algorithm used, there is a clickable link for more information. *The user clicks outside the popup window, dismissing it.*

Once dismissing the information popup, the underlying filtered tools page becomes visible. Screen 20 in Figure 18 below, shows the final view of the filtered tools activity before tool optimisation takes place. *The user moves the slider for “Maximise tool life” to the highest position.*

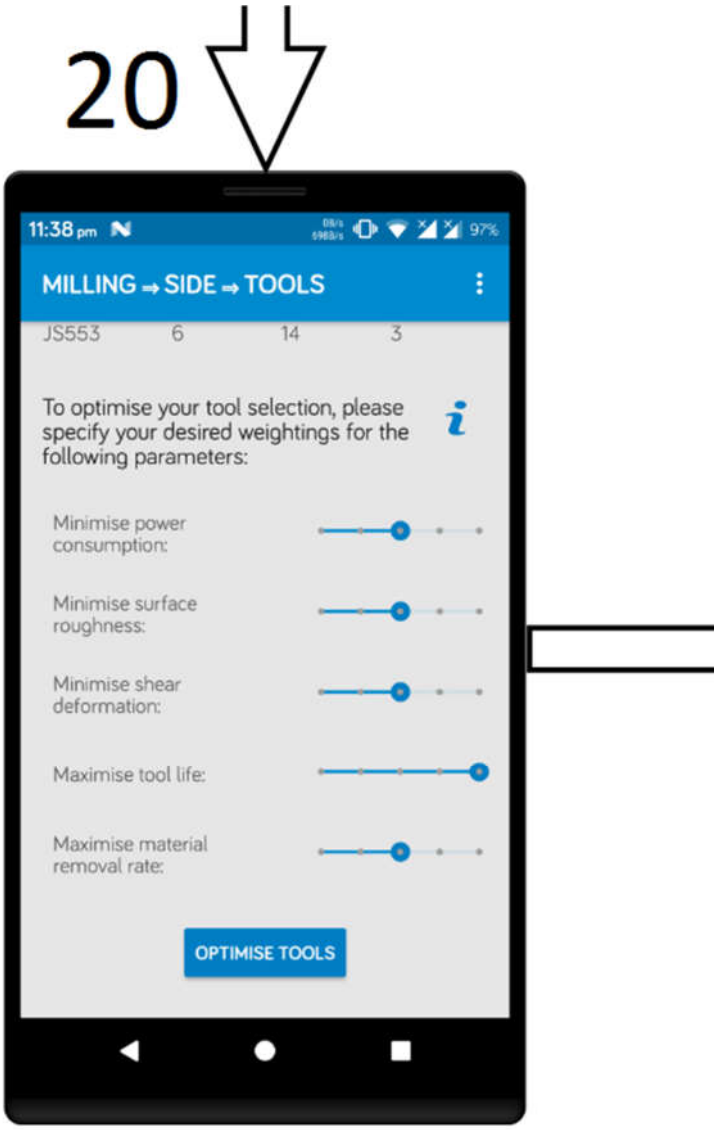


Figure 18: Screen 20

Screen 20: In this case, the user in this use-case desires to maximise the tool life, without sacrificing much in terms of the other criteria. *The user clicks the "Optimise Tools" button.*

After the "Optimise Tools" button is clicked, the application performs the necessary computations, and displays the final result in Screen 21 in Figure 19 below.

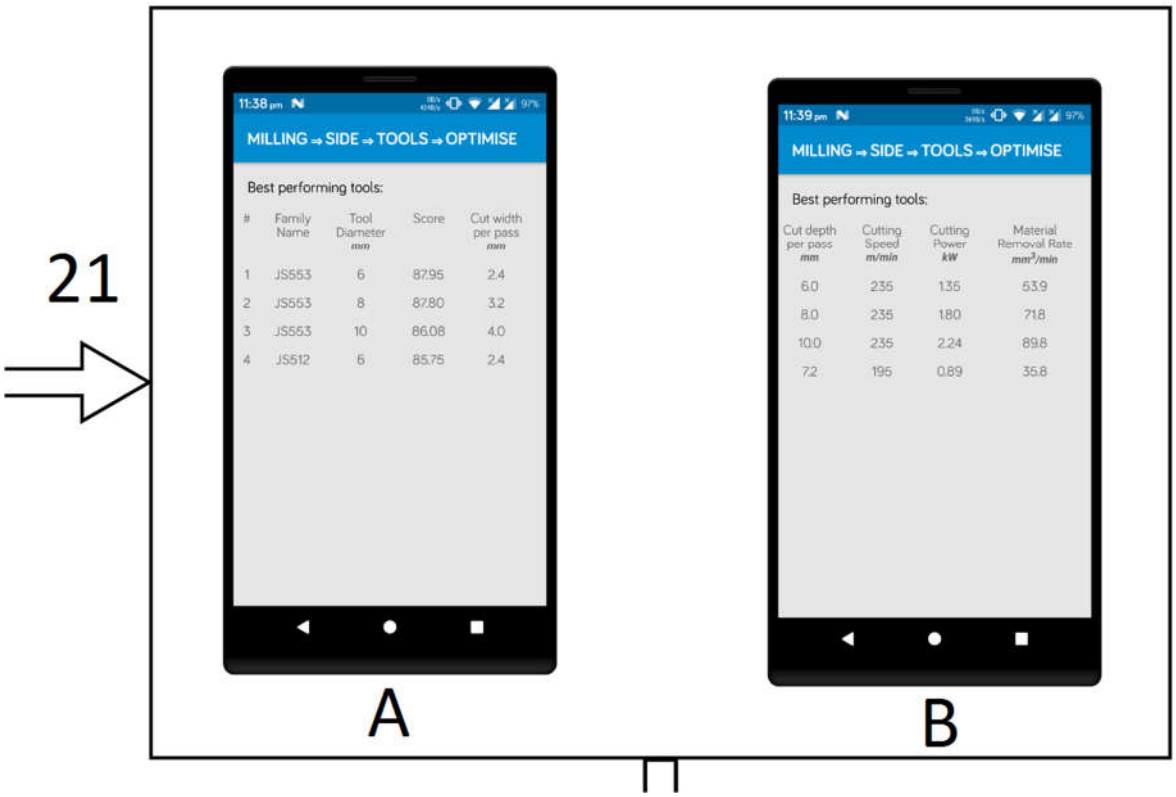


Figure 19: Screen 21

Screen 21: In this activity, initially only a list of the four best tools is visible. By scrolling horizontally, all of the columns of the list can be seen, as shown in Parts A and B. Note the "Score" column, which denotes the internal algorithm score that ranks the tools after the algorithm runs on every tool from the previous activity (Screen 18).

To see additional details about the recommended tools (for example, in order to purchase a tool), the user is required to simply click the respective tool in the list. The details are shown under the tool list, as shown in Screens 22 and 23 in Figure 20 below.

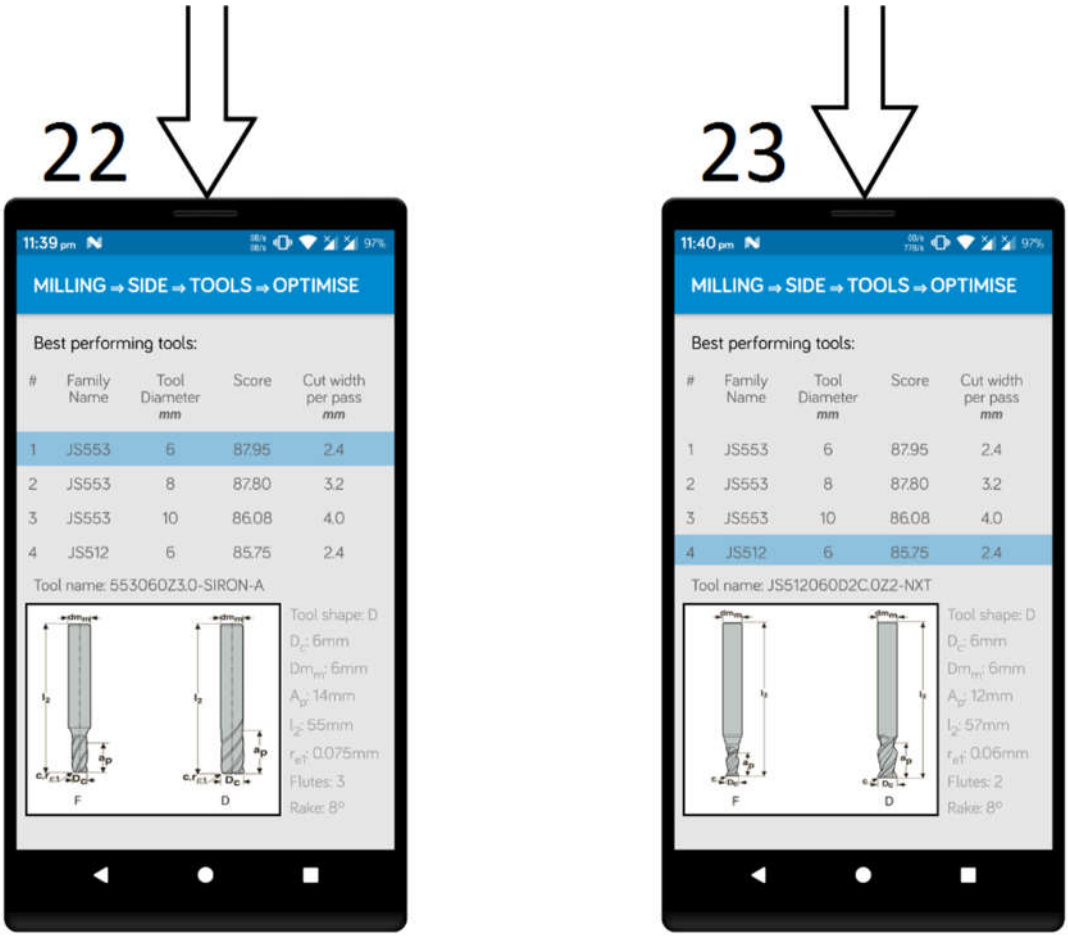


Figure 20: Screens 22 and 23

Screen 22: *The user selects the first tool in the list.* The first entry in the list represents the tool that obtained the highest overall score among the feasible tool alternatives. The unique tool name, tool diagram, as well as a variety of dimensions are visible in the details panel that appears. The dimensions can be referred to from the diagram of the tool, which had been adapted from the manufacturer's catalogue. With the tool name, the user can find a supplier to purchase from.

Screen 23: *The user selects the fourth tool in the list.* The fourth tool is of a different family than the other three tool in the list. Hence, the picture that has been loaded from the database, as well as the associated details, have changed to match the selected tool's details.

With this screen flow demonstration, the basic functionality and appearance of the developed application is known. In the following section (3.3), the technical elements and code that built the application will be discussed.

3.3. Implementation

For this application, the core programming is in java, the visual interface is xml-based, the database system is SQL, and graphics editing software was used to edit the visual elements. Due to the technical nature of the application construction, the fundamental concepts of the android development environment and project structure are referred to extensively.

This section will discuss the methods used and decisions taken with regards to creating the functionality required to address the thesis problem (Section xx) and the user interface that makes the functionality useful to people. For clarity, the implementation is discussed as follows: first, descriptions are provided for a few application-wide elements. These modules are used several times each in the application. Secondly, each of the four distinct activities in the application is discussed, as well as the machine management activity. A screenshot of the relevant activity is provided first, and has various elements labelled. These elements represent the unique aspects of the application and will be discussed in turn after the screenshot. Any other elements seen in the contents of the screen, are slightly different duplications of one of the labelled elements. Additional figures will be provided for each of the unique elements when additional clarity is required.

3.3.1. Application structure

The file structure for the application is shown in Figure 21 below. Each of the files listed are used within at least one of the five activities that are discussed further in this section. All of the programming code is stored in the `java` folder. The embedded database file is seen in the `assets` folder. The `res` directory, for resources, store pictures in the `drawable` folder. All visual layouts are stored in the `layout` folder, while static data is stored in the `values` folder.

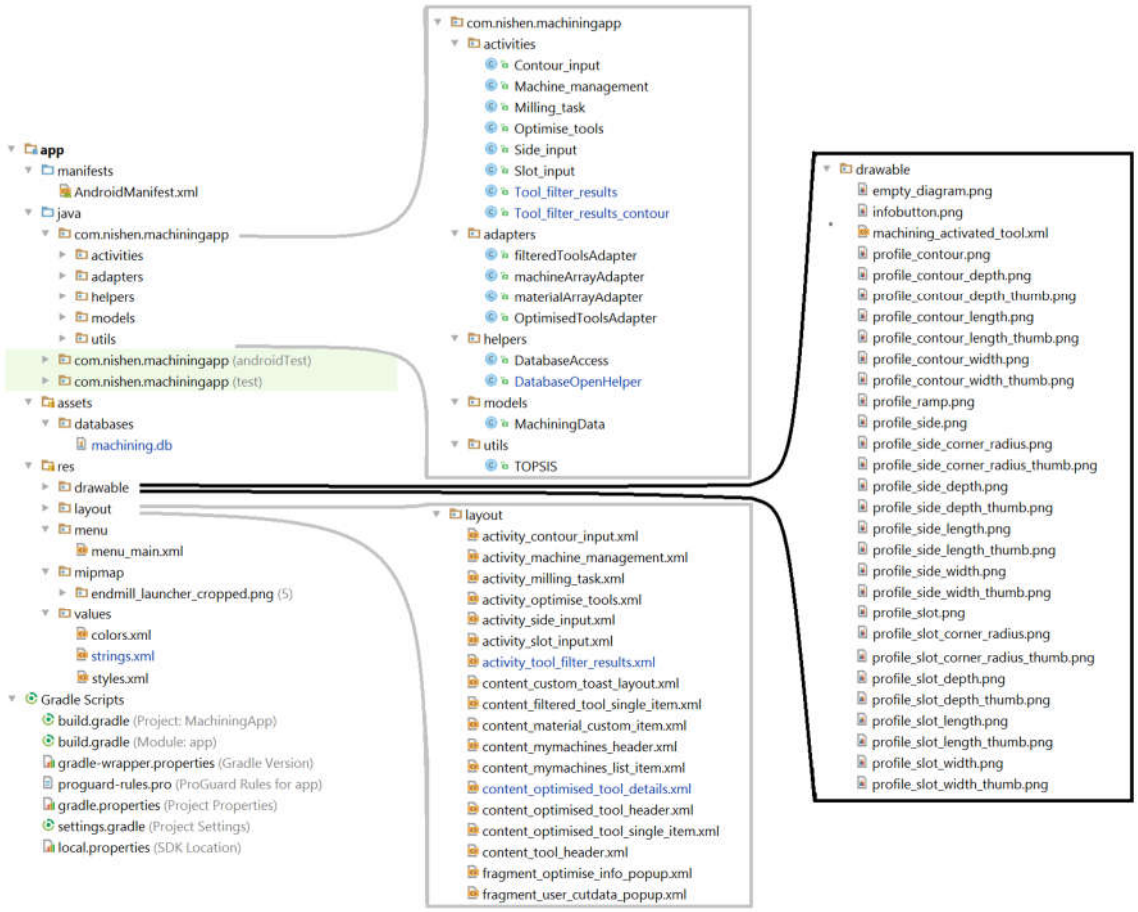


Figure 21: Project structure

3.3.2. SQL database

This application stores all static data in a SQL database. The database consists of five tables, storing the following data: workpiece materials, user machines, cutting tools, cutting data as well as diagrams of the tools.

This application is intended to utilise a cloud-based database or server, so that updates to the database can benefit the users in real-time. Also, if the tool database grows very large, the computations and optimisation will have to be run by the cloud server, requiring very high performance for a large user base. For the purpose of developing this application within a limited time frame, the fundamental application-database interaction is accomplished using a SQLite database. A SQLite database is embedded within the application, and has the same storage limitation as regular SQL databases (140 terabytes). However, the device's memory size limits the amount of data that can be retrieved from a query. This type of SQL database management was used for the following reasons:

- An Android device will be able to handle many more tools in the database than this initial release, while being within the SQLite limitations on android.
- No internet access is required. All the functions of the application can be used offline, therefore not limited by, for example, a weak phone or Wi-Fi signal, or other numerous

network connectivity issues. Constantly requiring a reliable internet connection to develop, test and debug the application is a significant impediment.

- In order for a regular SQL database to be accessible over the internet, a server must be hosted, requiring a monthly fee. Configuring a SQL server on a machine in a Local Area Network (LAN) is possible (for example, in a university's network), however the application will only be able to run when the device is connected to the same network as the server. It may also be possible to configure a public-facing address for the server to be accessed over the general internet, but advanced access to the routers/switches/hubs will be required, with no guarantee of working.
- SQLite is supported natively in Android with existing java classes present in Android Studio.

3.3.2.1. Approach

In order to utilise a SQLite database within an Android application, a database helper class must be used to manage database creation, read, update and delete (CRUD) operations. The native helper class present in Android is the `SQLiteOpenHelper` class (`android.database.sqlite.SQLiteOpenHelper`). A custom database helper will extend this basic (limited functions to directly access) `SQLiteOpenHelper` to easily offer CRUD functionality. In this application, a custom database helper, called `DatabaseOpenHelper` is used to open the database, while a `DatabaseAccess` class is used for CRUD operations. These two classes use the base functionality provide by an externally developed database helper, called `SQLiteAssetHelper`, by Jeff Gilfelt [ref].

The application's `DatabaseOpenHelper` class, which extends the SQLiteAssetHelper class, is shown below.

```
1.  package com.nishen.machiningapp.helpers;
2.
3.  import android.content.Context;
4.
5.  import com.readystatesoftware.sqliteasset.SQLiteAssetHelper;
6.
7.  public class DatabaseOpenHelper extends SQLiteAssetHelper {
8.      private static final String DATABASE_NAME = "machining.db";
9.      private static final int DATABASE_VERSION = 1;
10.
11.     public DatabaseOpenHelper(Context context) {
12.         super(context, DATABASE_NAME, null, DATABASE_VERSION);
13.     }
14. }
```

As can be seen from the code, `DatabaseOpenHelper` serves to point the real helper class (`SQLiteAssetHelper`) to the correct database (`machining.db`) and version (for updating). The external `SQLiteAssetHelper` class is imported into the project by adding it as a dependency in the Gradle build system's `build.gradle` file, as follows:

```
1.  dependencies {
2.      compile 'com.readystatesoftware.sqliteasset:sqliteassethelper:+'
3.  }
```

The `DatabaseAccess` class allows the application to access the embedded SQLite database either through various `SQLiteOpenHelper` functions or with direct SQL queries to the database. A condensed excerpt of `DatabaseAccess.java` is shown below.

```
1.  package com.nishen.machiningapp.helpers;
2.
3.  import android.database.sqlite.SQLiteDatabase;
4.  import android.database.sqlite.SQLiteOpenHelper;
5.
6.  public class DatabaseAccess {
7.      private SQLiteOpenHelper openHelper;
8.      private SQLiteDatabase database;
9.      private static DatabaseAccess instance;
10.
11.     /**
12.      * Private constructor to avoid object creation from outside classes.
13.      */
14.     private DatabaseAccess(Context context) {
15.         this.openHelper = new DatabaseOpenHelper(context);
16.     }
17.
18.     /**
19.      * Return a singleton instance of DatabaseAccess.
20.      */
21.     public static DatabaseAccess getInstance(Context context) {
22.         if (instance == null) {
23.             instance = new DatabaseAccess(context);
24.         }
25.         return instance;
26.     }
27.
28.     /**
29.      * Open the database connection.
30.      */
31.     public void open() {
32.         this.database = openHelper.getWritableDatabase();
33.     }
34.
35.     /**
36.      * Close the database connection.
37.      */
38.     public void close() {
39.         if (database != null) {
40.             this.database.close();
41.         }
42.     }
```

Note the `open()` method that obtains read/write permissions to the database, and the `close()` method that closes the database connection, preventing unwanted access. In order to query the database, the `rawquery(String)` method is used, an example of which is shown below.

```
1.  public Cursor getMaterialsCursor() {
2.          Cursor materialsCursor = database.rawQuery("SELECT SMG, Descrip-
    tion FROM Material", null);
3.          return materialsCursor;
4.
5.      }
```

In this case, the result set of the query is stored in memory as a cursor. The data in this cursor can be utilised at a later time. The rest of the documentation on the application's SQLite implementation is discussed as the relevant elements are explored in the respective activities further in this section.

3.3.3. Global data class

The standard practice in Android to transfer data between activities is through the use of `Bundles` when using `Intents`. However, when storing data over a long time, and between several activities, a `Bundle` is limited. Given the large amount of data this application works with, with regards to the tool result set, consisting of 14 columns for each tool, a robust persistent storage mechanism is needed. For this purpose, a global-access data storage class had been developed. Two variables, and the associated methods are shown in the global data class, `MachiningData`, below.

```
1.  public class MachiningData extends Application {
2.      private String selectedMaterial;
3.      private ArrayList<HashMap<String, String>> ToolList;
4.
5.      public String getSelectedMaterial() {
6.          return selectedMaterial;
7.      }
8.      public void setSelectedMaterial(String selectedMaterial) {
9.          this.selectedMaterial = selectedMaterial;
10.     }
11.
12.     public ArrayList<HashMap<String, String>> getToolList() {
13.         return ToolList;
14.     }
15.     public void setToolList(ArrayList<HashMap<String, String>> toolList) {
16.         ToolList = toolList;
17.     }
18. }
```

As can be seen from the excerpt, the `get-` and `set-` methods allow the global variables to be read and overwritten, respectively. The `MachiningData` class serves the purposes of the application perfectly, as it allows the variables within to be accessed and written at any time, from any class in the application.

3.3.4. Milling task activity

The `Milling_task` activity is the first screen that greets the user when the application is opened, and a labelled picture of the screen is shown in Figure 22 below.

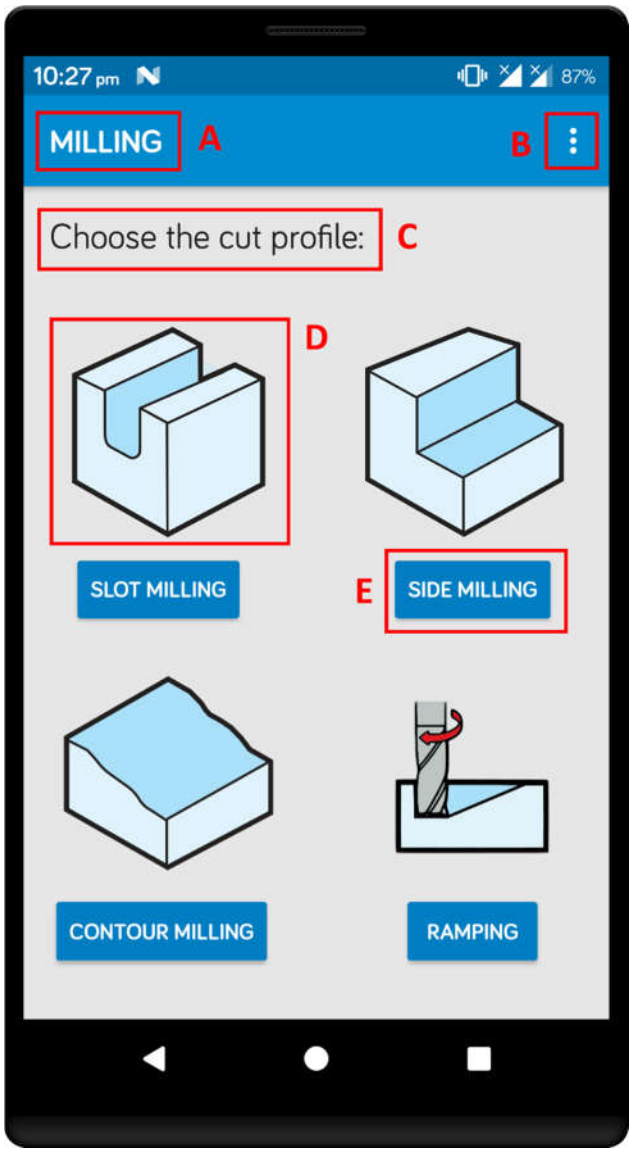


Figure 22: Labelled milling task screen

3.3.4.1. A – Action bar text

The action bar title text can be set manually in the activity's java file, under the `onCreate()` method, which performs all the contained actions when the activity is instantiated. Seen on line 6 below, the setTitle() method sets the text. This method is used in subsequent activities to set the title text.

```
1.  public class Milling_task extends AppCompatActivity {
2.
3.      @Override
4.      protected void onCreate(Bundle savedInstanceState) {
5.          super.onCreate(savedInstanceState);
6.          setTitle("Milling");
7.          setContentView(R.layout.activity_milling_task);
8.  }
```

3.3.4.2. B – Action bar menu

The menu button is automatically created when the activity contains an `onCreateOptionsMenu()` method, implemented as follows:

```
1.  public class Milling_task extends AppCompatActivity {
2.  @Override
3.      public boolean onCreateOptionsMenu(Menu menu) {
4.          // Inflate the menu; this adds items to the action bar if it is present.
5.          getMenuInflater().inflate(R.menu.menu_main, menu);//Menu Resource, Menu
6.          return true;
7.      }
8.
9.      @Override
10.     public boolean onOptionsItemSelected(MenuItem item) {
11.         switch (item.getItemId()) {
12.             case R.id.item1:
13.                 Intent intent = new Intent(this, Machine_management.class);
14.                 startActivity(intent);
15.                 return true;
16.             default:
17.                 return super.onOptionsItemSelected(item);
18.         }
19.     }
20. }
```

The `onOptionsItemSelected()` method is used to program the actions for each of the menu items defined in `menu_main.xml`, the xml layout file for the menu:

```
1.  <menu xmlns:android="http://schemas.android.com/apk/res/android" >
2.      <item  android:id="@+id/item1"
3.          android:title="Machine management"/>
4.  </menu>
```

3.3.4.3. C – Simple TextView

A TextView is used for simple text display. XML-based, this particular TextView has been configured within the `activity_milling_task.xml` layout file as follows:

```
1.  <TextView
2.              android:id="@+id/textView"
3.              android:layout_width="368dp"
4.              android:layout_height="38dp"
5.              android:layout_marginLeft="16dp"
6.              android:layout_marginStart="16dp"
7.              android:layout_marginTop="16dp"
8.              android:text="@string/profile_question"
9.              android:textAppearance="@android:style/TextAppearance.Material.Large"
10.             app:layout_constraintLeft_toLeftOf="parent"
11.             app:layout_constraintTop_toTopOf="parent" />
```

Many of the properties set are related to the position and size of the element. Note that the `android:text` property is set to `"@string/profile_question"`, referring to a value in `strings.xml`, as shown below.

```
1.  <resources>
2.      <string name="profile_question">Choose the cut profile:</string>
3.  </resources>
```

Throughout the application, TextViews are implemented in this way.

3.3.4.4. D – Image display

Images are loaded statically through an ImageView element. XML-based, the source image is read from the `drawable` folder in the project structure. The xml is shown below:

```
<ImageView
            android:id="@+id/imageView2"
            android:layout_width="130dp"
            android:layout_height="130dp"
            android:layout_marginLeft="24dp"
            android:layout_marginStart="24dp"
            android:layout_marginTop="32dp"
            android:onClick="slotbutton"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/textView"
            app:srcCompat="@drawable/profile_slot" />
```

All static image loading in the application is performed in this way.

3.3.4.5. E – Profile button

Most buttons contain both a visual (xml) aspect, and a programmatic (java) aspect. The button is defined in xml in the following way:

```
<Button
    android:id="@+id/button3"
    style="@android:style/Widget.Material.Button.Colored"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginRight="32dp"
    android:layout_marginTop="8dp"
    android:onClick="sidebutton"
    android:text="@string/side_milling"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/imageView3"
    tools:layout_constraintLeft_creator="1" />
```

Note that the `android:onClick` property is set to "`sidebutton`". This refers to the sidebutton() method present in the `Milling_task` activity, which serves to start the input activity for the corresponding cut profile. The `setProfile()` from `MachiningData.java` is used to set the global profile variable.

```
public class Milling_task extends AppCompatActivity {
    public void sidebutton(View view) {
        // Do something in response to button
        Intent intent = new Intent(this, Side_input.class);
        ((MachiningData)getApplicationContext()).setPro-
file("Side"); //set global Profile variable
        startActivity(intent);
    }
}
```

All buttons in the application are implemented in a similar way.

3.3.5. Input details activity

After selecting a desired cut profile, the application details are captured in the input details activity, in this case, `Side_input.java`. The various elements in this page is highlighted in Figure 23 below.

