2016 Pattern Recognition Association of South Africa and Robotics and Mechatronics
International Conference (PRASA-RobMech)
Stellenbosch, South Africa

Robot Movement Parameterization using Chess as a Case Study within an Education Environment

Herman Vermaak and Japie Janse van Rensburg
RGEMS Research Unit
Department of Electrical, Electronic and Computer Engineering
Central University of Technology, Free State,
Bloemfontein, South Africa
hvermaak@cut.ac.za

Abstract — **This paper describes a method of utilising a robot that plays a game of chess, against a human or itself, as a case study to try and improve the development structure of a Reconfigurable Assemble System within an educational environment. The focus is on the number of different products that can potentially be assembled and the different methods used in assembling the products. The future of this technology lies in the ability to adapt to numerous inputs that are able to deliver as many different outputs as possible. An industrial robot with the ability to play a game of chess, against itself or a human player, exhibits many of the requirements needed for future reconfigurable assemble systems. To play chess, the industrial robot should be able to record numerous external inputs, produce an immeasurable amount of outputs whilst adhering to pre-set rules. This case study allows for robotic training implemented on an industrial platform with real world application.**

I. Introduction

A reconfigurable system is one designed at the outset for rapid change in its structure, as well as its hardware and software components. Reconfigurable Assembly Systems (RAS) today are in high demand in order to quickly adjust its assembly capacity and functionality within a part family in response to sudden market changes or intrinsic system changes [1, 2, 3].

Dedicated Assembly Lines (DAL), or transfer lines are based on fixed automation and produce a company's core products or parts at high volume. Each dedicated line is typically designed to produce a single part at high assembly rate [4]. The DAL is part of a Dedicated Assembly System (DAS) and each DAL may have different capabilities.

Even though a DAL operates at low cost, it is only cost effective as long as market demands match the supply [4]. So as new products are required, or less of the current product, this could be a telling setback for a company.

Figure 1 shows that as additional products are included in the system, more assembly lines are required. Thus, in a DAS, the total cost of the system increases.

With a RAS, as in Figure 2, this is not the case, as the Reconfigurable Assembly Line (RAL) is adaptable.

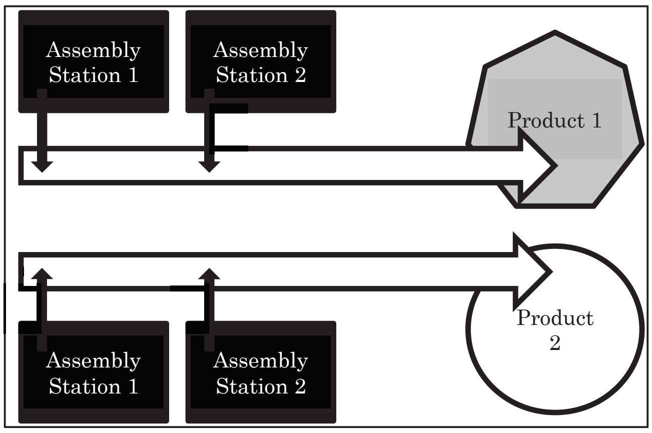


Figure 1: Dedicated Assembly Lines of two products in a Dedicated Assembly System.

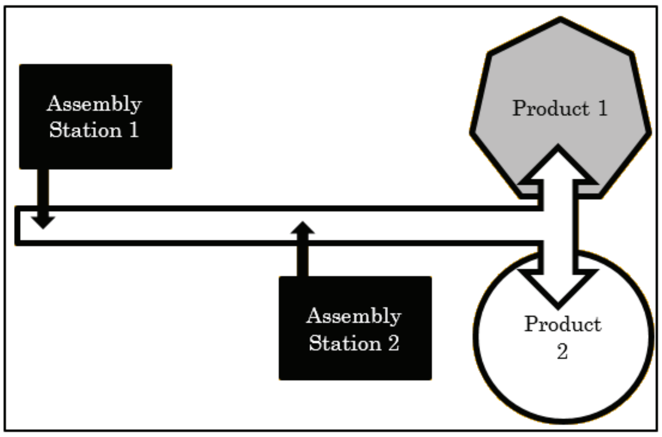


Figure 2: Reconfigurable Assembly Line of two products in a Reconfigurable Assembly Systems

The start-up cost for a RAS would be more than that of a DAS, but the flexibility and scalability of the RAS makes it a more viable option looking into the future. A comparison between RAL and DAL are shown in Table 1.

