Jigsaw Puzzle Building Robot With CNC Approach

Final Report

J.P. Strydom
13010736

Submitted as partial fulfilment of the requirements of Project EPR402
in the Department of Electrical, Electronic and Computer Engineering
University of Pretoria

November 2016

Study leader: Mr. H. Grobler

Part 1. Preamble

This report describes work that I did in designing and implementing an intelligent CNC robotic system, capable of solving and building a 12 to 24 piece jigsaw puzzle. The system's hardware and software units were built using mainly discrete and first principle components.

Project proposal and technical documentation
This main report contains a copy of the approved Project Proposal (Part 3 of the report) and technical documentation (Part 5 of the report). The latter provides details of the schematics, flow diagrams, software code, photos, and videos. I have included this appendix on a DVD that accompanies this report.

Project history
A few previous projects, such as the ones discussed in [1, 2, 3], all succeeded at solving some sort of jigsaw puzzle. These projects however, only produced software solutions. Another project succeeded in creating a puzzle-building robot [4]. This robot could however, only solve and construct a jigsaw puzzle provided that its pieces were all uniformly rectangular. By studying the techniques used in the above mentioned projects, I created a system, of my own design, capable of solving and building a 12 to 24 piece, traditionally shaped, jigsaw puzzle. Additionally, my robotic system can handle cases where up to three puzzle pieces are initially stacked on top of one another. Even though I make use of existing concepts and ideas, the core of my project was designed and implemented by myself.

Language editing
This document has been language edited by a knowledgeable person. By submitting this document in its present form, I declare that this is the written material that I wish to be examined on.

My language editor was C.L. Bastiaanse.

Language editor signature	*Date*

Declaration
I, Johannes Petrus Strydom understand what plagiarism is and have carefully studied the plagiarism policy of the University. I hereby declare that all the work described in this report is my own, except where explicitly indicated otherwise. Although I may have discussed the design and investigation with my study leader, fellow students or consulted various books, articles or the Internet, the design and investigative work is my own. I have mastered the design and I have made all the required calculations in my lab book (and/or they are reflected in this report) to authenticate this. I am not presenting a complete solution of someone else.

Wherever I have used information from other sources, I have given credit by proper and complete referencing of the source material so that it can be clearly discerned what is my own work and what was quoted from other sources. I acknowledge that failure to comply with the instructions regarding referencing will be regarded as plagiarism. If there is any doubt about the authenticity of my work, I am willing to attend an oral ancillary examination or evaluation about the work.

I certify that the Project Proposal appearing as the introduction section of the report is a verbatim copy of the approved Project Proposal.

J.P. Strydom	Date

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ASCII	American standard code for information interchange
CNC	Computer numerical control
GUI	Graphic user interface
HSV	Hue saturation value
IC	Integrated circuit
IDE	Integrated development environment
JTAG	Joint test action group
LCD	Liquid crystal display
PC	Personal computer
PCB	Printed circuit board
PNG	Portable network graph
PWM	Pulse width modulation
RGB	Red green blue
SPI	Serial peripheral interface
UML	Unified machine language
USB	Universal serial bus

Part 2. Summary

Jigsaw Puzzle Building Robot With CNC Approach

J.P. Strydom[1]
Email: u13010736@tuks.co.za
[1]Dept. of Electrical, Electronic and Computer Engineering, University of Pretoria, South Africa

***Abstract*—This report describes the work done pertaining to the design and implementation of an intelligent CNC robotic system, capable of autonomously solving and building a 12-piece, traditionally shaped, jigsaw puzzle, even when the jumbled puzzle contained overlapping pieces.**

What has been done

- **A literature survey on existing automated puzzle-solving techniques was completed.**
- **An algorithm was developed from first principles in Python, which could autonomously detect and classify jumbled jigsaw puzzle pieces.**
- **An algorithm was developed from first principles in Python, which could autonomously create a software solution to a jumbled jigsaw puzzle.**
- **A software simulation platform was developed from first principles in Python, which could test and verify the functionality of the system's main software components.**
- **A user interface was developed in Python with the aide of QtDesigner. The user interface's functionality was designed and developed from first principles.**
- **The system's hardware, which housed a microcontroller at its core, was designed and constructed from first principles.**
- **C code for the microcontroller was developed from first principles.**
- **An algorithm was developed from first principles in Python, which could translate a software puzzle solution into into physical puzzle solution. This was achieved by utilising a serial communication link between the systems's software and hardware units.**

What has been achieved

- **The system could autonomously solve and construct a 12-piece jigsaw puzzle in 210 seconds on average, with an accuracy of 97.44%.**
- **The system could separate overlapping piece clusters, consisting of up to three pieces, and could still construct a successful solution. The system had a piece separation likelihood of 85.33%.**
- **The final system had an average displacement error of 1mm and a maximum displacement error of 4mm.**

***Index Terms*—puzzle-solving; computer vision; CNC; gantry; image processing; robotics; automation**

I. Problem Identification

Since the introduction of robotics into the industrial world during the early 1960s [1], they have slowly helped pave the way to the fourth Industrial Revolution. Most modern intelligent robotic systems can perform specific tasks more quantitatively and qualitatively than any human ever could, and at a fraction of the cost [2]. However, some tasks still remain extremely difficult to implement in robotic systems, despite being very simple for humans to perform. Examples of industrial tasks such as these include strawberry-picking and plastic waste sorting - tasks which both still heavily rely on manual human labour [3, 4]. Another example of such a task is building a jigsaw puzzle, which is where the focus of this project lies.

The main aim of this project is to design and develop a robot capable of constructing a jigsaw puzzle similar to that which a young child can build. Despite not being a direct industrial problem, the techniques and design procedures involved in this project can very easily be adapted to fit a similar complex industrial problem. This project also focuses on researching current techniques and technologies associated with the various aspects of the problem and critically analysing them in order to tailor a solution best-suited for the problem.

The primary design challenge is to develop and integrate the system's hardware and software components. The software component should be able to capture an image of a jumbled puzzle and create solution to it. The hardware component should be able to physically construct a puzzle when it is provided with solution instructions from the software component. The core hardware and software components of this project is to be designed and implemented mainly from first principles.

II. Methods

The main approach to this project was to break the overall problem into small manageable components. By doing so, the problem became fully modularise, which eased design, development, and integration of the entire system. Each of these components were built and tested separately and, where necessary, were first broken down even further into smaller modular sub-units. This strategy made it more effective to locate errors and to verify the functionality of each unit without any interference from neighbouring units and/or sub-units.

A. Software Component

This component was ultimately responsible for analysing an initial puzzle layout[1] image, in order to derive a correct software solution to the puzzle at hand. This unit also provided the hardware component with all the commands needed to mechanically construct a puzzle solution. Additionally, it provided all the relevant system information to users via a graphical display. This unit was implemented on a PC and was developed in Python programming language in Spider IDE. The handy OpenCV libraries were used to facilitate a few basic image processing operations.

1) Piece Detection and Classification: The main goal of this component was to detect and classify puzzle pieces within an initial puzzle layout. Papers discussing techniques such as contour detection an line transformation [5, 6] were studied in order to derive an original piece-detection and piece-classification algorithm. Collectively, these algorithms performed the following core operations in order to detect and classify puzzle pieces.

- Capture a puzzle layout image.
- Apply transposition correction to the image.
- Filter and binarise the image.
- Detect contours within the image.
- Filter out non-piece contours.
- Extract each piece's shape, location, and orientation parameters from the valid piece contours.
- Extract and correct individual piece images.
- Store each piece's parameters in a data object.

Figure 1 below shows an indexed piece-classification image, as generated by the piece-classification algorithm. This image shows each piece's indent/tab/edge layout as well as each piece's offset angle.

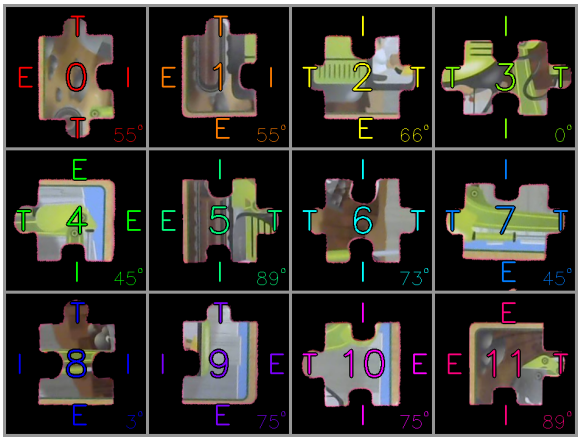


Fig. 1. Piece classification

[1]An initial puzzle layout consists of physical puzzle pieces that are randomly scattered, upward facing, and possibly overlapping. Such a layout will occupy no more than three quarters of the problem space.

2) Puzzle Solving: This component was responsible for interpreting the information it received from the piece-classification algorithm in order to derive a software puzzle solution. The two main approaches to puzzle-solving that were considered when constructing this component, was shape-matching and colour-matching. Shape-matching utilises piece edge type information (such as the locations of straight edges, tabs, and indents) to determine possible piece neighbours [7], and colour-matching utilises piece-colour information to determine the similarity between two pieces' colour content [8, 9]. These approaches were studied in order to derive an original puzzle-solving algorithm. This algorithm consisted of the following three core components which it used to derive a software solution.

1) Starting piece selection.
2) Shape-matching.
3) Colour-matching.

Figure 2 below shows the software flow diagram for the puzzle-solving algorithm. The processes show in **red** belong to the starting piece selection component, the ones shown in **green** belong to the shape-matching component, and the ones shown in **blue** belong to the colour-matching component .

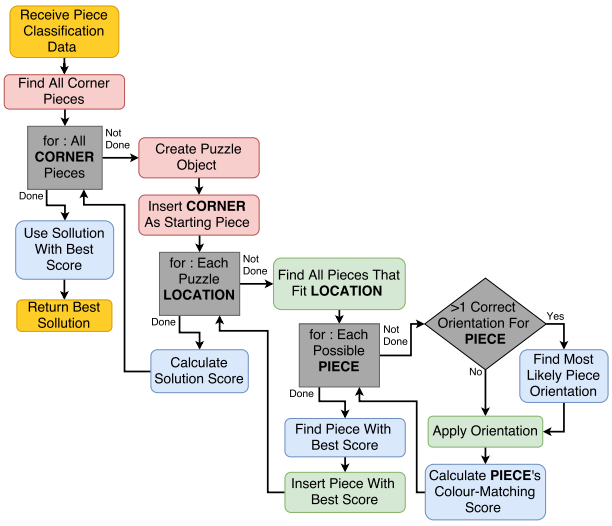


Fig. 2. Puzzle Solving Algorithm Flow Diagram [10]

The algorithm created four separate puzzle solutions, each starting from a separate corner pieces. Firstly, the algorithm would insert an initial corner piece and would then uses shape-matching to find all the corners' possible neighbouring pieces. Once all such pieces were found, they were correctly orientated to prepare them for colour-matching. Colour-matching was then applied to each candidate pieces in order to find the most likely piece neighbour. The most likely neighbour was the added to the puzzle. The algorithm would continue in this manner until all pieces were placed. Once all four solutions are complete, the algorithm would select the best one based on its puzzle score.

Figure 3 below shows an indexed solution generated by the puzzle-solving algorithm.

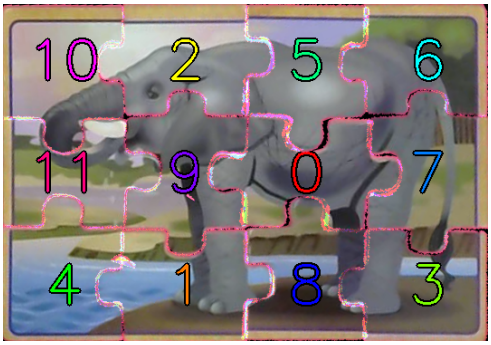


Fig. 3. Elephant puzzle solution

3) System Controller: The system controller regulated when certain software functions were called and dictated which actions to perform with their outputs. It functioned as a basic state machine, using the results obtained from the functions it conducts to transition between appropriate states. This controller thus served as a feedback control mechanism which allowed the software component to internally detect and correct errors.

Figure 4 below illustrates the system controller's flow diagram. Although not explicitly illustrated, this flow diagram functioned as a state machine. Once a function was completed, it would no longer be executed unless the system needed the components's functionality again. All the functions surrounded by a **thick black line** were treated as is shown by the sub-flow diagram in Figure 5, which illustrates the state machine behaviour. If a component should produce an unsuccessful output, the controller would simply reacquire a new image and reattempt all its processes until all of its components produce successful outputs.

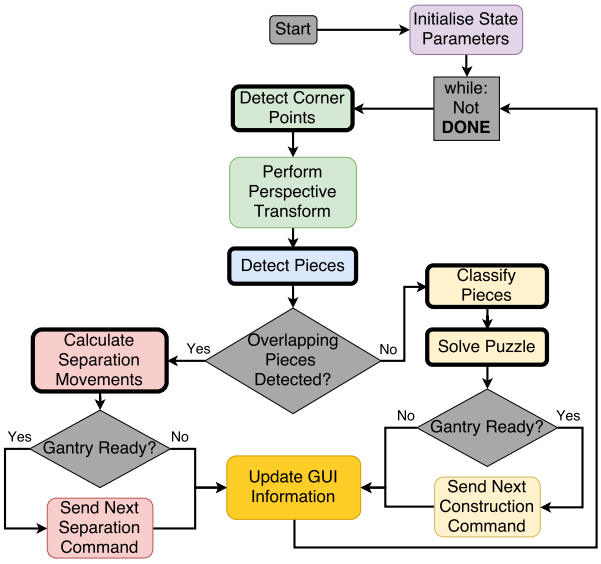


Fig. 4. System controller flow diagram [10]

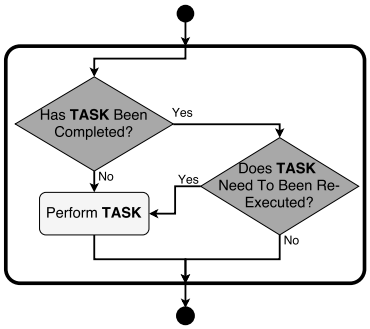


Fig. 5. System controller flow diagram expansion [10]

4) GUI: The GUI relayed information related to the puzzle-solving procedure to users, and streamlined the overall software experience. The system's GUI utilised a minimalistic layout to emphasise only the most important and relevant information, and to simplify the operation of the software. The GUI was dynamic as its height and width could easily be adjusted.

The GUI's image display window, contained four tabs. These tabs were used to switch between images and camera feeds related to the puzzle-solving process. These tabs contained the following information:

- the **Camera** tab showed the live camera feed,
- the **Detection** tab showed the puzzle piece detection view,
- the **Classification** tab showed the puzzle piece classification image, and
- the **Solution** tab showed the puzzle solution image.

The four buttons on the left of the user interface, when pressed, would capture and save the respective image in a PNG format in the *Images* sub folder (located where the puzzle solving program is installed).

The top right section of the user interface contained two LCD-style timers for the **Software Solution Time [ms]** and the **Hardware Solution Time [s]**. The bottom right section of the user interface contained a LCD-style **Piece Counter**, which displays the amount of valid pieces the system detected.

The **Start** button, at the bottom right of the user interface, initiated the puzzle-solving process when pressed. This button become unavailable while the puzzle solving system is in progress, and become available again once the system had completed the puzzle.

B. Hardware Component

This unit comprised of all the electronic, electro-mechanical, and mechanical hardware of the system. The main goal of this unit was to translate commands from the software component into a physical puzzle solution. Where applicable, this unit's sub-units were

prototyped and tested on breadboards. Once each sub-unit's functionality was satisfactory, the system was integrated and fully tested.

1) Electro-mechanical Hardware: The electro-mechanical hardware components of which the system is comprised, along with a brief description of their rolls in the overall system, are listed in Table I below.

Electro-Mechanical Hardware Units	
Servo Motors	• Activation and deactivation of the vacuum suction mechanism. • Raising and lowering the piece actuator.
Stepper Motors	• Controlling the x-axis movement of the gantry. • Controlling the y-axis movement of the gantry. • Rotating puzzle pieces.
Micro Switches	• Resetting the gantry to a known position. • Monitoring the gantry for x-axis limit breaches. • Monitoring the gantry for y-axis limit breaches.

TABLE I
ELECTRO-MECHANICAL HARDWARE UNITS

2) Mechanical Hardware: Table II below lists the mechanical hardware components that were designed and implemented for the hardware system, an briefly describes how they were constructed.

Mechanical Hardware Units	
Vacuum Suction Mechanism	A syringe, rubber tubing, and a suction cup was fitted together and attached to a servo motor, such that the motor could actuate the syringe. By doing so, this mechanism could create a vacuum and pick up puzzle pieces via the suction cup.
Piece Actuator	The suction cup of the vacuum suction mechanism was attached to one end of a short rod. The other end of the rod was attached to a axis such that the rod could raise and lower the suction cup. A servo motor was then attached to the rod so at to raise and lower the suction cup.
Piece Rotator	A stepper motor was attached to a gearing mechanism. One of the gears had a $6cm$ diameter and was positioned such that puzzle pieces could be placed on top of it. When the motor turned the gearing mechanism , the puzzle piece would rotate. A motor driver was used to run the motor.
Gantry	An x- and y-axis based gantry was used to transport puzzle pieces. The gantry utilised two stepper motors to control the its x- and y-axis respectively. Two motor drivers were used to run the motors.

TABLE II
MECHANICAL HARDWARE UNITS

3) Serial Communication: Serial communication was used to form the link between the system's hardware and software components. A MAX233 line transceiver chip was used to ensure that the communication voltage levels between the two sub-systems were at the correct levels. Table III below lists the RS-232 serial communication parameters that were used.

Baudrate	Databits	Parity	Stopbits	Flow Control
$38400Hz$	8 bits	None	1 bit	None

TABLE III
SERIAL COMMUNICATION PARAMETERS

In order for the two sub-systems to work together, they first had to understand each other. This was achieved by designing and implementing a shared communication protocol on both sub-systems. This protocol used a message format that both sub-systems could recognise and interpret, which enabled them to successfully send commands to one another.