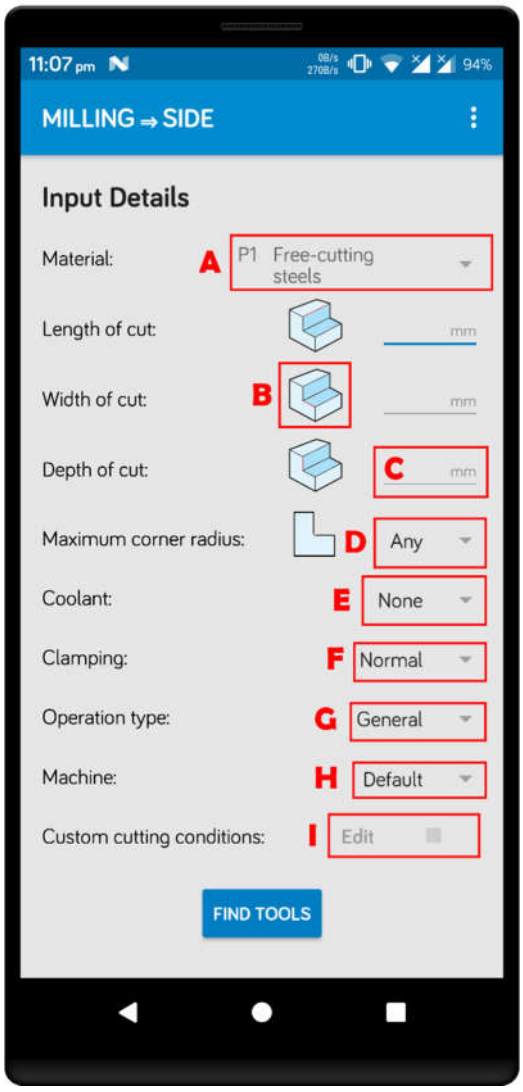


Figure 23: Labelled input details screen

3.3.5.1. A – Material spinner

The spinner element is simply defined and located on the xml layout, while the contents are populated programmatically. Being a multi-item data display element, a custom adapter class is needed to display the data. Loading the data from the database is done as follows:

```
materialList = new ArrayList<HashMap<String, String>>();
material_spinner = (Spinner)findViewById(R.id.material_spinner);
DatabaseAccess databaseAccess = DatabaseAccess.getInstance(getApplicationCon-
text());
databaseAccess.open();
Cursor materials =databaseAccess.getMaterialsCursor();
materials.moveToFirst();
while (!materials.isAfterLast()) {
    String SMG = materials.getString(0);
    String Description = materials.getString(1);
    HashMap<String, String> material = new HashMap<>();
    //add each value to temporary hashmap
    material.put("SMG", SMG);
    material.put("Description", Description);
    //add material to materialList
    materialList.add(material);
    materials.moveToNext();
}
materials.close();
```

Every time the SQLite database needs to be accessed, the same procedure is followed: a `DatabaseAccess` object is created, the query method is run, returning a result set in some form. The results are then formatted into a storable list. The `getMaterialsCursor()` method, which returns a `Cursor` object, is specified in `DatabaseAccess` as:

```
public Cursor getMaterialsCursor() {
    Cursor materialsCursor = database.rawQuery("SELECT SMG, Description FROM Mate-
rial", null);
    return materialsCursor;

}
```

Note that the list of materials is stored as an arraylist of hashmaps, or as an `ArrayList<HashMap<String, String>>` object. Hashmaps allow easy data storage with `String` type key-value pairs. The `materialList` is utilised by a custom `materialArrayAdapter` class as follows:

```
materialArrayAdapter adapter = new materialArrayAdapter(Side_input.this, material-
List);
material_spinner.setAdapter(adapter);
adapter.notifyDataSetChanged();
material_spinner.setOnItemSelectedListener(new materialSpinnerListener());
```

As can be seen above, a `materialArrayAdapter` is created based on the `materialList`, and the adapter is assigned to the spinner element. The `setOnItemSelectedListener()` method is used to capture the input from the material spinner, and is discussed further on in this section. The `materialArrayAdapter` class allows the data from the list to be displayed in the spinner, by being mapped to a custom item layout as follows:

```
1.  package com.nishen.machiningapp.adapters;
2.
3.  public class materialArrayAdapter extends BaseAdapter {
4.      public ArrayList<HashMap<String, String>> myMaterialList;
5.      public View getView(int position, View convertView, ViewGroup parent) {
6.          LayoutInflater inflater = activity.getLayoutInflater();
7.          if (convertView == null) {
8.              convertView = inflater.inflate(R.layout.content_material_cus-
    tom_item, null);
9.              viewHolder = new ViewHolder(convertView);
10.             convertView.setTag(viewHolder);
11.         } else {
12.             viewHolder = (ViewHolder) convertView.getTag();
13.         }
14.     // grab temporary material item from material arraylist
15.         HashMap<String, String> map = myMaterialList.get(position);
16.         viewHolder.SMG.setText(map.get("SMG"));
17.         viewHolder.Description.setText(map.get("Description"));
18.         return convertView;
19.     }
20.     private class ViewHolder {
21.         TextView SMG;
22.         TextView Description;
23.
24.         ViewHolder(View view) {
25.             SMG = (TextView) view.findViewById(R.id.SMG);
26.             Description = (TextView) view.findViewById(R.id.Description);
27.         }
28.     }
29. }
```

For compatibility with the `RecyclerView` functionality in Android (Lists are not completely loaded into memory, rather after ~10 items, the view item is recycled with new content), `ViewHolders` are used to store the individual item text. For the two-item custom adapter shown, the custom xml item layout is shown below.

```
1.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.      android:orientation="horizontal"
3.      android:layout_width="match_parent"
4.      android:layout_height="match_parent">
5.
6.      <TextView
7.          android:id="@+id/SMG"
8.          android:layout_width="wrap_content"
9.          android:layout_height="wrap_content"
10.         android:padding="6dp"
11.         android:textColor="@color/machining_dark_gray"
12.         android:textSize="16sp" />
13.
14.     <TextView
15.         android:id="@+id/Description"
16.         android:layout_width="wrap_content"
17.         android:layout_height="wrap_content"
18.         android:ellipsize="end"
19.         android:maxLines="2"
20.         android:padding="6dp"
21.         android:textColor="@color/machining_dark_gray"
22.         android:textSize="16sp" />
23.
24. </LinearLayout>
```

This configuration results in the dropdown list appearing like the image shown in Figure 24 below.

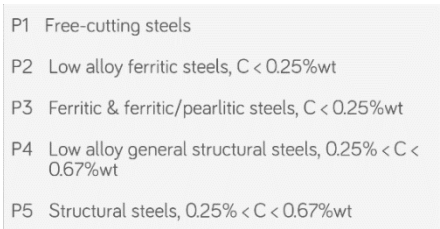


Figure 24: Material dropdown list

To capture the input, a `SpinnerListener` class is used:

```
1.  private class materialSpinnerListener implements OnItemSelectedListener {
2.      public void onItemSelected(AdapterView<?> par-
    ent, View view, int pos, long id) {
3.          String selectedMaterial = new String();
4.          selectedMaterial = String.valueOf(pos + 1);
5.          ((MachiningData)getApplicationContext()).setSelectedMaterial(selectedMate-
    rial);
6.          //          selectedMaterial = parent.getItemAtPosition(pos).toString();
7.      }
8.      public void onNothingSelected(AdapterView parent) {
9.          // Do nothing.
10.     }
11. }
```

When a material is selected in the spinner, the `Material` variable in the `MachiningData` class is written with the selection.

3.3.5.2. B – Zoomable image

The tap-to-zoom implementation used in the application is adapted from Google's guidelines [ref]. `ImageButtons` are used for the thumbnail image display. Once the `side_width_imagebutton` is clicked, the `zoomImageFromThumb()` method expands a corresponding bigger picture:

```
1.  final View side_length_view = findViewById(R.id.side_length_imagebutton);
2.  side_length_view.setOnClickListener(new View.OnClickListener() {
3.      @Override
4.      public void onClick(View view) {
5.          zoomImageFromThumb(side_length_view, R.drawable.profile_side_length);
6.      }
7.  });
```

The `zoomImageFromThumb()` method is lengthy, and was merely adapted to work within this application. The code can be found in the Side_input.java class in Appendix B.

Similar implementations are used for the `side_length_imagebutton`, `side_depth_imagebutton` and `side_corner_radius_imagebutton`.

3.3.5.3. C – Dimension textbox

Input textboxes utilise `EditText` elements to capture the input. The `cut_depth EditText` element has the following xml layout:

```
<EditText
    android:id="@+id/cut_depth"
    android:layout_width="77dp"
    android:layout_height="wrap_content"
    android:layout_marginEnd="16dp"
    android:layout_marginRight="16dp"
    android:ems="10"
    android:hint="mm"
    android:inputType="numberDecimal"
    android:textAlignment="textEnd"
    android:textSize="12sp"
    app:layout_constraintBaseline_toBaselineOf="@+id/textView5"
    app:layout_constraintRight_toRightOf="parent" />
```

A hint text of **"mm"** has been applied, resulting in the unit being shown when no text has been entered. The `android:inputType` property flags the devices keyboard to open to a numeric input mode when the `cut_depth` element is selected. The input data is stored in the global MachiningData class when the "Find Tools" button is pressed, after all the input have been specified, as shown below.

```
EditText CutDepth = (EditText)findViewById(R.id.cut_depth);
String cut_depth = CutDepth.getText().toString();
((MachiningData)getApplicationContext()).setCutDepth(cut_depth);
```

The `getText()` method is called on the `cut_depth` element, to transfer the input to a string. All the cut dimension `EditText` elements are processed in this way.

3.3.5.4. D – Corner radius spinner

The corner radius spinner is populated from the SQLite database, so a `DatabaseAccess` object must be created to obtain the result set needed. However, as only one column of data is requested in this case, the native `ArrayAdapter` in Android is sufficient. An `ArrayAdapter` is able to take a `List` object and populate a data element, such as a `Spinner`. The `List` for the corner radius spinner is obtained from the `DatabaseAccess` method, `unique_corner_radius()`, in the following way:

```
1.  public List<String> unique_corner_radius() {
2.      List<String> corner_radius_list = new ArrayList<>();
3.      Cursor cursor = database.rawQuery("SELECT DISTINCT re1 FROM Tool OR-
    DER BY re1 DESC", null);
4.      cursor.moveToFirst();
5.      while (!cursor.isAfterLast()) {
6.          corner_radius_list.add(cursor.getString(0));
7.          cursor.moveToNext();
8.      }
9.      cursor.close();
10.     return corner_radius_list;
11. }
```

The `moveToFirst()` method in line 4 is used to point the cursor selection to the first index. The while loop adds each cursor item (a corner radius value), to the `List` to `return`. This result set populates the corner radius spinner as shown below.

```
1.  cnr_radius_spinner = (Spinner) findViewById(R.id.corner_radius_spinner);
2.  DatabaseAccess cnr_radius_db = DatabaseAccess.getInstance(getApplicationCon-
    text());
3.  cnr_radius_db.open();
4.  List<String> cnr_radius_list = cnr_radius_db.unique_corner_radius();
5.  cnr_radius_list.add(0, "Any");
6.  cnr_radius_db.close();
7.  ArrayAdapter<String> cnr_radius_adapter = new ArrayAdapter<String>(Side_in-
    put.this, android.R.layout.simple_spinner_item, cnr_radius_list);
8.  cnr_radius_adapter.setDropDownViewResource(android.R.layout.simple_spin-
    ner_dropdown_item);
9.  cnr_radius_spinner.setAdapter(cnr_radius_adapter);
10. cnr_radius_spinner.setOnItemSelectedListener(new cornerRadiusSpinnerListener());
```

As shown in line 5, the option **"Any"**, is added manually at this point in the first position in the `List`. The `cornerRadiusSpinnerListener` class that captures the input data is similar to the `materialSpinnerListener`, and is shown below.

```
1.  private class cornerRadiusSpinnerListener implements OnItemSelectedListener {
2.      public void onItemSelected(AdapterView<?> par-
    ent, View view, int pos, long id) {
3.          String selectedCornerRadius = new String();
4.          selectedCornerRadius = parent.getItemAtPosition(pos).toString();
5.          ((MachiningData)getApplicationContext()).setCornerRadius(selectedCornerRa-
    dius);
6.      }
7.      public void onNothingSelected(AdapterView parent) {
8.          // Do nothing.
9.      }
10. }
```

3.3.5.5. E – Coolant spinner

The options presented for the coolant spinner element is static, and stored as a string-array in strings.xml as shown below.

```
<resources>
    <string-array name="coolant_list">
        <item>None</item>
        <item>Air</item>
        <item>Emulsion</item>
        <item>Mist Spray</item>
    </string-array>
<resources>
```

The list above is used to populate the `coolant_spinner` through the use of an `ArrayAdapter` as follows:

```
coolant_spinner = (Spinner)findViewById(R.id.coolant_spinner);
String[] coolantArray = getResources().getStringArray(R.array.coolant_list);
ArrayList<String> coolant_List = new ArrayList<String>(Arrays.asList(coolantAr-
ray));
ArrayAdapter<String> coolant_adapter = new ArrayAdapter<String>(this, an-
droid.R.layout.simple_spinner_item, coolant_List);
coolant_adapter.setDropDownViewResource(android.R.layout.simple_spin-
ner_dropdown_item);
coolant_spinner.setAdapter(coolant_adapter);
coolant_spinner.setOnItemSelectedListener(new coolantSpinnerListener());
```

The `coolantSpinnerListener` has been implemented in a similar way as the `cornerRadiusSpinnerListener, or materialSpinnerListener.`

3.3.5.6. F – Clamping spinner

The clamping spinner is implemented in the same way as the coolant spinner, with the following string-array providing the options:

```
<resources>
    <string-array name="clamping_list">
        <item>Unstable</item>
        <item>Normal</item>
        <item>Stable</item>
        <item>Very Stable</item>
    </string-array>
<resources>
```

3.3.5.7. G – Operation type spinner

The clamping spinner is implemented in the same way as the coolant spinner, with the following string-array providing the options:

```
<resources>
    <string-array name="operation_type_list">
        <item>Rough</item>
        <item>General</item>
        <item>Finish</item>
    </string-array>
<resources>
```

The default selection is set programmatically in the `Side_input` class, as follows:

```
1.  public class Side_input extends AppCompatActivity implements AdapterView.OnItem-
    SelectedListener {
2.      @Override
3.      protected void onCreate(Bundle savedInstanceState) {
4.          operation_type_spinner = (Spinner)findViewById(R.id.operation_type_spin-
    ner);
5.          operation_type_spinner.setSelection(1);
6.      }
7.  }
```

3.3.5.8. H – Machine spinner

The machine spinner is implemented similarly to the corner radius spinner, with the following SQLite query method in `DatabaseAccess.java`:

```
1.  public List<String> getmachines() {
2.      List<String> machine_list = new ArrayList<>();
3.      Cursor cursor = database.rawQuery("SELECT Name FROM Machine OR-
    DER BY ID ASC", null);
4.
5.      cursor.moveToFirst();
6.      while (!cursor.isAfterLast()) {
7.          machine_list.add(cursor.getString(0));
8.          cursor.moveToNext();
9.      }
10.     cursor.close();
11.     return machine_list;
12. }
```

3.3.5.9. I – Custom cutting conditions popup

Two elements are present in `Side_input` for interaction with the custom cutting conditions window. There is the "`Edit`" `TextView`, and a `Switch`. Simple xml implementations exist for the layout, while the main functionality is done programmatically, as shown below.

```
1.  user_cutdata_input_edit = (TextView) findViewById(R.id.EditUserCutData);
2.  user_cutdata_input_edit.setOnClickListener(new View.OnClickListener() {
3.      @Override
4.      public void onClick(View v) {
5.          UserCutdataWindow(true);
6.      }
7.  });
8.
9.  user_cutdata_switch = (Switch) findViewById(R.id.UserCutDataSwitch);
10. user_cutdata_switch.setOnCheckedChangeListener(new CompoundButton.OnCheckedChange-
    Listener() {
11.     @Override
12.     public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
13.         if (isChecked){
14.             user_cutdata_in-
    put_edit.setTextColor(getResources().getColor(R.color.colorPrimary));
15.             UserCutdataWindow(false);
16.         } else {
17.             user_cutdata_input_edit.setTextColor(getResources().getColor(an-
    droid.R.color.darker_gray));
18.         }
19.     }
20. });
```

The `EditUserCutData` `EditText` has an `OnClickListener` set to open the custom cutting conditions popup window through the `UserCutdataWindow()` method. The `UserCutDataSwitch` also opens the `UserCutDataWindow`, and is programmed to change

the colour of the `EditUserCutData EditText` depending on the status of the `Switch` (checked or not). The `UserCutdataWindow()` method inflates a popup layout file at a specified location in the following way:

```
1.  public void UserCutdataWindow(boolean Edit){
2.          try {
3.              // get a reference to the already created main layout
4.              final ScrollView mainLayout = (ScrollView) findViewById(R.id.con-
    tainer);
5.              // inflate the layout of the popup window
6.              LayoutInflater inflater = (LayoutInflater) getSystemService(LAYOUT_IN-
    FLATER_SERVICE);
7.              final View popupView = inflater.inflate(R.layout.fragment_user_cut-
    data_popup, null);
8.              // create the popup window
9.              //boolean focusable = false; // true lets taps out-
    side the popup also dismiss it
10.             final PopupWindow popupWindow = new PopupWin-
    dow(popupView, 1, 1, true);
11.             popupWindow.setWidth(850);
12.             popupWindow.setHeight(ViewGroup.LayoutParams.WRAP_CONTENT);
13.             popupWindow.setBackgroundDrawable(new ColorDrawable(Color.WHITE));
14.             // show the popup window
15.             popupWindow.showAtLocation(mainLayout, Gravity.CENTER, 0, 250);
16.         }
17. }
```

The layout file, fragment_user_cutdata_popup.xml, has TextView elements for text display, EditText elements to capture user input and Buttons for interaction. The element tree and graphical output is shown in Figure 25 below.

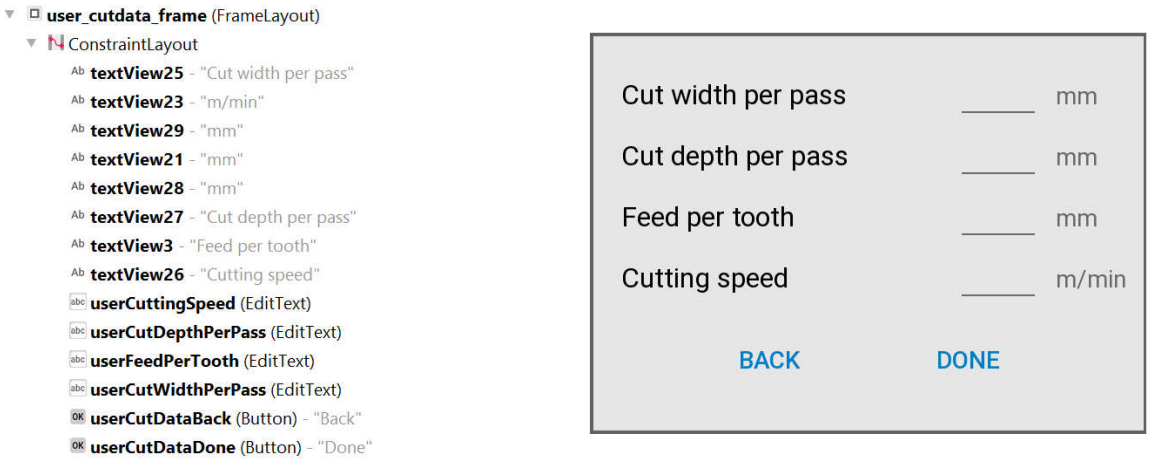


Figure 25: User cutting conditions window

In order to capture the user specified cut data, the code shown below is present under the `UserCutdataWindow()` method.

```
1.      public void UserCutdataWindow(boolean Edit){
2.              final EditText CutWidth = (EditText)popupView.findViewById(R.id.us-
    erCutWidthPerPass);
3.              /***3 other EditText here***/
4.              Button Done = (Button)popupView.findViewById(R.id.userCutDataDone);
5.              Done.setOnClickListener(new View.OnClickListener() {
6.                  @Override
7.                  public void onClick(View v) {
8.                      if (Cut-
    Width.getText().toString().trim().equals("") | CutDepth.getText().toString().trim()
    .equals("") | CuttingSpeed.getText().toString().trim().equals("") | FeedPer-
    Tooth.getText().toString().trim().equals("")  ) {
9.                          Toast.makeText(getApplicationCon-
    text(), "Please fill in the details", Toast.LENGTH_SHORT).show();
10.                     } else {
11.                         ((MachiningData) getApplicationContext()).setUserCut-
    Width(CutWidth.getText().toString());
12.                         /***Assign all inputs here***/
13.                         InputMethodManager imm = (InputMethodManager)getSystem-
    Service(Context.INPUT_METHOD_SERVICE);
14.                         imm.hideSoftInputFromWindow(popupView.getWindow-
    Token(), 0);
15.                         popupWindow.dismiss();
16.                     }
17.                 }
18.             });
19.             Button Back = (Button)popupView.findViewById(R.id.userCutDataBack);
20.             Back.setOnClickListener(new View.OnClickListener() {
21.                 @Override
22.                 public void onClick(View v) {
23.                     InputMethodManager imm = (InputMethodManager)getSystem-
    Service(Context.INPUT_METHOD_SERVICE);
24.                     imm.hideSoftInputFromWindow(popupView.getWindowToken(), 0);
25.                     popupWindow.dismiss();
26.                     user_cutdata_switch.toggle();
27.                 }
28.             });
29. }
```

The process has been shown for only one of the inputs in the above excerpt. To ensure that all the inputs are filled, an `if` statement is present in line 9, checking if any of the inputs are empty. The `else` block of code will assign the user cutting conditions to global variables in the `MachiningData` class. The "`Back`" button serves to dismiss the popup window, keyboard as well as set the `user_cutdata_switch` to the off-position. In order for the application to know whether to use the custom cutting conditions rather than the manufacturer recommended conditions, the status of the custom cutting conditions `Switch` is stored in a global variable when the "Find Tools" button is clicked to go to the next activity, in the following way:

```
1.   public void searchtools (View view) {
2.         Intent filter_tools = new Intent(getApplicationContext(), Tool_filter_re-
    sults.class);
3.         ((MachiningData)getApplicationContext()).setUserCutDataChecked(user_cut-
    data_switch.isChecked());
4.         startActivity(filter_tools);
5.  }
```

3.3.6. Filtered tools activity

After all the input details for the machining task has been entered, the tool database is filtered for usable tools and displayed as shown in Figure 26 below.

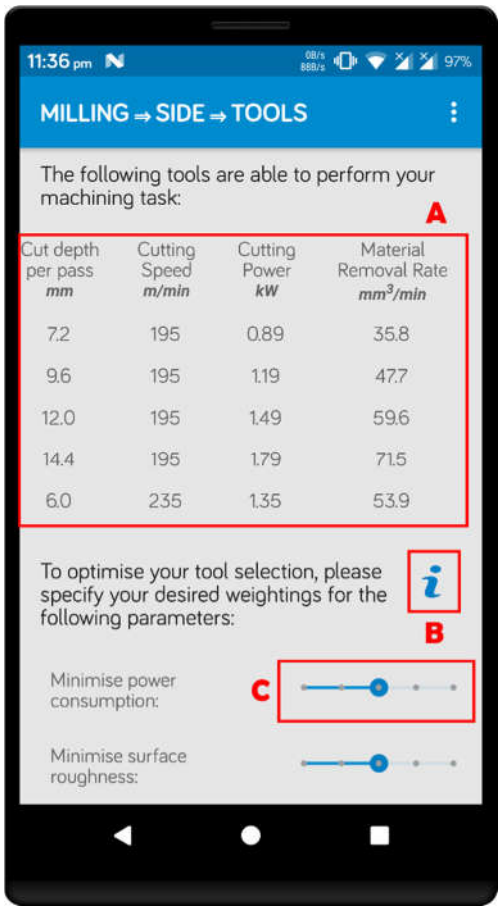


Figure 26: Labelled filtered tools screen

3.3.6.1. A – Filtered tool list

This `ListView` element serves to display the filtered tools and some associated data. The SQL query is performed using the `FilterToolsCursor()` method in `DatabaseAccess`, shown below.

```
1.  public Cursor FilterToolsCursor(String profile, String material) {
2.      Cursor materialCursor = database.rawQuery("SELECT SMG FROM Mate-
    rial WHERE ID =" + material, null);
3.      materialCursor.moveToFirst();
4.      String SMG = materialCursor.getString(0);
5.      Cursor cursor = database.rawQuery("SE-
    LECT Tool.Name, Dc, ap, zn, Part_No, Tool_Shape, dmm, l2, re1, rake, cool-
    ant, \"Ap/Dc\", \"Ae/Dc\", \"6\", \"8\", \"10\", \"12\", Vc FROM Tool, Cut-
    data WHERE Profile LIKE '%" + profile + "%' AND Tool.Mate-
    rial LIKE '%" + SMG + "%' AND Cutdata.Material LIKE '" + SMG + "' AND Cut-
    data.Name = Tool.Name AND Cutdata.Operation LIKE '" + profile + "'", null);
6.      return cursor;
7.  }
```

The large SQL query on line 5 performs the filtering, while the return `Cursor` is utilised to in the `Tool_filter_results` activity as follows:

```
1.  public class Tool_filter_results extends AppCompatActivity {
2.  @Override
3.      protected void onCreate(Bundle savedInstanceState) {
4.          String cut_profile = ((MachiningData)getApplicationContext()).getPro-
    file();
5.          setTitle(Html.fromHtml("Milling<big>⇒</big>" + cut_pro-
    file + "<big>⇒</big>Tools"));
6.          materialID = ((MachiningData)getApplicationContext()).getSelectedMate-
    rial();
7.          /** Populating Tool results listview **/
8.          tool_results_list= (ListView) findViewById(R.id.toolList);
9.          DatabaseAccess tool_search_db = DatabaseAccess.getInstance(this);
10.         tool_search_db.open();
11.         Cursor tool_search_list = tool_search_db.FilterToolsCursor(cut_profile, ma-
    terialID);
12.         tool_search_list.moveToFirst();
13.         while (!tool_search_list.isAfterLast()) {
14.             String Name = tool_search_list.getString(0);
15.             String Diameter = tool_search_list.getString(1);
16.             String CuttingLength = tool_search_list.getString(2);
17.             String FluteNumber = tool_search_list.getString(3);
18.             /**The rest of the column values are captured here**/
19.             String Cutting_speed = tool_search_list.getString(17);
20.        }
21. }
```

It can be seen in lines 4 and 5, that the cut profile selected at the milling task page is retrieved from the global variable, and used to programmatically set the action bar title text. In order to utilise the data retrieved from the database, all of the values are stored in temporary strings in the `while` loop, which iterates one tool at a time. As all of the individual tool information is available within each `while` loop iteration, the optimisation criteria calculations are also done within this `while` loop. Given that calculations in java must be performed on numerical data types, all the required data is converted in double values, which allow computations with the double precision 64-bit IEEE 754 values. The cutting condition value is chosen as shown below, with only the depth of cut per pass being shown.

```
1.  double diameter = Double.parseDouble(Diameter);
2.  double ApDc = Double.parseDouble(Ap_Dc);
3.
4.  double CutDepth;
5.  if (((MachiningData)getApplicationContext()).isUserCutDataChecked()){
6.      CutDepth = Double.parseDouble(((MachiningData)getApplicationContext()).getUser-
    CutDepth());
7.  } else {
8.      CutDepth = ApDc * diameter ;
9.  }
```

The `if` statement checks the stored status of the `user_cutdata_switch` in the previous activity. The other cutting parameters that are parsed in this way, are the width of cut, feed per tooth and cutting speed. With these values, the material removal rate for the tool can be calculated:

```
1.  double Vc = Double.parseDouble(Cutting_speed);
2.  double SpindleSpeed = Vc / (pi * diameter);
3.  double SpindleSpeed = Vc / (pi * diameter);
4.  double Fz = Double.parseDouble(Feed_per_tooth);
5.  double zn = Double.parseDouble(FluteNumber);
6.  double FeedVelocity = SpindleSpeed * Fz * zn;
7.  double MMR = CutDepth * CutWidth * FeedVelocity;
```

Next, the cutting power is calculated:

```
double SpecificCuttingEnergy = Double.parseDouble(kc);
double CuttingPower = MMR * SpecificCuttingEnergy;
double CuttingForce = SpecificCuttingEnergy * CutDepth * Fz;
```

The specific cutting energy value is obtained from the database using the `getMaterialData()` method in `DatabaseAccess` as shown below.

```
public Cursor getMaterialData(String materialID){
    Cursor cursor = database.rawQuery("SELECT SMG, HB, UTS, kc, Yield FROM Material WHERE ID = '" + materialID + "'", null);
    return cursor;
}
```

Next, the shear plane deformation can be calculated as follows:

```
double UTStrength = Double.parseDouble(UTS);
double YieldStrength = Double.parseDouble(Yield);
double ChipCompressionRatio = UTStrength/YieldStrength;  //Approximately cutting ratio
double Gamma0 = Double.parseDouble(rakeAngle); //tool rake angle
//Solving for phi (shear angle)
double phi = Math.atan(Math.cos(Gamma0) / (ChipCompressionRatio - Math.sin(Gamma0)));
//Solving for beta (tool-interface friction)
double Beta = (pi / 4) + Gamma0 - phi;

double ShearStrain = Math.abs(Math.cos(Gamma0) / (Math.sin(phi) - Math.cos(phi - Gamma0)));
```