II. Objectives of Project

The main objective of the project is to replicate a reconfigurable assembly system and improve on the diversity and accuracy regarding the output range. This is achieved by implementing an industrial robot to play a game of chess against itself or a human opponent.

TABLE 1: COMPARISON BETWEEN RAL AND DAL REGARDING SINGLE PRODUCT ASSEMBLY

RAL	DAL
Limitations • Expensive • Slow-single tool operation	**Limitations** • Not flexible – for a single part • Fixed capacity, not scalable
Advantages • Convertible • Scalable Capacity	**Advantages** • Low Cost • Fast

A disadvantage for an RAS system is the speed at which it operates. Because the DAS is optimized for a specific product whereas the RAS is adaptable and with that comes certain design sacrifices. This paper details on the concept of one system being able to manufacture different kinds of products and be flexible enough to change the process used to manufacture the products; thus being able to optimize assembly flow.

To achieve this, the idea is to maximize assembly stations' abilities regarding the assembly process. This is to fully take advantage of an assemble station's re-configurability.

The concept is that any station must be able to function as the other assembly process if required. This concept is illustrated in Figure 3 where the assembly stations can be moved to optimize the assembly flow. There are many advantages to this kind of system. Figure 3 illustrates the ability to adapt to assembly workloads and avoid a potential bottle-neck in the production line.

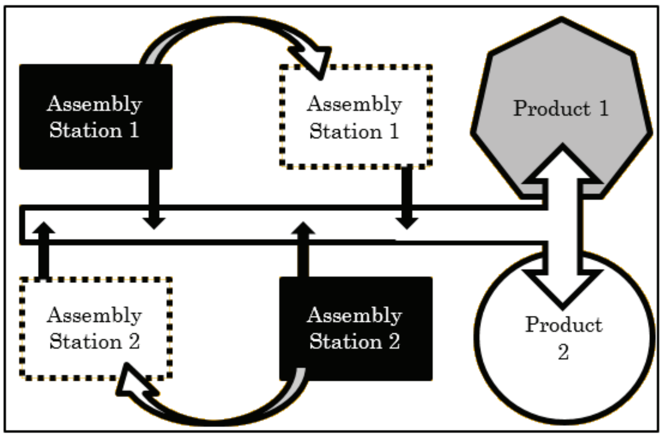


Figure 3: RAL with movable assembly stations to optimise assembly flow.

The system will monitor the assembly speed at every station to calculate possible backlogs. Should there be a backlog at a station another station will assist with that specific assembly point keeping the production functioning at an optimal rate. This will ensure during an assembly process there won't be many backlogs, if any. Optimizing the assembly flow should make RAS more lucrative to even large assembly companies.

III. RAS and Chess

A. Similarities

Chess was chosen to simulate the assembly of products because it shares the following similarities:

- The amount of different chess positions will be considered to be a specific product that will be manufactured.
- The chess pieces moves up to a specific position, the notation, will represent the method used to assemble the product.
- Every system must also adhere to certain rules, be it physics, like gravity, or the manner in which certain materials must be manipulated to achieve a certain output. In this regard the basic chess rules will be seen as those rules. That is, the amount of pieces available, how the pieces are allowed to move (Check, En Passant). .