4) Microcontroller and Firmware: At the heart of the hardware component, lies the *Atmel ATmega1284P-PU* 8-bit microcontroller. This component housed the firmware which was responsible for directing all the electro-mechanical components, and for communicating with the software component.

Figure 6 below shows the main flow diagram for this device. The **Initialisation** steps configure all the necessary device parameters, as described in the preceding sub-sub-sections. The **Wait For COMMAND** step is achieved by polling the microcontroller's USART flag. The **Apply COMMAND** step first interprets and then applies a received command, given that it is valid. **Send *Done* Command** step is achieved by sending the ASCII string *"D\r\n"* to the software system, indicating that it has completed its current command.

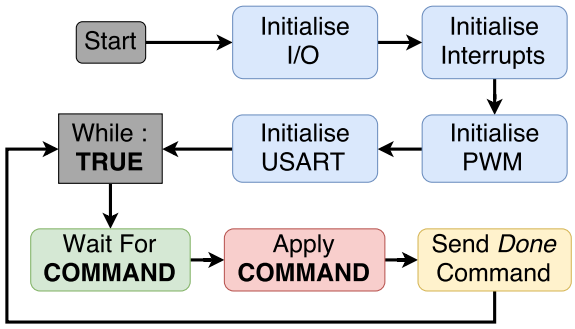


Fig. 6. Main flow diagram

Figure 7 below shows the flow diagram for the interrupt service routine of the device. (Note: **PORT A** of the device is connected to the limit monitoring switches.)

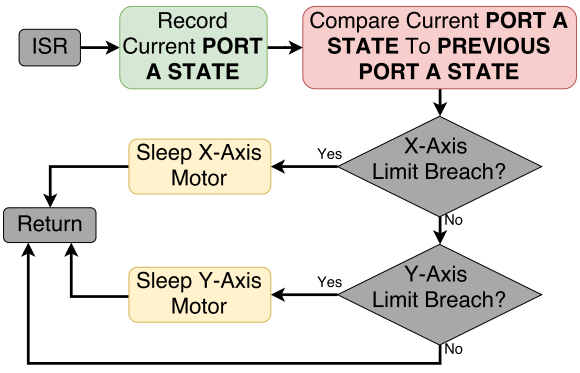


Fig. 7. ISR flow diagram

C. Integration

Once the software and hardware components were completed, a strategy to convert a software solution into a hardware solution needed to be devised. With a communication protocol already in place between the two sub-systems, they only needed to be calibrated and fine-tuned.

The software component produced puzzle construction commands in terms of pixel coordinates, where as the hardware system performed gantry translations in terms of stepper motor step counts. It was decided to perform the conversions from pixel coordinates to step count instructions on the software component. A few techniques were attempted but ultimately, a calibration mechanism proved to be the most successful. Calibration was achieved by sending the gantry to various step count coordinates and then monitoring each such coordinate's corresponding pixel coordinate. Once a large set of data was available, a spreadsheet was used to estimate a conversion equation. Figure 8 below shows the calibration output for the system's x-axis. The trend-line formula shown in the green box was then used to convert x-axis pixel coordinates into x-axis step count coordinates. Similarly, Figure 9 shows the calibration output for the system's y-axis, who's pixel coordinates are converted similarly to that of the x-axis.

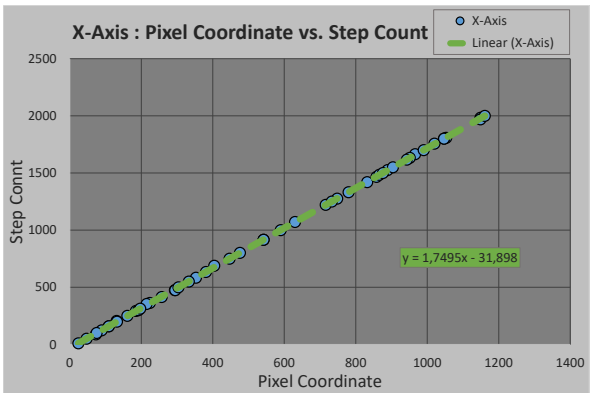


Fig. 8. Movement calibration: x-axis

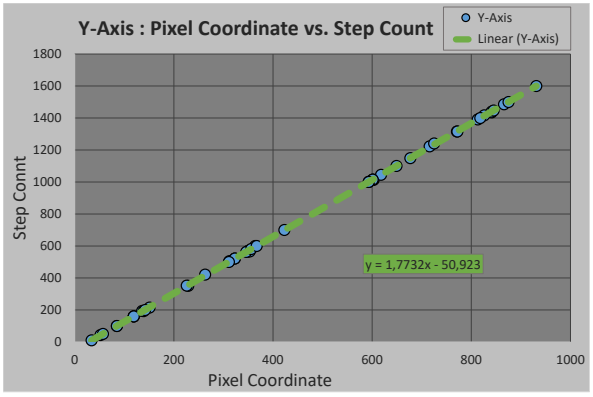


Fig. 9. Movement calibration: y-axis

An initial hardware and software movement strategy was implemented and tested. After this test, the strengths and weaknesses of the strategy in question were ascertained. These weaknesses were then used to make alterations and corrections to the strategy. This process was repeated until a sufficient successful strategy was obtained.

The final strategy saw a feedback control mechanism being introduced into the system. This mechanism was able to detect and correct various errors during the puzzle construction phase, which made the overall system more robust and reliable. The pseudo procedure of this strategy is listed below.

1) Rotate each piece correctly and return it to its original location.
2) Check if each piece has been correctly rotated, if not make corrections.
3) Recalculate each piece's centre location to alleviate possible errors introduced by piece rotations.
4) Temporarily stack pieces that obstruct the puzzle construction area at temporary holding locations.
5) Construct the entire puzzle in the puzzle construction area.
6) Gently nudge the corner pieces inward to compress the solution.

Figure 10 below shows this strategy in progress during phase 4 (as listed above). The red squares indicate the temporary holding locations and the blue square indicated the puzzle construction area.

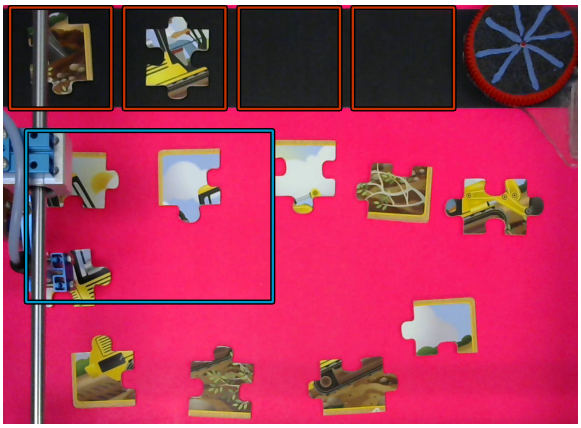


Fig. 10. Puzzle construction and stacking locations

III. Results

Various qualification tests were applied to the system's components in order to quantify their performance. For the majority of these tests, the system was presented with a variety of different jumbled puzzle layouts. This allowed the system to analyse the performance of its various components under different circumstances. In some cases, the system was presented with extremely harsh circumstances, which revealed the various component's performance limits.

The following lists the system's core components and shows some of the key results obtained from their qualification tests.

1) Piece detection:
   - Piece detection likelihood: 99.33%
2) Piece separation:
   - Piece separation likelihood: 85.33%
3) Piece classification:
   - Piece classification likelihood: 95.17%
4) Software solution:
   - Average solution accuracy: 96%
   - Average execution time: 584.95ms
5) Gantry accuracy:
   - Average deviation: 5.8 steps (1.1mm)
   - Maximum deviation: 19.3 steps (3.6mm)
6) Puzzle construction:
   - Average solution score: 97.44%
   - Flawless solution likelihood: 74%
   - Average solution time: 210.46s

IV. Discussion

Overall, the system and all of its core components performed exceptionally well. All the results indicated that the system's specifications were met, and that all the various sub-components were successfully integrated.

The system's robust design allowed it to detect and correct errors within its components, which allowed it to produce consistent and accurate puzzle solutions.

The most important results to take away from all the qualification tests is flawless solution likelihood and the average puzzle solution score. These results imply that roughly three out of every four puzzles will be solved and built perfectly, and those that are not perfect will most likely only have one piece that is not perfectly interlocked. This verifies that the entire system's most important objective has been reached.

V. Conclusion

It can thus be concluded that the design and implementation choices made throughout this project indicated sound engineering logic and reason. The system and its sub-components were designed and build efficiently and effectively, which contributed to the overall success of the system.

If this project should be attempted again, image feature detection could be a useful and interesting topic worth appending to this project. Once a software solution image is available, an image feature detection technique could be applied to it in order to detect features within the puzzle's picture. Although image feature detection by itself can be an extremely complex task, one can remedy this by limiting the system to only a few key features (such as an animal, building, or vehicle). This will make the task of feature detection more manageable and realistic in terms of the overall scope of the project.

The following lists two possible approaches to feature detection.

1) The system could make use of an artificially intelligent technique (such as a neural network or a hidden Markov model) to detect features. This will add a machine-learning component to the project.
2) The system could make use of a web-query technique (such as Google image search) to detect features. This will add a web-integration component to the project.

These approaches are merely suggestions, but they show that feature detection could add useful and interesting components to into the project's scope.

References

[1] RobotWorx. (2007) History of industrial robots. [Online]. Available: https://www.robots.com/education/industrial-history

[2] T. Schulz. (2013) Man vs. machine: Are any jobs safe from innovation? [Online]. Available: http://www.spiegel.de/international/business/innovation-and-automation-threatens-global-labor-market.html

[3] SBI. (2016) How are strawberries harvested. [Online]. Available: http://www.strawberries-for-strawberry-lovers.com/how-are-strawberries-harvested.html

[4] E. A. Bruno, "Automated sorting of plastics for recycling," University Of Minnesota, Tech. Rep., March 2007.

[5] M. R. Maire, "Contour detection & image segmentation," Ph.D. dissertation, University of California, Berkeley, 2009.

[6] N. Aggarwal and W. C. Karl, "Line detection in images through regularized hough transform," *IEEE transactions on image processing*, vol. 15, no. 3, pp. 582–591, 2006.

[7] D. J. Hoff and P. J. Olver, "Automatic solution of jigsaw puzzles," *Journal of mathematical imaging and vision*, vol. 49, no. 1, pp. 234–250, 2014.

[8] G. Paikin and A. Tal, "Solving multiple square jigsaw puzzles with missing pieces," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015, pp. 4832–4839.

[9] D. Pomeranz, M. Shemesh, and O. Ben-Shahar, "A fully automated greedy square jigsaw puzzle solver," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 9–16.

[10] Draw.io. (2015) version 5.0.3.7. 20-22 Wenlock Road, London, UK. [Online: accessed 21-July-2016]. [Online]. Available: https://www.draw.io/

Part 3. Project Proposal

This section contains the problem identification in the form of the complete approved Project Proposal, unchanged from the final approved version with the exception of minor grammatical corrections.

1. Problem Statement

Motivation. Since the introduction of robotics into the industrial world during the early 1960s [1], they have slowly helped pave the way to the fourth Industrial Revolution. Most modern intelligent robotic systems can perform specific tasks more quantitatively and qualitatively than any human ever could, and at a fraction of the cost [2]. However, some tasks still remain extremely difficult to implement in robotic systems, despite being very simple for humans to perform. Examples of industrial tasks such as these include strawberry-picking and plastic waste sorting - tasks which both still heavily rely on manual human labour [3, 4]. Another example of such a task is building a jigsaw puzzle, which is where the focus of this project will lie. This project's main aim will be to design and develop a robot capable of constructing a jigsaw puzzle similar to that which a young child can build. Despite not being a direct industrial problem, the techniques and design procedures involved in this project can very easily be adapted to fit a similar complex industrial problem.

Context. The problem of building a jigsaw puzzle will be solved by using a computer numerical control (CNC) approach, aided by digital image-processing techniques. CNC machines use a prepared program containing coded alphanumeric data to control the functions and motions of a machine tool [5]. Many companies are dedicated solely to the development and distribution of modular CNC hardware. This means that intricate components that cannot be designed and developed from first principles (for example, microprocessors, motors, and precision actuators) and components that fall outside of the core focus of this project (Purely mechanical components) will be easily available. The project will also heavily rely on image-processing techniques and puzzle-solving algorithms, both to which there are many existing approaches. Various puzzle-solving techniques and algorithms, such as the ones discussed in [6, 7, 8], will aid in the development of algorithms capable of reconstructing jigsaw puzzles. A similar project succeeded at creating a puzzle-building robot using image-processing techniques and a robotic arm [9]. This robot could solve and construct a jigsaw puzzle provided that the jumbled pieces were facing up with adequate spacing in between them. Using a CNC approach, this project will extend on the aforementioned project by adding a third dimension into its problem space[1]. This project will aim to replicate the results of the previous project even when jumbled pieces are stacked on top of each other. This will not only make it more difficult to mechanically manipulate pieces, it will also drastically increase the complexity of the image-processing component, thus making this new variation of the original project a viable topic to explore. This project will focus on researching current techniques and technologies associated with the various aspects of the problem and critically analyse them in order to tailor a solution best-suited for the problem.

[1] Problem space refers to the physical environment to which the system's vision and movement is limited.

Technical challenge. The technical challenges of this project lie in the design and development of its various hardware and software components, and the integration of all these components to form a fully functioning intelligent robotic system. Many of the hardware and software subcomponents will be designed from first principles and will thus require extensive investigation and testing of existing methods so as to determine where possible improvements or adaptations can be made. The following lists the specific technical challenges involved in this project:

- Creating algorithms capable of recognising stacked and unstacked puzzle pieces in a non-ideal image-processing environment.
- Creating algorithms capable of producing an efficient physical puzzle solution.
- Constructing a gantry robot and developing firmware capable of controlling it.
- Creating an electro-mechanical manipulator capable of handling stacked and unstacked pieces.
- Managing difficulties associated with mechanical control such as speed, stability, accuracy, stacked piece handling, dropped piece handling, gear slippage, and inertia.

Limitations. There are currently many limitations as to what robotic systems can do and it is important to consider the limitation related to this problem in order to design around them elegantly. The most obvious limitation is the system's limited processing power, which will put certain constraints on algorithm complexity. The limited processing power will also limit the amount and intricacy of pieces that the system can handle. As mentioned in [8], when very intricate puzzle shapes are considered, solving a 100-piece puzzle could take up to 36 hours. To avoid such a problem, the system will have to be restricted to a limited number of relatively simple puzzle pieces. The puzzle pieces' general shape will also heavily restrict the design and development of the mechanical manipulator especially considering it will have to manipulate stacked and unstacked pieces. The frame rate and image resolution of the digital camera will be limited, which will put constraints on the level of accuracy that the system can achieve.

2. User Requirement Statement

ELO 3: Design part of the project

An intelligent robotic system capable of constructing a jigsaw puzzle from pieces scattered in a three-dimensional problem space needs to be designed and developed.

2.1 MISSION REQUIREMENTS OF THE PRODUCT

The mission requirements of the system consists of the following.

- The system should be able to complete a jigsaw puzzle in roughly the same time as an average human toddler (between the ages of one- and three-years-old).
- The system should be able to detect and manipulate pieces within its entire problem space.
- The system should be able to handle a range of reasonably simple puzzle piece shapes.
- The system should be able to handle stacked puzzle pieces.
- The system should be able to complete a puzzle with reasonable accuracy.
- The system should be reliable, robust, and user-friendly.

2.2 FIELD CONDITIONS

The field conditions for the system comprises of the following.

- The system will be designed, tested, and operated in controlled laboratory conditions.
- The lighting available to the system will be controlled to create a stable image-processing environment.
- The system will handle all puzzle pieces on a controlled and uni-coloured platform with reference points to aid with calibration and orientation.
- The system does not have to be portable.
- The system that will be designed and developed will represent the full scale of the final system and not a scaled-down version thereof.

2.3 STUDENT TASKS

The student tasks pertaining to the system consists of the following.

- Acquire various jigsaw puzzles for testing purposes.
- Investigate suitable hardware and software components.
- Run tests on various candidate hardware and software components to determine those most suitable for the task.
- Develop an image-processing algorithm that can identify stacked and unstacked puzzle pieces.
- Develop an algorithm that can store all the necessary information about detected puzzle pieces in a format that can easily be interpreted and manipulated by software (A virtual puzzle piece object).
- Develop an algorithm that can intelligently manipulate virtual puzzle piece objects to deduce a solution.
- Develop a simulation platform capable of testing the image-processing algorithms independently of other functioning units.
- Develop an algorithm that can translate a virtual solution into the movements and actions required to execute the physical solution.
- Implement a solution for manipulating stacked puzzle pieces.
- Design and develop hardware and firmware for a gantry robot using CNC techniques.
- Investigate, design and develop an efficient communication scheme to integrate the various subcomponents.
- Create suitable performance tests and debugging tools to aid with the optimisation and development of the system.

ELO 4: Investigative part of the project

Before the puzzle-solving robot can be designed and developed, the most suitable software and hardware components first have to be determined. The most effective way to do so is to gather or create multiple candidate components and pass them through a series of performance tests. The results from such tests will indicate which techniques triumph over others, thus allowing the final system to be constructed with the most optimal and ideal components. The challenge is to create applicable and efficient performance tests and there are multiple factors to consider.

2.4 RESEARCH QUESTIONS

The research questions that need to be asked when choosing candidate components and when creating performance tests consists of the following.

- What is the highest puzzle piece shape complexity that will still be computationally feasible for the puzzle-solving algorithm?
- What type of mechanical hardware would be most suitable for interacting with jigsaw puzzle pieces?
- What image resolution would offer the best trade-off between speed and accuracy?
- What construction surface colour would best suit the image-processing component of the system?
- Which type of algorithm would be the most efficient at puzzle-piece recognition?
- Which type of algorithm would be the most efficient at puzzle-solving?
- What is the best approach to determining source and destination coordinates for the gantry robot?
- Which position-control techniques would provide high enough accuracy and speed to comply with the mission requirements?