Given that the TOPSIS algorithm normalises the tool scores for each criteria, linear factors in the scores do not affect the decision making process for that criteria. Hence in the tool life calculation shown below, the static factors are ignored.

```
double Fr = CuttingForce / Math.cos(Beta - Gamma0);
double Wn = Fr / Math.cos(Beta);// Normal load.
double length_of_cut = Double.parseDouble(((MachiningData)getApplicationContext()).getCutLength());
double ChipNumber = Length_of_cut / Fz;
double ContactLength = Math.sqrt(CutDepth * diameter);
double L = ChipNumber * ContactLength / zn ; // sliding distance per tooth
double ToolWear = (n * n) * ((Py * Em *(Math.pow(Wn, 3 / 2)))/(Kc * Kc * (Math.pow(H, 3 / 2)))) * L ;
```

Lastly, the surface roughness height is calculated:

```
double rn = Double.parseDouble(re1);
double Roughness = (Fz * Fz) / (31.2 * rn);
```

With all of the optimisation criteria scores calculated, all of the relevant tool details can be added to the `ArrayList` of `Hashmaps` to be stored in the global data class:

```
1.  public class Tool_filter_results extends AppCompatActivity {
2.      @Override
3.      protected void onCreate(Bundle savedInstanceState) {
4.          filteredTools = new ArrayList<>();
5.          tool_results_list= (ListView) findViewById(R.id.toolList);
6.          DatabaseAccess tool_search_db = DatabaseAccess.getInstance(this);
7.          tool_search_db.open();
8.          Cursor tool_search_list = tool_search_db.FilterToolsCursor(cut_profile, materialID);
9.          tool_search_list.moveToFirst();
10.         while (!tool_search_list.isAfterLast()) {
11.         /***Assign data and perform calculations***/
12.             HashMap<String, String> tool = new HashMap<>();
13.             HashMap<String, String> tool = new HashMap<>();
14.             //add each value to temporary hashmap
15.             tool.put("Name", Name);
16.             tool.put("Diameter", Diameter);
17.             /***Assign all properties to hashmap***/
18.             tool.put("CuttingPower", Cutting_power);
19.             tool.put("Roughness", Surface_roughness);
20.             tool.put("Shear", Shear_strain);
21.             tool.put("ToolLife", Tool_life);
22.             tool.put("MMR", Material_removal_rate);
23.             filteredTools.add(tool);
24.             tool_search_list.moveToNext();
25.         }
26.         tool_search_list.close();
27.         filteredToolsAdapter tool_filter_adapter = new filteredToolsAdapter(this, fil-
    teredTools);
28.         tool_results_list.setAdapter(tool_filter_adapter);
29.         tool_search_db.close();
30.         ((MachiningData)getApplicationContext()).setFilteredToolList(filteredTools);
```

The `filteredToolsAdapter` is used to populate the `toolList ListView`. Similar in implementation to the `materialArrayAdapter`, the an excerpt of the layout inflater, `getView()`, can be seen below.

```
1.  public class filteredToolsAdapter extends BaseAdapter {
2.      public View getView(int position, View convertView, ViewGroup parent) {
3.          //grab temporary tool item from arraylist of filtered tools
4.          HashMap<String, String> map = myToolsList.get(position);
5.
6.          DecimalFormat formatter0 = new DecimalFormat("#0");
7.          DecimalFormat formatter1 = new DecimalFormat("#0.0");
8.          DecimalFormat formatter2 = new DecimalFormat("#0.00");
9.
10.         double CuttingSpeed = Double.parseDouble(map.get("CuttingSpeed"));
11.         double CutWidth = Double.parseDouble(map.get("CutWidth"));
12.         double CutDepth = Double.parseDouble(map.get("CutDepth"));
13.         double MMR = Double.parseDouble(map.get("MMR"));
14.         double CuttingPower = Double.parseDouble(map.get("CuttingPower"));
15.
16.         viewHolder.Name.setText(map.get("Name"));
17.         viewHolder.Diameter.setText(map.get("Diameter"));
18.         viewHolder.CuttingLength.setText(map.get("CuttingLength"));
19.         viewHolder.FluteNumber.setText(map.get("FluteNumber"));
20.         viewHolder.CutDepthPerPass.setText(formatter1.format(CutDepth));
21.         viewHolder.CutWidthPerPass.setText(formatter1.format(CutWidth));
22.         viewHolder.MaterialRemovalRate.setText(formatter1.format(MMR));
23.         viewHolder.CuttingSpeed.setText(formatter0.format(CuttingSpeed));
24.         viewHolder.CuttingPower.setText(formatter2.format(CuttingPower));
25.     }
26. }
```

Note the use of the `DecimalFormat` objects to format the number display in the list. The position of the `toolList` in the xml structure of `activity_tool_filter_results.xml` is shown in Figure 27 below.

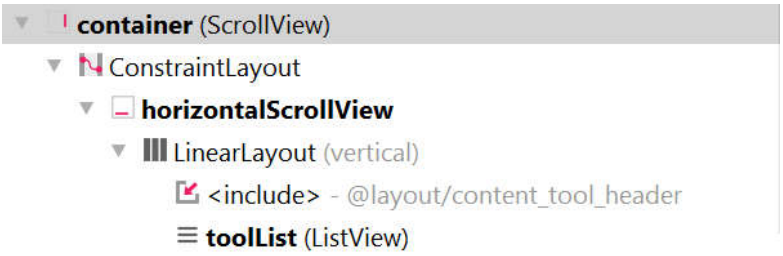


Figure 27: Tool list structure

It can be seen that the `ListView` element is below an `<include>` element, which allows the use of an xml layout defined separately. In this case, `content_tool_header` contains a layout for a header row for the list. This method allows the header of the list to be always visible, even when the list is scrolled. The header and `ListView` is nested within a `horizontalScrollView`, to allow horizontal scrolling due to the many list columns. The element tree for `content_tool_header` is shown in Figure 28 below.

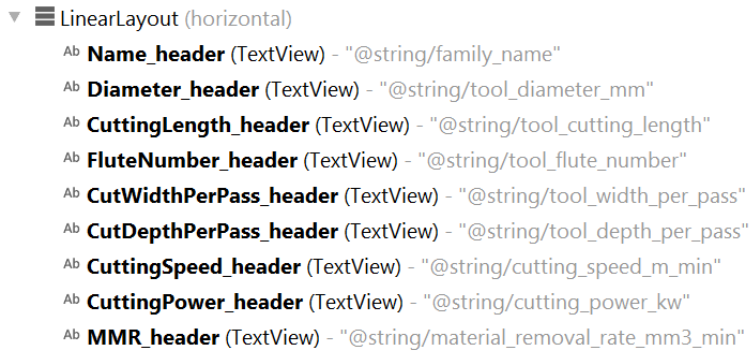


Figure 28: Tool table header structure

The header and custom list item together produce the result shown in Figure 29 below.

Family Name	Tool Diameter *mm*	Cutting Length *mm*	No. of Flutes	Cut width per pass *mm*	Cut depth per pass *mm*	Cutting Speed *m/min*	Cutting Power *kW*	Material Removal Rate *mm^3/min*
JS512	6	12	2	2.4	7.2	195	0.89	35.8
JS512	8	16	2	3.2	9.6	195	1.19	47.7
JS512	10	20	2	4.0	12.0	195	1.49	59.6
JS512	12	24	2	4.8	14.4	195	1.79	71.5
JS553	6	14	3	2.4	6.0	235	1.35	53.9

Figure 29: Optimised tool list

In order to apply the styling seen in the units for the header columns, the text in each of the `TextViews` are overwritten with html-styled text, as shown below for the "Tool Diameter" heading.

```
1.  public class Tool_filter_results extends AppCompatActivity {
2.      protected void onCreate(Bundle savedInstanceState) {
3.          Diameter_header = (TextView) findViewById(R.id.Diameter_header);
4.          Diameter_header.setText(Html.fromHtml("Tool Diameter
        <small><b><em>mm</em></b></small>"));
5.      }
6.  }
```

3.3.6.2. B – Optimisation information window

Due to the relative complexity of the optimisation algorithm, what is required from the user may not be immediately apparent. To add clarity, an information `ImageButton` was placed, to allow the user to inflate an information popup window. This is accomplished as follows:

```
1.  ImageButton OptimiseInfoButton = (ImageButton)findViewById(R.id.optimiseInfo-
    Button);
2.  OptimiseInfoButton.setOnClickListener(new View.OnClickListener() {
3.      @Override
4.      public void onClick(View v) {
5.          OptimiseInfoWindow();
6.      }
7.  });
```

The `OptimiseInfoWindow()` method is similar to the user cutting conditions popup window and serves to inflate the `fragment_optimise_info_popup.xml` layout, the result of which is shown in Figure 30 below.

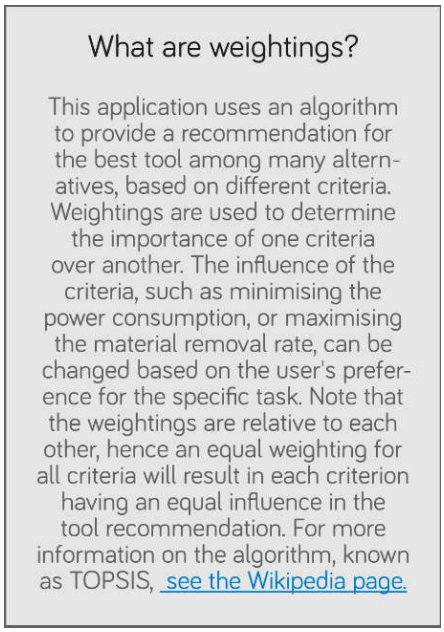


Figure 30: Optimization infopanel

The popup window consists of only two `TextViews`. The hyperlink in the description is made clickable with the `android:linksClickable="true"` attribute in the xml layout file.

3.3.6.3. C – Criteria weighting slider

The size and division of the `seekBars` used to set the criteria weighting, are defined in the xml layout, of which the power consumption `seekBar` is shown below.

```
<SeekBar
    android:id="@+id/powerSeekbar"
    style="@style/Widget.AppCompat.SeekBar.Discrete"
    android:layout_width="129dp"
    android:layout_height="wrap_content"
    android:layout_marginEnd="16dp"
    android:layout_marginRight="16dp"
    android:layout_marginTop="8dp"
    android:max="4"
    android:progress="2"
    android:progressTint="@color/colorPrimary"
    android:scaleX="1.2"
    android:scaleY="1.2"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="@+id/textView14" />
```

The values of the weightings are captured when the “Optimise Tools” `Button` is pressed to proceed to the next activity. At this point, the criteria weighting matrix is constructed and stored in the global data class, as shown below.

```
public class Tool_filter_results extends AppCompatActivity {
    public void OptimiseTools(View view) {
        Intent optimise_tools_intent = new Intent(getApplicationContext(), Optimise_tools.class);
        double [] CriteriaWeightingMatrix = new double[5];

        SeekBar power = (SeekBar)findViewById(R.id.powerSeekbar);
        int powerWeight = power.getProgress();
        SeekBar roughness = (SeekBar)findViewById(R.id.roughnessSeekBar);
        int roughnessWeight = roughness.getProgress();
        SeekBar shear = (SeekBar)findViewById(R.id.shearSeekBar);
        int shearWeight = shear.getProgress();
        SeekBar toolLife = (SeekBar)findViewById(R.id.toolLifeSeekBar);
        int toolLifeWeight = toolLife.getProgress();
        SeekBar MMR = (SeekBar)findViewById(R.id.mmrSeekBar);
        int mmrWeight = MMR.getProgress();

        CriteriaWeightingMatrix[0] = powerWeight * 1.0;
        CriteriaWeightingMatrix[1] = roughnessWeight * 1.0;
        CriteriaWeightingMatrix[2] = shearWeight * 1.0;
        CriteriaWeightingMatrix[3] = toolLifeWeight * 1.0;
        CriteriaWeightingMatrix[4] = mmrWeight * 1.0;

        ((MachiningData)getApplicationContext()).setCriteriaWeightingMatrix(CriteriaWeightingMatrix);

        startActivity(optimise_tools_intent);
    }
```

3.3.7. Optimised tools activity

The final step of the tool Figure 31 below.

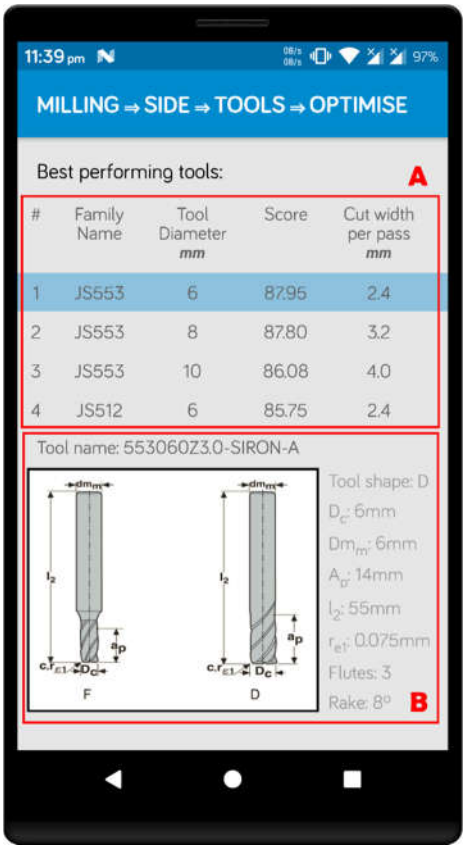


Figure 31:Labelled optimised tools screen

3.3.7.1. A – Optimised tool list

The final tool recommendation for the specified machining task is shown in the `OptimisedToolList` `ListView`, implemented in a similar way to the `toolList` from the previous activity. In this case, the ranking of the list must be determined by the optimisation algorithm. The criteria scores for each tool is retrieved and converted for use in calculations as shown below.

```
1.  public class Optimise_tools extends AppCompatActivity {
2.      @Override
3.      protected void onCreate(Bundle savedInstanceState) {
4.          filteredToolList = ((MachiningData)getApplicationContext()).getFilteredToolList();
5.          double [] CriteriaWeightingMatrix = ((MachiningData)getApplicationContext()).getCrite-
    riaWeightingMatrix();
6.          int rows_alternatives = filteredToolList.size();
7.          int columns_criteria = 5;
8.          //TOPSIS_List = new ArrayList<>();
9.          double [] [] TOPSISmatrix = new double[rows_alternatives][columns_criteria];
10.         for (int row = 0; row < rows_alternatives; row++) {
11.             HashMap<String, String> ToolData = filteredToolList.get(row);
12.
13.             String Power = ToolData.get("CuttingPower");
14.             String Roughness = ToolData.get("Roughness");
15.             String Shear = ToolData.get("Shear");
16.             String ToolLife = ToolData.get("ToolLife");
17.             String MMR = ToolData.get("MMR");
18.
19.             TOPSISmatrix[row][0] = Double.parseDouble(Power);       //Must minimise
20.             TOPSISmatrix[row][1] = Double.parseDouble(Roughness);   //Must minimise
21.             TOPSISmatrix[row][2] = Double.parseDouble(Shear);       //Must minimise
22.             TOPSISmatrix[row][3] = Double.parseDouble(ToolLife);    //Must minimise
23.             TOPSISmatrix[row][4] = Double.parseDouble(MMR);         //Must maximise
24.         }
25. }
```

The `for` loop in line 10 is used to assign each of the criteria scores for a tool to a `TOPSISmatrix` row. With the required data available, the optimisation algorithm, TOPSIS, is run as follows:

```
1.  TOPSIS machiningTOPSIS = new TOPSIS();
2.  machiningTOPSIS.setrows_alternatives(rows_alternatives);
3.  machiningTOPSIS.setcolumns_criteria(columns_criteria);
4.  machiningTOPSIS.setTOPSISmatrix(TOPSISmatrix);
5.  machiningTOPSIS.setCriteriaWeightingMatrix(CriteriaWeightingMatrix);
6.  double [] UnorderedToolScores = machiningTOPSIS.calculate();
```

As can be seen in the excerpt above, a TOPSIS object is created then the appropriate matrices are assigned through the `set-` methods. The output of the TOPSIS process is stored as a single array of `double` values, through the use of the `calculate()` method. The code for the TOPSIS process is lengthy, and mostly involves the transforms that are applied to find the best choice among multiple criteria. The full code can be found in appendix, while an excerpt containing only one of the transform functions is shown below.

```
public class TOPSIS {
        public double [] calculate() {
        calculateNormalizedDecisionMatrix();
        calculateNormalizedWeightingMatrix();
        calculateWeightedNormalizedDecisionMatrix();
        calculatepositiveIdealSolution();
        calculatenegativeIdealSolution();
        calculatePositiveSeparationFromIdeal();
        calculateNegativeSeparationFromIdeal();
        calculateClosenessCoefficient();
        return closenessCoefficient;
    }
        public double[][] calculateNormalizedDecisionMatrix() {
        double[] sumPowSqrt = new double[columns_criteria];
        normalizedDecisionMatrix = new double[rows_alternatives][columns_criteria];
        /* Calculate Normalized Decision Matrix */
        for (int col = 0; col < columns_criteria; col++) {
            double sumPow = 0.d;
            for (int row = 0; row < rows_alternatives; row++) {
                sumPow = sumPow + Math.pow(TOPSISmatrix[row][col], 2);
            }
            sumPowSqrt[col] = Math.sqrt(sumPow);
            for (int row = 0; row < TOPSISmatrix.length; row++) {
                normalizedDecisionMatrix[row][col] = TOPSISmatrix[row][col] / sumPowSqrt[col];
            }
        }
        return normalizedDecisionMatrix;
    }
}
```

It can be seen that the return variable for `calculate()`, is an array of double values called `closenessCoefficient`, which is stored as the `UnorderedToolScores` list in the `Optimise_tools` activity. These tool scores are added to the `filteredToolList` as shown below.

```
//add closeness coefficient (score) table into tool data table.
for (int row = 0; row < rows_alternatives; row++) {
    HashMap<String, String> TOPSISelement = filteredToolList.get(row);
    double toolScore = UnorderedToolScores[row];
    DecimalFormat formatter2 = new DecimalFormat("#0.00");
    String toolScoreShort = formatter2.format(toolScore*100);
    TOPSISelement.put("Score", toolScoreShort);
}
```

With the tool scores known, the result list can be sorted from best to worst. This is accomplished using a `ToolScoreComparator`, an Android built in function. The implementation is shown below.

```
public class ToolScoreComparator
        implements Comparator<HashMap<String, String>>
{
    @Override
    public int compare(HashMap<String, String> Tool1,
                       HashMap<String, String> Tool2)
    {
        return Double.compare(Double.parseDouble(Tool2.get("Score")), Double.parseDou-
ble(Tool1.get("Score")));
    }
}
```

The sort() method from the built-in Collections.java class is called to sort the tool list. After being sorted, the data is displayed on the `OptimisedToolList` through an `OptimisedToolsAdapter`, as shown below.

```
Collections.sort(filteredToolList, new ToolScoreComparator());

ArrayList<HashMap<String, String>> Top5Tools = new ArrayList<>();
for (int position = 0; position < 4; position++){
    Top5Tools.add(filteredToolList.get(position));
}
ListView OptimisedToolList = (ListView) findViewById(R.id.OptimisedToolList);
OptimisedToolsAdapter adapter = new OptimisedToolsAdapter(this,Top5Tools);
OptimisedToolList.setAdapter(adapter);
adapter.notifyDataSetChanged();
```

The `OptimisedToolsAdapter` populates the `ListView` in a similar way to the `filteredToolsAdapter`, used in the previous activity. With a custom header file, the `OptimisedToolList` is displayed as shown in Figure 32 below.

#	Family Name	Tool Diameter *mm*	Score	Cut width per pass *mm*	Cut depth per pass *mm*	Cutting Speed *m/min*	Cutting Power *kW*	Material Removal Rate *mm^3/min*
1	JS553	6	87.95	2.4	6.0	235	1.35	53.9
2	JS553	8	87.80	3.2	8.0	235	1.80	71.8
3	JS553	10	86.08	4.0	10.0	235	2.24	89.8
4	JS512	6	85.75	2.4	7.2	195	0.89	35.8

Figure 32: Complete optimised tool list

3.3.7.2. B – Tool details panel

The tool details panel is implemented as a popup window, similar to the information panel for the optimisation algorithm, as well as the custom cutting conditions window. The layout consists of the various `TextView` details, as well as an `ImageView` for the image display. When a list item is clicked, a simple click listener is set as shown below.

```
OptimisedToolList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int posi-
tion, long id) {
        ToolDetailsWindow(position);
    }
});
```

The `ToolDetailWindow()` method inflates the details panel, and the text and image is populated as follows, shown for three of the tool details `TextViews`:

```
1.  public void ToolDetailsWindow(int position){
2.          String FamilyName = filteredToolList.get(position).get("Name");
3.          String ToolName = filteredToolList.get(position).get("PartNumber");
4.          String ToolShape = filteredToolList.get(position).get("ToolShape");
5.          /***All the tool details***/
6.
7.          TextView Tool_name = (TextView) popupView.findViewById(R.id.ToolName);
8.          TextView Tool_shape = (TextView) popupView.findViewById(R.id.ToolShape);
9.          TextView dc = (TextView) popupView.findViewById(R.id.Dc);
10.         /***All the detail textviews***/
11.
12.         Tool_name.setText("Tool name: " + ToolName);
13.         Tool_shape.setText("Tool shape: " + ToolShape);
14.         dc.setText(Html.fromHtml("D<sub><small>c</small></sub>: " + Dc + "mm"));
15.
16.         ImageView ToolDiagram = (ImageView)popupView.findViewById(R.id.Opti-
    misedToolDiagram);
17.         DatabaseAccess databaseAccess = DatabaseAccess.getInstance(getApplication-
    Context());
18.         databaseAccess.open();
19.         Bitmap toolDiagram = databaseAccess.getToolDiagram(FamilyName);
20.         ToolDiagram.setImageBitmap(toolDiagram);
21. }
```

The `getToolDiagram()` method is used to obtain the image from the SQLite database. The excerpt from `DatabaseAccess.java` is shown below.

```
1.  public Bitmap getToolDiagram(String FamilyName) {
2.      Cursor cursor = database.rawQuery("SELECT Picture FROM Tool_pic-
    tures WHERE Name ='" + FamilyName +"'", null);
3.      cursor.moveToFirst();
4.      byte [] imageByteStream = cursor.getBlob(0);
5.      ByteArrayInputStream inputStream = new ByteArrayInputStream(imageByteStream);
6.      cursor.close();
7.      return BitmapFactory.decodeStream(inputStream);
8.  }
```

As images are stored as BLOB (Binary Large OBject) files, they must be retrieved as a `bytestream` and converted to a `Bitmap` that can be displayed.

3.3.8. Machine management activity

In order for the user to add and remove personal machines from the database, the `Machine_management` activity is used, of which the visual representation in Figure 33 below.

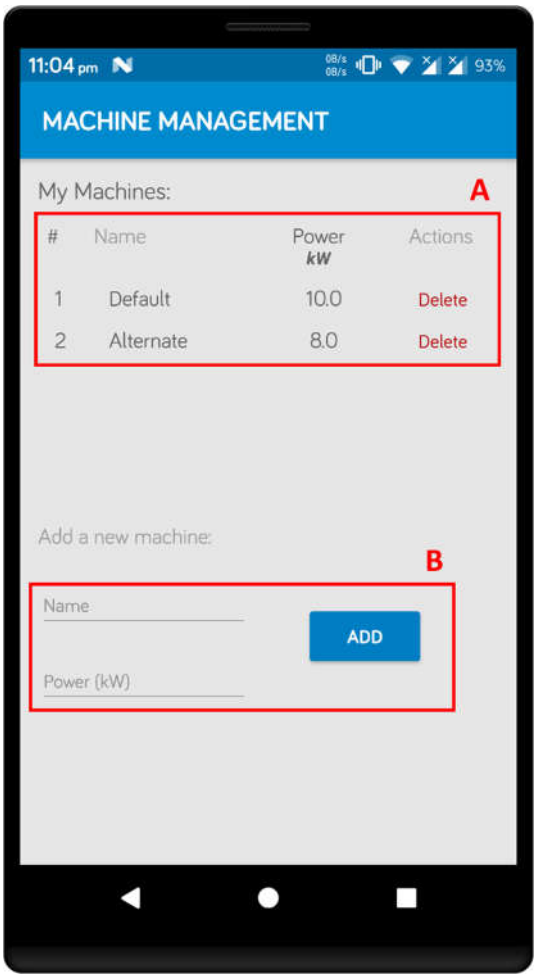


Figure 33: Labelled machine management screen

3.3.8.1. A – Machine list

The `MyMachinesList` is implemented like any of the previously discussed `ListViews`, with the only differing feature being the "delete" button functionality. This feature is done through the `machineArrayAdapter` that populates the machine list, by setting a click listener on the `TextView` element itself, as shown below.

```
1.  public class machineArrayAdapter extends BaseAdapter {
2.      public View getView(final int position, View convertView, final ViewGroup par-
    ent) {
3.          viewHolder.Delete.setOnClickListener(new View.OnClickListener() {
4.              @Override
5.              public void onClick(View v) {
6.                  DatabaseAccess databaseAccess = DatabaseAccess.getInstance(activ-
    ity.getApplicationContext());
7.                  databaseAccess.open();
8.                  databaseAccess.deleteMachine(myMachineList.get(posi-
    tion).get("Name"));
9.                  databaseAccess.close();
10.                 Intent intent = new Intent(activity.getApplicationContext(), Ma-
    chine_management.class);
11.                 context.startActivity(intent);
12.                 activity.finish();
13.                 Toast.makeText(activity.getApplicationContext(), "Deleted ma-
    chine", Toast.LENGTH_SHORT).show();
14.             }
15.         });
16.     }
17. }
```

As seen on line 13, a toast message is displayed upon the successful execution of the delete command.

3.3.8.2. B – Add machine

In order to provide the functionality for the user to add machines to the database, two `EditText` elements and a `Button` is used as follows:

```
1.  Name = (EditText)findViewById(R.id.MachineName);
2.  Power = (EditText)findViewById(R.id.MachinePower);
3.  AddMachine = (Button)findViewById(R.id.addMachine);
4.  AddMachine.setOnClickListener(new View.OnClickListener() {
5.      @Override
6.      public void onClick(View v) {
7.          String name = Name.getText().toString();
8.          String power = Power.getText().toString();
9.          String powerKW = Double.toString(Double.parseDouble(power)*1000);
10.         if (name.trim().equals("") || power.trim().equals("")){
11.             Toast.makeText(getApplicationContext(), "Please fill in the de-
    tails", Toast.LENGTH_SHORT).show();
12.         } else {
13.             DatabaseAccess databaseAccess = DatabaseAccess.getInstance(getApplicationContext());
14.             databaseAccess.open();
15.             databaseAccess.setMachine(name, powerKW);
16.             databaseAccess.close();
17.             startMachineManagement();
18.             InputMethodManager imm = (InputMethodManager)getSystemService(Context.IN-
    PUT_METHOD_SERVICE);
19.             imm.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(), 0);
20.             Toast.makeText(getApplicationContext(), "Added machine", Toast.LENGTH_SHORT).show();
21.         }
22.     }
23. });
```

As can be seen above, an onClick listener is set on the “Add” Button, and the if statement on line 10 prevents null entries. If the inputs are fine, the machine is added to the database, the keyboard is closed and a toast message is displayed.

4. Results and Discussion

The overall result of the application development discussed in section 3 is the fully developed application. The screen flow demonstration in section 3 is a comprehensive display of the entire visual interface of the application. In this section, an alternate use case will be investigated and the performance of the application will be discussed.

The desired machining task in the use case is a simple slot in S11 Titanium, the design of which is shown in Figure 34 below.

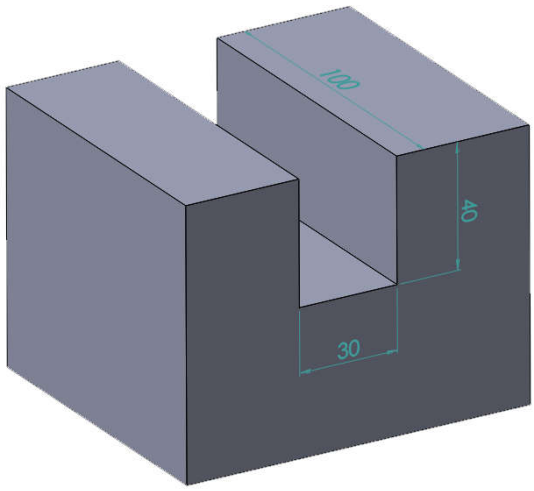


Figure 34: Slot milling model

A condensed screen flow diagram for the first three steps: selecting the slot milling task, inputting application details, and filtering the tool selection is shown in Figure 35 below.

Figure 35: Alternate use case screen flow diagram

If the input details have changed, the tool filter result changes accordingly. With the most significant change seen with the workpiece material, Figure 36 below shows the difference when H11 martensitic stainless steel is selected.