TABLE 2: CHESS AND A RAS COMPARISON

CHESS	RAS
Chess Position	Finished Product
Notation	Assembly Sequence
Chess Rules (FIDE)	Assembly Boundaries

Using chess as a case study equates to a lot of possibilities if you take into account that there are over 318 979 different ways to play the first four moves in chess [5]. Figure 4 shows an example of two different positions, Game 1 and Game 2. Those positions were reached after four moves. The emphasis will be on the big differences between the two positions. Specifically the amount of pieces present on the two boards as well as the different positions of the pieces on the boards.

In short it indicates that within four moves the product (Chess Position) differs greatly in present state and immediate modification abilities (Assembly Capabilities).

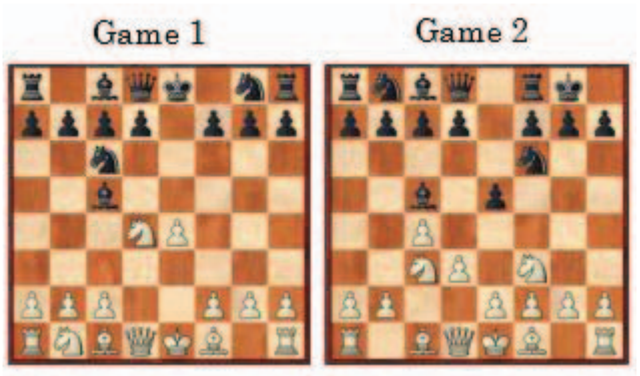
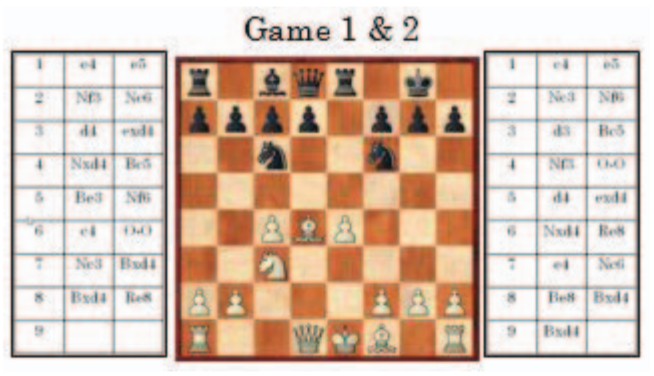


Figure 4: Two different chess positions after four moves [5]

However, there is still an option for both positions to be identical after a few moves. Figure 5 shows the position that both games 1 and 2 from Figure 4 reached after 8 and 9 moves respectfully. The chess position reached by both games, 1 and 2, after different amount of moves. The notation for both games is displayed on both sides of the chess board. Game 1's notation on the left and Game 2's notation on the right side of the board.



1	e4	e5
2	Nf3	Nc6
3	d4	exd4
4	Nxd4	Bc5
5	Be3	Nf6
6	c4	O-O
7	Nc3	Bxd4
8	Bxd4	Re8
9		

1	e4	e5
2	Nc3	Nf6
3	d3	Bc5
4	Nf3	O-O
5	d4	exd4
6	Nxd4	Re8
7	e4	Nc6
8	Be8	Bxd4
9	Bxd4	

Figure 5: The display as seen by user [5].

B. *Systemization*

To manufacture a product in the same way every time will require Portable Game Notation (PGN). PGN is a standard designed for the representation of chess-game-data using ASCII text files [6].

To play a game of chess will automatically subsume the ability to replicate a game using a PGN. That is why the focus of this project was to be able to first play a standard game of chess.

If the system being researched is able to play a game of chess, it is safe to assume that with the capabilities shown it should be able to be integrated with a RAS and instead of a game being played, a product will be manufactured.

IV. Methodology

A. *Requirements*

For an industrial robot to play chess, the following requirements must be met:

- The Robot needs to physically make moves on a chessboard.
- The moves of the human player need to be interpreted and the best move needs to be signalled to the Robot.