2.5 STUDENT TASKS AND EXPERIMENTS

In order to aid the design process and verify the specifications of the final system, various experiments have to be developed and conducted.

i) Specification experiments:

- The time the system takes to complete a puzzle will be monitored in order to comply with the specifications.
- Once a puzzle has been constructed, the correctness and accuracy of the completed puzzle will be visually inspected in order to comply with the specifications.
- A graphical user interface (GUI) will be designed and developed to show visually how the system extracts puzzle pieces from the camera image and how the puzzle-solving algorithm manipulates these pieces whilst attempting to find a solution.
- The motions of the gantry robot should be fluent enough so that the puzzle piece-gripping mechanism does not unintentionally drop pieces.
- The accuracy of the gantry robot will be measured by comparing its physical coordinates with that specified by the mechanical control subsystem.
- The software and hardware of the system will be stress-tested in order to ensure robustness of the system.

ii) Research experiments:

- Once a puzzle-solving algorithm has been created, different puzzles with varying piece shape complexities will be solved virtually in order to determine the maximum complexity that will still lead to a feasible computational time.
- By studying the puzzle-solving algorithm, the maximum image resolution the algorithm can handle within the time constraint will be determined.
- By applying the chosen puzzle piece recognition algorithm to multiple different puzzle pieces on several candidate construction surfaces, the surface that provides the quickest and most reliable piece recognition can be determined.
- A set of pictures will be passed through various puzzle-piece recognition algorithms and the one that detects pieces the most effectively and consistently across the entire set of images will be chosen.
- Once puzzle piece objects have been extracted from an image set, various candidate puzzle-solving algorithms will be applied to the set. The algorithm that solves the puzzles the quickest, on average, with the least amount of help (for example, not having the solution picture or the puzzle size available) will be selected.
- By using images with reference points, various methods of determining the physical location of the points can be tested. The method that provides the most accurate results across the test cases will be selected.
- The minimum speed required by the motors can be calculated by taking the maximum puzzle size and the time limit into consideration. Once mechanical hardware has been chosen, the minimum torque required from the motors can be derived from the minimum force required to operate the hardware at an adequate speed.

3. Functional Analysis

Figure 1 below shows the functional block diagram of the puzzle-building robotic system. It illustrates the main functional units of which the system comprises.

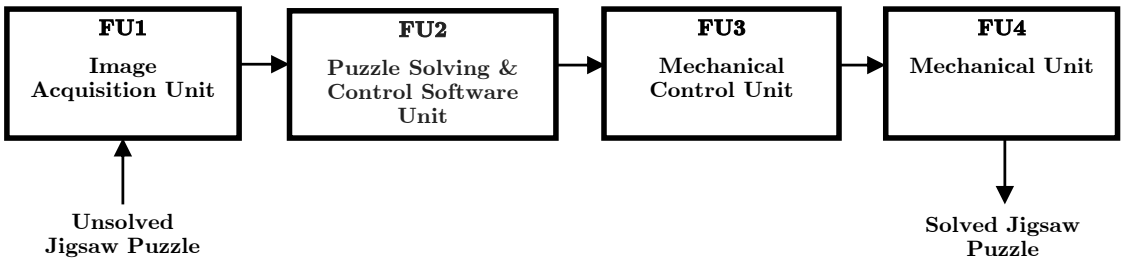


Figure 1.
Functional block diagram of the entire system

The image acquisition unit (FU1) will capture images by means of a digital camera and delivering them to the puzzle-solving and control software unit (FU2). Should extra lighting be required to assure image quality, it will form part of FU1. FU2, which will be implemented on a PC, will be responsible for all image-processing, computer vision, puzzle-solving and system control. Figure 2 below provides more detail about FU2. The mechanical control unit (FU3) will facilitate communication between FU2 and the mechanical unit (FU4). FU3 will interpret commands received from FU2 and apply the appropriate actions to FU4 in order to manipulate puzzle pieces. Figure 3 below describes FU3 in more detail. FU4 will consist of all the mechanical components of both the gantry system and the puzzle piece manipulator. It will perform all physical puzzle piece manipulations as instructed by FU3. The actions applied by FU4 will, in turn, be monitored by FU2 via FU1, which will serve as a feedback error correction mechanism.

Software processing is the main focus of this project. Figure 2 below provides a detailed functional description of the software components.

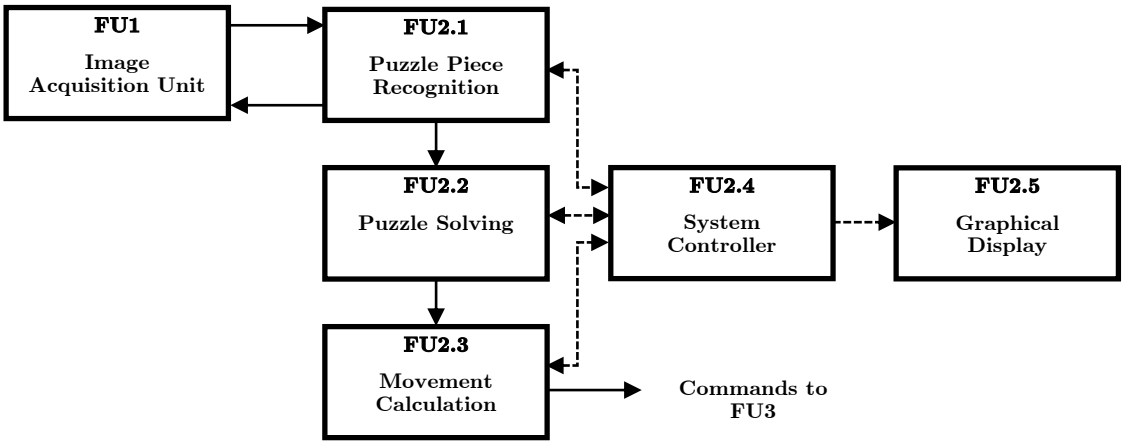


Figure 2.
Functional block diagram of the main software component

The system controller (FU2.4) will be responsible for running the entire system. It will initiate the system by prompting the puzzle piece recognition unit (FU2.1) to request an image from FU1. Upon receiving said image, FU2.1 will proceed to detect all visible puzzle pieces and convert them, and their location, into a format that the puzzle-solving unit (FU2.2) can interpret. If, however FU2.1 detects that pieces are stacked on top of one another, it will provide FU2.4 with the necessary information so that it can prompt the movement calculator (FU2.3) to devise a movement strategy to separate the stacked pieces in such a way that they can be adequately detected by FU2.1. Once such a movement strategy has been made, it will be sent to FU3, which will apply the necessary actions to the stacked pieces, allowing FU2.1 to continue its operations. Once all the piece information has been converted to the appropriate format, FU2.2 will calculate the desired location of each piece. This information will then be forwarded to FU2.3, which will derive the actions required to get the physical pieces to their correct locations. These actions will be sent to FU3, which will apply the desired actions to the jumbled pieces until they are all in the correct position. As the solution movements are applied, the system will use feedback analysis to ensure that errors are detected and appropriately handled. Additionally, FU2.4 will gather relevant information about the system's current state and make it available to users through the graphical display unit (FU2.5).

A further description of the electro-mechanical hardware and firmware that will be designed is provided in Figure 3 below.

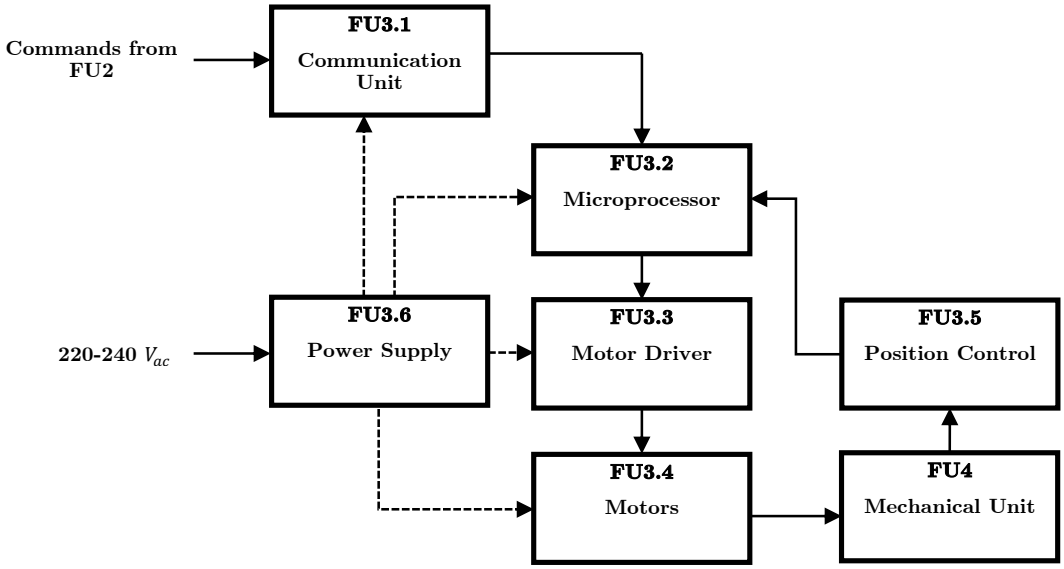


Figure 3.
Functional block diagram of the electro-mechanical hardware and firmware

The communication unit (FU3.1) will facilitate communication between the PC (FU2) and the microprocessor (FU3.2). The position control unit (FU3.5) will measure the exact position of the gantry system. FU3.2 will then use the gantry's current position, along with the command it received from FU2, to determine the appropriate rotational direction and speed for each motor (FU3.4), which it will then apply using the motor driver (FU3.3). FU3.4 will be connected to FU4 in such a way that it allows the gantry system to be translated linearly along the X, Y and Z-axis, allowing the piece-manipulating actuator to reach any position in the problem space. FU3 will be powered by the power supply unit (FU3.6), which will convert power from mains to the correct low-voltage levels required by the electronic components.

4. System Specifications

4.1 MISSION-CRITICAL SYSTEM SPECIFICATIONS

SPECIFICATION (IN MEASURABLE TERMS)	ORIGIN OR MOTIVATION OF THIS SPECIFICATION	HOW WILL YOU CONFIRM THAT YOUR SYSTEM COMPLIES WITH THIS SPECIFICATION?
The system should be able to solve and build a physical, kindergarten level, jigsaw puzzle within 15 minutes.	Various online discussions amongst parents suggest that most toddlers can solve 12 to 24 piece jigsaw puzzles in 10 to 30 minutes [10, 11].	When the system is initialised, it will activate a timer. Once the puzzle has been completed with sufficient accuracy, the time to completion will be displayed and moderated.
The system should be able to handle a minimum of 12 standard jigsaw puzzle pieces, each ranging between $40 \times 40 \times 1mm$ and $80 \times 80 \times 5mm$ in size with a maximum tab diameter of $40mm$.	Following the recommendations in [12], the ideal puzzle dimensions for toddlers have been selected. This will ensure that pieces can be feasibly handled by the piece manipulator.	During the operation of the system it will display the ability to manipulate pieces that fit the criteria.
The system should be able to handle cases where pieces in the initial puzzle layout[2] overlap, provided that such overlapping clusters consist of no more than three pieces.	In order to replicate the puzzle-building skills of a human toddler more closely, the system has to be capable of handling non-ideal initial puzzle layouts much like a child can.	In the initial puzzle layout, a few pieces will be placed in an overlapping position. The system will then demonstrate its capabilities by solving the puzzle despite these conditions.
The system should operate with a maximum displacement error of $10mm$.	Given the puzzle piece size, pieces would have to be no more than $10mm$ from their correct position to ensure that the solution is of high enough quality. This will allow the final puzzle image to be clearly visible and identifiable.	Upon completion of a puzzle, the displacement errors present will be visible and measurable.

Table 1.
Mission-critical system specification

[2] An initial puzzle layout consists of physical puzzle pieces that are randomly scattered, upward facing, and possibly overlapping. Such a layout will occupy no more than three quarters of the problem space.

4.2 FIELD CONDITIONS

SPECIFICATION	ORIGIN OR MOTIVATION
The system will be designed, tested and operated in controlled laboratory conditions with ambient lighting ranging between 300 and 500 lux.	These are the laboratory workbench conditions as recommended by the Illuminating Engineering Society in [13].
The system will handle all puzzle pieces on a controlled, and uni-coloured platform.	A controlled workspace will serve as a stable reference point for all calibration and image processing.

Table 2.
Field conditions

4.3 FUNCTIONAL UNIT SPECIFICATIONS

SPECIFICATION	ORIGIN OR MOTIVATION
FU1 should be able to capture images and supply them to FU2 in a digital format.	Once images are in a digital format, various software image-processing techniques can be applied to them.
FU2.1 should be able to process images of the initial puzzle layout and identify all the puzzle pieces, provided that they are all uniform, valid, undamaged and present.	This will allow puzzle pieces to be virtually manipulated by software.
FU2.2 should be able to solve the puzzle by manipulating virtual pieces in software.	This will be much more efficient, accurate and less time-consuming than a physical manipulation solution mechanism.
FU2.3 should be able to determine the most optimal route to deliver pieces to their correct location, given that a software solution exists.	This will allow the pieces to reach their correct locations quickly, using as few movements as possible.
FU2.4 must control and connect all the main software components.	This will allow the use of centralised control techniques.
FU2.5 will display all software and hardware operations, in real time, on a graphical user display.	This will enable users to validate the functionality of all the subcomponents.
FU3.1 must facilitate communication between the software (PC) and electro-mechanical components at a rate that will satisfy the time limitations.	This will allow the software component to indirectly control pieces and eventual solve the physical puzzle.

FU3.2 must control the entire electro-mechanical component by translating commands received from FU2.3 into destination coordinates. It will acquire the systems current location from FU3.5 and use it along with the target coordinates to determine the necessary physical manipulations.	This will provide the system with a local feedback control mechanism.
FU3.3, FU3.4 and FU4 will form a gantry robot, capable of moving a piece within the $0.4 \times 0.4 \times 0.05m$ problem space. The robot's movements have to be fluent enough to ensure that the piece manipulator has constant control over puzzle pieces whilst in transit.	Such a system will allow pieces anywhere in the problem space to be manipulated fluently and with little error.
FU3.5 will use potentiometers, the voltage of which will be related to a X, Y, or Z component, to determine the gantry's current position.	This will allow the system to have knowledge of its own location at all times, making manipulations possible.

Table 3.
Functional unit specifications

5. Deliverables

5.1 TECHNICAL DELIVERABLES

DELIVERABLE	DESIGNED AND IMPLEMENTED BY STUDENT	OFF-THE-SHELF
Jigsaw puzzles to construct during demonstration.		X
A digital camera capable of capturing images of the problem domain (FU1).		X
A digital camera interface capable of supplying the image-processing software with a data format that it can interpret (FU1).		X
A software component that can recognise puzzle pieces and store relevant information about each piece in an appropriate format (FU2.1).	X	
A software component capable of solving a digital version of the puzzle when provided with the necessary puzzle piece information (FU2.2).	X	
A software component that can determine appropriate puzzle piece manipulations that will lead to a physical puzzle solution, if provided with a digital puzzle solution (FU2.3).	X	
A software component capable of controlling and integrating various software components in order to derive a physical puzzle solution strategy from an aquittal image of an initial puzzle layout (FU2.4).	X	
A graphical user interface to display the software component's functionality and processing (FU2.5).	X	
Class and flow diagrams to describe and illustrate the interconnections and operation of the various software components.	X	
A software simulation platform that can verify the software components functionality independently.	X	
Communication hardware that can facilitate communication between the software unit and the mechanical control unit (FU3.1).		X

A microcontroller to run the firmware and interconnect the various mechanical control unit modules (FU3.2).		X
Motor drivers that will convert microcontroller signals to adequate high-voltage and high-current outputs by means of a power supply unit (FU3.3).		X
Electric motors capable of driving all the gantry mechanisms (FU3.4).		X
A position-control mechanism that can digitally measure the exact position of the gantry system (FU3.5).		X
A power supply unit that can convert power from mains to the correct low-voltage levels required by the electronic components (FU3.6).		X
Circuit diagrams of all the electronic hardware units.	X	
An electronic module, capable of connecting and integrating all the electro-mechanical hardware units, constructed from discrete components.	X	
Firmware capable of monitoring and controlling all the electro-mechanical units.	X	
A mechanical gantry system and puzzle piece manipulator (FU4).		X

Table 4.
Deliverables

5.2 DEMONSTRATION AT THE EXAMINATION

1. The demonstration will commence with the user initialising the automated calibration of the system.

2. After calibration, the system will be presented with an initial puzzle layout that conforms to the specifications.

3. Upon the user's command, the system will proceed to first separate stacked puzzle pieces from one another, given that any are present. The GUI will display the puzzle layout and will indicate the stacked and unstacked pieces that the system detects.

4. Once an unstacked initial puzzle state has been reached, the system will derive a virtual software solution of the puzzle. The GUI will systematically display the system's software puzzle-solving progress leading up to a complete virtual puzzle solution.

5. Once a virtual solution has been derived, the system will proceed to complete the puzzle with the necessary accuracy by using the gantry system along with the piece manipulator to move pieces, one by one, from their initial location to their correct position.

6. After the puzzle has been completed, the simulation platform will be used to describe and demonstrate the puzzle solving software process.

6. References

[1] RobotWorx. (2007) History of industrial robots. [Online]. Available: https://www.robots.com/education/industrial-history

[2] T. Schulz. (2013) Man vs. machine: Are any jobs safe from innovation? [Online]. Available: http://www.spiegel.de/international/business/speed-of-innovation-and-automation-threatens-global-labor-market-a-897412-2.html

[3] SBI. (2016) How are strawberries harvested. [Online]. Available: http://www.strawberries-for-strawberry-lovers.com/how-are-strawberries-harvested.html

[4] E. A. Bruno, "Automated sorting of plastics for recycling," University Of Minnesota, Tech. Rep., March 2007.

[5] Y. Koren, *Computer control of manufacturing systems*. McGraw-Hill New York, 1983.

[6] M. R. Maire, "Contour detection & image segmentation," Ph.D. dissertation, University of California, Berkeley, 2009.