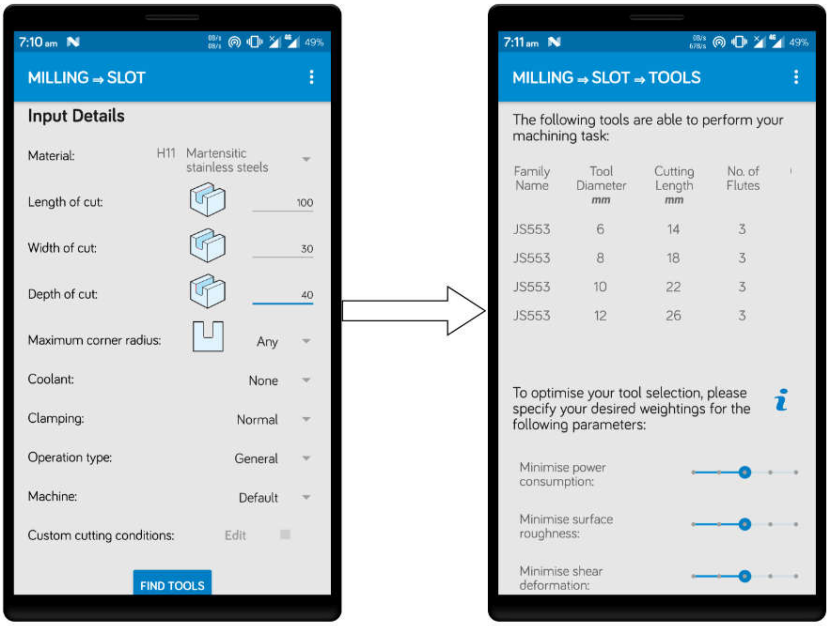


Figure 36: H11 material selection

Another input that can be modified is the custom cutting conditions. A comparison of the table data between manufacturer recommended cutting conditions and user specified cutting conditions is shown in Figure 37 below.

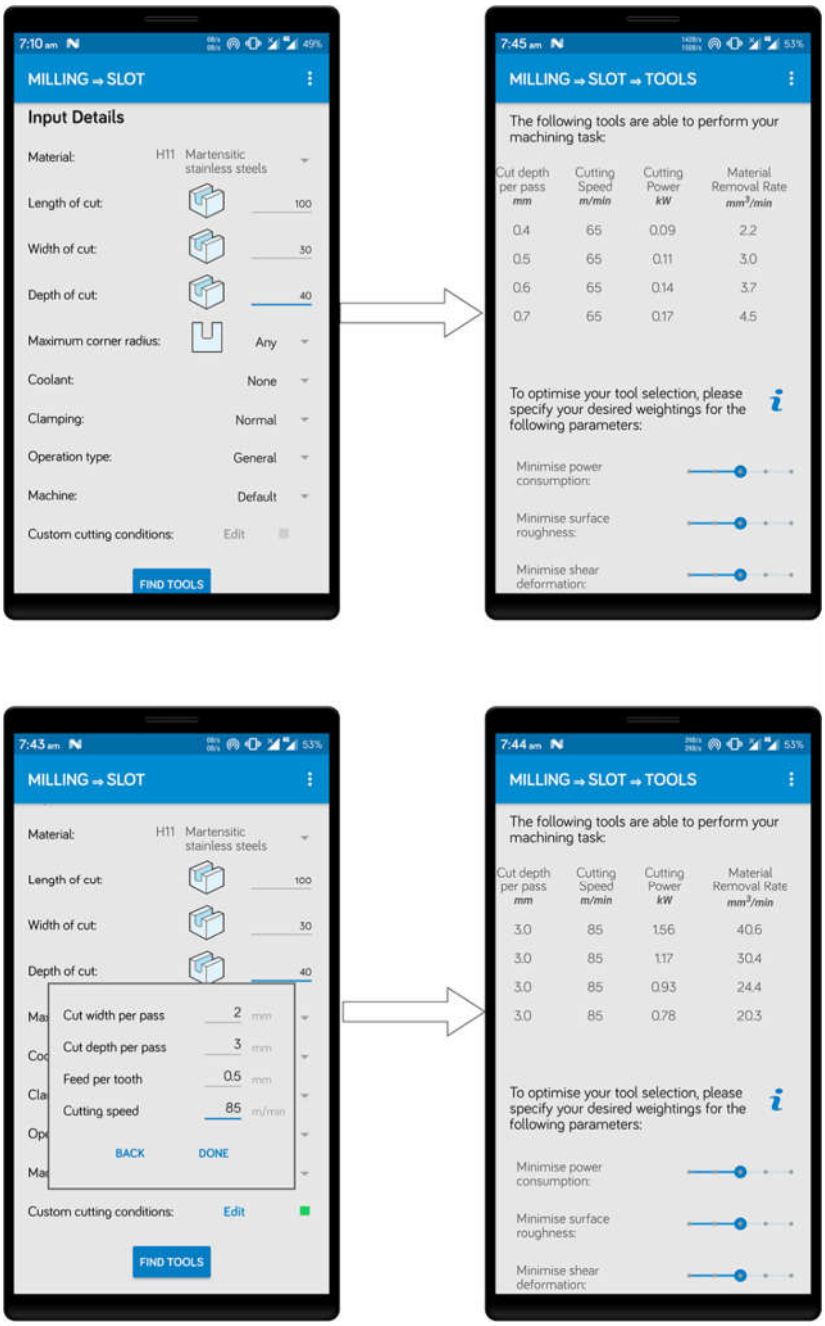


Figure 35: Effect of custom cutting conditions

From the original S11 use case described earlier, for the default weighting configuration, the optimisation result is shown in the screen flow diagram shown in Figure 38 below.

Figure 36: Default weightings

The effect of changing the criteria weightings can be seen in Figure 39 below. The complete list contents have been joined using image editing software.

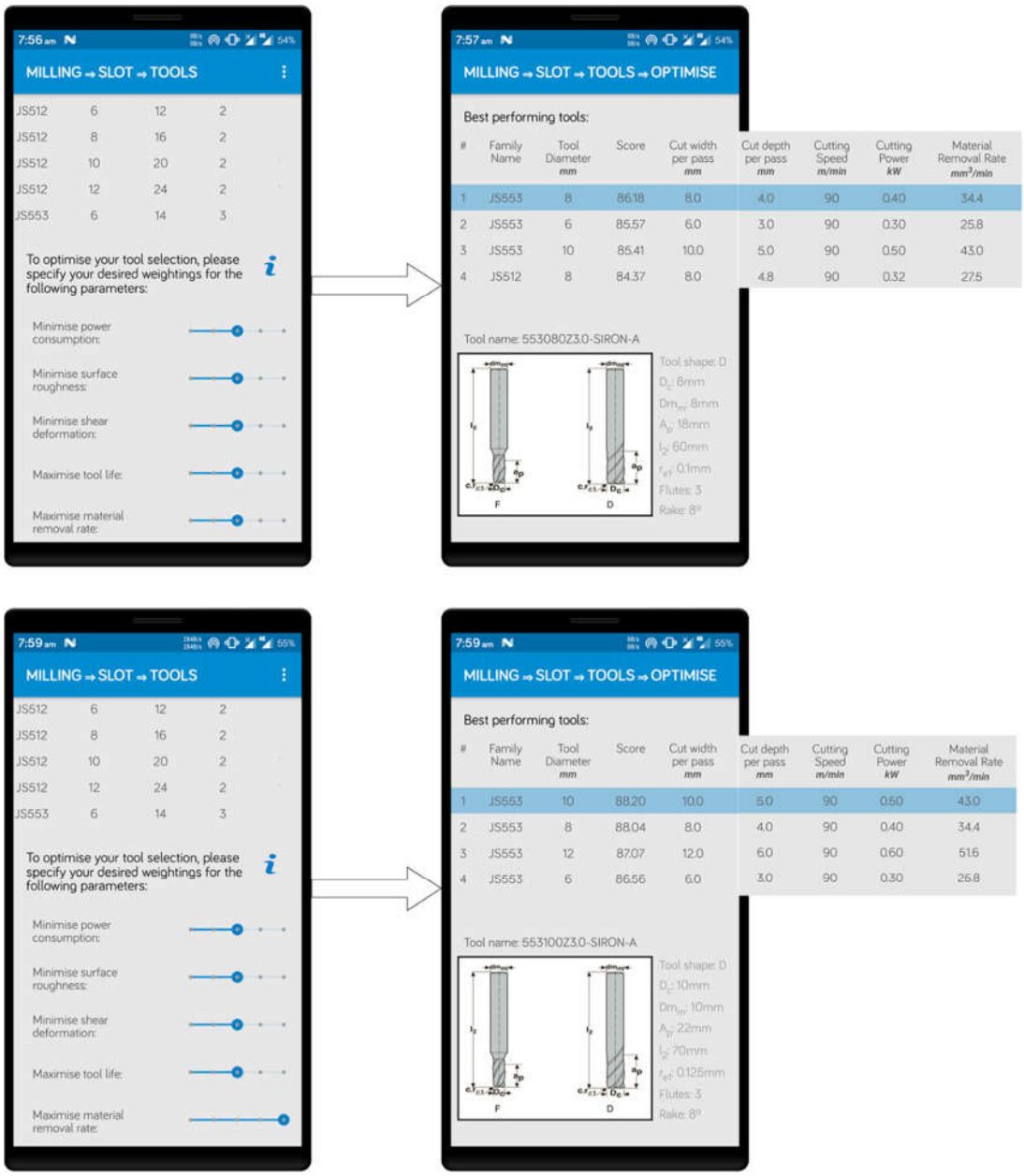


Figure 37: Effect of weighting change

As can be seen from figure, the effect of increasing the priority of maximising the material removal rate, has clearly affected the tool recommendation. Hence, the MCDM algorithm implementation is functional.

5. Conclusion

The goal of the project were to develop an application that can provide an appropriate cutting tool selection for an intended machining task. The application would be able to communicate with a cloud database and search for tools from multiple manufacturers, utilising dynamic filtering to narrow the result set.

The application that has been developed to achieve the goal utilises an embedded database, which uses that same technology as a cloud-based database, to store the tool data. Tools from any manufacturer can be added to the database, and is incorporated into the calculations provide the relevant tool data is entered into the database. An input page is used to gather the application details needed to filter the tool database for usable tools. In order to provide a tool recommendation, the MCDM algorithm TOPSIS was used, and shown to distinctly optimise the tool selection based on the input details, tool properties and criteria weightings.

The application was developed in Android Studio, tested with a Xiaomi Redmi Note 4 smartphone, and was found to be fully functional.

5.1 Future work

This initial release of the application has had the essential features developed for the functionality required to solve the thesis problem. Possible feature additions and improvements that can be made to the application in the future is discussed in this section.

Increasing the tool database

By adding more tools to the application's database, the optimisation algorithm has more tools to compare with, resulting in higher quality recommendations being given.

Machining formulae revision

The accuracy of the formulae used can be increased by obtaining new and more accurate information to calculate the optimisation criteria. More optimisation criteria can be added to the MCDM algorithm, such as reducing the residual stress.

Create cloud server database

As the basic functionality of the application is complete, the database can be migrated and accessed externally. Integrating the computing functionality to the server will also improve the performance of the application when the tool database becomes very large.

Add other types of cutting tools

The scope of this thesis has been limited to solid carbide end mills, but there is an abundance of alternate tools in the form of indexable inserts. To integrate a general purpose machining functionality to the application, both turning and holemaking operations must have tools to provide recommendations for.

Improve performance

The application has been developed with the aim of achieving the primary functions. Hence, there is room for a lot of code optimisations within the software stack of the application.

6. References

[1] “Techspex,” [Online]. Available: https://www.techspex.com/. [Accessed 19 10 2017].

[2] C. Peng, H. Du and W. Liao, “A research on the cutting database system based on machining features and TOPSIS,” *Robotics and Computer-Integrated Manufacturing,* vol. 43, pp. 96-104, 2017.

[3] M. Ribeiro and N. Coppini, “An applied database system for the optimization of cutting conditions and tool selection,” *Journal of Materials Processing Technology,* Vols. 92-93, pp. 317-374, 1999.

[4] B. Arezoo, K. Ridgway and A. Al-Ahmari, “Selection of cutting tools and conditions of machining operations using an expert system,” *Computers in Industry,* vol. 42, pp. 43-58, 2000.

[5] Y. Zhao, K. Ridgway and A. Al-Ahmari, “Integration of CAD and a cutting tool selection system,” *Computers & Industrial Engineering,* vol. 42, pp. 17-34, 2002.

[6] M. C. Cakir and K. Cavdar, “Development of a knowledge-based expert system for solving metal cutting problems,” *Materials and Design,* vol. 27, pp. 1027-1034, 2006.

[7] K. Edalew, H. Abdalla and R. Nash, “A computer-based intelligent system for automatic tool selection,” *Materials and Design,* vol. 22, pp. 337-351, 2001.

[8] P. G. Maropoulos, “Cutting tool selection: an intelligent methodology and its interfaces with technical and planning functions,” *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture,* vol. 206, no. 1, pp. 49-60, 1992.

[9] C. Hwang and K. Yoon, Multiple Attribute Decision Making - Methods and Application, New York: Springer, 1981.

[10] N. S. K. Reddy and P. V. Rao, “Selection of optimum tool geometry and cutting conditions using a surface roughnessprediction model for end milling,” *International Journal of Advanced Manufacturing Technology,* vol. 26, pp. 1202-1210, 2005.

[11] Y. Mizugaki, M. Hao, M. Sakamoto and H. Makino, “Optimal Tool Selection Based on Genetic Algorithm in a Geometric Cutting Simulation,” *Annals of the ClRP,* vol. 43, no. 1, pp. 433-436, 1994.

[12] C. Jensen, W. Red and J. Pi, “Tool selection for five-axis curvature matched machining,” *Computer-Aided Design,* vol. 34, pp. 251-266, 2002.

[13] H. M. Rho, R. Geelink, A. H. v. '. Erve and H. J. J. Kals, “An Integrated Cutting Tool Selection and Operation Sequencing Method,” *Annals of the CIRP,* vol. 41, no. 1, pp. 517-520, 1992.

[14] A. Oral and M. C. Cakir, “Automated cutting tool selection and cutting tool sequence optimisation for rotational parts,” *Robotics and Computer-Integrated Manufacturing,* vol. 20, pp. 127-141, 2004.

[15] I. Carpenter and P. Maropoulos, “Automatic tool selection for milling operations Part 1: cutting data generation,” *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture,* vol. 214, no. 4, pp. 271-282, 2000.

[16] O. Çakīr, A. Yardimeden, T. Ozben and E. Kilickap, “Selection of cutting fluids in machining processes,” *Journal of Achievements in Materials and Manufacturing Engineering,* vol. 25, no. 2, pp. 99-102, 2007.

[17] O. Onuoha, J. Abu, S. Lawal, E. Mudiare and M. Adeyemi, “Determining the Effect of Cutting Fluids on Surface Roughness in Turning AISI 1330 Alloy Steel Using Taguchi Method,” *Modern Mechanical Engi-,* vol. 6, pp. 51-59, 2016.

[18] M. Noordin, V. Venkatesh, C. Chan and A. Abdullah, “Performance Evaluation of Cemented Carbide Tools in Turning AISI 1010 Steel,” *Journal of Materials Processing Technology,* vol. 116, pp. 16-21, 2001.

[19] V. P. Astakhov, “The assessment of cutting tool wear,” *International Journal of Machine Tools & Manufacture,* vol. 44, pp. 637-647, 2004.

[20] E. Kılıckap, O. Cakır, M. Aksoy and A. Inan, “Study of tool wear and surface roughness in machining of homogenised SiC-p reinforced aluminium metal matrix composite,” *Journal of Materials Processing Technology,* Vols. 164-165, pp. 862-867, 2005.

[21] A. Makadia and J. Nanavati, “Optimisation of machining parameters for turning operations based on response surface methodology,” *Measurement,* vol. 46, pp. 1521-1529, 2013.

[22] Y. A. Hadi, “The relationship between tool length/diameter ratio and surface roughness in end milling applications,” *ISCV15,* vol. 15, pp. 1382-1389, 2008.

[23] J. Lorincz, “Using digital tools to optimize your cutting tools,” *Manufacturing Engineering Magazine,* pp. 55-62, 2014.

[24] J. Stark, “Principles of Mobile Interface Design,” 15 March 2012. [Online]. Available: http://www.slideshare.net/jonathanstark/principles-of-mobile-interface-design. [Accessed 11 10 2017].

[25] “Architecture,” Android Open Source Project, [Online]. Available: https://source.android.com/devices/architecture/. [Accessed 20 10 2017].

[26] C. Date, A guide to the SQL standard: a user's guide to the standard database language SQL, Addison-Wesley Professional., 1997.

[27] “Java Language Specification. Chapter 4. Types, Values, and Variables,” [Online]. Available: https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3. [Accessed 15 10 2017].

[28] D. GV, “Material removal rate,” Cadem Technologies P Ltd., 2017. [Online]. Available: http://cadem.com/cncetc/cnc-milling-turning-material-removal-rate/. [Accessed 27 05 2017].

[29] S. Coromant, “Milling formulas,” Sandvik Coromant, [Online]. Available: http://www.sandvik.coromant.com/en-us/knowledge/milling/formulas_and_definitions/formulas. [Accessed 25 05 2017].

[30] H. Rebai, Tribolgy and machine elements, Riihimäki: HAMK University of Applied, 2014.

[31] S. Kalpakjian and S. R. Schmid, Manufacturing processes for engineering materials, Upper Saddle River, N.J: Prentice Hall, 2003.

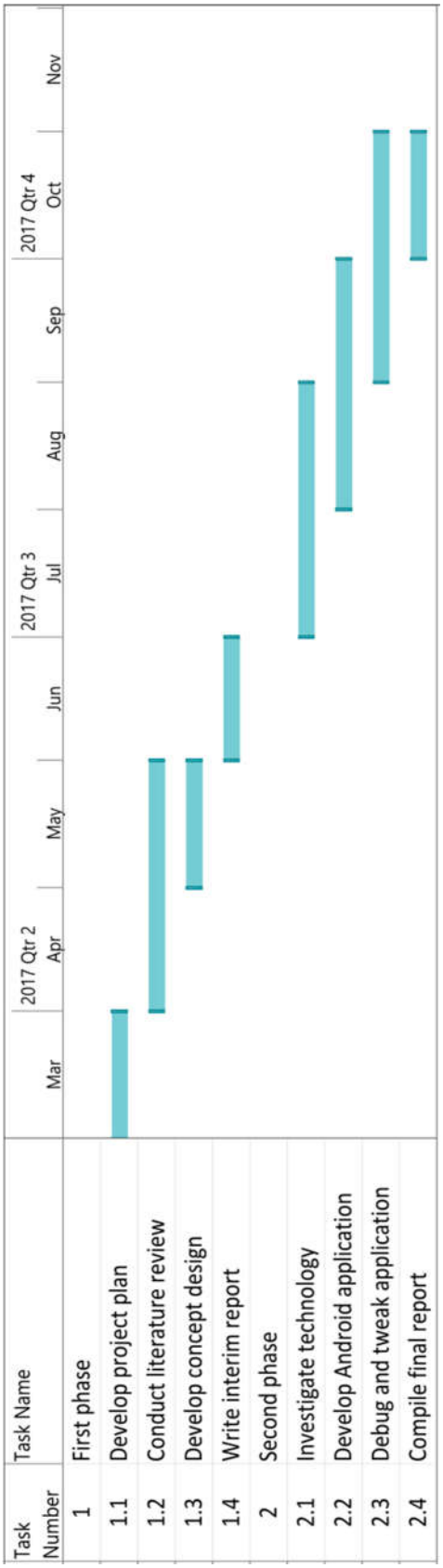
[32] A. Abdullah, L. Chia and Z. Samad, “The effect of feed rate and cutting speed on surface roughness,” *Asian Journal of Scientific Reseach,* vol. 1, no. 1, pp. 12-21, 2008.

[33] C.-L. Hwang, Y.-J. Lai and T.-Y. Liu, “A New Approach For Multiple Objective Decision Making,” *Computers Ops Res.,* vol. 20, no. 8, pp. 889-899, 1993.

[34] “Machining ISO workpiece materials,” [Online]. Available: http://www.industrysourcing.com/article/machining-iso-p-workpiece-materials. [Accessed 15 10 2017].

7. Appendix

7.1. Appendix A: Project Plan



7.2. Appendix B: Project Code

Activities

Milling_task.java

```
1.  package com.nishen.machiningapp.activities;
2.
3.  import android.content.Intent;
4.  import android.os.Bundle;
5.  import android.support.v7.app.AppCompatActivity;
6.  import android.view.LayoutInflater;
7.  import android.view.Menu;
8.  import android.view.MenuItem;
9.  import android.view.View;
10. import android.view.ViewGroup;
11. import android.widget.Button;
12. import android.widget.TextView;
13. import android.widget.Toast;
14.
15. import com.nishen.machiningapp.R;
16. import com.nishen.machiningapp.models.MachiningData;
17.
18. /**
19.  * Created by Nishen on 2017/09/20.
20.  */
21.
22. public class Milling_task extends AppCompatActivity {
23.
24.
25.     @Override
26.     protected void onCreate(Bundle savedInstanceState) {
27.         super.onCreate(savedInstanceState);
28.         setTitle("Milling");
29.         setContentView(R.layout.activity_milling_task);
30.
31.         // Capture our button from layout
32.         Button ramp_button = (Button)findViewById(R.id.button_ramp);
33.         ramp_button.setOnClickListener(rampListener);
34.
35.
36.
37.     }
38.
39.
40.     /** Called when the user taps the Slot Milling button */
41.     public void slotbutton(View view) {
42.         // Do something in response to button
43.         Intent intent = new Intent(this, Slot_input.class);
44.         ((MachiningData)getApplicationContext()).setProfile("Slot");
45.         startActivity(intent);
46.
47.         //finish();
48.     }
49.
50.     public void sidebutton(View view) {
51.         // Do something in response to button
52.         Intent intent = new Intent(this, Side_input.class);
53.         ((MachiningData)getApplicationContext()).setProfile("Side"); //set global P
    rofile variable
54.         startActivity(intent);
55.     }
56.
57.     public void contourbutton(View view) {
58.         // Do something in response to button
59.         Intent intent = new Intent(this, Contour_input.class);
```

```
        ((MachiningData)getApplicationContext()).setProfile("Contour");
        startActivity(intent);
    }


    // Toast for ramp button
    private View.OnClickListener rampListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            showCustomToast("Feature arriving in the future");
        }
    };


    //show custom Toast in android
    private void showCustomToast(String showToast) {
        LayoutInflater inflater = getLayoutInflater();
        View layout = inflater.inflate(R.layout.content_custom_toast_layout,
                (ViewGroup) findViewById(R.id.toast_layout_root));
        TextView text = (TextView) layout.findViewById(R.id.text);
        text.setText(showToast);
        Toast toast = new Toast(getApplicationContext());
        toast.setDuration(Toast.LENGTH_SHORT);
        toast.setView(layout);
        toast.show();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);//Menu Resource, Menu
        return true;
    }


    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
                switch (item.getItemId()) {
                    case R.id.item1:
                        Intent intent = new Intent(this, Machine_management.class);

                        startActivity(intent);
                        return true;


                    default:
                        return super.onOptionsItemSelected(item);
                }
            }


        }
```