To accomplish the goals of the project set out by the requirements, the project was divided into three sub categories:

- Software
    1. GUI
    2. Chess Engine
    3. Robot movement
- Hardware
    1. Robot
    2. Gripper
    3. Chessboard and pieces
- Integration
    1. Between Hardware and Software

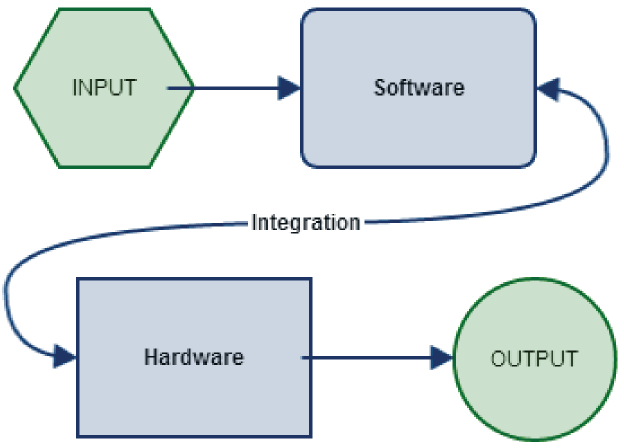


Figure 6: Shows the flow of the system with regards to the Hardware and the Software needed by system to play Chess

B. *Hardware*

1) *Robot*

An industrial robot was used. Industrial Robots are tough and tireless [7]. An industrial robot was chosen because of the following reasons:

- It can perform complex tasks
- Tireless precision of robot
- Easily integrated with external modules like sensors and/or adaptive controls.

Precision is a major advantage as this will ensure that, once programmed, the intervals for required calibration will be long. The robot we used is the KUKA KR5 Sixx R850 and is shown in Figure 7. This robot is very versatile and small enough to be moved for added mobility. Different pieces needed to be picked and placed, a gripper would be required, and thus easy integration added simplicity to the design.

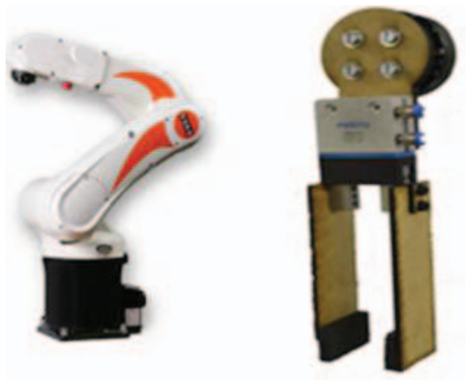


Figure 7: KUKA KR5 Sixx R850 Robot and the Gripper being used in this system

2) *Gripper*

One of the key elements of this project is the ability to pick and place pieces. This is accomplished by using a gripper. There are many different kinds of grippers. Examples are vacuum, hydraulic and servo-electric to name a few. Pneumatic grippers were chosen due to being a clean, cheap and reliable means to grip the chess pieces. The gripper used in the system is shown in Figure 7.

3) *Chessboard and Chess Pieces*

Chess pieces have a complex design in the sense that different pieces require different means to be picked and placed on the chess board. Thus the complexity of the gripper increases. But instead of creating a more complex gripper, a decision was made to alter the bases of the chess pieces. This made it possible to use a standard gripping method for all the chess pieces involved. The custom base is visible in Figure 8.

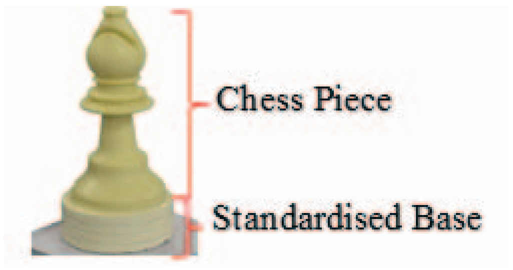


Figure 8: Illustrates the use of the custom base with chess pieces

C. *Software and Integration Protocol*

The software for this project is the core of what needs to be achieved. Figure 9 shows the flow of the system regarding the software and what base of communication is being used.