[7] G. Paikin and A. Tal, "Solving multiple square jigsaw puzzles with missing pieces," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015, pp. 4832–4839.

[8] D. J. Hoff and P. J. Olver, "Automatic solution of jigsaw puzzles," *Journal of mathematical imaging and vision*, vol. 49, no. 1, pp. 234–250, 2014.

[9] N. Erell and R. Nagar, "Robotic jigsaw puzzle solver," Ben-Gurion University of the Negev, Tech. Rep., 2012.

[10] I. K. Sandhu. (2016) Gifted children and puzzles. [Online]. Available: http://www.brainy-child.com/expert/gifted-child-puzzle.shtml

[11] NetMums.com. (2012) 3 year olds and jigsaws. [Online]. Available: http://www.netmums.com/coffeehouse/children-parenting-190/toddlers-pre-schoolers-12-months-4-years-59/869067-3-year-olds-jigsaws.html

[12] M. Bryla and W. Lipinski. (2013) How to choose a puzzle for children? [Online]. Available: https://www.castorland.pl/en/how-to-choose-puzzle

[13] R. Redford, J. E. Kaufman, H. Haynes, and C. Crouch, "Illuminating engineering society," *Journal of the Illuminating Engineering Society*, vol. 2, no. 4, 1973.

Part 4. Main Report

1. Literature Study

The main aim of this project was to design and develop an intelligent robotic system, capable of solving and constructing a toddler-level jigsaw puzzle. The implementation of such a system required advanced knowledge of various image-processing techniques and puzzle-solving algorithms. This section focuses on summarising and contrasting literature on the aforementioned techniques and algorithms, whilst also discussing how they aided in the design and implementation of this project.

1.1 CONTOUR DETECTION

A Ph.D dissertation by M.R. Maire [1] discusses, in great detail, image segmentation by means of contour detection. Contour detection, in essence, uses mathematical techniques to estimate where contours reside within images. Additionally, the presented contour detection algorithm provides the hierarchy of all the contours it detects. This hierarchy shows which parent contour surrounds each individual contour, if such a surrounding parent contour exists, and it shows which contours reside within the same surrounding parent contour. The dissertation also discusses and contrasts various image filters that, when applied prior to contour detection, improve the accuracy and efficiency of the contour detection algorithm.

This contour detection algorithm, along with all its features, proved to be ideal for detecting overlapping and non-overlapping puzzle pieces and for classifying piece shapes. This was achieved by using the dimensions and defects of such contours, which is discussed in more detail in Section 2. The comparison on pre-image filters this paper provides, allowed an effective pre-detection filter to be identified, which increased the algorithm's piece-detection accuracy.

1.2 HOUGH LINE TRANSFORM

The Hough Line Transform, as discussed in [5], is shown to be extremely useful in determining the location and orientation of straight lines in images. This is not only of great importance to the fields to computer vision and image processing, but proved useful in the detection of puzzle piece angles.

The Hough Line Transform achieves line recognition by first passing an image through an edge detector. Such an edge detector then provides the line detection algorithm with all the edge points within the image. The algorithm then determines which of these edge points lie within the same straight line. If enough edge points lie on the same straight line, the algorithm records the parameters of the line and appends it to a list of detected lines. Once all edge points have been examined, the algorithm returns the parameters off all the straight lines it detected.

By applying this algorithm to puzzle piece images, the system could detect at what angle the pieces were, and thus by what angle they needed to be corrected. This is discussed in more detail in Section 2.

1.3 PUZZLE SOLVING

1.3.1 Colour-Matching Approach

Once puzzle pieces were identified and classified, a puzzle-solving algorithm needed to be applied. An algorithm suggested by G. Paikin and A. Tal [2] provides a solution to a jigsaw puzzle consisting of square pieces only. The main challenge behind this approach is that only colour can be considered, as all pieces are uniform in shape. This algorithm can function even when the following conditions are present.

- There are missing pieces.
- The orientation of the pieces are unknown.
- The size of the puzzles are unknown.
- The input contains pieces from several different puzzles.

The algorithm used the following three basic steps.

1. Calculate the piece compatibility.
2. Find the optimal starting piece.
3. Assemble the puzzle.

Step one – This step was achieved by using a compatibility metric of the authors' own design. Given two pieces, p_i and p_j, the compatibility function $C(p_i,\ p_j,\ r)$ calculates the likelihood that p_i and p_j are neighbouring pieces in the spatial relation r where $r \in \{up,\ down,\ left,\ right\}$. This compatibility function was then used alongside a "best buddy" metric. This metric, as introduced by D. Pomeranz in [6], labels two pieces as "best buddies" if, and only if, both agree that the other piece is their most likely neighbour in a certain spatial relation. It is extremely unlikely for two pieces to be labelled as "best buddies" if they are not real neighbours, making the combination of these two algorithms an effective means to find neighbouring pieces.

Step two – This step involved finding an optimal piece with which to start the puzzle construction. As the assembly function operated in a greedy way, it was important to find an effective starting point so as to avoid early errors, which could lead to later failures. The more distinct a starting piece is, the less likely the assembly step is to fail. The "best buddy" metric was used to judge how distinct a piece was. If a piece has a "best buddy" for all four of its edges, it was considered to be distinct and thus a suitable starting piece.

Step three – This step used the information from the previous two steps to conduct a greedy construction algorithm. Essentially, this algorithm would continue to add "best buddy" pieces to the already solved cluster of pieces until a solution was deemed complete.

1.3.2 Shape-Matching Approach

Another puzzle-solving algorithm was proposed by D.J. Hoff and P.J. Olver in [3]. This algorithm can solve a scrambled jigsaw puzzle in software by only considering the shapes of puzzle pieces. This algorithm can function as long as:

- the individual pieces have four well-defined sides, each containing either a tab or an indent,
- each piece has four primary neighbours at most, one on each side (except, of course, for pieces on the puzzle boundary) that are fitted together via the tabs and indents, and
- the boundary of the solved puzzle is an easily identified smooth shape, such as a rectangle.

The algorithm operates in the following three basic steps.

1. Piece boundary matching.
2. Piece locking.
3. Puzzle assembly.

Step one – This step used an algorithm based on the solution to the equivalence problem for plane curves based on extended Euclidean-invariant signatures developed in [7]. In essence, this algorithm first constructed a mathematical curve from each puzzle piece's edge. By doing so, the algorithm could then analyse the equivalence between opposing edges in order to estimate the most likely edge pairs. Additionally, this algorithm employed the logical fact that a tab can only be adjacent to an indent (and vice versa) to set up possible piece-pairs prior to calculation. Doing so drastically decreased calculation time and hence increased efficiency.

Step two – This step utilised the satisfying sensation humans experience when "snapping" puzzle pieces into place. By digitally simulating this sensation with a simple physics engine, the algorithm could accurately judge when pieces were in their most optimal inter-locking location.

Step three – This step first found a suitable starting piece (usually a corner piece, if one is present) and then used the data obtained from the previous two steps to construct the puzzle in software.

1.3.3 Chosen Approach

It can thus be seen that there are two main approaches when it comes to puzzle-solving algorithms, namely shape-matching and colour-matching. There are advantages and disadvantages to both these approaches, and they can both be implemented in many different ways. Table 5 below shows the main advantages and disadvantages associated with both approaches.

In order to make the system as effective as possible, it was designed as such so that shape- and colour-matching functions alongside each other. This allows the system to make use of the advantages of both approaches, whilst not suffering from any of their drawbacks. More detail about this can be found in Section 2.

Shape-Matching	Colour-Matching
Advantages	
When trying to find candidate piece neighbours, shape-matching makes it very easy to identify whether pieces are compatible with one another (tab vs. indent) or not (tab vs. tab, indent vs. indent). This reduces the amount of comparisons that the algorithm needs to make, which will, in turn, increase its solution speed.	Colour-matching will produce a reliable value indicating, with confidence, the likelihood of two pieces being neighbours. This will allow for the construction of a stable solution every time, which increases the accuracy of its solutions.
Disadvantages	
When more than two pieces are candidate neighbours, it becomes difficult to distinguish which piece fits best. Additionally, in order for such cases to be successfully handled, a very high image resolution might be required, as to capture finer shape details. This might be infeasible considering the time and monetary constraints of this project.	Colour-matching would have to be applied to all possible piece pairs at every step of the puzzle solving algorithm, as no distinction between incompatible neighbours can be made. This will increase the overall calculation time of the algorithm, leading to a slower solution time.

Table 5.
Comparison of puzzle solving approaches

1.4 PREVIOUS PROJECTS

The project described in [4] succeeded at creating a puzzle-building robot using image-processing techniques and a robotic arm. This robot could solve and construct a jigsaw puzzle consisting of rectangular pieces, provided that the jumbled pieces were facing upward with adequate spacing in between them.

After adding a third dimension into its problem space[1], this project succeeded at extending on the aforementioned project. This project is able to replicate the results of the previous project even when jumbled pieces are stacked on top of each other. This did not only make it more difficult to mechanically manipulate pieces, it also drastically increased the complexity of the image-processing component, which made this new variation of the original project a viable topic to explore.

[1] Problem space refers to the physical environment to which the system's vision and movement is limited.

2. Design and Implementation

2.1 APPROACH

The main approach to this project was to break the overall problem into small manageable components. By doing so, the problem became fully modularise, which eased design, development, and integration of the entire system. The main functional block diagram of the system is shown in Figure 4 below.

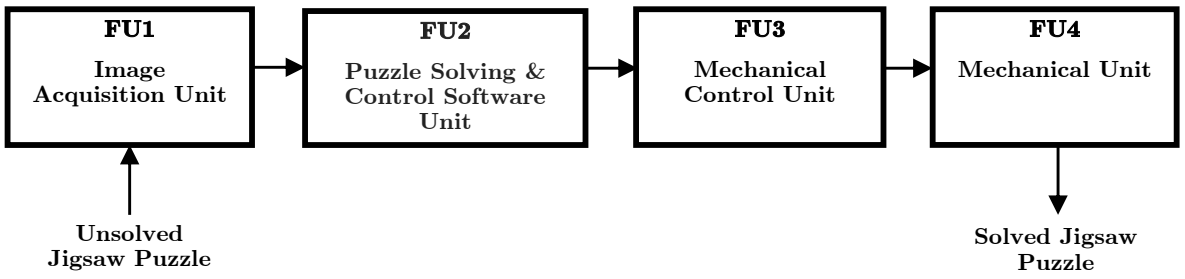


Figure 4.
Functional block diagram of the entire system

These units were built and tested separately and, where necessary, were first broken down even further into smaller modular sub-units. This strategy made it more effective to locate errors and to verify the functionality of each unit without any interference from neighbouring units and/or sub-units.

2.1.1 FU1 - Image Acquisition Unit

This unit is responsible for capturing images and transporting them to FU2. The hardware that this functional unit comprises of is a *Logitec C310* 720p web camera, a camera-to-USB interface, and an adjustable camera mount. This camera was chosen as it is affordable, reliable, and has an image resolution high enough to fulfil the needs of the system.

2.1.2 FU2 - Puzzle Solving and Control Software Unit

This unit is ultimately responsible for analysing an initial puzzle layout[2] image, in order to derive a correct software solution to the puzzle at hand. This unit also provides FU3 with all the commands needed to mechanically construct a puzzle solution. Additionally, it provides all the relevant system information to users via a graphical display. This unit was implemented on a PC and was developed in *Python* programming language in *Spider* IDE. The handy *OpenCV* libraries were used to facilitate a few basic image processing operations. The sub-functional block diagram of FU2 is shown in Figure 5 below.

[2]An initial puzzle layout consists of physical puzzle pieces that are randomly scattered, upward facing, and possibly overlapping. Such a layout will occupy no more than three quarters of the problem space.

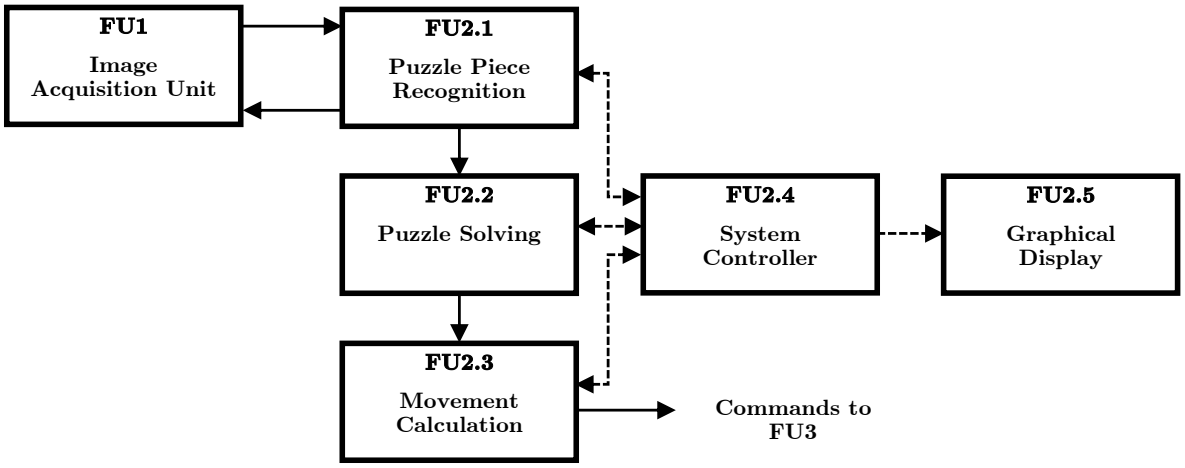


Figure 5.
Functional block diagram of the software component

FU2.1 - Puzzle Piece Recognition

The main goal of this sub-unit is to detect and classify puzzle pieces. The design was thus focused on solving problems that might hinder the success, accuracy, and/or reliability of these goals.

If piece detection and classification is performed on an image that is not centred in relation to the puzzle-construction surface, it will be extremely difficult to translate the piece information into physical real world parameters. This would be owing to the system having no reference to the construction surface. To solve this problem, fixed markers were placed on the construction surface. With the aid of image-thresholding and image-transformation techniques, these markers could be detected, and the system could be adjusted accordingly. This allowed all piece information to be directly and accurately translatable into real world parameters. As an alternative approach, these markers' pixel coordinates could be hard-coded into the system, as to simplify the procedure and to decrease the algorithm's runtime. The drawback of this alternative approach, however, is that any slight movement of the camera would cause an unwanted offset in the system. It was thus decided to stick to a dynamic approach that could adapt to a wide range of camera offsets.

In order to detect pieces, the system needs to be able distinguish between foreground and background. One approach would be to use the construction surface's colour to perform image thresholding in the RGB colour space. The problem with this approach is that the RGB colour space is extremely sensitive to varying lighting conditions. It was thus decided to rather do image thresholding in the HSV colour space, which is far less susceptible to inconsistent lighting. To improve the image thresholding even further, a simple image filter is applied prior to thresholding. Once image-filtering and thresholding have been successfully applied, an image can easily be binarised in order to prepare it for contour-detection.

Once a binarised image is available, contour detection is applied in order to locate all piece edges. An image cannot be perfectly binarised and, as a result, the contour detection algorithm often produces many additional, unwanted, contours of various strange shapes and sizes. Luckily, the erraticness of these contours can be used to identify and avoid them. The system has been calibrated to a specific puzzle piece shape and size, which allows the algorithm to identify, with sufficient accuracy, which contours are related to puzzle pieces and which to discard. These

calibrations were tweaked even further, which allowed this method to identify stacked puzzle pieces, as their shape and size are similar to that of a single puzzle piece, only slightly larger. It was thus easy to distinguish piece contours from error contours as well as overlapping piece contours from single piece contours.

To separate overlapping pieces, adequate spacing is first found by searching for areas large enough to accommodate such overlapping pieces. By searching for such areas in a binarised image, the algorithm simply needs to find a large enough area that contains purely background. Once space has been found, the algorithm determines an adequate pick-up location for all the overlapping pieces that need to be separated. Essentially, such a pick-up point is found by locating a spot inside a pieces' contour large enough for the piece actuator. This is achieved with the *Point Polygon Test*.

In order to classify non-overlapping pieces, their contours are used to obtain information about their shape, orientation, and centre location. Firstly, the centre pixel coordinates of a piece's contour is used to identify its centre location. Each contour, and its corresponding puzzle piece, is then saved into a separate new sub-image file. The *Hough Line Transform* is then applied to each of these sub-images in order to estimate the angle that each piece needs to be rotated by in order to orientate the piece upright[3]. Once each piece is orientated upright, the location and magnitude of defects present in the contours are used to identify specific piece edge types (such as straight edge, tab, or indent). Each puzzle piece's information is then stored into data objects, which is used by FU2.2 to derive a software puzzle solution.

FU2.2 - Puzzle Solving

This sub-unit is responsible for interpreting the information it receives from FU2.1 in order to derive a software puzzle solution. As mentioned in Sub-section 1.3, the two main approaches to this is by means of colour-matching or shape-matching, the advantages and disadvantages of which are shown in Table 5 above. As already stated, a combination of these two methods is used to find a solution. Shape-matching uses piece edge type information (such as the locations of straight edges, tabs, and indents) to determine possible piece neighbours, and colour-matching uses piece colour information to determine the similarity between two pieces' colour content. By first finding all valid piece neighbours with shape-matching, colour-matching is then used to find the most likely neighbouring piece out of this group of candidate pieces. By first selecting an adequate starting piece, the algorithm then iterates through this process until all pieces are in a suitable solution location.

To increase the likelihood of finding a correct solution, multiple solutions are build from different starting pieces. Such solutions are then scored and the best solution is chosen. Two solution scoring techniques were considered. The first accumulated a solution's colour-matching values to assign a score, and the second used the amount of background visible in the software solution image to assign a score. The former was found to be inconsistent and unreliable, where as the latter provided consistent and accurate results. As such, the latter technique was chosen.

[3]Upright refers to a piece orientation in which each of its sides are either parallel or perpendicular to all the borders of the image.