Side_input.java

```
package com.nishen.machiningapp.activities;

/**
```

```
4.   * Created by Nishen on 2017/09/19.
5.   */
6.
7.  import android.animation.Animator;
8.  import android.animation.AnimatorListenerAdapter;
9.  import android.animation.AnimatorSet;
10. import android.animation.ObjectAnimator;
11. import android.content.Context;
12. import android.content.Intent;
13. import android.database.Cursor;
14. import android.graphics.Color;
15. import android.graphics.Point;
16. import android.graphics.Rect;
17. import android.graphics.drawable.ColorDrawable;
18. import android.os.AsyncTask;
19. import android.support.v7.app.AppCompatActivity;
20. import android.os.Bundle;
21. import android.text.Html;
22. import android.view.Gravity;
23. import android.view.LayoutInflater;
24. import android.view.Menu;
25. import android.view.MenuItem;
26. import android.view.View;
27. import android.view.ViewGroup;
28. import android.view.WindowManager;
29. import android.view.animation.DecelerateInterpolator;
30. import android.view.inputmethod.InputMethodManager;
31. import android.widget.AdapterView;
32. import android.widget.Button;
33. import android.widget.CompoundButton;
34. import android.widget.EditText;
35. import android.widget.ImageView;
36. import android.widget.PopupWindow;
37. import android.widget.ScrollView;
38. import android.widget.Spinner;
39. import android.widget.AdapterView.OnItemSelectedListener;
40.
41. import java.util.ArrayList;
42. import java.util.Arrays;
43. import java.util.HashMap;
44. import java.util.List;
45. import android.widget.ArrayAdapter;
46. import android.widget.Switch;
47. import android.widget.TextView;
48. import android.widget.Toast;
49.
50. import com.nishen.machiningapp.R;
51. import com.nishen.machiningapp.adapters.materialArrayAdapter;
52. import com.nishen.machiningapp.helpers.DatabaseAccess;
53. import com.nishen.machiningapp.models.MachiningData;
54.
55.
56. public class Side_input extends AppCompatActivity implements AdapterView.OnItemSele
    ctedListener {
57.
58.
59.     /**
60.      * Hold a reference to the current animator, so that it can be canceled mid-
    way.
61.      */
62.     private Animator mCurrentAnimator;
63.
64.     /**
65.      * The system "short" animation time duration, in milliseconds. This duration i
    s ideal for
66.      * subtle animations or animations that occur very frequently.
```

```
67.      */
68.     private int mShortAnimationDuration;
69.
70.
71.     Spinner material_spinner;
72.     Spinner cnr_radius_spinner;
73.     Spinner coolant_spinner;
74.     Spinner operation_type_spinner;
75.     Spinner machine_spinner;
76.     Spinner clamping_spinner;
77.     Switch user_cutdata_switch;
78.     TextView user_cutdata_input_edit;
79.     //String UserCutWidth;
80.     //String UserCutDepth;
81.     //String UserCuttingSpeed;
82.
83.
84.     @Override
85.     protected void onCreate(Bundle savedInstanceState) {
86.         super.onCreate(savedInstanceState);
87.         String cut_profile = ((MachiningData)getApplicationContext()).getProfile();

88.         setTitle(Html.fromHtml("Milling<big>⇒</big>" +cut_profile));
89.         setContentView(R.layout.activity_side_input);
90.         //materialList = new ArrayList<>();
91.         //cnr_radius_list = new ArrayList<>();
92.
93.         getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_HI
    DDEN);
94.
95. /**  Function to load the materials spinner data from SQLite database
96.  material_spinner = (Spinner)findViewById(R.id.material_spinner);
97.  DatabaseAccess databaseAccess = DatabaseAccess.getInstance(this);
98.  databaseAccess.open();
99.  Cursor materials =databaseAccess.getMaterialsCursor();
100.         materials.moveToFirst();
101.         while (!materials.isAfterLast()) {
102.         String SMG = materials.getString(0);
103.         String Description = materials.getString(1);
104.         HashMap<String, String> material = new HashMap<>();
105.         //add each value to temporary hashmap
106.         material.put("SMG", SMG);
107.         material.put("Description", Description);
108.         //add material to materialList
109.         materialList.add(material);
110.         materials.moveToNext();
111.         }
112.         materials.close();
113.
114.         materialArrayAdapter adapter = new materialArrayAdapter(Slot_input.this, ma
    terialList);
115.         this.material_spinner.setAdapter(adapter);
116.         /**  Function to load the materials spinner data from SQLite database */
117.
118.                 //material_spinner.setOnItemSelectedListener(new materialSpinnerList
    ener());
119.
120.                 new SlotInputAsyncTask().execute();
121.
122.                 //Corner radius dropdown spinner
123.                 cnr_radius_spinner = (Spinner) findViewById(R.id.corner_radius_spinn
    er);
124.                 DatabaseAccess cnr_radius_db = DatabaseAccess.getInstance(getApplica
    tionContext());
125.                 cnr_radius_db.open();
```

```
126.                List<String> cnr_radius_list = cnr_radius_db.unique_corner_radius();
127.                cnr_radius_list.add(0, "Any");
128.
129.                cnr_radius_db.close();
130.                ArrayAdapter<String> cnr_radius_adapter = new ArrayAdapter<String>(S
    ide_input.this, android.R.layout.simple_spinner_item, cnr_radius_list);
131.                cnr_radius_adapter.setDropDownViewResource(android.R.layout.simple_s
    pinner_dropdown_item);
132.                cnr_radius_spinner.setAdapter(cnr_radius_adapter);
133.
134.                //Corner radius dropdown spinner
135.                cnr_radius_spinner.setOnItemSelectedListener(new cornerRadiusSpinner
    Listener());
136.
137.        // Zoomable image buttons
138.                final View side_length_view = findViewById(R.id.side_length_imagebut
    ton);
139.                side_length_view.setOnClickListener(new View.OnClickListener() {
140.                    @Override
141.                    public void onClick(View view) {
142.                        zoomImageFromThumb(side_length_view, R.drawable.profile_side
    _length);
143.                    }
144.                });
145.
146.                final View side_width_view = findViewById(R.id.side_width_imagebutto
    n);
147.                side_width_view.setOnClickListener(new View.OnClickListener() {
148.                    @Override
149.                    public void onClick(View view) {
150.                        zoomImageFromThumb(side_width_view, R.drawable.profile_side_
    width);
151.                    }
152.                });
153.
154.                final View side_depth_view = findViewById(R.id.side_depth_imagebutto
    n);
155.                side_depth_view.setOnClickListener(new View.OnClickListener() {
156.                    @Override
157.                    public void onClick(View view) {
158.                        zoomImageFromThumb(side_depth_view, R.drawable.profile_side_
    depth);
159.                    }
160.                });
161.
162.                final View side_cnr_radius_view = findViewById(R.id.side_corner_radi
    us_imagebutton);
163.                side_cnr_radius_view.setOnClickListener(new View.OnClickListener() {
164.                    @Override
165.                    public void onClick(View view) {
166.                        zoomImageFromThumb(side_cnr_radius_view, R.drawable.profile_
    side_corner_radius);
167.                    }
168.                });
169.
170.                // Retrieve and cache the system's default "short" animation time.
171.                mShortAnimationDuration = getResources().getInteger(android.R.intege
    r.config_shortAnimTime);
172.        // Zoomable image buttons
173.
174.                coolant_spinner = (Spinner)findViewById(R.id.coolant_spinner);
175.                String[] coolantArray = getResources().getStringArray(R.array.coolan
    t_list);
```

```
176.                ArrayList<String> coolant_List = new ArrayList<String>(Arrays.asList
    (coolantArray));
177.                ArrayAdapter<String> coolant_adapter = new ArrayAdapter<String>(this
    , android.R.layout.simple_spinner_item, coolant_List);
178.                coolant_adapter.setDropDownViewResource(android.R.layout.simple_spin
    ner_dropdown_item);
179.                coolant_spinner.setAdapter(coolant_adapter);
180.
181.                coolant_spinner.setOnItemSelectedListener(new coolantSpinnerListener
    ());
182.
183.                operation_type_spinner = (Spinner)findViewById(R.id.operation_type_s
    pinner);
184.                String[] operationTypeArray = getResources().getStringArray(R.array.
    operation_type_list);
185.                ArrayList<String> operationType_list = new ArrayList<String>(Arrays.
    asList(operationTypeArray));
186.                ArrayAdapter<String> operationType_adapter = new ArrayAdapter<String
    >(this, android.R.layout.simple_spinner_item, operationType_list);
187.                operationType_adapter.setDropDownViewResource(android.R.layout.simpl
    e_spinner_dropdown_item);
188.                operation_type_spinner.setAdapter(operationType_adapter);
189.                operation_type_spinner.setSelection(1);
190.                //TODO utilise operation type to filter tools (no explicit argument)

191.
192.                operation_type_spinner.setOnItemSelectedListener(new operationTypeSp
    innerListener());
193.
194.                machine_spinner = (Spinner) findViewById(R.id.machine_spinner);
195.                DatabaseAccess machine_db = DatabaseAccess.getInstance(this);
196.                machine_db.open();
197.                List<String> machine_list = machine_db.getmachines();
198.                ArrayAdapter<String> machine_adapter = new ArrayAdapter<String>(this
    , android.R.layout.simple_spinner_item, machine_list);
199.                machine_adapter.setDropDownViewResource(android.R.layout.simple_spin
    ner_dropdown_item);
200.                machine_spinner.setAdapter(machine_adapter);
201.                machine_db.close();
202.                //TODO utilise machine power to limit tool selection
203.
204.                material_spinner.setOnItemSelectedListener(new machineSpinnerListene
    r());
205.
206.                clamping_spinner = (Spinner)findViewById(R.id.clamping_spinner);
207.                String[] clampingArray = getResources().getStringArray(R.array.clamp
    ing_list);
208.                ArrayList<String> clamping_list = new ArrayList<String>(Arrays.asLis
    t(clampingArray));
209.                ArrayAdapter<String> clampingAdapter = new ArrayAdapter<String>(this
    , android.R.layout.simple_spinner_item, clamping_list);
210.                clampingAdapter.setDropDownViewResource(android.R.layout.simple_spin
    ner_dropdown_item);
211.                clamping_spinner.setAdapter(clampingAdapter);
212.                clamping_spinner.setSelection(1);
213.                //TODO utilise clamping value
214.
215.                clamping_spinner.setOnItemSelectedListener(new clampingSpinnerListen
    er());
216.
217.                user_cutdata_input_edit = (TextView) findViewById(R.id.EditUserCutDa
    ta);
218.                user_cutdata_input_edit.setOnClickListener(new View.OnClickListener(
    ) {
219.                    @Override
220.                    public void onClick(View v) {
```

```
                        UserCutdataWindow(true);
                    }
                });

                user_cutdata_switch = (Switch) findViewById(R.id.UserCutDataSwitch);

                user_cutdata_switch.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
                    @Override
                    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                        if (isChecked){
                            user_cutdata_input_edit.setTextColor(getResources().getColor(R.color.colorPrimary));
                            UserCutdataWindow(false);
                        } else {
                            user_cutdata_input_edit.setTextColor(getResources().getColor(android.R.color.darker_gray));
                        }
                    }
                });









            }   //onCreate




            public void UserCutdataWindow(boolean Edit){
                try {
                    // get a reference to the already created main layout
                    final ScrollView mainLayout = (ScrollView) findViewById(R.id.container);

                    // inflate the layout of the popup window
                    LayoutInflater inflater = (LayoutInflater) getSystemService(LAYOUT_INFLATER_SERVICE);
                    final View popupView = inflater.inflate(R.layout.fragment_user_cutdata_popup, null);

                    // create the popup window
                    //boolean focusable = false; // true lets taps outside the popup also dismiss it
                    final PopupWindow popupWindow = new PopupWindow(popupView, 1, 1, true);
                    popupWindow.setWidth(850);
                    popupWindow.setHeight(ViewGroup.LayoutParams.WRAP_CONTENT);

                    popupWindow.setBackgroundDrawable(new ColorDrawable(Color.WHITE));
                    // show the popup window
                    popupWindow.showAtLocation(mainLayout, Gravity.CENTER, 0, 250);


                    final EditText CutWidth = (EditText)popupView.findViewById(R.id.userCutWidthPerPass);
                    final EditText CutDepth = (EditText)popupView.findViewById(R.id.userCutDepthPerPass);
                    final EditText CuttingSpeed = (EditText)popupView.findViewById(R.id.userCuttingSpeed);
```

```
272.                     final EditText FeedPerTooth = (EditText)popupView.findViewById(R
    .id.userFeedPerTooth);
273.
274.                     /**if (Edit && !UserCutWidth.trim().equals("") && !UserCutDepth.
    trim().equals("") && UserCuttingSpeed.trim().equals("")){
275.                      CutWidth.setText(UserCutWidth);
276.                      CutDepth.setText(UserCutDepth);
277.                      CuttingSpeed.setText(UserCuttingSpeed);
278.                      }*/
279.
280.                     //UserCutWidth = CutWidth.getText().toString();
281.                     //UserCutDepth = CutDepth.getText().toString();
282.                     //UserCuttingSpeed = CuttingSpeed.getText().toString();
283.                     Button Done = (Button)popupView.findViewById(R.id.userCutDataDon
    e);
284.                     Done.setOnClickListener(new View.OnClickListener() {
285.                         @Override
286.                         public void onClick(View v) {
287.                             if (CutWidth.getText().toString().trim().equals("") | Cu
    tDepth.getText().toString().trim().equals("") | CuttingSpeed.getText().toString().t
    rim().equals("") | FeedPerTooth.getText().toString().trim().equals("")  ) {
288.
289.                                 Toast.makeText(getApplicationContext(), "Please fill
    in the details", Toast.LENGTH_SHORT).show();
290.
291.                             } else {
292.                                 ((MachiningData) getApplicationContext()).setUserCut
    Width(CutWidth.getText().toString());
293.                                 ((MachiningData) getApplicationContext()).setUserCut
    Depth(CutDepth.getText().toString());
294.                                 ((MachiningData) getApplicationContext()).setUserCut
    tingSpeed(CuttingSpeed.getText().toString());
295.                                 ((MachiningData) getApplicationContext()).setUserFee
    dPerTooth(FeedPerTooth.getText().toString());
296.                                 InputMethodManager imm = (InputMethodManager)getSyst
    emService(Context.INPUT_METHOD_SERVICE);
297.                                 imm.hideSoftInputFromWindow(popupView.getWindowToken
    (), 0);
298.                                 popupWindow.dismiss();
299.                             }
300.                         }
301.                     });
302.
303.                     Button Back = (Button)popupView.findViewById(R.id.userCutDataBac
    k);
304.                     Back.setOnClickListener(new View.OnClickListener() {
305.                         @Override
306.                         public void onClick(View v) {
307.                             InputMethodManager imm = (InputMethodManager)getSystemSe
    rvice(Context.INPUT_METHOD_SERVICE);
308.                             imm.hideSoftInputFromWindow(popupView.getWindowToken(),
    0);
309.
310.                             popupWindow.dismiss();
311.                             user_cutdata_switch.toggle();
312.
313.
314.                         }
315.                     });
316.
317.                     // dismiss the popup window when touched
318.                     //popupView.setOnTouchListener(new View.OnTouchListener() {
319.                     //   @Override
320.                     //   public boolean onTouch(View v, MotionEvent event) {
321.                     //       popupWindow.dismiss();
322.                     //       return true;
```

```
                    //  }
                    // });
                }

                catch (Exception e) {
                    e.printStackTrace();
                }
            }

            public void searchtools (View view) {
                Intent filter_tools = new Intent(getApplicationContext(), Tool_filte
    r_results.class);
                //Bundle input_data_bundle = new Bundle();
                //insert data into bundle

                //input_data_bundle.putString("coolant", "");
                //input_data_bundle.putString("clamping", "");
                //input_data_bundle.putString("operation_type", "");
                //input_data_bundle.putString("machine", "");

                //filter_tools.putExtras(input_data_bundle);

                //Grab cut dimensions and add to global variables
                EditText CutLength = (EditText)findViewById(R.id.cut_length);
                String cut_length = CutLength.getText().toString();
                ((MachiningData)getApplicationContext()).setCutLength(cut_length);

                EditText CutWidth = (EditText)findViewById(R.id.cut_width);
                String cut_width = CutWidth.getText().toString();
                ((MachiningData)getApplicationContext()).setCutWidth(cut_width);


                EditText CutDepth = (EditText)findViewById(R.id.cut_depth);
                String cut_depth = CutDepth.getText().toString();
                ((MachiningData)getApplicationContext()).setCutDepth(cut_depth);

                ((MachiningData)getApplicationContext()).setUserCutDataChecked(user_
    cutdata_switch.isChecked());


                startActivity(filter_tools);
                finish();
            }

            /**
             * "Zooms" in a thumbnail view by assigning the high resolution image to
    a hidden "zoomed-in"
             * image view and animating its bounds to fit the entire activity conten
    t area. More
             * specifically:
             * <p>
             * <ol>
             * <li>Assign the high-res image to the hidden "zoomed-
    in" (expanded) image view.</li>
             * <li>Calculate the starting and ending bounds for the expanded view.</
    li>
             * <li>Animate each of four positioning/sizing properties (X, Y, SCALE_X
    , SCALE_Y)
             * simultaneously, from the starting bounds to the ending bounds.</li>
             * <li>Zoom back out by running the reverse animation on click.</li>
             * </ol>
             *
             * @param thumbView  The thumbnail view to zoom in.
             * @param imageResId The high-
    resolution version of the image represented by the thumbnail.
             */
```

```
381.          private void zoomImageFromThumb(final View thumbView, int imageResId) {
382.              // If there's an animation in progress, cancel it immediately and pr
    oceed with this one.
383.              if (mCurrentAnimator != null) {
384.                  mCurrentAnimator.cancel();
385.              }
386.
387.              // Load the high-resolution "zoomed-in" image.
388.              final ImageView expandedImageView = (ImageView) findViewById(R.id.pr
    ofile_side_length_big);
389.              expandedImageView.setImageResource(imageResId);
390.
391.              // Calculate the starting and ending bounds for the zoomed-
    in image. This step
392.              // involves lots of math. Yay, math.
393.              final Rect startBounds = new Rect();
394.              final Rect finalBounds = new Rect();
395.              final Point globalOffset = new Point();
396.
397.              // The start bounds are the global visible rectangle of the thumbnai
    l, and the
398.              // final bounds are the global visible rectangle of the container vi
    ew. Also
399.              // set the container view's offset as the origin for the bounds, sin
    ce that's
400.              // the origin for the positioning animation properties (X, Y).
401.              thumbView.getGlobalVisibleRect(startBounds);
402.              findViewById(R.id.container).getGlobalVisibleRect(finalBounds, globa
    lOffset);
403.              startBounds.offset(-globalOffset.x, -globalOffset.y);
404.              finalBounds.offset(-globalOffset.x, -globalOffset.y);
405.
406.              // Adjust the start bounds to be the same aspect ratio as the final
    bounds using the
407.              // "center crop" technique. This prevents undesirable stretching dur
    ing the animation.
408.              // Also calculate the start scaling factor (the end scaling factor i
    s always 1.0).
409.              float startScale;
410.              if ((float) finalBounds.width() / finalBounds.height()
411.                      > (float) startBounds.width() / startBounds.height()) {
412.                  // Extend start bounds horizontally
413.                  startScale = (float) startBounds.height() / finalBounds.height()
    ;
414.                  float startWidth = startScale * finalBounds.width();
415.                  float deltaWidth = (startWidth - startBounds.width()) / 2;
416.                  startBounds.left -= deltaWidth;
417.                  startBounds.right += deltaWidth;
418.              } else {
419.                  // Extend start bounds vertically
420.                  startScale = (float) startBounds.width() / finalBounds.width();
421.                  float startHeight = startScale * finalBounds.height();
422.                  float deltaHeight = (startHeight - startBounds.height()) / 2;
423.                  startBounds.top -= deltaHeight;
424.                  startBounds.bottom += deltaHeight;
425.              }
426.
427.              // Hide the thumbnail and show the zoomed-
    in view. When the animation begins,
428.              // it will position the zoomed-
    in view in the place of the thumbnail.
429.              thumbView.setAlpha(0f);
430.              expandedImageView.setVisibility(View.VISIBLE);
431.
```

```
                // Set the pivot point for SCALE_X and SCALE_Y transformations to th
    e top-left corner of
                // the zoomed-in view (the default is the center of the view).
                expandedImageView.setPivotX(0f);
                expandedImageView.setPivotY(0f);

                // Construct and run the parallel animation of the four translation
    and scale properties
                // (X, Y, SCALE_X, and SCALE_Y).
                AnimatorSet set = new AnimatorSet();
                set
                        .play(ObjectAnimator.ofFloat(expandedImageView, View.X, star
    tBounds.left,
                                finalBounds.left))
                        .with(ObjectAnimator.ofFloat(expandedImageView, View.Y, star
    tBounds.top,
                                finalBounds.top))
                        .with(ObjectAnimator.ofFloat(expandedImageView, View.SCALE_X
    , startScale, 1f))
                        .with(ObjectAnimator.ofFloat(expandedImageView, View.SCALE_Y
    , startScale, 1f));
                set.setDuration(mShortAnimationDuration);
                set.setInterpolator(new DecelerateInterpolator());
                set.addListener(new AnimatorListenerAdapter() {
                    @Override
                    public void onAnimationEnd(Animator animation) {
                        mCurrentAnimator = null;
                    }

                    @Override
                    public void onAnimationCancel(Animator animation) {
                        mCurrentAnimator = null;
                    }
                });
                set.start();
                mCurrentAnimator = set;

                // Upon clicking the zoomed-
    in image, it should zoom back down to the original bounds
                // and show the thumbnail instead of the expanded image.
                final float startScaleFinal = startScale;
                expandedImageView.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View view) {
                        if (mCurrentAnimator != null) {
                            mCurrentAnimator.cancel();
                        }

                        // Animate the four positioning/sizing properties in paralle
    l, back to their
                        // original values.
                        AnimatorSet set = new AnimatorSet();
                        set
                                .play(ObjectAnimator.ofFloat(expandedImageView, View
    .X, startBounds.left))
                                .with(ObjectAnimator.ofFloat(expandedImageView, View
    .Y, startBounds.top))
                                .with(ObjectAnimator
                                        .ofFloat(expandedImageView, View.SCALE_X, st
    artScaleFinal))
                                .with(ObjectAnimator
                                        .ofFloat(expandedImageView, View.SCALE_Y, st
    artScaleFinal));
                        set.setDuration(mShortAnimationDuration);
                        set.setInterpolator(new DecelerateInterpolator());
                        set.addListener(new AnimatorListenerAdapter() {
```

```
                            @Override
                            public void onAnimationEnd(Animator animation) {
                                thumbView.setAlpha(1f);
                                expandedImageView.setVisibility(View.GONE);
                                mCurrentAnimator = null;
                            }

                            @Override
                            public void onAnimationCancel(Animator animation) {
                                thumbView.setAlpha(1f);
                                expandedImageView.setVisibility(View.GONE);
                                mCurrentAnimator = null;
                            }
                        });
                        set.start();
                        mCurrentAnimator = set;
                    }
                });
            }

            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int positio
    n, long id) {

            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {

            }

            private class materialSpinnerListener implements OnItemSelectedListener
    {
                public void onItemSelected(AdapterView<?> parent, View view, int pos
    , long id) {
                    String selectedMaterial = new String();
                    selectedMaterial = String.valueOf(pos + 1);
                    ((MachiningData)getApplicationContext()).setSelectedMaterial(sel
    ectedMaterial);
                    //          selectedMaterial = parent.getItemAtPosition(pos).toS
    tring();
                }
                public void onNothingSelected(AdapterView parent) {
                    // Do nothing.
                }
            }

            private class cornerRadiusSpinnerListener implements OnItemSelectedListe
    ner {
                public void onItemSelected(AdapterView<?> parent, View view, int pos
    , long id) {
                    String selectedCornerRadius = new String();
                    selectedCornerRadius = parent.getItemAtPosition(pos).toString();

                    ((MachiningData)getApplicationContext()).setCornerRadius(selecte
    dCornerRadius);
                }
                public void onNothingSelected(AdapterView parent) {
                    // Do nothing.
                }
            }

            private class coolantSpinnerListener implements OnItemSelectedListener {

                public void onItemSelected(AdapterView<?> parent, View view, int pos
    , long id) {
```

```
541.                     String selectedCoolant = new String();
542.                     selectedCoolant = parent.getItemAtPosition(pos).toString();
543.                     ((MachiningData)getApplicationContext()).setCoolant(selectedCool
    ant);
544.                 }
545.                 public void onNothingSelected(AdapterView parent) {
546.                     // Do nothing.
547.                 }
548.             }
549.
550.             private class clampingSpinnerListener implements OnItemSelectedListener
    {
551.                 public void onItemSelected(AdapterView<?> parent, View view, int pos
    , long id) {
552.                     String selectedClamping = new String();
553.                     selectedClamping = parent.getItemAtPosition(pos).toString();
554.                     ((MachiningData)getApplicationContext()).setClamping(selectedCla
    mping);
555.                 }
556.                 public void onNothingSelected(AdapterView parent) {
557.                     // Do nothing.
558.                 }
559.             }
560.
561.             private class operationTypeSpinnerListener implements OnItemSelectedList
    ener {
562.                 public void onItemSelected(AdapterView<?> parent, View view, int pos
    , long id) {
563.                     String selectedOperationType = new String();
564.                     selectedOperationType = parent.getItemAtPosition(pos).toString()
    ;
565.                     ((MachiningData)getApplicationContext()).setOperationType(select
    edOperationType);
566.                 }
567.                 public void onNothingSelected(AdapterView parent) {
568.                     // Do nothing.
569.                 }
570.             }
571.
572.             private class machineSpinnerListener implements OnItemSelectedListener {

573.                 public void onItemSelected(AdapterView<?> parent, View view, int pos
    , long id) {
574.                     String selectedMachine = new String();
575.                     selectedMachine = parent.getItemAtPosition(pos).toString();
576.                     ((MachiningData)getApplicationContext()).setMachine(selectedMach
    ine);
577.                 }
578.                 public void onNothingSelected(AdapterView parent) {
579.                     // Do nothing.
580.                 }
581.             }
582.
583.             private class SlotInputAsyncTask extends AsyncTask<Void, Void, Void> {
584.
585.                 ArrayList<HashMap<String, String>> materialList;
586.                 List<String> cnr_radius_list;
587.
588.                 //@Override
589.                 //protected void onPreExecute() {}
590.
591.                 @Override
592.                 protected Void doInBackground(Void...arg0) {
593.
594.                     materialList = new ArrayList<HashMap<String, String>>();
```

```
595.                     /**  Function to load the materials spinner data from SQLite dat
    abase */
596.                     material_spinner = (Spinner)findViewById(R.id.material_spinner);
597.                     DatabaseAccess databaseAccess = DatabaseAccess.getInstance(getAp
    plicationContext());
598.                     databaseAccess.open();
599.                     Cursor materials =databaseAccess.getMaterialsCursor();
600.                     materials.moveToFirst();
601.                     while (!materials.isAfterLast()) {
602.                         String SMG = materials.getString(0);
603.                         String Description = materials.getString(1);
604.                         HashMap<String, String> material = new HashMap<>();
605.                         //add each value to temporary hashmap
606.                         material.put("SMG", SMG);
607.                         material.put("Description", Description);
608.                         //add material to materialList
609.                         materialList.add(material);
610.                         materials.moveToNext();
611.                     }
612.                     materials.close();
613.
614.                     /**  Function to load the materials spinner data from SQLite dat
    abase */
615.
616.
617.
618.
619.                     return null;
620.                 }
621.
622.                 //@Override
623.                 //protected void onProgressUpdate() {}
624.
625.                 @Override
626.                 protected void onPostExecute(Void result) {
627.
628.                     /**  Setting the materials spinner data from SQLite database */
629.                     materialArrayAdapter adapter = new materialArrayAdapter(Side_inp
    ut.this, materialList);
630.                     material_spinner.setAdapter(adapter);
631.                     adapter.notifyDataSetChanged();
632.                     material_spinner.setOnItemSelectedListener(new materialSpinnerLi
    stener());
633.
634.
635.
636.
637.                 }
638.
639.             }
640.
641.             @Override
642.             public boolean onCreateOptionsMenu(Menu menu) {
643.                 // Inflate the menu; this adds items to the action bar if it is pres
    ent.
644.                 getMenuInflater().inflate(R.menu.menu_main, menu);//Menu Resource, M
    enu
645.                 return true;
646.             }
647.
648.
649.             @Override
650.             public boolean onOptionsItemSelected(MenuItem item) {
651.                 switch (item.getItemId()) {
```

```
652.                     case R.id.item1:
653.                         Intent intent = new Intent(this, Machine_management.class);
654.
655.                         startActivity(intent);
656.                         return true;
657.
658.                     default:
659.                         return super.onOptionsItemSelected(item);
660.                 }
661.             }
662.
663.
664.         }
```