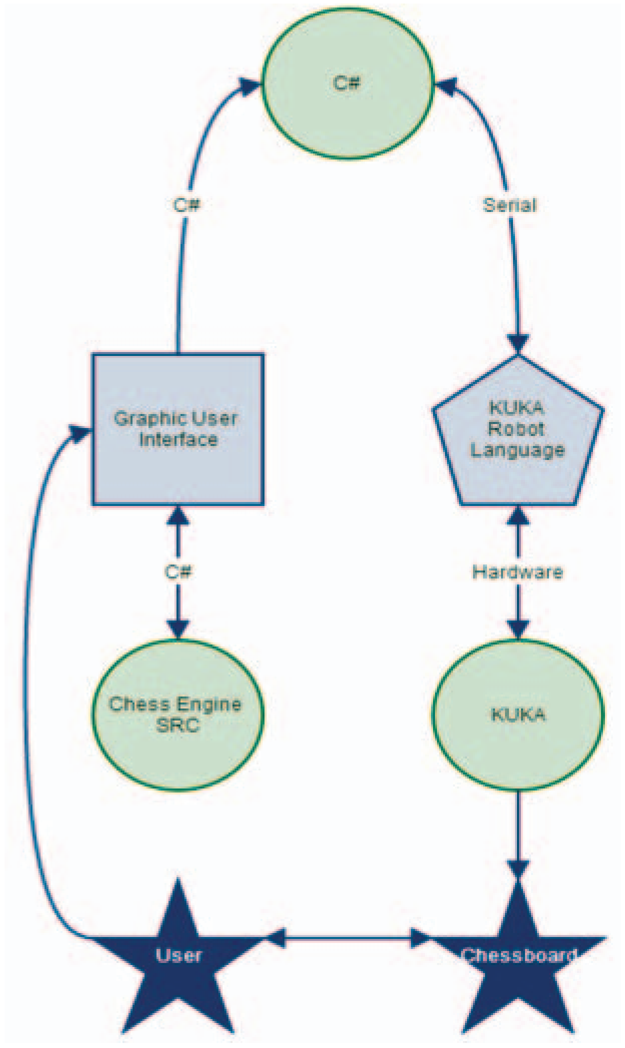


Figure 9: A flow diagram explaining how the software integrates with the system

The user communicates with the KUKA Robot using a Graphical User Interface (GUI). The "brain" of the PC that calculates the correct move using algorithms is the chess engine [8]. Both the chess engine and the GUI are open source code written in C# [9].

Code was then needed to retrieve the move calculated by the chess engine and relay it to the KUKA in a form that it can understand. This code, written in C#, was integrated between the GUI and the chess engine.

Kuka Robot Language (KRL) is the programming language used by the KUKA robot to perform hardware tasks. Serial was used to communicate between KRL on the KUKA and C# on the PC.

KRL differs a bit from C#, but the means of communication between C# and KRL was limited to only sending the moves the KUKA requires. Thus, the KRL has no code to accommodate what is being calculated using the algorithms.

The advantage of this approach meant that the robot was independent of the chess program. Meaning if the user required the robot to move from A1 to B8, the robot would do so

This form of independency means that it would be easy to use another robot, should the current one experience difficulty performing a certain task.

Also, functionality would also increase. Should the KUKA need to be used for a different purpose apart from chess, the move calibration is ready, meaning that the transition between software would be almost seamless. Figure 10 is showing the project in actions with 1 being the Gripper, 2 the Human, 3 the KUKA and 4 the Chessboard.