FU2.3 - Movement Calculation

This is the only sub-unit that directly interacts with FU3 and, as such, was designed and calibrated alongside it. This sub-unit uses solutions found by FU2.2 and the piece information found by FU2.1 to create movement commands which it then sends to FU3. These commands include actions such as piece rotations, piece movements, temporary piece stacking, and piece placement. These commands are sent to FU3 via a serial communication channel, which is facilitated by *PySerial*. The main challenge here is to convert pixel coordinates into accurate real world piece manipulation commands, which FU3 can interpret. One approach is to relate the image dimensions to the gantry dimensions, but this approach was found to be rather inaccurate. Ultimately, a calibration mechanism was created, which proved to offer the best results. A detailed description of this mechanism, and its results, can be found in Sub-sub-section 2.2.3.

FU2.4 - System Controller

This sub-unit is simply in charge of controlling the other sub-units of FU2. It calls upon these sub-units and monitors their outputs in order to determine when other sub-units are needed. If a specific task should fail, this sub-unit will simply recall preceding sub-units until the task is successful. If, for example, a correct solution cannot be found, this sub-unit will instruct FU2.1 to acquire a new image from FU1 and to repeat all its functionality on said new image. Once FU2.1 is successful with all its operations, the controller will instruct FU2.2 to repeat its functionality on the new dataset obtained from FU2.1. Additionally, this sub-unit provides FU2.5 with all the relevant system information.

FU2.5 - Graphical Display

This sub-unit was designed in *QtDesigner* and was converted to *Python* code by means of *PyQt*. This GUI was designed with a minimal interface so as to provide a streamlined experience. This sub-unit allows users to initialise the entire system by means of a start button, after which it displays the software solution time, hardware solution time, and the puzzle piece count. Most importantly, this sub-unit provides users with the camera live feed, piece detection image, piece classification image, and software solution image, which demonstrates and validates the functionality of the entire software unit. It also provides buttons that can capture and store each of these respective images.

2.1.3 FU3 - Mechanical Control Unit

This unit comprises of all the electronic and electro-mechanical hardware of the system. The main goal of this unit is to translate commands from FU2 onto FU4, which will facilitate puzzle construction. Where applicable, this unit's sub-units were prototyped and tested on breadboards. Once each sub-unit's functionality was satisfactory, the system was integrated and fully tested. The sub-functional block diagram of FU3 is shown in Figure 6 below.

It is important to note that a minor change has been made to this functional unit since the initial concept design. The original FU3.5 (Position Control) was found to be unnecessary, as the system functioned accurately enough without the need for such feedback control. Instead, limit-monitoring switches were used to align the gantry and to check for limit-breaching movements.

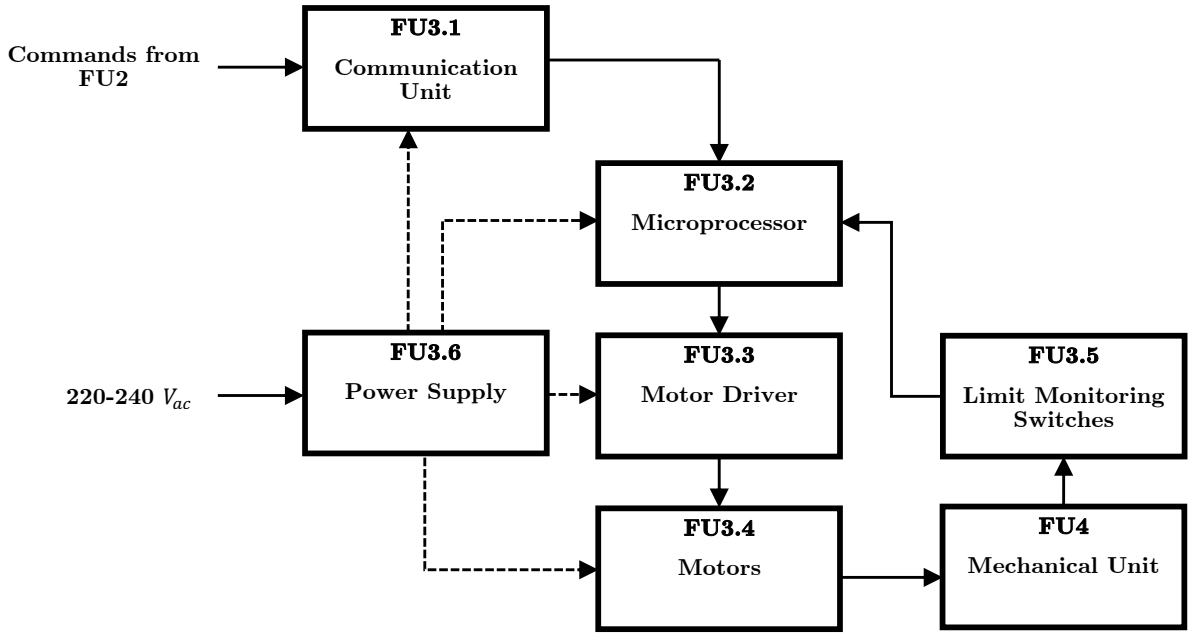


Figure 6.
Functional block diagram of electro-mechanical hardware and firmware

This sub-unit was tested by rigorously applying all possible commands to the system via *Putty*. The outputs of each command was then closely monitored so as to ensure they were performed accurately and correctly.

FU3.1 - Communication Unit

This sub-unit consists of a *MAX233* line transceiver, which facilitates communication between the PC and the microprocessor by ensuring all serial communication voltages are at the correct levels. This device was chosen as it contains built-in noise filtering capacitor, which decreases the likelihood of communication errors.

FU3.2 - Microprocessor

This sub-unit is responsible for housing the firmware that orchestrates the entire unit. Initially, it was decided to use an *ATmega328P-PU* microprocessor as it is affordable and widely available. At the time of implementation, however, all the *Atmel AVR JTAGICE mkII* programming devices available had broken SPI programming interfaces, which is the only interface the *ATmega328P-PU* supports. It was thus decided to use a device that supports a JTAG programming interface, as this interface was still intact of a few programming devices. The *ATmega1284P-PU* microprocessor was thus chosen, as its architecture is similar to the that of the *ATmega328P-PU*, which allowed for development to remain mostly unaffected. The firmware for this device was programmed in *C* programming language in *Atmel Studio* IDE.

An external crystal oscillator was connected to this device to provide it with a stable clock signal. Such an external oscillator is needed as the device's internal clock is not stable enough to facilitate high speed serial communication. Capacitors were connected to the device's power lines to smooth out any power supply ripple.

FU3.3 - Motor Driver

This sub-unit consists of various *DRV8825* high-current stepper motor drivers, which each control one of the system's stepper motors. These devices enabled the microprocessor to control high-current stepper motor with simple low-current logic.

FU3.4 - Motors

Two $1.5A$ stepper motors are used to translate the x- and y-axis of the gantry. A $0.5A$ stepper motor, along with a gearing configuration, is used to rotate pieces. A $90g$ $1.3kg/cm$ servo motor is used to raise and lower pieces and a $200g$ $15kg/cm$ servo motor is used to engage and release the vacuum mechanism.

FU3.5 - Limit Monitoring Switches

These switches are interfaced with the microprocessor such that they can be managed on an interrupt basis. When any of these switches are pressed, the program routine is halted, the responsible switch is identified, and the necessary actions are taken. This method ensured that the gantry does not move out of bounds, which could cause damage to the system.

FU3.6 - Power Supply

A $12V$ $20A$ switching mode power supply is used to power the entire system. Along with this power supply, various *7805T* linear voltage regulators are used to lower the $12V$ input to $5V$, which is the operational voltage for most of the system's components. More than one voltage regulator is used in order to reduce the current demand from each regulator, too much of which would cause overheating and failure.

2.1.4 FU4 - Mechanical Unit

This unit facilitates mechanical puzzle construction. Puzzle construction is completed with a robotic gantry, puzzle piece manipulator, and puzzle piece rotator; the the mechanical hardware of which all form this functional unit. The gantry operates along the x- and y-axis whereas the manipulator functions along the z-axis. The piece rotator is used to correctly orientate pieces. The destination location and desired orientation of each piece, as obtained from FU2, is sent to FU3 via a communication channel. FU3 then instructs this unit collect and transport each piece to its correct location and orientation.

The specific gantry used for this project dictated the physical capabilities of the system. It was thus important to review this gantry's limitations in order to elegantly design around them. Table 6 below shows the specifications of the *MakeBlock Gantry Kit (v2)*, which serves as the robotic transportation medium for puzzle construction.

MakeBlock Gantry Kit (v2) Specifications	
Physical Dimensions $(L \times W \times H)$	$620mm \times 620mm \times 140mm$
Working Area $(X \times Y)$	$310mm \times 390mm$
Max Working Speed	$50mm/s$
Max Carry Weight	$250g$

Table 6.
Gantry kit specifications

Some of these specifications were important to consider as they placed a few constraints on the puzzle solving problem. To name a few:

1. adequate space for construction might not always be available owing to spacial constraints,

2. the piece manipulator cannot exceed $250g$ unless the bulk of its components are mounted elsewhere, and
3. puzzle pieces might have to be purposefully stacked to make room for puzzle construction.

All these constraints, along with their implications, are addressed in the detailed Sub-section 2.2 below.

2.2 DESIGN AND IMPLEMENTATION

The majority of this system's functionality and complexity resides within FU2 (Puzzle Solving and Control Software Unit) and FU3 (Mechanical Control Unit). As a result, these units have each been sub-divided into smaller functional units, as described in Sub-section 2.1 above. In order to test and validate the functionality of these sub-units, however, the smaller main units, namely FU1 (Image Acquisition Unit) and FU4 (Mechanical Unit), will be needed. FU2 cannot function without inputs from FU1, and these units will thus be designed and implemented alongside each other. The combination of FU1 and FU2 will hereafter be refereed to as the *PC System*. Similarly, FU3 cannot function without interactions with FU4, and will be designed and implemented together. The combination of FU3 and FU4 will hereafter be referred to as the *Gantry System*.

2.2.1 PC System

Reflection Error Correction

The initial camera setup had the camera facing perpendicular to the puzzle surface. This caused the direct reflection from the lights to skew certain colour values in specific areas. To overcome this, the camera was placed at an angle such that the camera would not directly receive strongly reflected light. This is illustrated in Figure 7 below.

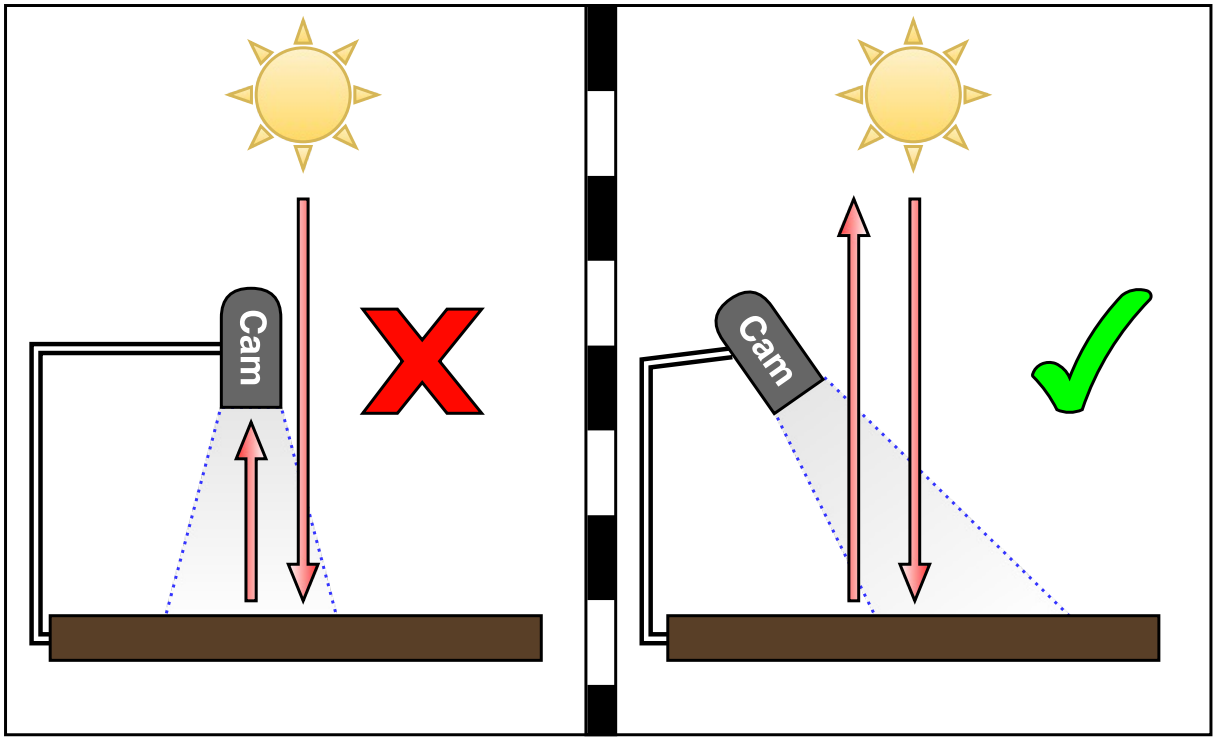


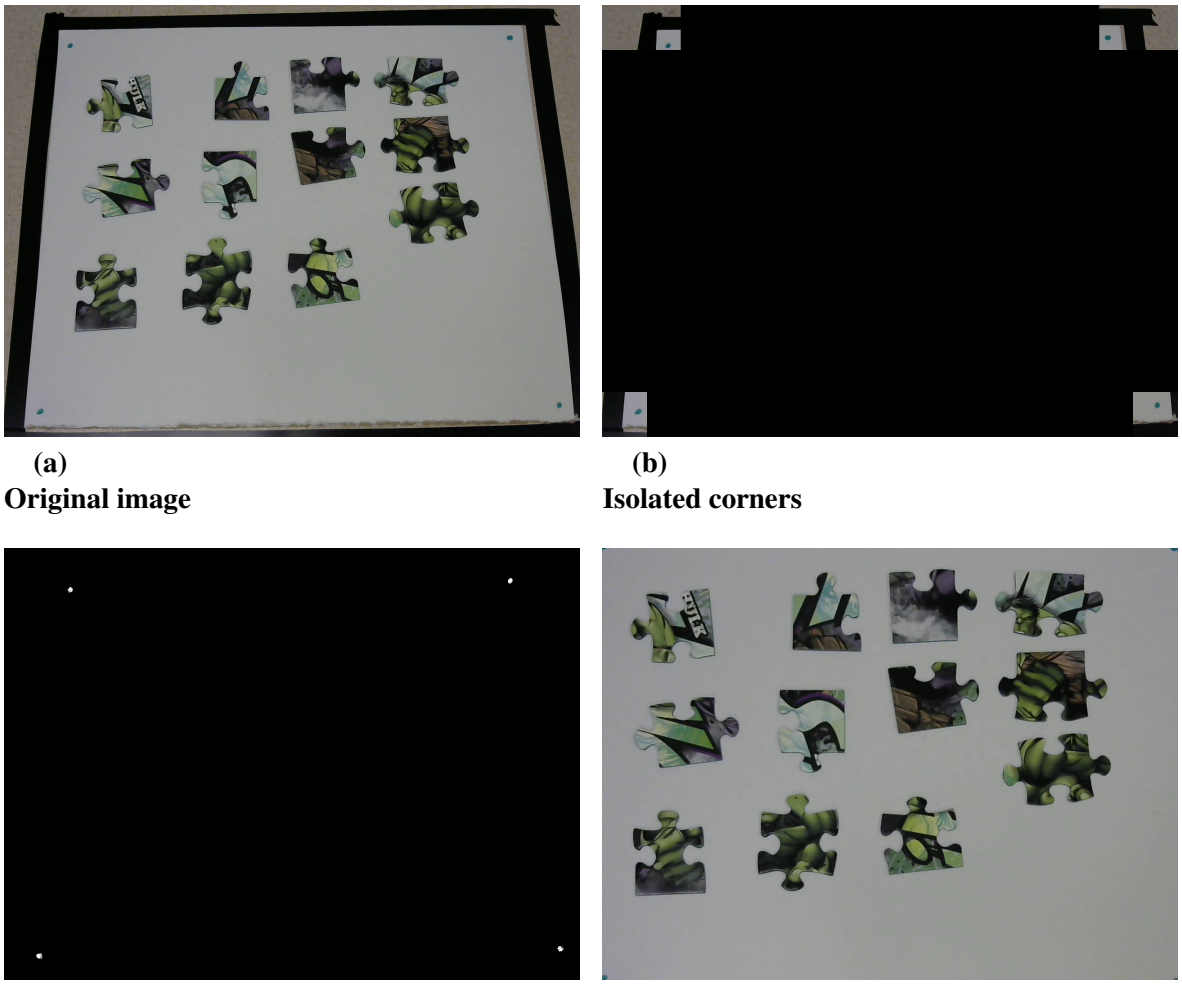
Figure 7.
Reflection error mitigation technique [8]

The non-perpendicular angle of the camera did, however, introduce a few drawbacks, mainly in that it skewed the system's vision of the puzzle layout. Luckily, this error was fully resolved by the use of a well-calibrated, intelligent perspective transformation algorithm.

At its core, this algorithm takes various points on a skewed image and manipulates them to predetermined points; by doing so, shifts and changes the perspective of the entire image to a more desirable one. Figure 8 below depicts the steps this algorithm takes, with Figure 8a showing the original image. First the image corners are isolated by applying and image masking function. This masking function is dynamic as it will continue to increase the foreground corner size if it cannot detect all the coloured corner markers, and will continue to do so until they are all found. Additionally, this sub-function has been carefully calibrated, such that it is successful on most of its first attempts, thus additional time consuming iteration are not needed. Figure 8b shows the corner isolation step and Figure 8c shows the system detecting the markers. The transform function is then applied to said points in order to make them the new corner points of the final image, resulting in Figure 8d, the final unskewed image.

The minor drawback of this algorithm is that some image resolution on the top part of the image is lost, as it has effectively been digitally zoomed. Since the alteration is not drastic in any way, these effects should be, at most, near-negligible.

A major advantage to this algorithm, however, is that it provides the system with four fixed reference points to the physical construction surface. These reference points are of vital importance as they provide the system with an accurate mechanism with which to translate pixel coordinates into physical real world coordinates.