Tool_filter_results.java

```
1.  package com.nishen.machiningapp.activities;
2.
3.  import android.database.Cursor;
4.  import android.graphics.Color;
5.  import android.graphics.drawable.ColorDrawable;
6.  import android.os.Bundle;
7.  import android.content.Intent;
8.  import android.support.v7.app.AppCompatActivity;
9.  import android.text.Html;
10. import android.text.method.LinkMovementMethod;
11. import android.view.Gravity;
12. import android.view.LayoutInflater;
13. import android.view.Menu;
14. import android.view.MenuItem;
15. import android.view.View;
16. import android.view.ViewGroup;
17. import android.widget.ImageButton;
18. import android.widget.ListView;
19. import android.widget.PopupWindow;
20. import android.widget.ScrollView;
21. import android.widget.SeekBar;
22. import android.widget.TextView;
23.
24. import com.nishen.machiningapp.R;
25. import com.nishen.machiningapp.activities.Machine_management;
26. import com.nishen.machiningapp.activities.Optimise_tools;
27. import com.nishen.machiningapp.adapters.filteredToolsAdapter;
28. import com.nishen.machiningapp.helpers.DatabaseAccess;
29. import com.nishen.machiningapp.models.MachiningData;
30.
31. import java.util.ArrayList;
32. import java.util.HashMap;
33.
34. import  java.lang.Math;
35.
36. /**
37.  * Created by Nishen on 2017/09/19.
38.  */
39.
40. public class Tool_filter_results extends AppCompatActivity {
41.
42.     ListView tool_results_list;
43.     TextView Diameter_header;
44.     TextView CuttingLength_header;
45.     TextView CutDepthPerPass_header;
46.     TextView CutWidthPerPass_header;
```

```
47.     TextView CuttingSpeed_header;
48.     TextView CuttingPower_header;
49.     TextView MMR_header;
50.     ArrayList<HashMap<String, String>> filteredTools;
51.
52.     @Override
53.     protected void onCreate(Bundle savedInstanceState) {
54.         super.onCreate(savedInstanceState);
55.         String cut_profile = ((MachiningData)getApplicationContext()).getProfile();

56.         setTitle(Html.fromHtml("Milling<big>⇒</big>" + cut_profile + "<big>⇒</big>
    Tools"));
57.         setContentView(R.layout.activity_tool_filter_results);
58.         String materialID;
59.         filteredTools = new ArrayList<>();
60.
61.
62. //Format header columns
63.         Diameter_header = (TextView) findViewById(R.id.Diameter_header);
64.         Diameter_header.setText(Html.fromHtml("Tool Diameter <small><b><em>mm</em><
    /b></small>"));
65.
66.         CuttingLength_header = (TextView) findViewById(R.id.CuttingLength_header);

67.         CuttingLength_header.setText(Html.fromHtml("Cutting Length <small><b><em>mm
    </em></b></small>"));
68.
69.         CutDepthPerPass_header = (TextView) findViewById(R.id.CutDepthPerPass_heade
    r);
70.         CutDepthPerPass_header.setText(Html.fromHtml("Cut depth per pass <small><b>
    <em>mm</em></b></small>"));
71.
72.         CutWidthPerPass_header = (TextView) findViewById(R.id.CutWidthPerPass_heade
    r);
73.         CutWidthPerPass_header.setText(Html.fromHtml("Cut width per pass <small><b>
    <em>mm</em></b></small>"));
74.
75.         CuttingSpeed_header = (TextView) findViewById(R.id.CuttingSpeed_header);
76.         CuttingSpeed_header.setText(Html.fromHtml("Cutting Speed <small><b><em>m/mi
    n</em></b></small>"));
77.
78.         CuttingPower_header = (TextView) findViewById(R.id.CuttingPower_header);
79.         CuttingPower_header.setText(Html.fromHtml("Cutting Power <small><b><em>kW</
    em></b></small>"));
80.
81.         MMR_header = (TextView) findViewById(R.id.MMR_header);
82.         MMR_header.setText(Html.fromHtml("Material Removal Rate <small><b><em>mm<su
    p><small>3</small></sup>/min</em></b></small>"));
83.
84. //Format header columns
85.
86. // Grab input data from MachiningData global class
87.         //Bundle tool_search_bundle = getIntent().getExtras();
88.         materialID = ((MachiningData)getApplicationContext()).getSelectedMaterial()
    ;
89.
90.         //String cut_length = tool_search_bundle.getString("cut_length");
91.         //String cut_width = tool_search_bundle.getString("cut_width");
92.         //String cut_depth = tool_search_bundle.getString("cut_depth");
93.         String max_corner_radius = ((MachiningData)getApplicationContext()).getCorn
    erRadius();
94.         //String coolant = tool_search_bundle.getString("coolant");
95.         //String clamping = tool_search_bundle.getString("clamping");
96.         //String operation_type = tool_search_bundle.getString("operation_type");
97.         //String machine = tool_search_bundle.getString("machine");
98.
```

```
99.
100.                /** Populating Tool results listview **/
101.                tool_results_list= (ListView) findViewById(R.id.toolList);
102.                //View content_tool_header = getLayoutInflater().inflate(R.layout.co
    ntent_tool_header, null);
103.                //tool_results_list.addHeaderView(content_tool_header);
104.                DatabaseAccess tool_search_db = DatabaseAccess.getInstance(this);
105.                tool_search_db.open();
106.
107.            //Parse input data for database query
108.                Cursor tool_search_list = tool_search_db.FilterToolsCursor(cut_profi
    le, materialID);
109.                //Parse input data for database query
110.                tool_search_list.moveToFirst();
111.                while (!tool_search_list.isAfterLast()) {
112.                    String Name = tool_search_list.getString(0);
113.                    String Diameter = tool_search_list.getString(1);
114.                    String CuttingLength = tool_search_list.getString(2);
115.                    String FluteNumber = tool_search_list.getString(3);
116.                    String PartNumber = tool_search_list.getString(4);
117.                    String ToolShape = tool_search_list.getString(5);
118.                    String dmm = tool_search_list.getString(6);
119.                    String l2 = tool_search_list.getString(7);
120.                    String re1 = tool_search_list.getString(8);
121.                    String rakeAngle = tool_search_list.getString(9);
122.                    String Coolant = tool_search_list.getString(10);
123.                    String Ap_Dc = tool_search_list.getString(11);
124.                    String Ae_Dc = tool_search_list.getString(12);
125.                    String Fz6 = tool_search_list.getString(13);
126.                    String Fz8 = tool_search_list.getString(14);
127.                    String Fz10 = tool_search_list.getString(15);
128.                    String Fz12 = tool_search_list.getString(16);
129.                    String Cutting_speed = tool_search_list.getString(17);
130.
131.
132.
133.
134.
135.                    Cursor MaterialData = tool_search_db.getMaterialData(materialID)
    ;
136.                    MaterialData.moveToFirst();
137.                    String HB = MaterialData.getString(1);
138.                    String UTS = MaterialData.getString(2);
139.                    String kc = MaterialData.getString(3);
140.                    String Yield = MaterialData.getString(4);
141.
142.
143.        /**     Calculate power, etc.. and filter hashmap for individual tool **/
144.                    //Assign feed/tooth (mm) based on Diameter
145.                    String Feed_per_tooth = "0.1";
146.                    if (Diameter == "6"){
147.                        Feed_per_tooth = Fz6;
148.                    } else if (Diameter == "8"){
149.                        Feed_per_tooth = Fz8;
150.                    }else if (Diameter == "10"){
151.                        Feed_per_tooth = Fz10;
152.                    }else if (Diameter == "12"){
153.                        Feed_per_tooth = Fz12;
154.                    }
155.                    //Assign feed/tooth (mm) based on Diameter
156.
157.                    double pi = Math.PI;
158.
159.                    //Material removal rate calculations
160.                    double diameter = Double.parseDouble(Diameter);
161.                    double ApDc = Double.parseDouble(Ap_Dc);
```

```
162.
163.                     double CutDepth;
164.                     if (((MachiningData)getApplicationContext()).isUserCutDataChecke
    d()){
165.                         CutDepth = Double.parseDouble(((MachiningData)getApplication
    Context()).getUserCutDepth());
166.                     } else {
167.                         CutDepth = ApDc * diameter ;
168.                     }
169.
170.                     double AeDc = Double.parseDouble(Ae_Dc);
171.
172.                     double CutWidth;
173.                     if (((MachiningData)getApplicationContext()).isUserCutDataChecke
    d()){
174.                         CutWidth = Double.parseDouble(((MachiningData)getApplication
    Context()).getUserCutWidth());
175.                     } else {
176.                         CutWidth = AeDc * diameter ;
177.                     }
178.                     double Vc;
179.                     if (((MachiningData)getApplicationContext()).isUserCutDataChecke
    d()){
180.                         Vc = Double.parseDouble(((MachiningData)getApplicationContex
    t()).getUserCuttingSpeed());
181.                     } else {
182.                         Vc = Double.parseDouble(Cutting_speed);
183.                     }
184.
185.
186.                     double SpindleSpeed = Vc / (pi * diameter);
187.                     double Fz;
188.                     if (((MachiningData)getApplicationContext()).isUserCutDataChecke
    d()){
189.                         Fz = Double.parseDouble(((MachiningData)getApplicationContex
    t()).getUserFeedPerTooth());
190.                     } else {
191.                         Fz = Double.parseDouble(Feed_per_tooth);
192.                     }
193.                     double zn = Double.parseDouble(FluteNumber);
194.                     double FeedVelocity = SpindleSpeed * Fz * zn;
195.
196.                     double MMR = CutDepth * CutWidth * FeedVelocity; // Q
197.                     //Material removal rate calculations
198.
199.                     String CuttingSpeed = Double.toString(Vc);
200.                     String Cut_depth = Double.toString(CutDepth);
201.                     String Cut_width = Double.toString(CutWidth);
202.                     //String Material_removal_rate = Double.toString(MMR);
203.
204.
205.                     String Material_removal_rate = Double.toString(MMR);
206.
207.                     //Cutting power calculations
208.                     double SpecificCuttingEnergy = Double.parseDouble(kc); // Ks
209.                     double CuttingPower = MMR * SpecificCuttingEnergy;
210.                     //Cutting power calculations
211.
212.                     String Cutting_power = Double.toString(CuttingPower / (60 * 1000
    )); //kW formatting
213.
214.                     //Cutting force calculations
215.
216.                     double CuttingForce = SpecificCuttingEnergy * CutDepth * Fz;
217.                     //Cutting force calculations
218.
```

```
219.
220.                     //Shear plane deformation calculations
221.
222.                     double UTStrength = Double.parseDouble(UTS);
223.                     double YieldStrength = Double.parseDouble(Yield);
224.                     double ChipCompressionRatio = UTStrength/YieldStrength;  //Very
    similar to cutting ratio
225.
226.                     double Gamma0 = Double.parseDouble(rakeAngle); //tool rake angle

227.
228.                     //ChipCompressionRatio = cos(phi - Gamma0) / sin (phi)
229.                     //Solving for phi (shear angle)
230.                     double phi = Math.atan(Math.cos(Gamma0) / (ChipCompressionRatio
    - Math.sin(Gamma0)));
231.
232.
233.                     //double phi = (pi / 4)-(Beta - Gamma0) ;
234.                     //Solving for beta (tool-interface friction)
235.                     double Beta = (pi / 4) + Gamma0 - phi;
236.
237.                     double ShearStrain = Math.abs(Math.cos(Gamma0) / (Math.sin(phi)
    - Math.cos(phi - Gamma0)));
238.
239.                     //Shear plane deformation calculations
240.
241.                     String Shear_strain = Double.toString(ShearStrain);
242.
243.                     //Tool wear calculations
244.                     double n = 0.5; // work hardening factor of material. May be unn
    ecessary in comparison
245.                     double Py = 1; // yield strength of material.
246.                     double Em = 1; // Elastic modulus of material.
247.                     double Kc = 1; //Fracture toughness of material.
248.                     double H = 1; // hardness of material.
249.
250.                     double Fr = CuttingForce / Math.cos(Beta - Gamma0);
251.                     double Wn = Fr / Math.cos(Beta);// Normal load.
252.
253.
254.                     String length_of_cut = ((MachiningData)getApplicationContext()).
    getCutLength(); // from input page
255.                     //if (Length_of_cut < 1) {
256.                     double  Length_of_cut = 10.0;
257.                     //}
258.
259.                     double ChipNumber = Length_of_cut / Fz;
260.
261.                     double ContactLength = Math.sqrt(CutDepth * diameter);
262.
263.                     double L = ChipNumber * ContactLength / zn ; // sliding distance
    per tooth
264.
265.                     double ToolWearOrig = (n * n) * ((Py * Em *(Math.pow(Wn, 3 / 2))
    )/(Kc * Kc * (Math.pow(H, 3 / 2)))) * L ;
266.                     double ToolWear = Math.abs(ToolWearOrig / 1000);
267.
268.                     //Tool wear calculations
269.
270.                     String Tool_life = Double.toString(ToolWear);
271.
272.
273.                     //Surface roughness calculations
274.                     //tool tip radius
275.                     double rn = Double.parseDouble(re1);
276.                     double RoughnessOrig = (Fz * Fz) / (31.2 * rn) ; //Ra (mm)
```

```
277.                    double Roughness = RoughnessOrig * 1000;
278.                    //Surface roughness calculations
279.
280.                    String Surface_roughness = Double.toString(Roughness);
281.
282.
283.                    /**     Calculate power, etc.. and filter hashmap for individual
    tool  **/
284.
285.                    HashMap<String, String> tool = new HashMap<>();
286.                    //add each value to temporary hashmap
287.
288.                    tool.put("Name", Name);
289.                    tool.put("Diameter", Diameter);
290.                    tool.put("CuttingLength", CuttingLength);
291.                    tool.put("FluteNumber", FluteNumber);
292.                    tool.put("PartNumber", PartNumber);
293.                    tool.put("ToolShape", ToolShape);
294.                    tool.put("dmm", dmm);
295.                    tool.put("l2", l2);
296.                    tool.put("re1", re1);
297.                    tool.put("rakeAngle", rakeAngle);
298.                    tool.put("Coolant", Coolant);
299.                    tool.put("CutDepth", Cut_depth);
300.                    tool.put("CutWidth", Cut_width);
301.                    tool.put("Fz", Feed_per_tooth);
302.                    tool.put("CuttingSpeed", CuttingSpeed);
303.                    tool.put("CuttingPower", Cutting_power);
304.                    tool.put("Roughness", Surface_roughness);
305.                    tool.put("Shear", Shear_strain);
306.                    tool.put("ToolLife", Tool_life);
307.                    tool.put("MMR", Material_removal_rate);
308.                    filteredTools.add(tool);
309.
310.
311.
312.
313.                    tool_search_list.moveToNext();
314.                }   //while loop
315.
316.
317.
318.                tool_search_list.close();
319.
320.
321.
322.                filteredToolsAdapter tool_filter_adapter = new filteredToolsAdapter(
    this, filteredTools);
323.                tool_results_list.setAdapter(tool_filter_adapter);
324.                tool_search_db.close();
325.
326.       /** Populating Tool results listview **/
327.
328.                ImageButton OptimiseInfoButton = (ImageButton)findViewById(R.id.opti
    miseInfoButton);
329.                OptimiseInfoButton.setOnClickListener(new View.OnClickListener() {
330.                    @Override
331.                    public void onClick(View v) {
332.                        OptimiseInfoWindow();
333.                    }
334.                });
335.
336.
337.
338.
```

```
339.                ((MachiningData)getApplicationContext()).setFilteredToolList(filtere
    dTools);
340.
341.            } //onCreate
342.
343.
344.            public void OptimiseInfoWindow(){
345.                try {
346.                    // get a reference to the already created main layout
347.                    final ScrollView mainLayout = (ScrollView) findViewById(R.id.con
    tainer);
348.
349.                    // inflate the layout of the popup window
350.                    LayoutInflater inflater = (LayoutInflater) getSystemService(LAYO
    UT_INFLATER_SERVICE);
351.                    final View popupView = inflater.inflate(R.layout.fragment_optimi
    se_info_popup, null);
352.
353.                    // create the popup window
354.                    //boolean focusable = true; // lets taps outside the popup also
    dismiss it
355.                    final PopupWindow popupWindow = new PopupWindow(popupView, 1, 1,
    true);
356.                    popupWindow.setWidth(900);
357.                    popupWindow.setHeight(ViewGroup.LayoutParams.WRAP_CONTENT);
358.
359.                    popupWindow.setBackgroundDrawable(new ColorDrawable(Color.WHITE)
    );
360.                    // show the popup window
361.                    popupWindow.showAtLocation(mainLayout, Gravity.CENTER, 0, 0);
362.
363.                    // TextView has links specified by putting <a> tags in the strin
    g
364.                    // resource.  By default these links will appear but not
365.                    // respond to user input.  To make them active, you need to
366.                    // call setMovementMethod() on the TextView object.
367.
368.                    TextView OptimiseInfoText = (TextView)popupView.findViewById(R.i
    d.OptimiseInfoText);
369.                    OptimiseInfoText.setMovementMethod(LinkMovementMethod.getInstanc
    e());
370.
371.
372.
373.                    }
374.
375.                    // dismiss the popup window when touched
376.                    //popupView.setOnTouchListener(new View.OnTouchListener() {
377.                    //   @Override
378.                    //   public boolean onTouch(View v, MotionEvent event) {
379.                    //       popupWindow.dismiss();
380.                    //       return true;
381.                    //  }
382.                    // });
383.
384.
385.                catch (Exception e) {
386.                    e.printStackTrace();
387.                }
388.            }
389.
390.
391.
392.            public void OptimiseTools(View view) {
393.                Intent optimise_tools_intent = new Intent(getApplicationContext(), O
    ptimise_tools.class);
```

```
394.                double [] CriteriaWeightingMatrix = new double[5];
395.
396.                //grab optimisation parameter weightings
397.                SeekBar power = (SeekBar)findViewById(R.id.powerSeekbar);
398.                int powerWeight = power.getProgress();
399.                //String PowerWeight = Integer.toString(powerWeight);
400.                ////((MachiningData)getApplicationContext()).setPowerWeight(powerWeigh
    t * 1.0);
401.
402.                SeekBar roughness = (SeekBar)findViewById(R.id.roughnessSeekBar);
403.                int roughnessWeight = roughness.getProgress();
404.                ////((MachiningData)getApplicationContext()).setRoughnessWeight(roughn
    essWeight * 1.0);
405.
406.                SeekBar shear = (SeekBar)findViewById(R.id.shearSeekBar);
407.                int shearWeight = shear.getProgress();
408.                ////((MachiningData)getApplicationContext()).setShearWeight(shearWeigh
    t);
409.
410.                SeekBar toolLife = (SeekBar)findViewById(R.id.toolLifeSeekBar);
411.                int toolLifeWeight = toolLife.getProgress();
412.                ////((MachiningData)getApplicationContext()).setToolLifeWeight(toolLif
    eWeight);
413.
414.                SeekBar MMR = (SeekBar)findViewById(R.id.mmrSeekBar);
415.                int mmrWeight = MMR.getProgress();
416.                ////((MachiningData)getApplicationContext()).setMmrWeight(mmrWeight);

417.
418.                CriteriaWeightingMatrix[0] = powerWeight * 1.0;
419.                CriteriaWeightingMatrix[1] = roughnessWeight * 1.0;
420.                CriteriaWeightingMatrix[2] = shearWeight * 1.0;
421.                CriteriaWeightingMatrix[3] = toolLifeWeight * 1.0;
422.                CriteriaWeightingMatrix[4] = mmrWeight * 1.0;
423.
424.
425.                ((MachiningData)getApplicationContext()).setCriteriaWeightingMatrix(
    CriteriaWeightingMatrix);
426.
427.
428.
429.
430.                startActivity(optimise_tools_intent);
431.            }
432.
433.
434.
435.            @Override
436.            public boolean onCreateOptionsMenu(Menu menu) {
437.                // Inflate the menu; this adds items to the action bar if it is pres
    ent.
438.                getMenuInflater().inflate(R.menu.menu_main, menu);//Menu Resource, M
    enu
439.                return true;
440.            }
441.
442.
443.            @Override
444.            public boolean onOptionsItemSelected(MenuItem item) {
445.                switch (item.getItemId()) {
446.                    case R.id.item1:
447.                        Intent intent = new Intent(this, Machine_management.class);

448.
449.                        startActivity(intent);
450.                        return true;
```

```
451.
452.                     default:
453.                         return super.onOptionsItemSelected(item);
454.                 }
455.             }
456.
457.
458.         }
```

Tool_filter_results_contour.java

```
1.  package com.nishen.machiningapp.activities;
2.
3.  import android.content.Intent;
4.  import android.database.Cursor;
5.  import android.graphics.Color;
6.  import android.graphics.drawable.ColorDrawable;
7.  import android.os.Bundle;
8.  import android.support.v7.app.AppCompatActivity;
9.  import android.text.Html;
10. import android.text.method.LinkMovementMethod;
11. import android.view.Gravity;
12. import android.view.LayoutInflater;
13. import android.view.Menu;
14. import android.view.MenuItem;
15. import android.view.View;
16. import android.view.ViewGroup;
17. import android.widget.ImageButton;
18. import android.widget.ListView;
19. import android.widget.PopupWindow;
20. import android.widget.ScrollView;
21. import android.widget.SeekBar;
22. import android.widget.TextView;
23.
24. import com.nishen.machiningapp.R;
25. import com.nishen.machiningapp.activities.Machine_management;
26. import com.nishen.machiningapp.activities.Optimise_tools;
27. import com.nishen.machiningapp.adapters.filteredToolsAdapter;
28. import com.nishen.machiningapp.helpers.DatabaseAccess;
29. import com.nishen.machiningapp.models.MachiningData;
30.
31. import java.util.ArrayList;
32. import java.util.HashMap;
33.
34. /**
35.  * Created by Nishen on 2017/09/19.
36.  */
37.
38. public class Tool_filter_results_contour extends AppCompatActivity {
39.
40.     ListView tool_results_list;
41.     TextView Diameter_header;
42.     TextView CuttingLength_header;
43.     TextView CutDepthPerPass_header;
44.     TextView CutWidthPerPass_header;
45.     TextView CuttingSpeed_header;
46.     TextView CuttingPower_header;
47.     TextView MMR_header;
48.     ArrayList<HashMap<String, String>> filteredTools;
49.
50.     @Override
51.     protected void onCreate(Bundle savedInstanceState) {
52.         super.onCreate(savedInstanceState);
```

```
53.         String cut_profile = ((MachiningData)getApplicationContext()).getProfile();
54.         setTitle(Html.fromHtml("Milling<big>⇒</big>" + cut_profile + "<big>⇒</big> Tools"));
55.         setContentView(R.layout.activity_tool_filter_results);
56.         String materialID;
57.         filteredTools = new ArrayList<>();
58.
59.
60. //Format header columns
61.         Diameter_header = (TextView) findViewById(R.id.Diameter_header);
62.         Diameter_header.setText(Html.fromHtml("Tool Diameter <small><b><em>mm</em></b></small>"));
63.
64.         CuttingLength_header = (TextView) findViewById(R.id.CuttingLength_header);
65.         CuttingLength_header.setText(Html.fromHtml("Cutting Length <small><b><em>mm</em></b></small>"));
66.
67.         CutDepthPerPass_header = (TextView) findViewById(R.id.CutDepthPerPass_header);
68.         CutDepthPerPass_header.setText(Html.fromHtml("Cut depth per pass <small><b><em>mm</em></b></small>"));
69.
70.         CutWidthPerPass_header = (TextView) findViewById(R.id.CutWidthPerPass_header);
71.         CutWidthPerPass_header.setText(Html.fromHtml("Cut width per pass <small><b><em>mm</em></b></small>"));
72.
73.         CuttingSpeed_header = (TextView) findViewById(R.id.CuttingSpeed_header);
74.         CuttingSpeed_header.setText(Html.fromHtml("Cutting Speed <small><b><em>m/min</em></b></small>"));
75.
76.         CuttingPower_header = (TextView) findViewById(R.id.CuttingPower_header);
77.         CuttingPower_header.setText(Html.fromHtml("Cutting Power <small><b><em>kW</em></b></small>"));
78.
79.         MMR_header = (TextView) findViewById(R.id.MMR_header);
80.         MMR_header.setText(Html.fromHtml("Material Removal Rate <small><b><em>mm<sup><small>3</small></sup>/min</em></b></small>"));
81.
82. //Format header columns
83.
84. // Grab input data from MachiningData global class
85.         //Bundle tool_search_bundle = getIntent().getExtras();
86.         materialID = ((MachiningData)getApplicationContext()).getSelectedMaterial();
87.
88.         //String cut_length = tool_search_bundle.getString("cut_length");
89.         //String cut_width = tool_search_bundle.getString("cut_width");
90.         //String cut_depth = tool_search_bundle.getString("cut_depth");
91.         String max_corner_radius = ((MachiningData)getApplicationContext()).getCornerRadius();
92.         //String coolant = tool_search_bundle.getString("coolant");
93.         //String clamping = tool_search_bundle.getString("clamping");
94.         //String operation_type = tool_search_bundle.getString("operation_type");
95.         //String machine = tool_search_bundle.getString("machine");
96.
97.
98.         /** Populating Tool results listview **/
99.         tool_results_list= (ListView) findViewById(R.id.toolList);
100.                 //View content_tool_header = getLayoutInflater().inflate(R.layout.content_tool_header, null);
101.                 //tool_results_list.addHeaderView(content_tool_header);
102.                 DatabaseAccess tool_search_db = DatabaseAccess.getInstance(this);
103.                 tool_search_db.open();
```

```
104.
105.           //Parse input data for database query
106.               Cursor tool_search_list = tool_search_db.FilterToolsCursorContour(cu
    t_profile, materialID);
107.               //Parse input data for database query
108.               tool_search_list.moveToFirst();
109.               while (!tool_search_list.isAfterLast()) {
110.                   String Name = tool_search_list.getString(0);
111.                   String Diameter = tool_search_list.getString(1);
112.                   String CuttingLength = tool_search_list.getString(2);
113.                   String FluteNumber = tool_search_list.getString(3);
114.                   String PartNumber = tool_search_list.getString(4);
115.                   String ToolShape = tool_search_list.getString(5);
116.                   String dmm = tool_search_list.getString(6);
117.                   String l2 = tool_search_list.getString(7);
118.                   String re1 = tool_search_list.getString(8);
119.                   String rakeAngle = tool_search_list.getString(9);
120.
121.                   Cursor MaterialData = tool_search_db.getMaterialData(materialID)
    ;
122.                   MaterialData.moveToFirst();
123.                   String HB = MaterialData.getString(1);
124.                   String UTS = MaterialData.getString(2);
125.                   String kc = MaterialData.getString(3);
126.                   String Yield = MaterialData.getString(4);
127.
128.
129.       /**     Calculate power, etc.. and filter hashmap for individual tool **/
130.
131.                   double pi = Math.PI;
132.
133.                   //Material removal rate calculations
134.                   double diameter = Double.parseDouble(Diameter);
135.
136.
137.                   double CutDepth = Double.parseDouble(((MachiningData)getApplicat
    ionContext()).getUserCutDepth());
138.                   double CutWidth = Double.parseDouble(((MachiningData)getApplicat
    ionContext()).getUserCutWidth());
139.                   double Vc = Double.parseDouble(((MachiningData)getApplicationCon
    text()).getUserCuttingSpeed());
140.
141.                   double SpindleSpeed = Vc / (pi * diameter);
142.                   String Feed_per_tooth = ((MachiningData)getApplicationContext())
    .getUserFeedPerTooth();
143.                   double Fz = Double.parseDouble(Feed_per_tooth);
144.                   double zn = Double.parseDouble(FluteNumber);
145.                   double FeedVelocity = SpindleSpeed * Fz * zn;
146.
147.                   double MMR = CutDepth * CutWidth * FeedVelocity; // Q
148.                   //Material removal rate calculations
149.
150.                   String CuttingSpeed = Double.toString(Vc);
151.                   String Cut_depth = Double.toString(CutDepth);
152.                   String Cut_width = Double.toString(CutWidth);
153.                   //String Material_removal_rate = Double.toString(MMR);
154.
155.
156.                   String Material_removal_rate = Double.toString(MMR);
157.
158.                   //Cutting power calculations
159.                   double SpecificCuttingEnergy = Double.parseDouble(kc); // Ks
160.                   double CuttingPower = MMR * SpecificCuttingEnergy;
161.                   //Cutting power calculations
162.
```

```
163.                    String Cutting_power = Double.toString(CuttingPower / (60 * 1000
    )); //kW formatting
164.
165.                    //Cutting force calculations
166.
167.                    double CuttingForce = SpecificCuttingEnergy * CutDepth * Fz;
168.                    //Cutting force calculations
169.
170.
171.                    //Shear plane deformation calculations
172.
173.                    double UTStrength = Double.parseDouble(UTS);
174.                    double YieldStrength = Double.parseDouble(Yield);
175.                    double ChipCompressionRatio = UTStrength/YieldStrength;  //Very
    similar to cutting ratio
176.
177.                    double Gamma0 = Double.parseDouble(rakeAngle); //tool rake angle

178.
179.                    //ChipCompressionRatio = cos(phi - Gamma0) / sin (phi)
180.                    //Solving for phi (shear angle)
181.                    double phi = Math.atan(Math.cos(Gamma0) / (ChipCompressionRatio
    - Math.sin(Gamma0)));
182.
183.
184.                    //double phi = (pi / 4)-(Beta - Gamma0) ;
185.                    //Solving for beta (tool-interface friction)
186.                    double Beta = (pi / 4) + Gamma0 - phi;
187.
188.                    double ShearStrain = Math.abs(Math.cos(Gamma0) / (Math.sin(phi)
    - Math.cos(phi - Gamma0)));
189.
190.                    //Shear plane deformation calculations
191.
192.                    String Shear_strain = Double.toString(ShearStrain);
193.
194.                    //Tool wear calculations
195.                    double n = 0.5; // work hardening factor of material. May be unn
    ecessary in comparison
196.                    double Py = 1; // yield strength of material.
197.                    double Em = 1; // Elastic modulus of material.
198.                    double Kc = 1; //Fracture toughness of material.
199.                    double H = 1; // hardness of material.
200.
201.                    double Fr = CuttingForce / Math.cos(Beta - Gamma0);
202.                    double Wn = Fr / Math.cos(Beta);// Normal load.
203.
204.
205.                    String length_of_cut = ((MachiningData)getApplicationContext()).
    getCutLength(); // from input page
206.                    //if (Length_of_cut < 1) {
207.                    double  Length_of_cut = 10.0;
208.                    //}
209.
210.                    double ChipNumber = Length_of_cut / Fz;
211.
212.                    double ContactLength = Math.sqrt(CutDepth * diameter);
213.
214.                    double L = ChipNumber * ContactLength / zn ; // sliding distance
    per tooth
215.
216.                    double ToolWearOrig = (n * n) * ((Py * Em *(Math.pow(Wn, 3 / 2))
    )/(Kc * Kc * (Math.pow(H, 3 / 2)))) * L ;
217.                    double ToolWear = Math.abs(ToolWearOrig / 1000);
218.
219.                    //Tool wear calculations
```

```
220.
221.                     String Tool_life = Double.toString(ToolWear);
222.
223.
224.                     //Surface roughness calculations
225.                     //tool tip radius
226.                     double rn = Double.parseDouble(re1);
227.                     double RoughnessOrig = (Fz * Fz) / (31.2 * rn) ; //Ra (mm)
228.                     double Roughness = RoughnessOrig * 1000;
229.                     //Surface roughness calculations
230.
231.                     String Surface_roughness = Double.toString(Roughness);
232.
233.
234.                     /**     Calculate power, etc.. and filter hashmap for individual
    tool  **/
235.
236.                     HashMap<String, String> tool = new HashMap<>();
237.                     //add each value to temporary hashmap
238.
239.                     tool.put("Name", Name);
240.                     tool.put("Diameter", Diameter);
241.                     tool.put("CuttingLength", CuttingLength);
242.                     tool.put("FluteNumber", FluteNumber);
243.                     tool.put("PartNumber", PartNumber);
244.                     tool.put("ToolShape", ToolShape);
245.                     tool.put("dmm", dmm);
246.                     tool.put("l2", l2);
247.                     tool.put("re1", re1);
248.                     tool.put("rakeAngle", rakeAngle);
249.                     tool.put("CutDepth", Cut_depth);
250.                     tool.put("CutWidth", Cut_width);
251.                     tool.put("Fz", Feed_per_tooth);
252.                     tool.put("CuttingSpeed", CuttingSpeed);
253.                     tool.put("CuttingPower", Cutting_power);
254.                     tool.put("Roughness", Surface_roughness);
255.                     tool.put("Shear", Shear_strain);
256.                     tool.put("ToolLife", Tool_life);
257.                     tool.put("MMR", Material_removal_rate);
258.                     filteredTools.add(tool);
259.
260.
261.
262.
263.                     tool_search_list.moveToNext();
264.                 }   //while loop
265.
266.
267.
268.                 tool_search_list.close();
269.
270.
271.
272.                 filteredToolsAdapter tool_filter_adapter = new filteredToolsAdapter(
    this, filteredTools);
273.                 tool_results_list.setAdapter(tool_filter_adapter);
274.                 tool_search_db.close();
275.
276.        /** Populating Tool results listview **/
277.
278.                 ImageButton OptimiseInfoButton = (ImageButton)findViewById(R.id.opti
    miseInfoButton);
279.                 OptimiseInfoButton.setOnClickListener(new View.OnClickListener() {
280.                     @Override
281.                     public void onClick(View v) {
282.                         OptimiseInfoWindow();
```

```
283.                     }
284.                 });
285.
286.
287.
288.
289.
290.                 ((MachiningData)getApplicationContext()).setFilteredToolList(filtere
    dTools);
291.
292.             } //onCreate
293.
294.
295.             public void OptimiseInfoWindow(){
296.                 try {
297.                     // get a reference to the already created main layout
298.                     final ScrollView mainLayout = (ScrollView) findViewById(R.id.con
    tainer);
299.
300.                     // inflate the layout of the popup window
301.                     LayoutInflater inflater = (LayoutInflater) getSystemService(LAYO
    UT_INFLATER_SERVICE);
302.                     final View popupView = inflater.inflate(R.layout.fragment_optimi
    se_info_popup, null);
303.
304.                     // create the popup window
305.                     //boolean focusable = true; // lets taps outside the popup also
    dismiss it
306.                     final PopupWindow popupWindow = new PopupWindow(popupView, 1, 1,
    true);
307.                     popupWindow.setWidth(900);
308.                     popupWindow.setHeight(ViewGroup.LayoutParams.WRAP_CONTENT);
309.
310.                     popupWindow.setBackgroundDrawable(new ColorDrawable(Color.WHITE)
    );
311.                     // show the popup window
312.                     popupWindow.showAtLocation(mainLayout, Gravity.CENTER, 0, 0);
313.
314.                     // TextView has links specified by putting <a> tags in the strin
    g
315.                     // resource.  By default these links will appear but not
316.                     // respond to user input.  To make them active, you need to
317.                     // call setMovementMethod() on the TextView object.
318.
319.                     TextView OptimiseInfoText = (TextView)popupView.findViewById(R.i
    d.OptimiseInfoText);
320.                     OptimiseInfoText.setMovementMethod(LinkMovementMethod.getInstanc
    e());
321.
322.
323.
324.                     }
325.
326.                     // dismiss the popup window when touched
327.                     //popupView.setOnTouchListener(new View.OnTouchListener() {
328.                     //   @Override
329.                     //   public boolean onTouch(View v, MotionEvent event) {
330.                     //       popupWindow.dismiss();
331.                     //       return true;
332.                     //  }
333.                     // });
334.
335.
336.                 catch (Exception e) {
337.                     e.printStackTrace();
338.                 }
```

```
339.            }
340.
341.
342.
343.            public void OptimiseTools(View view) {
344.                Intent optimise_tools_intent = new Intent(getApplicationContext(), O
    ptimise_tools.class);
345.                double [] CriteriaWeightingMatrix = new double[5];
346.
347.                //grab optimisation parameter weightings
348.                SeekBar power = (SeekBar)findViewById(R.id.powerSeekbar);
349.                int powerWeight = power.getProgress();
350.                //String PowerWeight = Integer.toString(powerWeight);
351.                ////((MachiningData)getApplicationContext()).setPowerWeight(powerWeigh
    t * 1.0);
352.
353.                SeekBar roughness = (SeekBar)findViewById(R.id.roughnessSeekBar);
354.                int roughnessWeight = roughness.getProgress();
355.                ////((MachiningData)getApplicationContext()).setRoughnessWeight(roughn
    essWeight * 1.0);
356.
357.                SeekBar shear = (SeekBar)findViewById(R.id.shearSeekBar);
358.                int shearWeight = shear.getProgress();
359.                ////((MachiningData)getApplicationContext()).setShearWeight(shearWeigh
    t);
360.
361.                SeekBar toolLife = (SeekBar)findViewById(R.id.toolLifeSeekBar);
362.                int toolLifeWeight = toolLife.getProgress();
363.                ////((MachiningData)getApplicationContext()).setToolLifeWeight(toolLif
    eWeight);
364.
365.                SeekBar MMR = (SeekBar)findViewById(R.id.mmrSeekBar);
366.                int mmrWeight = MMR.getProgress();
367.                ////((MachiningData)getApplicationContext()).setMmrWeight(mmrWeight);

368.
369.                CriteriaWeightingMatrix[0] = powerWeight * 1.0;
370.                CriteriaWeightingMatrix[1] = roughnessWeight * 1.0;
371.                CriteriaWeightingMatrix[2] = shearWeight * 1.0;
372.                CriteriaWeightingMatrix[3] = toolLifeWeight * 1.0;
373.                CriteriaWeightingMatrix[4] = mmrWeight * 1.0;
374.
375.
376.                ((MachiningData)getApplicationContext()).setCriteriaWeightingMatrix(
    CriteriaWeightingMatrix);
377.
378.
379.
380.
381.                startActivity(optimise_tools_intent);
382.            }
383.
384.
385.
386.            @Override
387.            public boolean onCreateOptionsMenu(Menu menu) {
388.                // Inflate the menu; this adds items to the action bar if it is pres
    ent.
389.                getMenuInflater().inflate(R.menu.menu_main, menu);//Menu Resource, M
    enu
390.                return true;
391.            }
392.
393.
394.            @Override
395.            public boolean onOptionsItemSelected(MenuItem item) {
```

```
396.                switch (item.getItemId()) {
397.                    case R.id.item1:
398.                        Intent intent = new Intent(this, Machine_management.class);
399.
400.                        startActivity(intent);
401.                        return true;
402.
403.                    default:
404.                        return super.onOptionsItemSelected(item);
405.                }
406.            }
407.
408.
409.        }
```