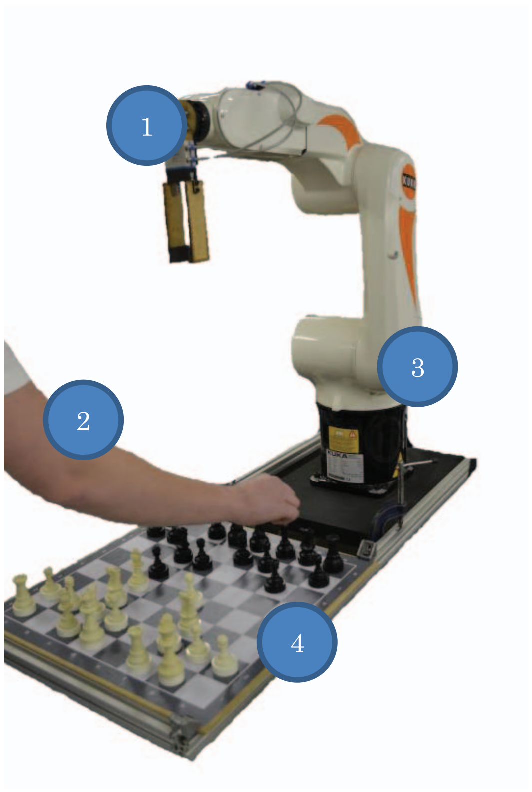


Figure 10: The project in action.

V. Experiments and Results

There are three main focus points regarding the experiments and results. They are:

- Communication
- Movements
- Calibration

A. Communication

Serial was the chosen medium of communication between the KUKA and the software. The communication was stable, fast and limitless with regards to input range. OPC and direct I/O pins were also considered. But due to the lack of speed with OPC and the limited available I/O ports, serial was the obvious option.

B. Movements

The KUKA is accurate and reliable regarding its ability to repeat programmed movements. It has an accuracy rating that's easily less than a millimetre. That level of accuracy is not required for this project as the correction distance is almost a centimetre. Figure 11 below shows a move in progress.

Figure 11: Shows a move in progress

There are 4 different movement types that were required:

1. Normal Move: Move piece from its current square to a destination square.
2. Capture Move: The piece occupying the destination square is moved to the Capture Bin (A box next to the board). Thereafter a *Normal Move* is performed.
3. En Passant: Like the *Capture Move*, except the Pawn that is moved to the Capture Bin is not occupying the destination square.
4. Castling: Consists of two *Normal Moves*. Involving the King (To be moved first) and the Rook.

C. Calibration

Every position on the chessboard was programmed and saved in the KUKA's memory. A better option would have been to utilise relative movements as this makes the entire process much more flexible and reusable. But due to the complexity to use variables for positions that can change regarding an initial position, it was decided to use the rigid approach of programming every position required for the proof of concept.

The advantage is that every position the KUKA can move to is perfectly programmed and a mistake is unlikely.

The disadvantage is that when the board is moved or adjusted slightly, the positions needs to be calibrated again.

VI. Future work

The current system does not run independently. I.e. when a user plays against the robot, the user is required to make a move on the board physically and in the software using the GUI. A camera will be used to solve this problem by monitoring the board and the movements of the individual pieces that are used. This will also assist with the problems regarding calibration. As currently each square is indivualy programmed to accommodate cases where the chess squares are not all the same size or the board isn't as level as it should be.

VII. Conclusion

As stated in the introduction, the purpose of this project is to optimize a RAS. Based on the results, it is more than plausible to implement a station that has pre-programmed values which the robot uses in various diverse ways to manufacture a product.

The detail of the product that can be manufactured relies on the resolution of the grid programmed. Figure 12 illustrates a station and product using a 5x5 (low resolution) grid.

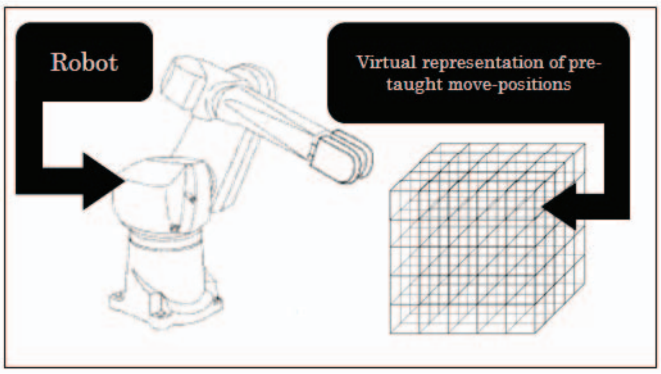


Figure 12: Explains the setup of a station. The industrial robot is on the left and the virtual assembly grid on the right.