(a)
Original image

(b)
Isolated corners

(c)
Corner colour mark identification

(d)
Transformed output

Figure 8.
Perspective transform algorithm process [4]

Piece Detection and Classification

Now that a reflection-free, reference image is available, the system can proceed to detect and classify all present pieces, as described in Sub-sub-section 2.1.2 above.

Figure 9 below illustrates the flow diagram of the piece detection and classification algorithms. The blue block refers to the calibration steps and will be dependant on factors, such as puzzle shape and background colour. The green blocks represent the steps performed by piece detection, as described in Sub-sub-section 2.1.2 above. The red blocks are steps that overlap between piece detection and piece classification, as its functionality is used by both phases. The yellow block represents the piece classification step, as described in Sub-sub-section 2.1.2 above. The purple blocks represent steps executed by other functional units and sub-units that are not strongly related to the functionality and operation of this functional sub-unit. The orange blocks represent interaction with other functional units and sub-units directly connected to this sub-unit.

[4]These images were autonomously generated by the *PC System* and have since not been altered in any way. Any visual graphics present on these images were drawn by the *PC System* so as to demonstrate its functionality.

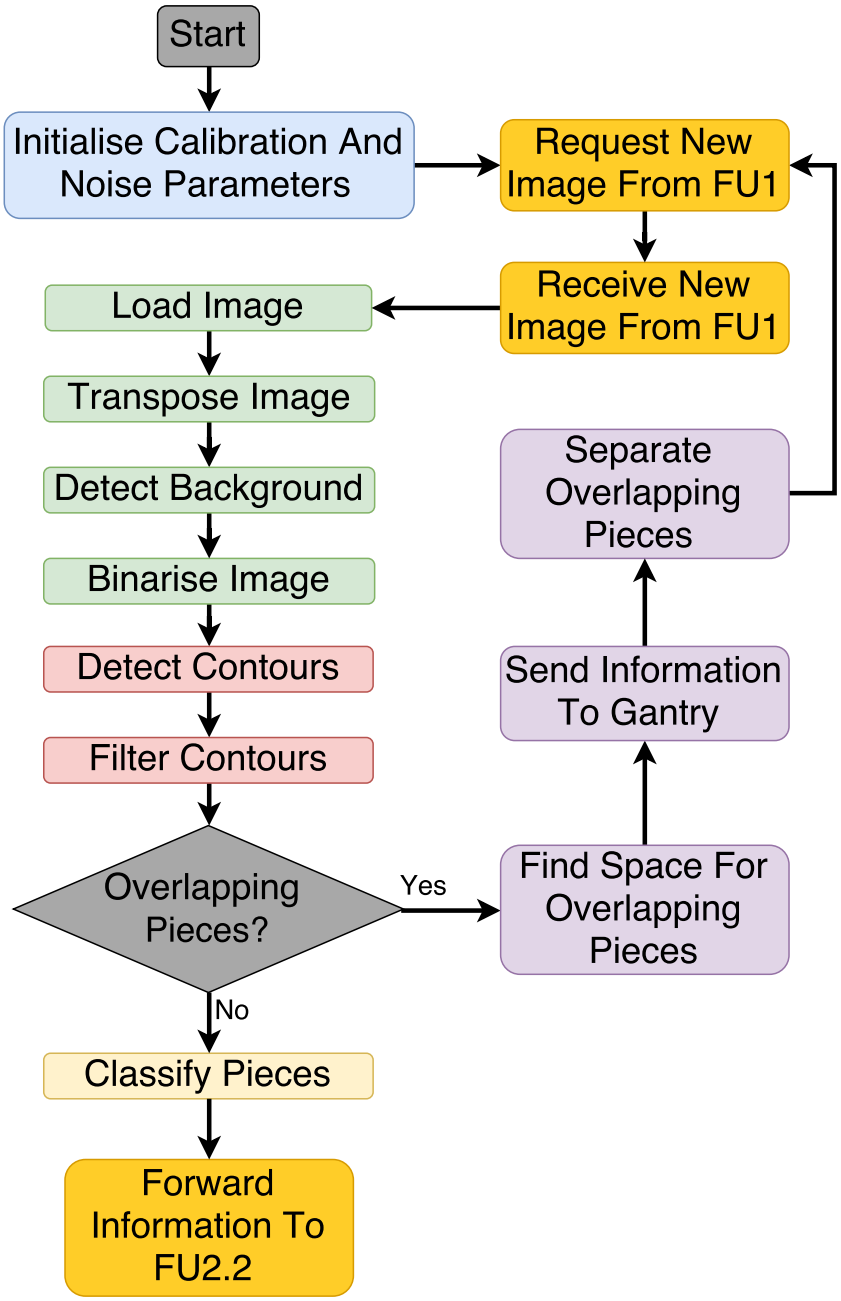
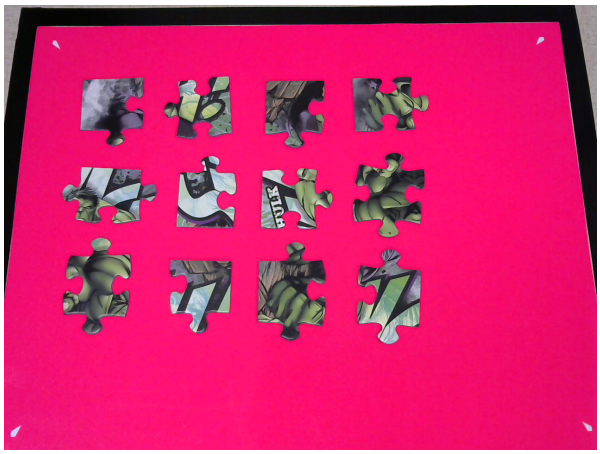


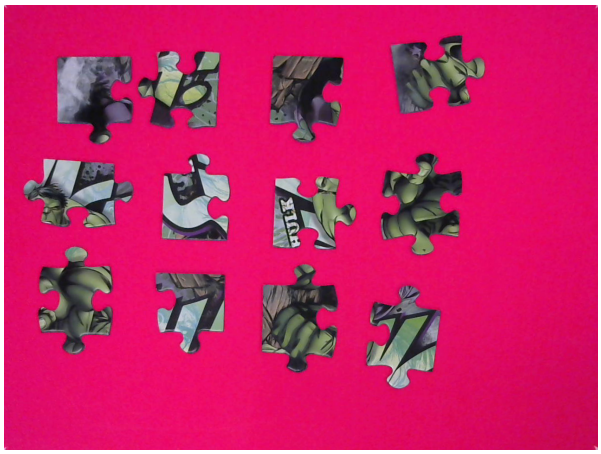
Figure 9.
Piece detection and classification flow diagram [8]

The piece detection algorithm can detect, index, and label puzzle pieces and it can indicate which of these pieces belong to an overlapping cluster. The algorithm was designed to handle a large amount of colour deviation, such as the ones introduced by shadows and reflections, and to operate in real-time (>20 FPS). This algorithm is capable of smoothly and accurately identifying puzzle edges, which makes it easier to classify pieces.

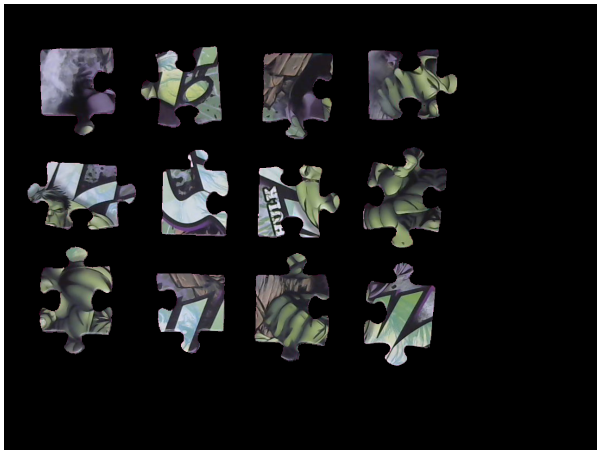
Figure 10 below shows the outputs for all the intermediary steps that form the piece detection algorithm, as described in Sub-sub-section 2.1.2 above and as illustrated by the green blocks in Figure 9.



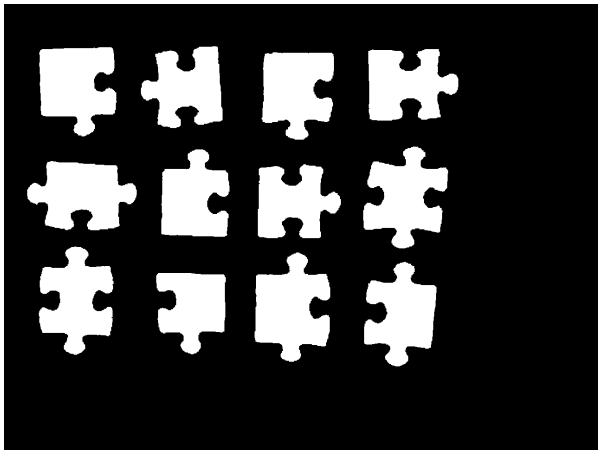
(a)
Original image



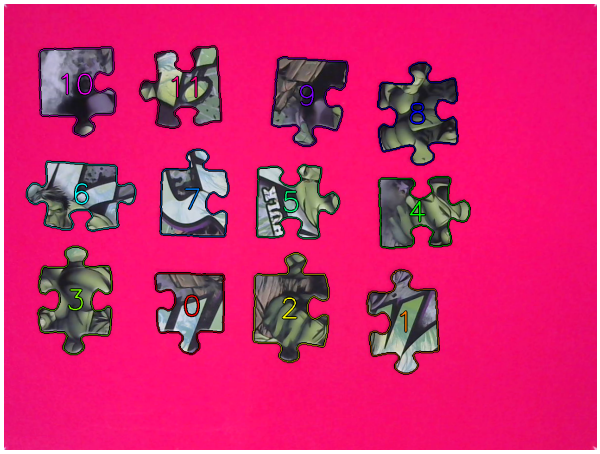
(b)
Transposed image



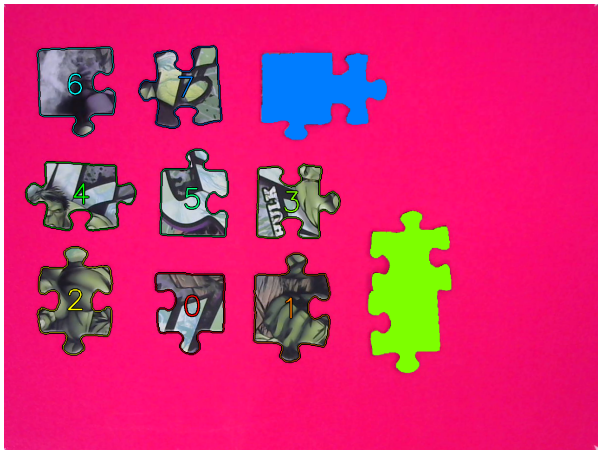
(c)
Image with blacked-out background



(d)
Binarised image



(e)
Detected pieces



(f)
Detected overlapping pieces

Figure 10.
Piece detection algorithm process [4]

Figure 11 below show the algorithm functioning with a shadow present in the left half of the image. This was achieved by using the *Hue Saturation Value* (HSV) colour space. *Hue* represents the specific colour (ranging from 0 to 180), *Saturation* represents the amount of white the *Hue* colour contains (ranging from 0 to 255), and *Value* represents the amount of black the *Hue* colour contains (ranging from 0 to 255). This is better illustrated in Figure 12 below. The system thus deals with shadows by searching for a specific *Hue* and a wide range to *Values*; this will allow darker, shadow-covered, parts of the colour to still be detected. Similarly, the system deals with minor reflections, by searching for a specific *Hue* and a wide range of *Saturations*.

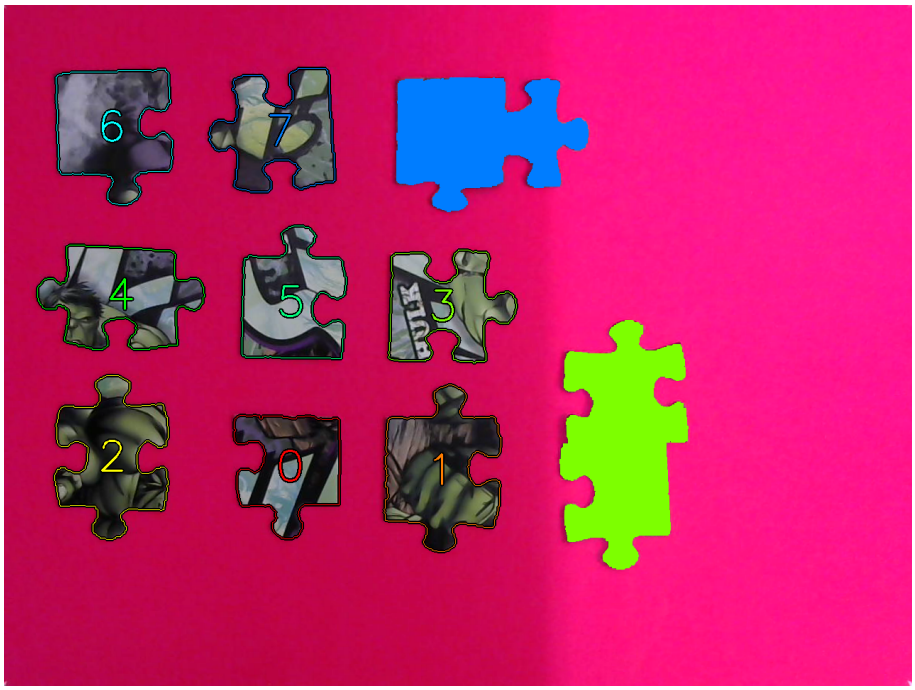


Figure 11.
Piece detection with strong shadow present [4]

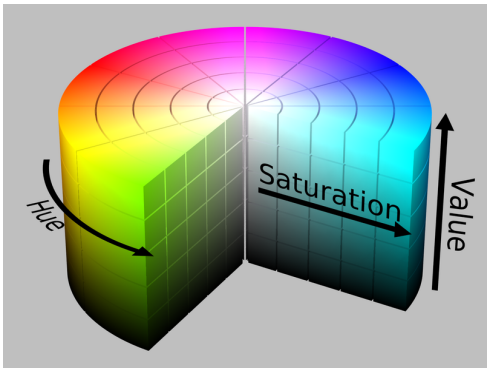


Figure 12.
HSV colour wheel [9]

Before all the pieces can be classified, all the overlapping pieces need to be separated. Sub-sub-section 2.1.2 above describes, in part, how overlapping pieces can be identified, and how suitable pick-up points on such overlapping pieces can be found.

Firstly, the overlapping piece contours obtained from the piece detection algorithm are used to draw a fit line through these contours. Such a fit line is one that attempts to pass through the majority of a contour's centre of mass. In essence, such a line passes through a contour and, whilst inside it, stays as far away as possible from its boundaries whilst maintaining a straight trajectory. Once such lines have been found for each overlapping piece contour, the algorithm determines where along this line the puzzle piece actuator will be able to attach itself. For this, the *Point Polygon Test* is used. The *Point Polygon Test* determines if a point is within a contour or not. This test works by first projecting a line, from the point in question, in a random direction. If this line crosses the contour an even number of times (including zero) it is outside the contour, and if it crosses the contour an odd amount of times, it is inside the contour. By utilising this test, the algorithm searched along a fit line to determine which points are within its contour. Additionally, this test also provided the distance to the nearest contour boundary from such a point. This method is thus used to find all the points within the contour that are large enough for the actuator to pick the piece up from.

Next, a suitable vacant location is found for each overlapping piece. This is done by using the binarised image obtained from the piece detection algorithm. This step searches the binarised image for a location big enough to fit the maximum piece dimensions. This ensures that no separation attempt leads to another heap of overlapping pieces. This is done by searching various quadrants of the binarised image for the necessary space. Only a few quadrants are considered at first, for efficiency's sake, but if no available space is found, the amount of search quadrants are increased until a vacant location is found.

Figure 13 below shows the algorithm in action by demonstrating the pick-up points and drop-off destinations for each of the overlapping piece clusters, along with other additional pick-up points, should the current ones fail. Figure 14 below shows the algorithm adjusting its drop-off locations (in real time) when a foreign object is introduced into one of its current destination areas.