Optimise_tools.java

```
1.  package com.nishen.machiningapp.activities;
2.
3.  import android.graphics.Bitmap;
4.  import android.graphics.Color;
5.  import android.graphics.drawable.ColorDrawable;
6.  import android.support.constraint.ConstraintLayout;
7.  import android.support.v7.app.AppCompatActivity;
8.  import android.os.Bundle;
9.  import android.text.Html;
10. import android.view.Gravity;
11. import android.view.LayoutInflater;
12. import android.view.View;
13. import android.view.ViewGroup;
14. import android.widget.AdapterView;
15. import android.widget.ImageView;
16. import android.widget.ListView;
17. import android.widget.PopupWindow;
18. import android.widget.TextView;
19.
20. import com.nishen.machiningapp.adapters.OptimisedToolsAdapter;
21. import com.nishen.machiningapp.R;
22. import com.nishen.machiningapp.utils.TOPSIS;
23. import com.nishen.machiningapp.helpers.DatabaseAccess;
24. import com.nishen.machiningapp.models.MachiningData;
25.
26. import java.text.DecimalFormat;
27. import java.util.ArrayList;
28. import java.util.Collections;
29. import java.util.Comparator;
30. import java.util.HashMap;
31.
32. public class Optimise_tools extends AppCompatActivity {
33.     ArrayList<HashMap<String, String>> filteredToolList;
34.     TextView Diameter_header;
35.     TextView CutDepthPerPass_header;
36.     TextView CutWidthPerPass_header;
37.     TextView CuttingSpeed_header;
38.     TextView CuttingPower_header;
39.     TextView MMR_header;
40.     TextView testV;
41.
42.     @Override
43.     protected void onCreate(Bundle savedInstanceState) {
44.         super.onCreate(savedInstanceState);
```

```
45.         String cut_profile = ((MachiningData)getApplicationContext()).getProfile();
46.         setTitle(Html.fromHtml("Milling<big>⇒</big>" + cut_profile + "<big>⇒</big>
    Tools<big>⇒</big>Optimise"));
47.         setContentView(R.layout.activity_optimise_tools);
48.
49.
50.
51.         //Format header columns
52.         Diameter_header = (TextView) findViewById(R.id.Diameter_header);
53.         Diameter_header.setText(Html.fromHtml("Tool Diameter <small><b><em>mm</em><
    /b></small>"));
54.
55.         CutDepthPerPass_header = (TextView) findViewById(R.id.CutDepthPerPass_heade
    r);
56.         CutDepthPerPass_header.setText(Html.fromHtml("Cut depth per pass <small><b>
    <em>mm</em></b></small>"));
57.
58.         CutWidthPerPass_header = (TextView) findViewById(R.id.CutWidthPerPass_heade
    r);
59.         CutWidthPerPass_header.setText(Html.fromHtml("Cut width per pass <small><b>
    <em>mm</em></b></small>"));
60.
61.         CuttingSpeed_header = (TextView) findViewById(R.id.CuttingSpeed_header);
62.         CuttingSpeed_header.setText(Html.fromHtml("Cutting Speed <small><b><em>m/mi
    n</em></b></small>"));
63.
64.         CuttingPower_header = (TextView) findViewById(R.id.CuttingPower_header);
65.         CuttingPower_header.setText(Html.fromHtml("Cutting Power <small><b><em>kW</
    em></b></small"));
66.
67.         MMR_header = (TextView) findViewById(R.id.MMR_header);
68.         MMR_header.setText(Html.fromHtml("Material Removal Rate <small><b><em>mm<su
    p><small>3</small></sup>/min</em></b></small>"));
69.
70. //Format header columns
71.
72.        //new MachiningTOPSIS().execute();
73.
74.         filteredToolList = ((MachiningData)getApplicationContext()).getFilteredTool
    List();
75.         double [] CriteriaWeightingMatrix = ((MachiningData)getApplicationContext()
    ).getCriteriaWeightingMatrix();
76.         int rows_alternatives = filteredToolList.size();
77.         int columns_criteria = 5;
78.
79.         //TOPSIS_List = new ArrayList<>();
80.         double [] [] TOPSISmatrix = new double[rows_alternatives][columns_criteria]
    ;
81.
82.         for (int row = 0; row < rows_alternatives; row++) {
83.             HashMap<String, String> ToolData = filteredToolList.get(row);
84.             HashMap<String, String> TOPSISelement = new HashMap<>();
85.
86.             DecimalFormat formatter2 = new DecimalFormat("#0.00");
87.
88.             String Name = ToolData.get("Name");
89.             String Diameter = ToolData.get("Diameter");
90.             String Power = ToolData.get("CuttingPower");
91.             String Roughness = ToolData.get("Roughness");
92.             String Shear = ToolData.get("Shear");
93.             String ToolLife = ToolData.get("ToolLife");
94.             String MMR = ToolData.get("MMR");
95.
96.             TOPSISmatrix[row][0] = Double.parseDouble(Power);       //Must minimise
```

```
97.             TOPSISmatrix[row][1] = Double.parseDouble(Roughness);   //Must minimise
98.             TOPSISmatrix[row][2] = Double.parseDouble(Shear);       //Must minimise
99.             TOPSISmatrix[row][3] = Double.parseDouble(ToolLife);    //Must minimise
100.                TOPSISmatrix[row][4] = Double.parseDouble(MMR);         //Must m
    aximise
101.
102.            }
103.
104.
105.
106.
107.            TOPSIS machiningTOPSIS = new TOPSIS();
108.            machiningTOPSIS.setrows_alternatives(rows_alternatives);
109.            machiningTOPSIS.setcolumns_criteria(columns_criteria);
110.            machiningTOPSIS.setTOPSISmatrix(TOPSISmatrix);
111.            machiningTOPSIS.setCriteriaWeightingMatrix(CriteriaWeightingMatrix);
112.            double [] UnorderedToolScores = machiningTOPSIS.calculate();
113.
114.
115.            //add closeness coefficient (score) table into tool data table.
116.            for (int row = 0; row < rows_alternatives; row++) {
117.                HashMap<String, String> TOPSISelement = filteredToolList.get(row
    );
118.                double toolScore = UnorderedToolScores[row];
119.                DecimalFormat formatter2 = new DecimalFormat("#0.00");
120.                String toolScoreShort = formatter2.format(toolScore*100);
121.                TOPSISelement.put("Score", toolScoreShort);
122.            }
123.
124.
125.            Collections.sort(filteredToolList, new ToolScoreComparator());
126.
127.            ArrayList<HashMap<String, String>> Top5Tools = new ArrayList<>();
128.
129.            for (int position = 0; position < 4; position++){
130.                Top5Tools.add(filteredToolList.get(position));
131.            }
132.
133.
134.
135.            ListView OptimisedToolList = (ListView) findViewById(R.id.OptimisedT
    oolList);
136.            OptimisedToolsAdapter adapter = new OptimisedToolsAdapter(this,Top5T
    ools);
137.            OptimisedToolList.setAdapter(adapter);
138.            adapter.notifyDataSetChanged();
139.
140.            testV = (TextView)findViewById(R.id.ToolName);
141.
142.
143.            /**OptimisedToolList.setOnItemSelectedListener(new AdapterView.OnIte
    mSelectedListener() {
144.                @Override
145.                public void onItemSelected(AdapterView<?> parent, View view, int
    position, long id) {
146.                    String FamilyName = filteredToolList.get(position).get("Name
    ").toLowerCase();
147.                    final int imgId = getResources().getIdentifier(FamilyName ,
    "drawable", getPackageName());
148.                    //ToolDetailsWindow(FamilyName);
149.                }
150.
```

```
151.                     @Override
152.                     public void onNothingSelected(AdapterView<?> parent) {
153.
154.                     }
155.                 });*/
156.
157.                 OptimisedToolList.setOnItemClickListener(new AdapterView.OnItemClick
    Listener() {
158.                     @Override
159.                     public void onItemClick(AdapterView<?> parent, View view, int po
    sition, long id) {
160.                         ToolDetailsWindow(position);
161.                     }
162.                 });
163.
164.
165.             } //onCreate
166.
167.
168.             public class ToolScoreComparator
169.                     implements Comparator<HashMap<String, String>>
170.             {
171.                 @Override
172.                 public int compare(HashMap<String, String> Tool1,
173.                                    HashMap<String, String> Tool2)
174.                 {
175.                     return Double.compare(Double.parseDouble(Tool2.get("Score")), Do
    uble.parseDouble(Tool1.get("Score"))); //Tool2 compared to Tool1 so that sorting is
    in descending order.
176.                 }
177.             }
178.
179.
180.
181.
182.
183.             public void ToolDetailsWindow(int position){
184.                 try {
185.                     // get a reference to the already created main layout
186.                     final ConstraintLayout mainLayout = (ConstraintLayout) findViewB
    yId(R.id.container);
187.
188.                     // inflate the layout of the popup window
189.                     LayoutInflater inflater = (LayoutInflater) getSystemService(LAYO
    UT_INFLATER_SERVICE);
190.                     final View popupView = inflater.inflate(R.layout.content_optimis
    ed_tool_details, null);
191.
192.                     // create the popup window
193.                     //boolean focusable = false; // lets taps outside the popup also
    dismiss it
194.                     final PopupWindow popupWindow = new PopupWindow(popupView, 1, 1,
    false);
195.                     popupWindow.setWidth(ViewGroup.LayoutParams.MATCH_PARENT);
196.                     popupWindow.setHeight(ViewGroup.LayoutParams.WRAP_CONTENT);
197.
198.                     popupWindow.setBackgroundDrawable(new ColorDrawable(Color.WHITE)
    );
199.                     // show the popup window
200.                     popupWindow.showAtLocation(mainLayout, Gravity.BOTTOM, 0, 0);
201.
202.                     String FamilyName = filteredToolList.get(position).get("Name");

203.                     //String FamilyName = filteredToolList.get(position).get("Name")
    .toLowerCase();
```

```
                    String ToolName = filteredToolList.get(position).get("PartNumber
");
                    String ToolShape = filteredToolList.get(position).get("ToolShape
");
                    String Dc = filteredToolList.get(position).get("Diameter");
                    String dmm = filteredToolList.get(position).get("dmm");
                    String ap = filteredToolList.get(position).get("CuttingLength");

                    String l2 = filteredToolList.get(position).get("l2");
                    String re1 = filteredToolList.get(position).get("re1");
                    String flutes = filteredToolList.get(position).get("FluteNumber"
);
                    String rake = filteredToolList.get(position).get("rakeAngle");

                    TextView Tool_name = (TextView) popupView.findViewById(R.id.Tool
Name);
                    TextView Tool_shape = (TextView) popupView.findViewById(R.id.Too
lShape);
                    TextView dc = (TextView) popupView.findViewById(R.id.Dc);
                    TextView Dmm = (TextView) popupView.findViewById(R.id.dmm);
                    TextView Ap = (TextView) popupView.findViewById(R.id.ap);
                    TextView L2 = (TextView) popupView.findViewById(R.id.l2);
                    TextView Re1 = (TextView) popupView.findViewById(R.id.re1);
                    TextView Flutes = (TextView) popupView.findViewById(R.id.Flutes)
;
                    TextView Rake = (TextView) popupView.findViewById(R.id.Rake);

                    Tool_name.setText("Tool name: " + ToolName);
                    Tool_shape.setText("Tool shape: " + ToolShape);
                    dc.setText(Html.fromHtml("D<sub><small>c</small></sub>: " + Dc +
"mm"));
                    Dmm.setText(Html.fromHtml("Dm<sub><small>m</small></sub>: " + dm
m + "mm"));
                    Ap.setText(Html.fromHtml("A<sub><small>p</small></sub>: " + ap +
"mm"));
                    L2.setText(Html.fromHtml("l<sub><small>2</small></sub>: " + l2 +
"mm"));
                    Re1.setText(Html.fromHtml("r<sub><small>e1</small></sub>: " + re
1 + "mm"));
                    Flutes.setText("Flutes: " + flutes);
                    Rake.setText(Html.fromHtml("Rake: " + rake + "<sup><small>o</sma
ll></sup>"));

                    ImageView ToolDiagram = (ImageView)popupView.findViewById(R.id.O
ptimisedToolDiagram);
                    //int imageID = getResources().getIdentifier(FamilyName, "drawab
le", getPackageName());
                    //ToolDiagram.setBackground(getDrawable(imageID));
                    DatabaseAccess databaseAccess = DatabaseAccess.getInstance(getAp
plicationContext());
                    databaseAccess.open();
                    Bitmap toolDiagram = databaseAccess.getToolDiagram(FamilyName);

                    ToolDiagram.setImageBitmap(toolDiagram);
                    //TODO check for zoomable and pannable with a click. Flushing pi
ctures from ram.
                }


                catch (Exception e) {
                    e.printStackTrace();
                }
            };


        }
```

Machine_management.java

```
1.  package com.nishen.machiningapp.activities;
2.
3.  import android.content.Context;
4.  import android.content.Intent;
5.  import android.database.Cursor;
6.  import android.support.v7.app.AppCompatActivity;
7.  import android.os.Bundle;
8.  import android.text.Html;
9.  import android.view.View;
10. import android.view.inputmethod.InputMethodManager;
11. import android.widget.Button;
12. import android.widget.EditText;
13. import android.widget.ListView;
14. import android.widget.TextView;
15. import android.widget.Toast;
16.
17. import com.nishen.machiningapp.R;
18. import com.nishen.machiningapp.helpers.DatabaseAccess;
19. import com.nishen.machiningapp.adapters.machineArrayAdapter;
20.
21. import java.util.ArrayList;
22. import java.util.HashMap;
23.
24. public class Machine_management extends AppCompatActivity {
25.     EditText Power;
26.     EditText Name;
27.     Button AddMachine;
28.
29.     @Override
30.     protected void onCreate(Bundle savedInstanceState) {
31.         super.onCreate(savedInstanceState);
32.         setTitle("Machine Management");
33.         setContentView(R.layout.activity_machine_management);
34.
35.         TextView machinePowerHeader = (TextView)findViewById(R.id.Machine_power_hea
    der);
36.         machinePowerHeader.setText(Html.fromHtml("Power <small><b><em>kW</em></b></
    small>"));
37.
38.
39.         final ListView MachinesList = (ListView)findViewById(R.id.MyMachinesList);

40.         ArrayList<HashMap<String, String>> MyMachineList = new ArrayList<HashMap<St
    ring, String>>();
41.         DatabaseAccess machinesDB = DatabaseAccess.getInstance(getApplicationContex
    t());
42.         machinesDB.open();
43.         Cursor MyMachineCursor = machinesDB.getMyMachines();
44.         MyMachineCursor.moveToFirst();
45.         while (!MyMachineCursor.isAfterLast()) {
46.             HashMap<String, String> machine = new HashMap<>();
47.             //add each value to temporary hashmap
48.             machine.put("Name", MyMachineCursor.getString(0));
49.             machine.put("Power", MyMachineCursor.getString(1));
50.             //add machine to machine list
51.             MyMachineList.add(machine);
52.             MyMachineCursor.moveToNext();
53.         }
54.         MyMachineCursor.close();
55.
```

```
56.         final machineArrayAdapter machineArrayAdapter = new machineArrayAdapter(get
    ApplicationContext(),Machine_management.this, MyMachineList);
57.         MachinesList.setAdapter(machineArrayAdapter);
58.         machineArrayAdapter.notifyDataSetChanged();
59.
60.
61.
62.
63.
64.         Name = (EditText)findViewById(R.id.MachineName);
65.
66.         Power = (EditText)findViewById(R.id.MachinePower);
67.
68.         AddMachine = (Button)findViewById(R.id.addMachine);
69.         AddMachine.setOnClickListener(new View.OnClickListener() {
70.             @Override
71.             public void onClick(View v) {
72.                 String name = Name.getText().toString();
73.                 String power = Power.getText().toString();
74.                 String powerKW = Double.toString(Double.parseDouble(power)*1000);
75.
76.                 if (name.trim().equals("") || power.trim().equals("")){
77.                     Toast.makeText(getApplicationContext(), "Please fill in the det
    ails", Toast.LENGTH_SHORT).show();
78.                 } else {
79.                     DatabaseAccess databaseAccess = DatabaseAccess.getInstance(getA
    pplicationContext());
80.                     databaseAccess.open();
81.                     databaseAccess.setMachine(name, powerKW);
82.                     databaseAccess.close();
83.                     startMachineManagement();
84.
85.                     InputMethodManager imm = (InputMethodManager)getSystemService(C
    ontext.INPUT_METHOD_SERVICE);
86.                     imm.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
    0);
87.                     Toast.makeText(getApplicationContext(), "Added machine", Toast.
    LENGTH_SHORT).show();
88.                 }
89.             }
90.         });
91.
92.     } //OnCreate
93.
94.     public void startMachineManagement(){
95.         Intent intent = new Intent(this, Machine_management.class);
96.         startActivity(intent);
97.         finish();
98.     }
99.
100.
101.
102.    }
```

Adapters

filteredToolsAdapter.java

```
1.  package com.nishen.machiningapp.adapters;
2.
3.  import android.app.Activity;
4.  import android.view.LayoutInflater;
```

```
5.  import android.view.View;
6.  import android.view.ViewGroup;
7.  import android.widget.BaseAdapter;
8.  import android.widget.TextView;
9.
10. import com.nishen.machiningapp.R;
11.
12. import java.text.DecimalFormat;
13. import java.util.ArrayList;
14. import java.util.HashMap;
15.
16. /**
17.  * Created by Nishen on 2017/10/07.
18.  */
19.
20. public class filteredToolsAdapter extends BaseAdapter {
21.
22.
23.     private ArrayList<HashMap<String, String>> myToolsList;
24.     private Activity activity;
25.     private ViewHolder viewHolder;
26.
27.
28.
29.     public filteredToolsAdapter(Activity activity, ArrayList<HashMap<String, String
    >> myToolsList) {
30.         super();
31.         this.activity = activity;
32.         this.myToolsList = myToolsList;
33.     }
34.
35.     @Override
36.     public int getCount() {
37.         return myToolsList.size();
38.     }
39.
40.     @Override
41.     public Object getItem(int position) {
42.         return myToolsList.get(position);
43.     }
44.
45.     @Override
46.     public long getItemId(int position) {
47.         return 0;
48.     }
49.
50.     public View getView(int position, View convertView, ViewGroup parent) {
51.         //ViewHolder viewHolder;
52.         LayoutInflater inflater = activity.getLayoutInflater();
53.
54.         if (convertView == null) {
55.             convertView = inflater.inflate(R.layout.content_filtered_tool_single_it
    em, null);
56.
57.             viewHolder = new ViewHolder(convertView);
58.             convertView.setTag(viewHolder);
59.
60.
61.         } else {
62.             viewHolder = (ViewHolder) convertView.getTag();
63.         }
64.         //grab temporary tool item from arraylist of filtered tools
65.         HashMap<String, String> map = myToolsList.get(position);
66.
67.         DecimalFormat formatter0 = new DecimalFormat("#0");
68.         DecimalFormat formatter1 = new DecimalFormat("#0.0");
```

```
69.         DecimalFormat formatter2 = new DecimalFormat("#0.00");
70.
71.         double CuttingSpeed = Double.parseDouble(map.get("CuttingSpeed"));
72.         double CutWidth = Double.parseDouble(map.get("CutWidth"));
73.         double CutDepth = Double.parseDouble(map.get("CutDepth"));
74.         double MMR = Double.parseDouble(map.get("MMR"));
75.         double CuttingPower = Double.parseDouble(map.get("CuttingPower"));
76.
77.         viewHolder.Name.setText(map.get("Name"));
78.         viewHolder.Diameter.setText(map.get("Diameter"));
79.         viewHolder.CuttingLength.setText(map.get("CuttingLength"));
80.         viewHolder.FluteNumber.setText(map.get("FluteNumber"));
81.         viewHolder.CutDepthPerPass.setText(formatter1.format(CutDepth));
82.         viewHolder.CutWidthPerPass.setText(formatter1.format(CutWidth));
83.         viewHolder.MaterialRemovalRate.setText(formatter1.format(MMR));
84.         viewHolder.CuttingSpeed.setText(formatter0.format(CuttingSpeed));
85.         viewHolder.CuttingPower.setText(formatter2.format(CuttingPower));
86.
87.         return convertView;
88.     }
89.
90.     private class ViewHolder {
91.         TextView Name;
92.         TextView Diameter;
93.         TextView CuttingLength;
94.         TextView FluteNumber;
95.         TextView CutDepthPerPass;
96.         TextView CutWidthPerPass;
97.         TextView MaterialRemovalRate;
98.         TextView CuttingSpeed;
99.         TextView CuttingPower;
100.
101.                public ViewHolder(View view) {
102.                    Name = (TextView)view.findViewById(R.id.Name);
103.                    Diameter = (TextView) view.findViewById(R.id.Diameter);
104.                    CuttingLength = (TextView) view.findViewById(R.id.CuttingLength)
    ;
105.                    FluteNumber = (TextView) view.findViewById(R.id.FluteNumber);
106.                    CutDepthPerPass = (TextView) view.findViewById(R.id.CutDepthPerP
    ass);
107.                    CutWidthPerPass = (TextView) view.findViewById(R.id.CutWidthPerP
    ass);
108.                    MaterialRemovalRate = (TextView) view.findViewById(R.id.MMR);
109.                    CuttingSpeed = (TextView) view.findViewById(R.id.CuttingSpeed);

110.                    CuttingPower = (TextView) view.findViewById(R.id.CuttingPower);

111.                }
112.
113.            }
114.
115.
116.        }
```

machineArrayAdapter.java

```
1.  package com.nishen.machiningapp.adapters;
2.
3.  import android.app.Activity;
4.  import android.content.Context;
5.  import android.content.Intent;
```

```
6.  import android.view.LayoutInflater;
7.  import android.view.View;
8.  import android.view.ViewGroup;
9.  import android.widget.BaseAdapter;
10. import android.widget.TextView;
11. import android.widget.Toast;
12.
13. import com.nishen.machiningapp.R;
14. import com.nishen.machiningapp.activities.Machine_management;
15. import com.nishen.machiningapp.helpers.DatabaseAccess;
16.
17. import java.text.DecimalFormat;
18. import java.util.ArrayList;
19. import java.util.HashMap;
20.
21. /**
22.  * Created by Nishen on 2017/10/07.
23.  */
24.
25. public class machineArrayAdapter extends BaseAdapter {
26.
27.
28.     public ArrayList<HashMap<String, String>> myMachineList;
29.     private Context context;
30.     Activity activity;
31.     ViewHolder viewHolder;
32.
33.     public machineArrayAdapter(Context context,Activity activity, ArrayList<HashMap
    <String, String>> myMachineList) {
34.         super();
35.         this.context = context;
36.         this.activity = activity;
37.         this.myMachineList = myMachineList;
38.     }
39.
40.     @Override
41.     public int getCount() {
42.         return myMachineList.size();
43.     }
44.
45.     @Override
46.     public Object getItem(int position) {
47.         return myMachineList.get(position);
48.     }
49.
50.     @Override
51.     public long getItemId(int position) {
52.         return 0;
53.     }
54.
55.     public View getView(final int position, View convertView, final ViewGroup paren
    t) {
56.
57.         LayoutInflater inflater = activity.getLayoutInflater();
58.
59.         if (convertView == null) {
60.             convertView = inflater.inflate(R.layout.content_mymachines_list_item, n
    ull);
61.
62.             viewHolder = new ViewHolder(convertView);
63.             convertView.setTag(viewHolder);
64.         } else {
65.             viewHolder = (ViewHolder) convertView.getTag();
66.         }
67.     // grab temporary material from material arraylist
68.         HashMap<String, String> map = myMachineList.get(position);
```

```
69.
70.         DecimalFormat formatter2 = new DecimalFormat("#0.0");
71.         Double PowerShort = Double.parseDouble(map.get("Power"))/1000;
72.
73.         viewHolder.Number.setText(Integer.toString(position+1));
74.         viewHolder.Name.setText(map.get("Name"));
75.         viewHolder.Power.setText(formatter2.format(PowerShort));
76.
77.         viewHolder.Delete.setOnClickListener(new View.OnClickListener() {
78.             @Override
79.             public void onClick(View v) {
80.                 DatabaseAccess databaseAccess = DatabaseAccess.getInstance(activity
    .getApplicationContext());
81.                 databaseAccess.open();
82.                 databaseAccess.deleteMachine(myMachineList.get(position).get("Name"
    ));
83.                 databaseAccess.close();
84.                 Intent intent = new Intent(activity.getApplicationContext(), Machin
    e_management.class);
85.                 context.startActivity(intent);
86.                 activity.finish();
87.                 Toast.makeText(activity.getApplicationContext(), "Deleted machine",
    Toast.LENGTH_SHORT).show();
88.             }
89.         });
90.
91.
92.
93.         return convertView;
94.     }
95.
96.     private class ViewHolder {
97.         TextView Number;
98.         TextView Name;
99.         TextView Power;
100.                TextView Delete;
101.
102.                ViewHolder(View view) {
103.                    Number = (TextView) view.findViewById(R.id.Number);
104.                    Name = (TextView) view.findViewById(R.id.Name);
105.                    Power = (TextView) view.findViewById(R.id.Machine_power);
106.                    Delete = (TextView)view.findViewById(R.id.DeleteMachine);
107.
108.                }
109.            }
110.        }
```

materialArrayAdapter.java

```
1.  package com.nishen.machiningapp.adapters;
2.
3.  import android.app.Activity;
4.  import android.view.LayoutInflater;
5.  import android.view.View;
6.  import android.view.ViewGroup;
7.  import android.widget.BaseAdapter;
8.  import android.widget.Spinner;
9.  import android.widget.TextView;
10.
11. import com.nishen.machiningapp.R;
12.
```

```
13. import java.util.ArrayList;
14. import java.util.HashMap;
15. import java.util.Map;
16.
17. /**
18.  * Created by Nishen on 2017/10/07.
19.  */
20.
21. public class materialArrayAdapter extends BaseAdapter {
22.
23.
24.     public ArrayList<HashMap<String, String>> myMaterialList;
25.     Activity activity;
26.     ViewHolder viewHolder;
27.
28.     public materialArrayAdapter(Activity activity, ArrayList<HashMap<String, String
    >> myMaterialList) {
29.         super();
30.         this.activity = activity;
31.         this.myMaterialList = myMaterialList;
32.     }
33.
34.     @Override
35.     public int getCount() {
36.         return myMaterialList.size();
37.     }
38.
39.     @Override
40.     public Object getItem(int position) {
41.         return myMaterialList.get(position);
42.     }
43.
44.     @Override
45.     public long getItemId(int position) {
46.         return 0;
47.     }
48.
49.     public View getView(int position, View convertView, ViewGroup parent) {
50.
51.         LayoutInflater inflater = activity.getLayoutInflater();
52.
53.         if (convertView == null) {
54.             convertView = inflater.inflate(R.layout.content_material_custom_item, n
    ull);
55.
56.             viewHolder = new ViewHolder(convertView);
57.             convertView.setTag(viewHolder);
58.         } else {
59.             viewHolder = (ViewHolder) convertView.getTag();
60.         }
61.     // grab temporary material from material arraylist
62.         HashMap<String, String> map = myMaterialList.get(position);
63.
64.         viewHolder.SMG.setText(map.get("SMG"));
65.         viewHolder.Description.setText(map.get("Description"));
66.
67.         return convertView;
68.     }
69.
70.     private class ViewHolder {
71.         TextView SMG;
72.         TextView Description;
73.
74.         ViewHolder(View view) {
75.             SMG = (TextView) view.findViewById(R.id.SMG);
76.             Description = (TextView) view.findViewById(R.id.Description);
```

```
        }
    }
}
```