A robot station will be assigned a workspace (virtual grid) where the assembly will take place. An actual example of this is shown below in Figure 13, where there are two products that are different after the same amount of assembly processes.

If a station can be configured to assemble any product using the grid system, it holds the following advantages:

- Different stations can dynamically change the products being manufactured depending on what is required by the system
- Even when a station is being serviced, assembly does not need to be stopped as the other stations can also fill the void, even though the assembly process will be slower.

- Easy integration between hardware and software. As a new product must only be designed and inserted for the station to start assembly.

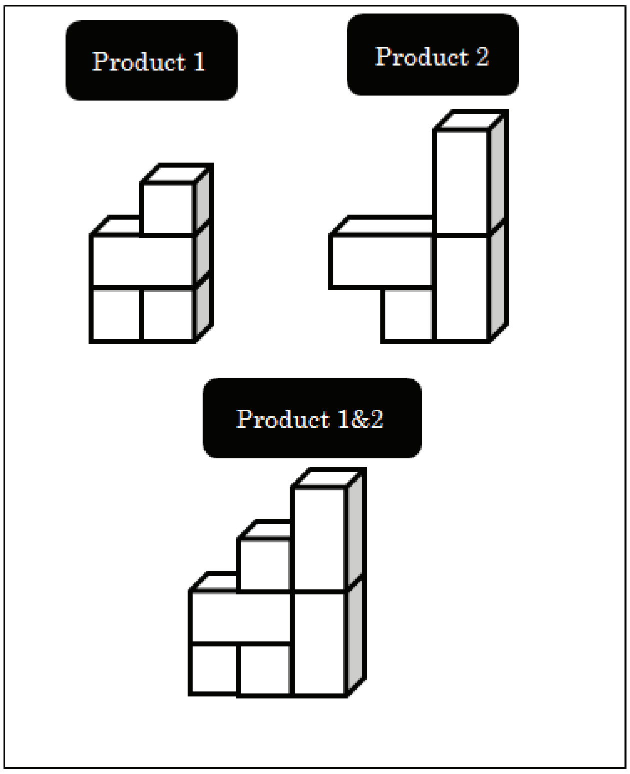


Figure 13: Shows two different products after adding 4 blocks in different ways as well as the combination product that both can reach

In the education environment to simulate RAS activity, each chess position resembles a product and the moves that lead to a position are regarded as the manufacturing process.

ACKNOWLEDGEMENTS

The Central University of Technology and the research unit RGEMS for their support and resources.

REFERENCES

[1] *EPSRC: Engineering and Physical Sciences Research Council*, http://www.epsrc.ac.uk/research/ourportfolio/themes/assemblythefuture/activities/Pages/fRAS.aspx, last accessed in August 2016
[2] Koren, Y et al., “Reconfigurable Assembly Systems,” in *A Keynote Paper*, 2nd ed., vol. 48, 1999, pp. 6-12
[3] *Michigan Engineering*, http://erc.engin.umich.edu/, last accessed in August 2016.
[4] Koren, Y, Reconfigurable Manufacturing Systems and Transformable Factories”, Springer Media, 2001, ISBN 3540293973, Pages 371.
[5] Enzine *@rticles*, http://ezinearticles.com/?Secret-of- chess&id= 1717732, last accessed in August 2016
[6] *Internet Chess Club*, http://www6.chessclub.com/help/PGN-spec, last accessed in September 2015
[7] Rajput, R.K., *Robotics And Industrial Automation*, Fist ed., 2008
[8] *Chess*, http://www.chess.com/blog/zaifrun/creating-a-chess-engine-from-scratch-part-1, last accessed in August 2016
[9] http://www.codeproject.com/Articles/36112/Chess-Program-in-C, last accessed in August 2016