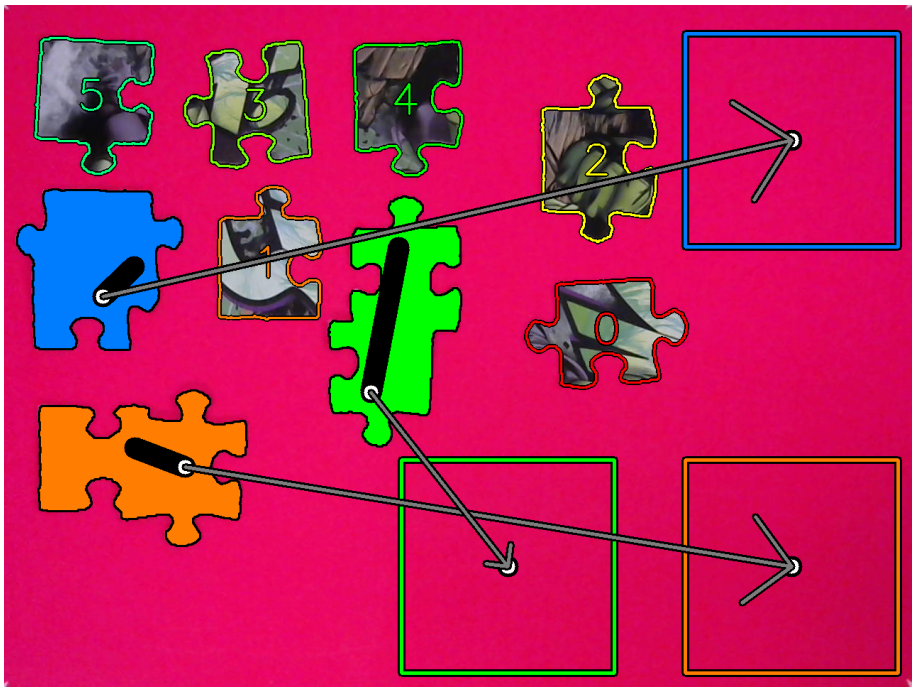


Figure 13.
Overlapping pieces separation [4]

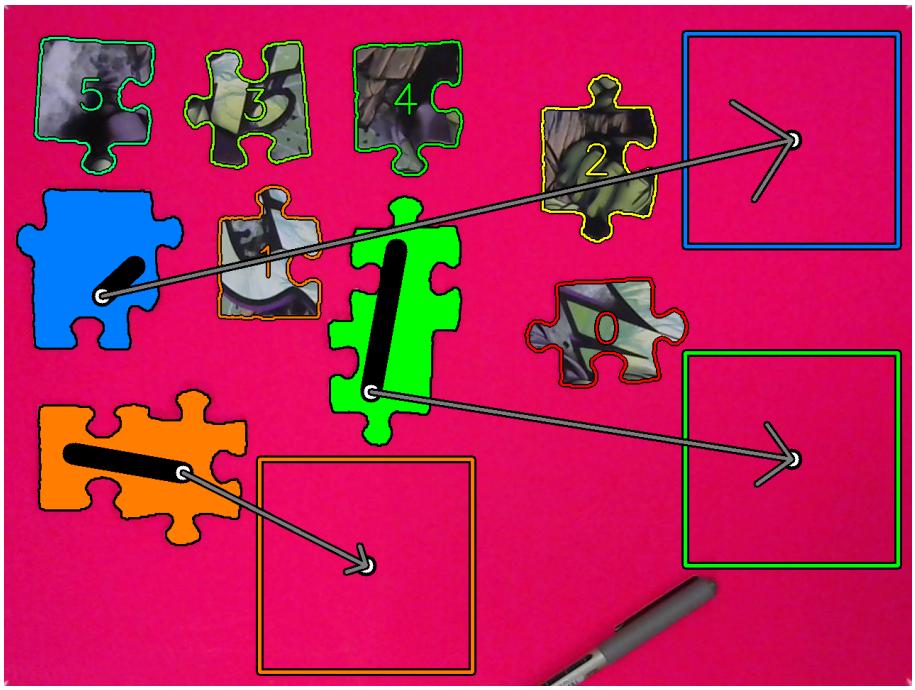


Figure 14.
Overlapping pieces separation with foreign object [4]

Once all pieces have been separated, the system can proceed to classify them. Piece classification is done by systematically creating an object for each puzzle piece. These puzzle piece objects will contain all the information and functionality needed to derive a solution from them. The UML class diagram for the puzzle piece class is shown below in Figure 15.

Puzzle_Piece
+ index : int + solve_index : int + center_location : int[2] + stack_height : int + it_class : int[4] + angle : int + file_name : string
+ print_index() : void + print_it_class() : void + print_center_location() : void + print_angle() : void + is_corner() : bool + is_edge() : bool + corner_location() : int + set_corner(int) : void + rotate_90_ccw(int) : void + copy() : Puzzle_Piece

Figure 15.
Puzzle piece class diagram [8]

Most of these members can be found by using the piece contours obtained for the piece detection algorithm.

+ index – A piece's index is obtained by using its corresponding contour's index in the valid piece contour list.

+ solve_index – This index is assigned by the puzzle solving algorithm. This parameter indicates the order in which the solution will be built.

+ center_location – A piece's centre location can be found by using the *moments* of its corresponding contour. An image *moment* is a certain particular weighted average of the image pixels' intensities, or a function of such *moments*, usually chosen to have some attractive property or interpretation [10]. Listing 1 below shows the Python code that can achieve this. Here, contours is a list of all the valid piece contours, and index is the index of the piece being examined.

```
M = cv2.moments(contour[index])

center_x = int(M['m10']/M['m00'])
center_y = int(M['m01']/M['m00'])

center_location = tuple([center_y,center_x])
```

Listing 1.
Finding a contour's center

+ stack_height – A piece's stack height is initially assigned as zero, and indicates the amount of pieces that reside underneath a piece. This parameter is useful to check to which depth the piece actuator needs to lower in order to pick up specific pieces. When a piece is moved and stacked, this parameter is updated accordingly.

+ it_class – This parameter refers to a piece's indent/tab/edge layout, basically, where a piece's indents/tabs/edges are in relation to its orientation. This is quite a complex parameter to obtain and thus requires a more detailed explanation.

First, the *Hough Line Transform* is used to obtain details about all the straight lines the algorithm can detect. A line can be represented as $y = mx + c$ or in parametric form, as $\rho = x\cos\theta + y\sin\theta$ where ρ is the perpendicular distance from origin to the line, and θ is the angle formed by this perpendicular line and horizontal axis measured counter-clockwise [11]. Figure 16 below illustrates the parametric form.

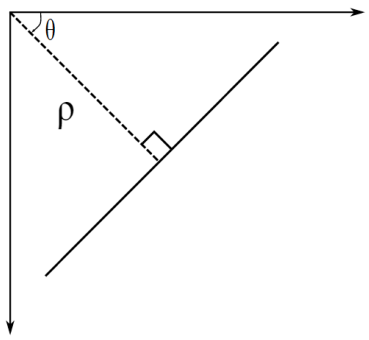
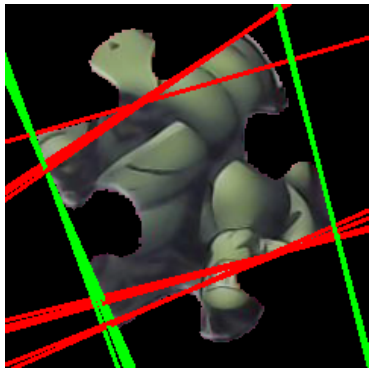


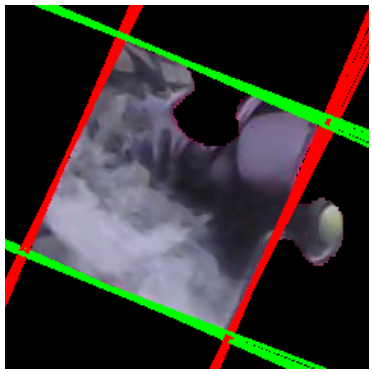
Figure 16.
Parametric line form [11]

The *Hough Line Transform* returns an array containing the ρ and θ values for each line that it detects. Because most puzzle pieces will have a roughly square shape, the majority of the lines are expected to be, approximately, either parallel or perpendicular to one another. Thus, by averaging all the line-angles that are close enough to one another, an angle can be deduced for each of these respective line groups. To better illustrate, Figure 17a and Figure 17c below show the detected piece line clusters for two separate puzzle pieces. The green lines are all roughly parallel and so are the red lines. By using the average angle (θ) of the line group containing the most lines, an accurate estimation of each piece's off-setted angle can be derived and corrected, as can be seen in Figure 17b and Figure 17d. As the purpose here is simply to orientate each piece upright, the piece is rotated by the remainder of the average angle (θ) divided by 90° (i.e. θ *mod* 90). This helps to minimise the magnitude of rotations that the piece rotator needs to perform, which improves the mechanical puzzle construction speed.

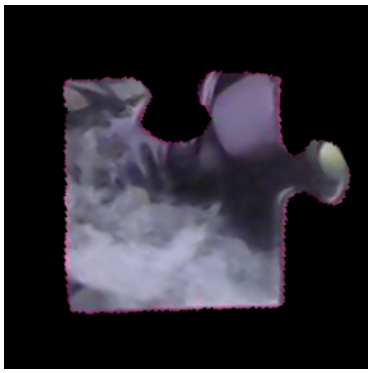


(a)
Piece 1: line detection

(b)
Piece 1: adjusted image



(c)
Piece 2: line detection



(d)
Piece 2: adjusted image

Figure 17.
Piece line detection [4]

Once a piece image has been correctly orientated, its contour is reacquired and used to identify indents/tabs/edges. This is done by making use of the *Convex Hull* and *Convexity Defects* functions available in *OpenCV*. These two functions together estimate a "best fit" straight contour around the original contour and then calculates the deviation points and distances from the original contour to the "best fit" contour. This is used to identify tabs as they will have the greatest deviation from this "best fit" contour. As the images are now orientated correctly, the

locations of the remaining unclassified sides are roughly known, and their distance from the piece centre can be used to judge whether they are edges or tabs.

Once this step is done all the indent/tab/edge information is captured in a matrix. Each column in this matrix represents an artefact (edge, tab or indent) on the piece. A -1 represents an indent, a 0 represents an edge, and a 1 represents a tab. Column 0 refers to the piece's current top side where column 1 refers to its current right side, and so forth. To illustrate, a classification array of $[1,\ 0,\ 0,\ -1]$ mean that the piece, going clockwise from the top, has a tab, followed by straight edges on its right and its bottom, ending on an indent on its left.

+ angle – A piece's angle indicates by how many degrees a software piece is positioned counter-clockwise from its physical angle; in essence, by how many degrees a piece needs to be rotated to be upright. This angle is obtained from the *Hough Line Transform* step above.

+ file_name – This refers to the file name of the individual piece images. This approach allows each object to only save the file name of an image, rather than the entire image itself. When a piece image is needed, it can simply be opened by using the piece's file name.

Table 7 below lists and describes all the methods of the puzzle piece class.

Puzzle Piece Class Method Description	
+ print_index() **+ print_it_class()** **+ print_center_location()** **+ print_angle()**	These functions are used for debugging purposes, as they print out relevant puzzle information to the console.
+ is_corner()	Checks if the piece is a corner piece. If the piece is a corner piece, return *True*, else return *False*.
+ is_edge()	Checks if the piece is an edge piece. If the piece is an edge piece, return *True*, else return *False*.
+ corner_location()	Retrieves the corner location of the piece if it is a corner piece. Corner location : 0 = top-left; 1 = top-right; 2 = bottom-right; bottom-left; -1 = not a corner.
+ set_corner(int x)	Rotates the piece until its corner is at location x and updates the piece's parameters accordingly. Corner location : 0 = top-left; 1 = top-right; 2 = bottom-right; bottom-left.
+ rotate_90_ccw(int x)	Rotates the piece 90° counter clockwise x times and updates the piece's parameters.
+ copy()	Creates and returns a deep copy of the piece.

Table 7.
Puzzle piece class method description

Figures 18 to 20 below illustrates the integrated functionality of the piece detection and classification algorithms. Figure 18 below shows an initial puzzle layout, Figure 19 shows the system detecting the layout's pieces, and Figure 20 shows each individual puzzle piece as it is classified and indexed by the system. For Figure 20, an *I* indicates an indent, a *T* indicates a tab, and an *E* indicates an edge.

Figure 18.
Initial puzzle layout [4]

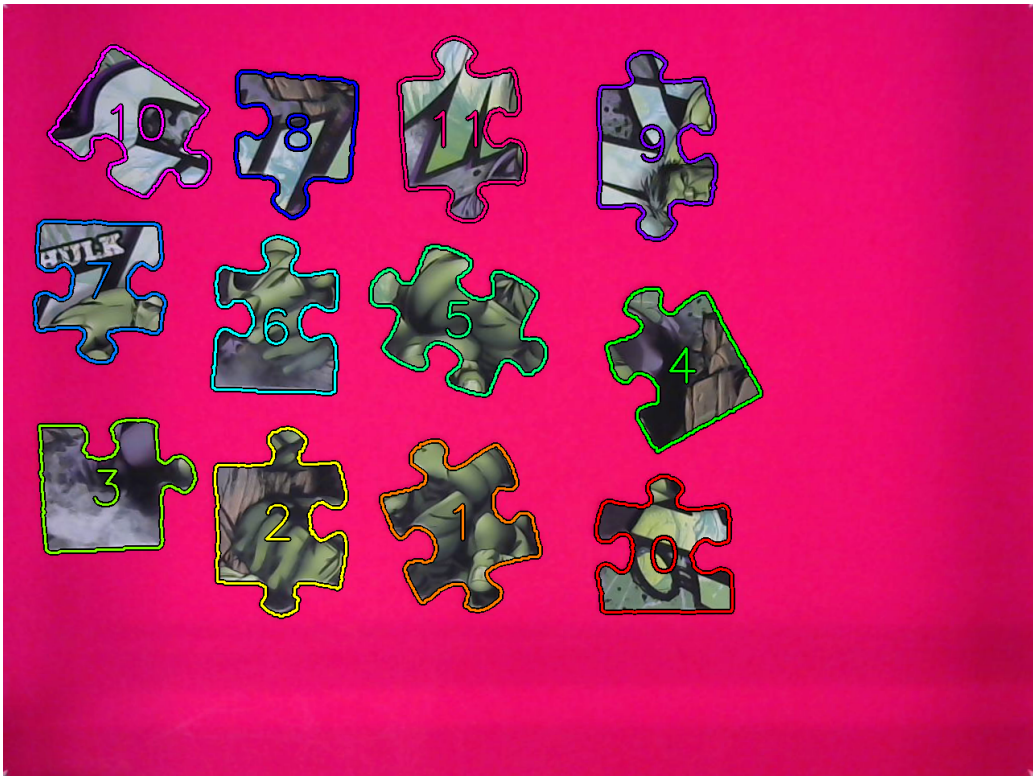


Figure 19.
Piece detection [4]

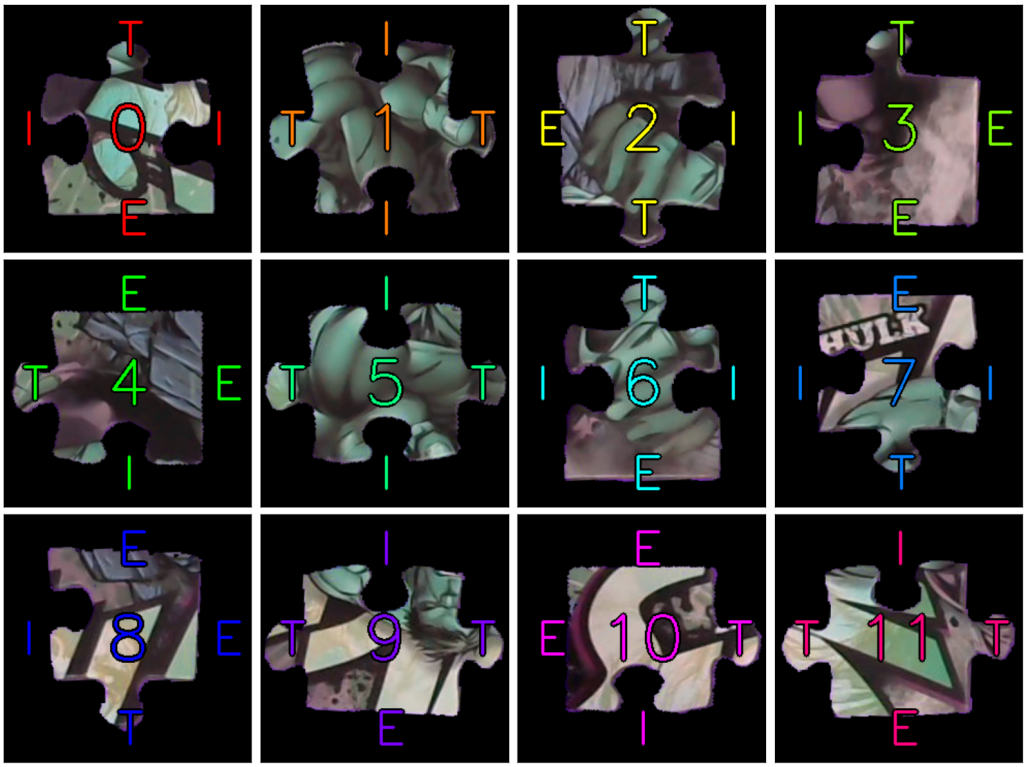


Figure 20.
Piece classification [4]

Puzzle Solving

Now that all the puzzle pieces can be correctly orientated and classified in software, a puzzle-solving algorithm is applied to derive a solution. This puzzle-solving algorithm partially mimics the puzzle-solving approach of a human, as it will start with corner and edge pieces and progress towards the middle until a solution is found. This algorithm is thus be broken into three core operations, namely:

1. starting piece selection,
2. shape-matching, and
3. colour-matching.

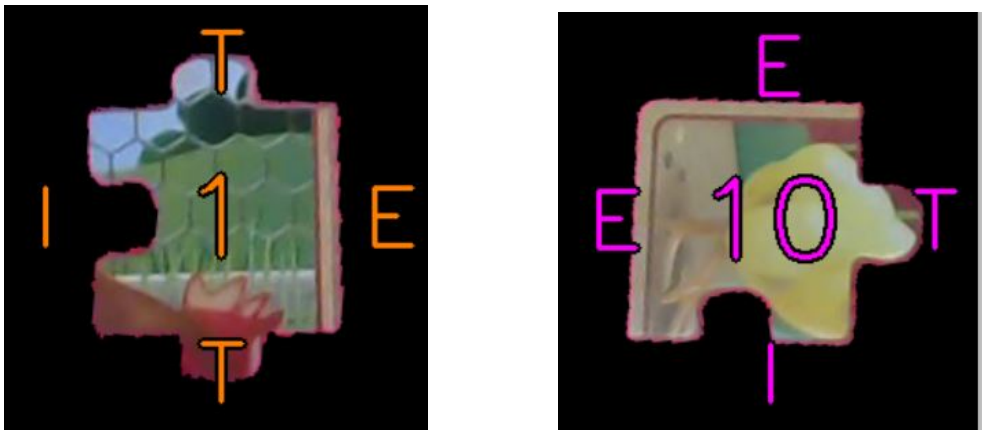
These operations do not necessarily operate independently, but their functionality forms the core structures of the puzzle-solving algorithm.

As most humans would do, a corner piece is chosen as a starting piece. Such a corner piece is distinguished by means of its indent/tab/edge layout, as discussed previously. To improve accuracy, all four corner pieces are used as starting pieces to create four unique, separate, solutions. Each piece's deep copy function is utilised to create four separate and identical puzzle sets, which allows the four puzzle to be built independently from one another. A solution score (discussed below) is then assigned to each such puzzle solution, which is then used to find the most optimal solution out of the four choices.