OptimisedToolsAdapter.java

```
package com.nishen.machiningapp.adapters;

import android.app.Activity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

import com.nishen.machiningapp.R;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.HashMap;

/**
 * Created by Nishen on 2017/10/07.
 */

public class OptimisedToolsAdapter extends BaseAdapter {


    public ArrayList<HashMap<String, String>> BIGmatrix;
    Activity activity;


    HashMap<String, String> element;

    ViewHolder viewHolder;

    public OptimisedToolsAdapter(Activity activity, ArrayList<HashMap<String, Strin
g>> BIGmatrix) {
        super();
        this.activity = activity;
        this.BIGmatrix = BIGmatrix;
    }

    @Override
    public int getCount() {
        return BIGmatrix.size();
    }


    @Override
    public Object getItem(int position) {
        return BIGmatrix.get(position);
    }


    @Override
    public long getItemId(int position) {
        return 0;
    }

    public View getView(int position, View convertView, ViewGroup parent) {
```

```
55.
56.         LayoutInflater inflater = activity.getLayoutInflater();
57.
58.         if (convertView == null) {
59.             convertView = inflater.inflate(R.layout.content_optimised_tool_single_i
    tem, null);
60.
61.             viewHolder = new ViewHolder(convertView);
62.             convertView.setTag(viewHolder);
63.         } else {
64.             viewHolder = (ViewHolder) convertView.getTag();
65.         }
66.     // grab temporary material from material arraylist
67.         element = BIGmatrix.get(position);
68.
69.         DecimalFormat formatter0 = new DecimalFormat("#0");
70.         DecimalFormat formatter1 = new DecimalFormat("#0.0");
71.         DecimalFormat formatter2 = new DecimalFormat("#0.00");
72.
73.         double CuttingSpeed = Double.parseDouble(element.get("CuttingSpeed"));
74.         double CutWidth = Double.parseDouble(element.get("CutWidth"));
75.         double CutDepth = Double.parseDouble(element.get("CutDepth"));
76.         double MMR = Double.parseDouble(element.get("MMR"));
77.         double CuttingPower = Double.parseDouble(element.get("CuttingPower"));
78.
79.         viewHolder.position.setText(Integer.toString(position+1));
80.         viewHolder.name.setText(element.get("Name"));
81.         viewHolder.diameter.setText(element.get("Diameter"));
82.         viewHolder.score.setText(element.get("Score"));
83.         viewHolder.CutDepthPerPass.setText(formatter1.format(CutDepth));
84.         viewHolder.CutWidthPerPass.setText(formatter1.format(CutWidth));
85.         viewHolder.CuttingSpeed.setText(formatter0.format(CuttingSpeed));
86.         viewHolder.power.setText(formatter2.format(CuttingPower));
87.         viewHolder.mmr.setText(formatter1.format(MMR));
88.
89.
90.         return convertView;
91.     }
92.
93.     private class ViewHolder {
94.         TextView position;
95.         TextView name;
96.         TextView diameter;
97.         TextView score;
98.         TextView CutDepthPerPass;
99.         TextView CutWidthPerPass;
100.                TextView CuttingSpeed;
101.                TextView power;
102.                TextView mmr;
103.
104.
105.                ViewHolder(View view) {
106.                    position = (TextView)view.findViewById(R.id.Position);
107.                    name = (TextView)view.findViewById(R.id.Name);
108.                    diameter = (TextView)view.findViewById(R.id.Diameter);
109.                    score = (TextView)view.findViewById(R.id.Score);
110.                    CutWidthPerPass = (TextView) view.findViewById(R.id.CutWidthPerP
    ass);
111.                    CutDepthPerPass = (TextView) view.findViewById(R.id.CutDepthPerP
    ass);
112.                    CuttingSpeed = (TextView) view.findViewById(R.id.CuttingSpeed);

113.                    power = (TextView)view.findViewById(R.id.CuttingPower);
114.                    mmr = (TextView)view.findViewById(R.id.MMR);
115.
116.                }
```

```
            }
        }
```

Helpers

DatabaseOpenHelper.java

```
package com.nishen.machiningapp.helpers;

        import android.content.Context;

        import com.readystatesoftware.sqliteasset.SQLiteAssetHelper;

public class DatabaseOpenHelper extends SQLiteAssetHelper {
    private static final String DATABASE_NAME = "machining.db";
    private static final int DATABASE_VERSION = 1;

    public DatabaseOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
}
```

DatabaseAccess.java

```
package com.nishen.machiningapp.helpers;

/**
 * Created by Nishen on 2017/09/18.
 */

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;

import java.io.ByteArrayInputStream;
import java.util.ArrayList;
import java.util.List;

public class DatabaseAccess {
    private SQLiteOpenHelper openHelper;
    private SQLiteDatabase database;
    private static DatabaseAccess instance;

    /**
     * Private constructor to avoid object creation from outside classes.
     *
     * @param context
     */
    private DatabaseAccess(Context context) {
        this.openHelper = new DatabaseOpenHelper(context);
    }


    /**
     * Return a singleton instance of DatabaseAccess.
```

```
36.      *
37.      * @param context the Context
38.      * @return the instance of DatabaseAccess
39.      */
40.     public static DatabaseAccess getInstance(Context context) {
41.         if (instance == null) {
42.             instance = new DatabaseAccess(context);
43.         }
44.         return instance;
45.     }
46.
47.
48.     /**
49.      * Open the database connection.
50.      */
51.     public void open() {
52.         this.database = openHelper.getWritableDatabase();
53.     }
54.
55.
56.     /**
57.      * Close the database connection.
58.      */
59.     public void close() {
60.         if (database != null) {
61.             this.database.close();
62.         }
63.     }
64.
65.
66.     /**
67.      * Read all xxxxx from the database.
68.      *
69.      * @return a List of quotes
70.      */
71.
72.     //String method. Suitable for single textview.
73.     public List<String> getMaterials() {
74.         List<String> material_list = new ArrayList<>();
75.         Cursor cursor = database.rawQuery("SELECT Description FROM Material GROUP B
    Y ID", null);
76.         cursor.moveToFirst();
77.         while (!cursor.isAfterLast()) {
78.             material_list.add(cursor.getString(0));
79.             cursor.moveToNext();
80.         }
81.         cursor.close();
82.         return material_list;
83.     }
84.
85.     public Cursor getMaterialsCursor() {
86.         Cursor materialsCursor = database.rawQuery("SELECT SMG, Description FROM Ma
    terial", null);
87.         return materialsCursor;
88.
89.     }
90.
91.     public Cursor FilterToolsCursor(String profile, String material) {
92.         Cursor materialCursor = database.rawQuery("SELECT SMG FROM Material WHERE I
    D =" + material, null);
93.         materialCursor.moveToFirst();
94.         String SMG = materialCursor.getString(0);
95.         Cursor cursor = database.rawQuery("SELECT Tool.Name, Dc, ap, zn, Part_No, T
    ool_Shape, dmm, l2, re1, rake, coolant, \"Ap/Dc\", \"Ae/Dc\", \"6\", \"8\", \"10\",
    \"12\", Vc FROM Tool, Cutdata WHERE Profile LIKE '%" + profile + "%' AND Tool.Mater
```

```
    ial LIKE '%" + SMG + "%' AND Cutdata.Material LIKE '" + SMG + "' AND Cutdata.Name =
    Tool.Name AND Cutdata.Operation LIKE '" + profile + "'", null);
96.         return cursor;
97.     }
98.     public Cursor FilterToolsCursorContour(String profile, String material) {
99.         Cursor materialCursor = database.rawQuery("SELECT SMG FROM Material WHERE I
    D =" + material, null);
100.                materialCursor.moveToFirst();
101.                String SMG = materialCursor.getString(0);
102.                Cursor cursor = database.rawQuery("SELECT Tool.Name, Dc, ap, zn, Par
    t_No, Tool_Shape, dmm, l2, re1, rake FROM Tool WHERE Profile LIKE '%" + profile + "
    %' AND Tool.Material LIKE '%" + SMG + "%'", null);
103.                return cursor;
104.            }
105.
106.
107.
108.
109.            public List<String> unique_corner_radius() {
110.                List<String> corner_radius_list = new ArrayList<>();
111.                Cursor cursor = database.rawQuery("SELECT DISTINCT re1 FROM Tool ORD
    ER BY re1 DESC", null);
112.
113.                cursor.moveToFirst();
114.                while (!cursor.isAfterLast()) {
115.                    corner_radius_list.add(cursor.getString(0));
116.                    cursor.moveToNext();
117.                }
118.                cursor.close();
119.                return corner_radius_list;
120.            }
121.
122.            public List<String> getmachines() {
123.                List<String> machine_list = new ArrayList<>();
124.                Cursor cursor = database.rawQuery("SELECT Name FROM Machine ORDER BY
    ID ASC", null);
125.
126.                cursor.moveToFirst();
127.                while (!cursor.isAfterLast()) {
128.                    machine_list.add(cursor.getString(0));
129.                    cursor.moveToNext();
130.                }
131.                cursor.close();
132.                return machine_list;
133.            }
134.
135.            public Cursor getMyMachines() {
136.                Cursor cursor = database.rawQuery("SELECT Name, Power FROM Machine",
    null);
137.                return cursor;
138.            }
139.
140.            public Cursor getMaterialData(String materialID){
141.                Cursor cursor = database.rawQuery("SELECT SMG, HB, UTS, kc, Yield FR
    OM Material WHERE ID = '" + materialID + "'", null);
142.                return cursor;
143.            }
144.
145.            public void setMachine(String Name, String Power){
146.                //database.rawQuery("INSERT into Machine(Name, Power) VALUES ('" + N
    ame +"'," + Power + ");", null);
147.                ContentValues contentValues = new ContentValues();
148.                contentValues.put("Name", Name);
149.                contentValues.put("Power", Power);
150.                database.insert("Machine",null, contentValues);
151.            }
```

```
152.
153.            public void deleteMachine (String name){
154.                //database.rawQuery("DELETE FROM Machine WHERE ID = '" + position +
    "'", null);
155.
156.                database.delete("Machine", "Name = '" + name + "'", null);
157.            }
158.
159.            public Bitmap getToolDiagram(String FamilyName) {
160.                Cursor cursor = database.rawQuery("SELECT Picture FROM Tool_pictures
    WHERE Name ='" + FamilyName +"'", null);
161.                cursor.moveToFirst();
162.                byte [] imageByteStream = cursor.getBlob(0);
163.                ByteArrayInputStream inputStream = new ByteArrayInputStream(imageByt
    eStream);
164.                cursor.close();
165.                return BitmapFactory.decodeStream(inputStream);
166.                //return BitmapFactory.decodeByteArray(imageByteStream, 0, imageByte
    Stream.length);
167.            }
168.
169.
170.        }
```

Models

MachiningData.java

```
1.  package com.nishen.machiningapp.models;
2.
3.  import android.app.Application;
4.  import android.support.annotation.CallSuper;
5.
6.  import java.util.ArrayList;
7.  import java.util.HashMap;
8.
9.  /**
10.  * Created by Nishen on 2017/10/11.
11.  */
12.
13. public class MachiningData extends Application {
14.     private String Profile;
15.     private String selectedMaterial;
16.     private String CutLength;
17.     private String CutWidth;
18.     private String CutDepth;
19.     private String CornerRadius;
20.     private String Coolant;
21.     private String Clamping;
22.     private String OperationType;
23.     private String Machine;
24.
25.     boolean UserCutDataChecked;
26.
27.     private String userCutWidth;
28.     private String userCutDepth;
29.     private String userCuttingSpeed;
30.     private String userFeedPerTooth;
31.
32.     private ArrayList<HashMap<String, String>> FilteredToolList;
33.     private ArrayList<HashMap<String, String>> ToolList;
34.
35.     double [] CriteriaWeightingMatrix;
36.     double [] [] TOPSISmatrix;
37.     double [] UnorderedToolScores;
```

```
38.
39.
40.
41.
42.
43.     public String getProfile() {
44.         return Profile;
45.     }
46.     public void setProfile(String profile) {
47.         Profile = profile;
48.     }
49.
50.     public String getSelectedMaterial() {
51.         return selectedMaterial;
52.     }
53.     public void setSelectedMaterial(String selectedMaterial) {
54.         this.selectedMaterial = selectedMaterial;
55.     }
56.
57.
58.     public String getCutLength() {
59.         return CutLength;
60.     }
61.     public void setCutLength(String cutLength) {
62.         CutLength = cutLength;
63.     }
64.
65.     public String getCutWidth() {
66.         return CutWidth;
67.     }
68.     public void setCutWidth(String cutWidth) {
69.         CutWidth = cutWidth;
70.     }
71.
72.     public String getCutDepth() {
73.         return CutDepth;
74.     }
75.     public void setCutDepth(String cutDepth) {
76.         CutDepth = cutDepth;
77.     }
78.
79.     public String getCornerRadius() {
80.         return CornerRadius;
81.     }
82.     public void setCornerRadius(String cornerRadius) {
83.         CornerRadius = cornerRadius;
84.     }
85.
86.     public String getCoolant() {
87.         return Coolant;
88.     }
89.     public void setCoolant(String coolant) {
90.         Coolant = coolant;
91.     }
92.
93.     public String getClamping() {
94.         return Clamping;
95.     }
96.
97.     public void setClamping(String clamping) {
98.         Clamping = clamping;
99.     }
100.
101.          public String getOperationType() {
102.              return OperationType;
103.          }
```

```
104.
105.            public void setOperationType(String operationType) {
106.                OperationType = operationType;
107.            }
108.
109.            public String getMachine() {
110.                return Machine;
111.            }
112.
113.            public void setMachine(String machine) {
114.                Machine = machine;
115.            }
116.
117.            public boolean isUserCutDataChecked() {
118.                return UserCutDataChecked;
119.            }
120.
121.            public void setUserCutDataChecked(boolean userCutDataChecked) {
122.                UserCutDataChecked = userCutDataChecked;
123.            }
124.
125.            public String getUserCutWidth() {
126.                return userCutWidth;
127.            }
128.
129.            public void setUserCutWidth(String userCutWidth) {
130.                this.userCutWidth = userCutWidth;
131.            }
132.
133.            public String getUserCutDepth() {
134.                return userCutDepth;
135.            }
136.
137.            public void setUserCutDepth(String userCutDepth) {
138.                this.userCutDepth = userCutDepth;
139.            }
140.
141.            public String getUserCuttingSpeed() {
142.                return userCuttingSpeed;
143.            }
144.
145.            public void setUserCuttingSpeed(String userCuttingSpeed) {
146.                this.userCuttingSpeed = userCuttingSpeed;
147.            }
148.
149.            public String getUserFeedPerTooth() {
150.                return userFeedPerTooth;
151.            }
152.
153.            public void setUserFeedPerTooth(String userFeedPerTooth) {
154.                this.userFeedPerTooth = userFeedPerTooth;
155.            }
156.
157.            public ArrayList<HashMap<String, String>> getFilteredToolList() {
158.                return FilteredToolList;
159.            }
160.            public void setFilteredToolList(ArrayList<HashMap<String, String>> filte
    redToolList) {
161.                FilteredToolList = filteredToolList;
162.            }
163.
164.            public ArrayList<HashMap<String, String>> getToolList() {
165.                return ToolList;
166.            }
167.            public void setToolList(ArrayList<HashMap<String, String>> toolList) {
168.                ToolList = toolList;
```

```
169.            }
170.
171.            public double[] getCriteriaWeightingMatrix() {
172.                return CriteriaWeightingMatrix;
173.            }
174.            public void setCriteriaWeightingMatrix(double[] criteriaWeightingMatrix)
    {
175.                CriteriaWeightingMatrix = criteriaWeightingMatrix;
176.            }
177.
178.            public double[][] getTOPSISmatrix() {
179.                return TOPSISmatrix;
180.            }
181.            public void setTOPSISmatrix(double[][] TOPSISmatrix) {
182.                this.TOPSISmatrix = TOPSISmatrix;
183.            }
184.
185.            public double[] getUnorderedToolScores() {
186.                return UnorderedToolScores;
187.            }
188.
189.            public void setUnorderedToolScores(double[] unorderedToolScores) {
190.                UnorderedToolScores = unorderedToolScores;
191.            }
192.
193.            @Override
194.            public void onCreate(){
195.                super.onCreate();//reinitialise variables
196.            }
197.
198.        }
```

Utilities

TOPSIS.java

```
1.  //TOPSIS Code for machining tool data optimisation
2.
3.  package com.nishen.machiningapp.utils;
4.  import java.lang.Math;
5.
6.  public class TOPSIS {
7.      private double[] CriteriaWeightingMatrix;
8.      private double[][] TOPSISmatrix;
9.      private int rows_alternatives, columns_criteria;
10.
11.     private double[][] normalizedDecisionMatrix;
12.     private double[] normalizedWeightingMatrix;
13.     private double[][] weightedNormalizedDecisionMatrix;
14.     private double[] positiveIdealSolution;
15.     private double[] negativeIdealSolution;
16.     private double[] positiveSeparationFromIdeal;
17.     private double[] negativeSeparationFromIdeal;
18.     private double[] closenessCoefficient;
19.     private double[] sortclosenessCoefficient;
20.
21.
22.     public double [] calculate() {
23.         calculateNormalizedDecisionMatrix();
24.         calculateNormalizedWeightingMatrix();
25.         calculateWeightedNormalizedDecisionMatrix();
26.         calculatepositiveIdealSolution();
27.         calculatenegativeIdealSolution();
```

```
28.         calculatePositiveSeparationFromIdeal();
29.         calculateNegativeSeparationFromIdeal();
30.         calculateClosenessCoefficient();
31.         return closenessCoefficient;
32.     }
33.
34.     public double[][] calculateNormalizedDecisionMatrix() {
35.         double[] sumPowSqrt = new double[columns_criteria];
36.         normalizedDecisionMatrix = new double[rows_alternatives][columns_criteria];

37.
38.         /* Calculate Normalized Decision Matrix */
39.         for (int col = 0; col < columns_criteria; col++) {
40.
41.             double sumPow = 0.d;
42.             for (int row = 0; row < rows_alternatives; row++) {
43.                 sumPow = sumPow + Math.pow(TOPSISmatrix[row][col], 2);
44.             }
45.             sumPowSqrt[col] = Math.sqrt(sumPow);
46.             for (int row = 0; row < TOPSISmatrix.length; row++) {
47.                 normalizedDecisionMatrix[row][col] = TOPSISmatrix[row][col] / sumPo
    wSqrt[col];
48.             }
49.         }
50.
51.         return normalizedDecisionMatrix;
52.     }
53.
54.     public double[] calculateNormalizedWeightingMatrix() {
55.         normalizedWeightingMatrix = new double[columns_criteria];
56.
57.         double sumPow = 0.d;
58.         double sumPowSqrt = 0.d;
59.         for (int row = 0; row < columns_criteria; row++) {
60.             sumPow = sumPow + Math.pow(CriteriaWeightingMatrix[row], 2);
61.         }
62.         sumPowSqrt = Math.sqrt(sumPow);
63.         for (int row = 0; row < columns_criteria; row++) {
64.             normalizedWeightingMatrix[row] = (CriteriaWeightingMatrix[row]) / sumPo
    wSqrt;
65.         }
66.         return normalizedWeightingMatrix;
67.     }
68.
69.     public double [][] calculateWeightedNormalizedDecisionMatrix()
70.     {
71.         weightedNormalizedDecisionMatrix = new double[rows_alternatives][columns_cr
    iteria];
72.         for(int col = 0; col<columns_criteria;col++)
73.         {
74.             for(int row = 0;row<rows_alternatives;row++)
75.             {
76.                 weightedNormalizedDecisionMatrix[row][col] = normalizedDecisionMatr
    ix[row][col] * normalizedWeightingMatrix[col];
77.             }
78.         }
79.
80.         return weightedNormalizedDecisionMatrix;
81.
82.     }
83.
84.     public double [] calculatepositiveIdealSolution()
85.     {
86.         positiveIdealSolution = new double[columns_criteria];
87.         double max = 0.d;
88.         for(int col = 0; col<columns_criteria;col++)
```

```
89.         {
90.             max = 0d;
91.
92.             if (col == 4) {                                                 //maximisin
    g the MMR
93.                 for(int row = 0;row<rows_alternatives;row++) {
94.                     if (weightedNormalizedDecisionMatrix[row][col] > max) {
95.                         max = weightedNormalizedDecisionMatrix[row][col];
96.                     }
97.                     positiveIdealSolution[col] = max;
98.                 }
99.             } else {
100.
101.                    for(int row = 0;row<rows_alternatives;row++)
102.                    {
103.                        if(weightedNormalizedDecisionMatrix[row][col]<max)
    //Minimising first 4 parameters
104.                        {
105.                            max=weightedNormalizedDecisionMatrix[row][col];
106.                        }
107.                        positiveIdealSolution[col]=max;
108.                    }
109.                }
110.
111.            }
112.
113.            return positiveIdealSolution;
114.        }
115.
116.        public double [] calculatenegativeIdealSolution()
117.        {
118.            negativeIdealSolution = new double[columns_criteria];
119.            double min = 0d;
120.
121.            for(int col = 0; col<columns_criteria;col++)
122.            {
123.                min = 1;
124.                if (col == 4){                                              //minimise M
    MR
125.                    for(int row = 0;row<rows_alternatives;row++)
126.                    {
127.                        min = 500;
128.
129.                        if(weightedNormalizedDecisionMatrix[row][col]<min)
130.                        {
131.                            min=weightedNormalizedDecisionMatrix[row][col];
132.                        }
133.                        negativeIdealSolution[col]=min;
134.                    }
135.                } else {
136.
137.                    for (int row = 0; row < rows_alternatives; row++) {     //ma
    ximise eg. power consumption
138.
139.                        if (weightedNormalizedDecisionMatrix[row][col] > min) {

140.                            min = weightedNormalizedDecisionMatrix[row][col];
141.                        }
142.                        negativeIdealSolution[col] = min;
143.                    }
144.                }
145.            }
146.
147.            return negativeIdealSolution;
148.        }
149.
```

```
150.            public double []calculatePositiveSeparationFromIdeal()
151.            {
152.                positiveSeparationFromIdeal= new double[rows_alternatives];
153.                double [] temp = new double[rows_alternatives];
154.
155.                for(int i = 0; i<rows_alternatives;i++)
156.                {
157.                    temp[i]=0d;
158.                }
159.
160.                for(int row = 0; row<rows_alternatives;row++)
161.                {
162.                    for(int col = 0;col<columns_criteria;col++)
163.                    {
164.                        temp[row] = temp[row] + Math.pow((weightedNormalizedDecision
    Matrix[row][col]- positiveIdealSolution[col]), 2);
165.
166.                    }
167.
168.                    positiveSeparationFromIdeal[row]= Math.sqrt(temp[row]);
169.                }
170.
171.                return positiveSeparationFromIdeal;
172.            }
173.
174.            public double [] calculateNegativeSeparationFromIdeal()
175.            {
176.                negativeSeparationFromIdeal= new double[rows_alternatives];
177.                double [] temp = new double[rows_alternatives];
178.
179.                for(int i = 0; i<rows_alternatives;i++)
180.                {
181.                    temp[i]=0d;
182.                }
183.                for(int row = 0; row<rows_alternatives;row++)
184.                {
185.                    for(int col = 0;col<columns_criteria;col++)
186.                    {
187.                        temp[row] = temp[row] + Math.pow((weightedNormalizedDecision
    Matrix[row][col]- negativeIdealSolution[col]), 2);
188.
189.                    }
190.
191.                    negativeSeparationFromIdeal[row]= Math.sqrt(temp[row]);
192.                }
193.
194.                return negativeSeparationFromIdeal;
195.            }
196.
197.            public double [] calculateClosenessCoefficient() //similarity to the gre
    atest distance from worst solution
198.            {
199.                closenessCoefficient = new double[rows_alternatives];
200.                for(int i = 0; i<rows_alternatives;i++)
201.                {
202.                    closenessCoefficient[i] = negativeSeparationFromIdeal[i]/(negati
    veSeparationFromIdeal[i] + positiveSeparationFromIdeal[i]);
203.                }
204.
205.
206.                return closenessCoefficient;
207.            }
208.
209.            public double [] calculateSortClosenessCoefficient()        //Will sort
    after joining table data
210.            {
```

```
211.                sortclosenessCoefficient = new double[rows_alternatives];
212.
213.                return sortclosenessCoefficient;
214.            }
215.
216.            public double[] getnegativeIdealSolution() {
217.                return negativeIdealSolution;
218.            }
219.
220.            public double[][] getNormalizedDecisionMatrix() {
221.                return normalizedDecisionMatrix;
222.            }
223.
224.            public double[] getpositiveIdealSolution() {
225.                return positiveIdealSolution;
226.            }
227.
228.            public double[] getclosenessCoefficient() {
229.                return closenessCoefficient;
230.            }
231.
232.            public double[] getSortclosenessCoefficient() {
233.                return sortclosenessCoefficient;
234.            }
235.
236.            public double[][] getWeightedNormalizedDecisionMatrix() {
237.                return weightedNormalizedDecisionMatrix;
238.            }
239.
240.
241.
242.            public int getrows_alternatives() {
243.                return rows_alternatives;
244.            }
245.
246.            public void setrows_alternatives(int rows_alternatives) {
247.                this.rows_alternatives = rows_alternatives;
248.            }
249.
250.            public int getcolumns_criteria() {
251.                return columns_criteria;
252.            }
253.
254.            public void setcolumns_criteria(int columns_criteria) {
255.                this.columns_criteria = columns_criteria;
256.            }
257.
258.            public double[][] getTOPSISmatrix() {
259.                return TOPSISmatrix;
260.            }
261.
262.            public void setTOPSISmatrix(double[][] TOPSISmatrix) {
263.                this.TOPSISmatrix = TOPSISmatrix;
264.            }
265.
266.            public double[] getCriteriaWeightingMatrix() {
267.                return CriteriaWeightingMatrix;
268.            }
269.
270.            public void setCriteriaWeightingMatrix(double[] CriteriaWeightingMatrix)
    {
271.                this.CriteriaWeightingMatrix = CriteriaWeightingMatrix;
272.            }
273.
274.            public double[] getnegativeSeparationFromIdeal() {
275.                return negativeSeparationFromIdeal;
```

```
276.            }
277.
278.            public void setnegativeSeparationFromIdeal(double[] negativeSeparationFr
    omIdeal) {
279.                this.negativeSeparationFromIdeal = negativeSeparationFromIdeal;
280.            }
281.
282.            public double[] getpositiveSeparationFromIdeal() {
283.                return positiveSeparationFromIdeal;
284.            }
285.
286.            public void setpositiveSeparationFromIdeal(double[] positiveSeparationFr
    omIdeal) {
287.                this.positiveSeparationFromIdeal = positiveSeparationFromIdeal;
288.            }
289.
290.            public void setclosenessCoefficient(double[] closenessCoefficient) {
291.                this.closenessCoefficient = closenessCoefficient;
292.            }
293.
294.            public void setnegativeIdealSolution(double[] negativeIdealSolution) {
295.                this.negativeIdealSolution = negativeIdealSolution;
296.            }
297.
298.            public void setNormalizedDecisionMatrix(double[][] normalizedDecisionMat
    rix) {
299.                this.normalizedDecisionMatrix = normalizedDecisionMatrix;
300.            }
301.
302.            public void setpositiveIdealSolution(double[] positiveIdealSolution) {
303.                this.positiveIdealSolution = positiveIdealSolution;
304.            }
305.
306.            public void setSortclosenessCoefficient(double[] sortclosenessCoefficien
    t) {
307.                this.sortclosenessCoefficient = sortclosenessCoefficient;
308.            }
309.
310.            public void setWeightedNormalizedDecisionMatrix(double[][] weightedNorma
    lizedDecisionMatrix) {
311.                this.weightedNormalizedDecisionMatrix = weightedNormalizedDecisionMa
    trix;
312.            }
313.
314.            public double[] getNormalizedWeightingMatrix() {
315.                return normalizedWeightingMatrix;
316.            }
317.
318.        }
```