Shape matching is once again achieved by utilising the piece indent/tab/edge layout, with the aid of a *Fit Matrix*. This *Fit Matrix* is a 3D-matrix, representing the layout information of a puzzle

and is used to find possible locations and orientations for puzzle pieces. The first two axis of this matrix represents the puzzle layout, and its exact values are calculated by making use of the puzzle size and the amount of edge pieces. For example, a puzzle with 12 pieces, of which 8 are edges, will be 4 by 3, and an 18 piece puzzle with 10 edges will be 6 by 3. These form the x- and y-axis for the matrix, and as the *MakeBlock Gantry Kit(v2)* has roughly a 3:4 aspect ratio, the larger of the two dimensions will be assigned to the x-axis. For all the experiments performed below, a 4 by 3 matrix size was used.

The third dimension of a *Fit Matrix* represents each puzzle location's *Fit Value* (not to be confused with each other). Another way to view this is to see a *Fit Matrix* as a 2D-matrix containing 1D *Fit Values*, thus making the *Fit Matrix* 3D. In order to better understand how *Fit Values* are used, let's first review how a piece's indent/tab/edge layout is represented. This is done with a 1D-matrix, where the zero-ith position refers to the artefact (indent/tab/edge $\Rightarrow -1/1/0$) on the piece's top, the second to its right, the third to its bottom, and the fourth to its left. To illustrate, the piece shown on the left in Figure 21 below has an indent/tab/edge matrix of $[1,\ 0,\ 1,\ -1]$, and the piece on the right has an indent/tab/edge matrix of $[0,\ 1,\ -1,\ 0]$.



(a)
I/T/E Matrix ⇒ [1, 0, 1, -1]

(b)
I/T/E Matrix ⇒ [0, 1, -1, 0]

Figure 21.
Indent/Tab/Edge matrix example [4]

When a piece is inserted into a puzzle at a specific location, that location's corresponding *Fit Matrix* value (third dimension) becomes the piece's indent/tab/edge matrix. When a location is vacant however, its corresponding *Fit Matrix* value represents that location's *Fit Value*. This *Fit Value* is designed such that the element-wise addition between it and a piece's indent/tab/edge matrix will all result in all zeros if, and only if, the piece can fit in the *Fit Value's* corresponding location.

In other words, if puzzle location $[x,\ y]$ is vacant, $\textit{Fit Matrix}_{[x,\ y]}$ (i.e. the *Fit Value* at $[x,\ y]$) is used to check if a piece can fit in at location $[x,\ y]$. If a piece is in location $[x,\ y]$ however, $\textit{Fit Matrix}_{[x,\ y]}$ contains that piece's indent/tab/edge matrix.

To better illustrate this, consider Figure 22 below. For the sake of this image, all coordinates are presented in an $[x,\ y]$ format.

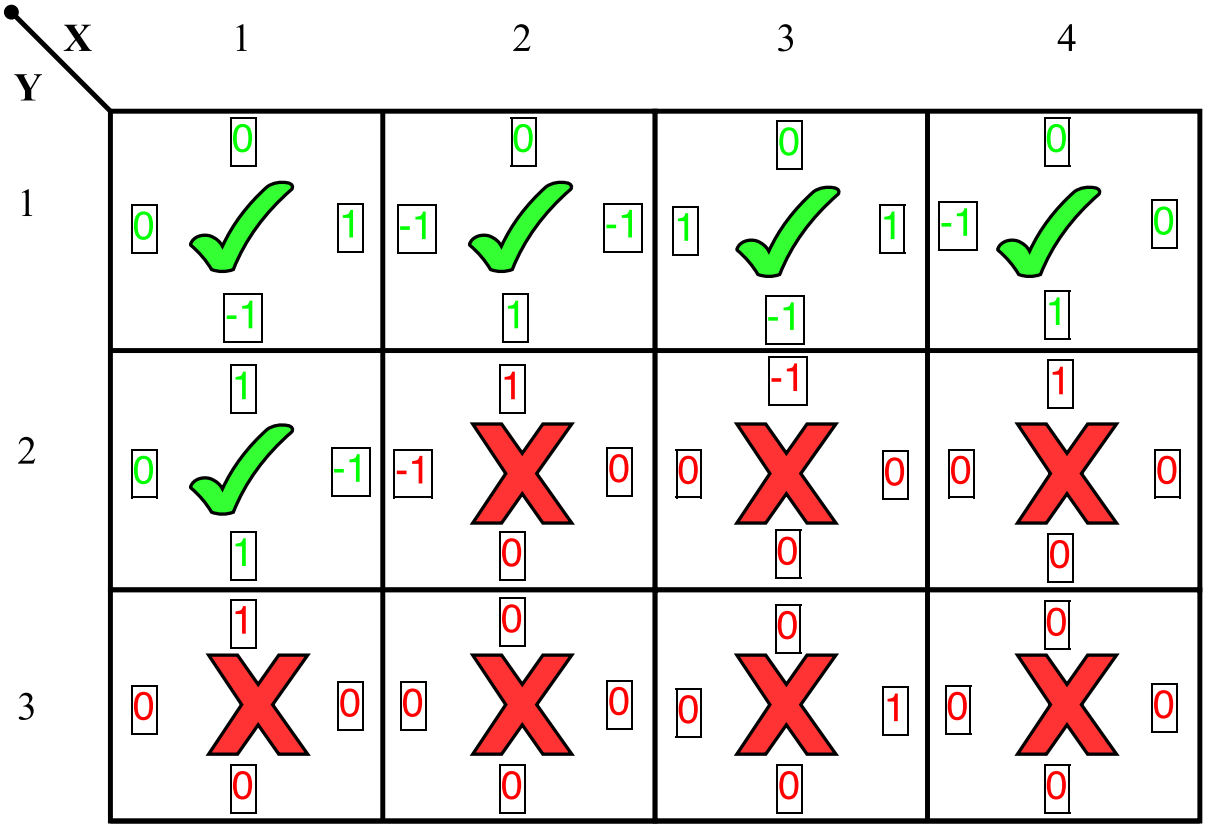


Figure 22.
Fit matrix example [8]

In this example, locations [1, 1], [2, 1], [3, 1], [4, 1], and [1, 2] (indicated with green check marks) are occupied by pieces, and the values contained within them are the residing pieces' indent/tab/edge matrix values. The remaining locations are all vacant (indicated with red crosses) and their values represent the location's *Fit Values*.

Prior to any *Fit Value* calculations, simple logic is applied to the problem to see whether such calculations would even be necessary. The simple logic used here includes, but is not limited to:

- only corner pieces can occupy corner locations,
- only edge pieces can occupy edge locations, and
- only inner pieces can occupy inner locations.

Once a piece is deemed worthy for a *Fit Value* calculation, the algorithm commences.

For a piece to fit into location [2, 2], it would have to have an indent/tab/edge matrix of $[-1, X, X, 1]$, or a rotated version thereof, where the X's represent "don't care" values. To handle these "don't care" values, a simple mask is applied to the calculation as to disregard indent/tab/edge values adjacent to vacant location.

Assume a piece with an indent/tab/edge matrix of $[1, -1, 1, -1]$. This is clearly an inner piece and would be eligible as a candidate piece for location [2, 2]. Equations 2.1 to 2.3 below show all the steps taken to perform a *Fit Value* calculation. (Note: all matrix operations refer to element-wise operations)

Initiate variables:

$$\begin{aligned} FitValue &= FitMatrix_{[2,\,2]} \\ &= [1,\ 0,\ 0,\ -1] \\ Mask &= [1,\ 0,\ 0,\ 1] \quad \text{(As locations [3, 2] and [2, 3] are vacant)} \\ Piece_{Indent|Tab|Edge} &= [1\ ,-1,\ 1,\ -1] \end{aligned} \tag{2.1}$$

Apply mask:

$$\begin{aligned} Temp &= Mask \times Piece_{Indent|Tab|Edge} \\ &= [1,\ 0,\ 0,\ 1] \times [1\ ,-1,\ 1,\ -1] \\ &= [1,\ 0,\ 0,\ -1] \end{aligned} \tag{2.2}$$

Calculate return value:

$$\begin{aligned} RetVal &= Temp + FitValue \\ &= [1,\ 0,\ 0,\ -1] + [1,\ 0,\ 0,\ -1] \\ &= [2,\ 0,\ 0,\ -2] \end{aligned} \tag{2.3}$$

It can be seen from Equation 2.3 that the example piece does not fit into location [2, 2], in its current orientation, as its *RetVal* contains non-zeros; but, the piece still has three other orientations to consider. Equations 2.4 to 2.7 below show the steps taken to perform a *Fit Value* calculation on the same example piece as in Equations 2.1 to 2.3, but with the piece rotated 90° counter-clockwise.

Initiate variables:

$$\begin{aligned} FitValue &= FitMatrix_{[2,\,2]} \\ &= [1,\ 0,\ 0,\ -1] \\ Mask &= [1,\ 0,\ 0,\ 1] \\ Piece_{Indent|Tab|Edge} &= [1\ ,-1,\ 1,\ -1] \end{aligned} \tag{2.4}$$

Rotate piece 90° counter-clockwise:

$$\begin{aligned} Piece_{Indent|Tab|Edge} &= Piece_{Indent|Tab|Edge} \ll {}^{5} \\ &= [1\ ,-1,\ 1,\ -1] \ll \\ &= [-1,\ 1,\ -1,\ 1] \end{aligned} \tag{2.5}$$

Apply mask:

$$\begin{aligned} Temp &= Mask \times Piece_{Indent|Tab|Edge} \\ &= [1,\ 0,\ 0,\ 1] \times [-1,\ 1,\ -1,\ 1] \\ &= [-1,\ 0,\ 0,\ 1] \end{aligned} \tag{2.6}$$

[5]The ≪ symbol indicates a wrap-around shift left operation.

Calculate return value:

$$\begin{aligned} RetVal &= Temp + FitValue \\ &= [1,\ 0,\ 0,\ -1] + [-1,\ 0,\ 0,\ 1] \\ &= [0,\ 0,\ 0,\ 0] \end{aligned} \tag{2.7}$$

It can be seen from Equations 2.4 to 2.7 that the example piece can fit into location [2, 2], after one 90° counter-clockwise rotation, as its *RetVal* contains only zeros. As the piece is symmetric, the same can be said for the piece after a 270° counter-clockwise rotation. This example piece can thus fit in location [2, 2] after one or three counter-clockwise 90° turns.

Once the most likely piece for a location is found, by means of colour-matching, it will be inserted and the corresponding *Fit Matrix* value will be updated. Once this has been done, the remaining vacant *Fit Matrix* values are updated. This is done by propagating edge values from occupied locations over to vacant ones. To illustrate, if the example piece were to be inserted into location [2, 2] after one 90° counter-clockwise rotation, $Fit\ Matrix_{[2,\ 2]}$ would become $[-1,\ 1,\ -1,\ 1]$. Next, the inserted piece's vacant neighbouring locations would be updated, such that $Fit\ Matrix_{[3,\ 2]}$ would become $[-1,\ 0,\ 0,\ 1]$ and $Fit\ Matrix_{[3,\ 2]}$ would become $[1,\ 0,\ 0,\ 0]$.

During puzzle construction, a situation can also occur where the puzzle being constructed, is built with its longest side along the shorter puzzle matrix axis. To correct such an occurrence, the puzzle matrix and *Fit Matrix* is simply rotated 90° counter-clockwise, and each *Fit Value* is shifted left once. This allows all solutions to have the same dimensions, which is very useful, as it allows the algorithm to compare various solutions directly to one another easily, in order to find the best one.

Colour matching, at its core, utilises image histograms to compare the similarity between two pieces. Such an image histogram is created by counting the amount of pixels containing the same value. The *x*-axis of such a histogram represents the pixel values (in this case RGB values) and the *y*-axis represents the amount of pixels containing the corresponding pixel values. Thus, such a histogram will have a *z*-axis, which represents the RGB values respectively. To give an example, Figure 23 below shows a colour image on the left, and the image's histograms on the right. The image on the left was used to create the red, green, and blue histograms on the right. A total of 50 bins were used to generate the histograms, hence the *x*-axis' range of 0-50.

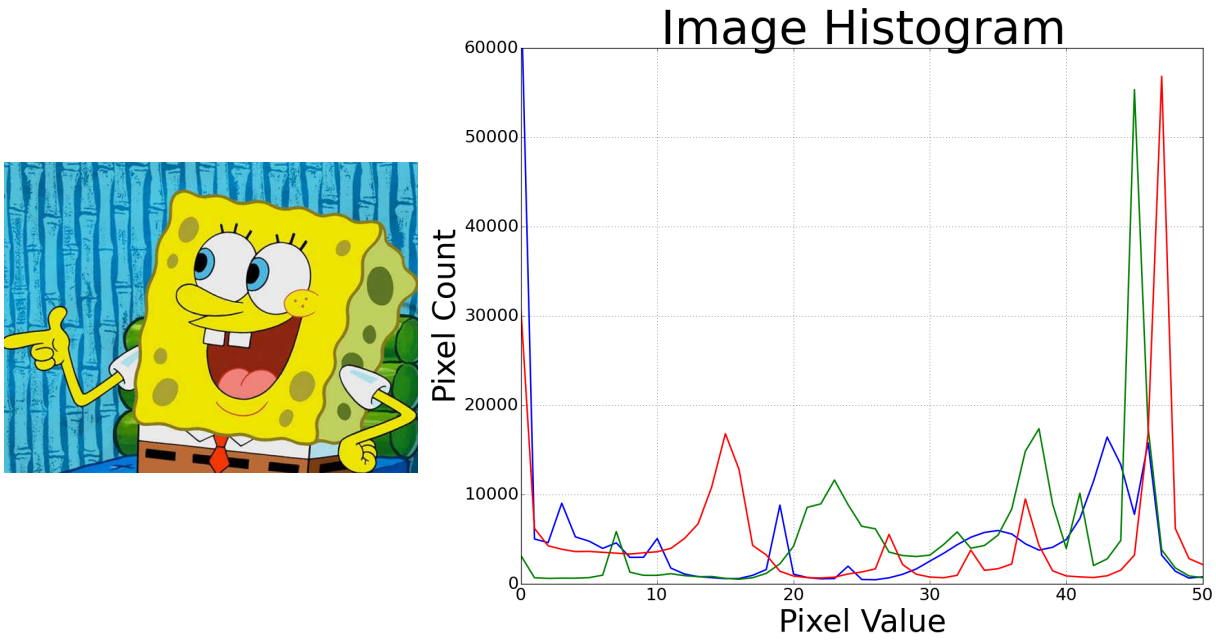
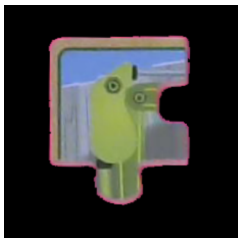


Figure 23.
RGB colour image histogram example

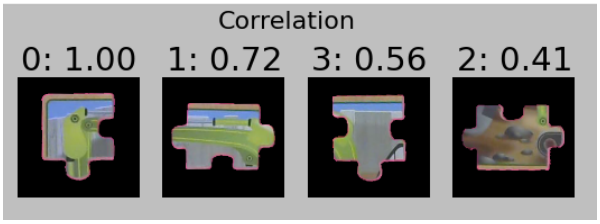
Once such a histogram has been constructed, mathematical operation (such as convolution and correlation) can be applied to it. In particular, two such histograms can be compared to one another in order to calculate the similarities between them and their source images. There are a few tools available in the *OpenCV* libraries that provide this functionality. The compareHist() function allows for the comparison between two image histograms, and has four methods of operation, namely:

- Correlation,
- Chi-squared,
- Intersection, and
- Hellinger.

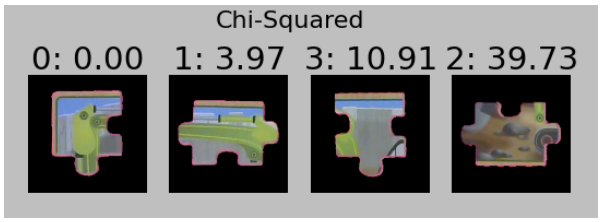
All these methods provide a numerical value that represent the similarity between two image histograms. The *Chi-squared* and *Hellinger* methods provide a low value when images are similar, and a high value when they are distinct. The *Correlation* and *Intersection* methods provide a high value when images are similar, and a low value when they are distinct. To illustrate this, Figure 24 below shows the similarity values between four puzzle pieces for each of the abovementioned methods. In each case, the puzzle pieces were compared to the test piece in Figure 24a. The pieces, along with their indices and similarity values, are ordered left to right from most similar to least similar. Piece 0 is the exact same piece as the test piece and thus received a perfect similarity score from all the methods. Piece 1 is the correct neighbouring piece to the test piece, and was labelled as such by all the methods. Piece 2 is entirely distinct from the test piece and as a result was found to be the least similar piece according to all four methods. Piece 3 was similar to the test piece but not a direct neighbour and as such was given a reasonable similarity value.



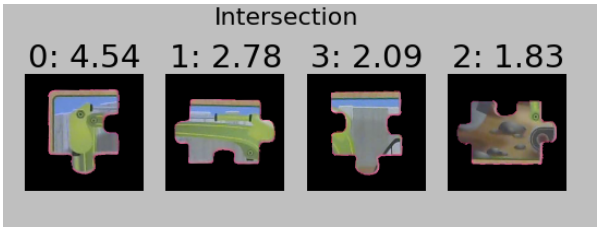
(a)
Comparison piece



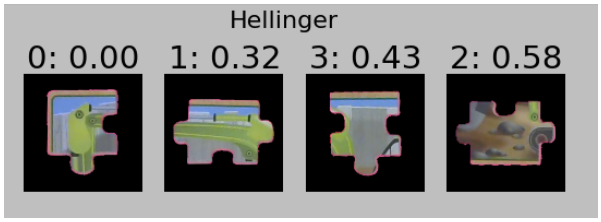
(b)
Correlation



(c)
Chi-squared



(d)
Intersection



(e)
Hellinger

Figure 24.
Histogram comparison methods [4]

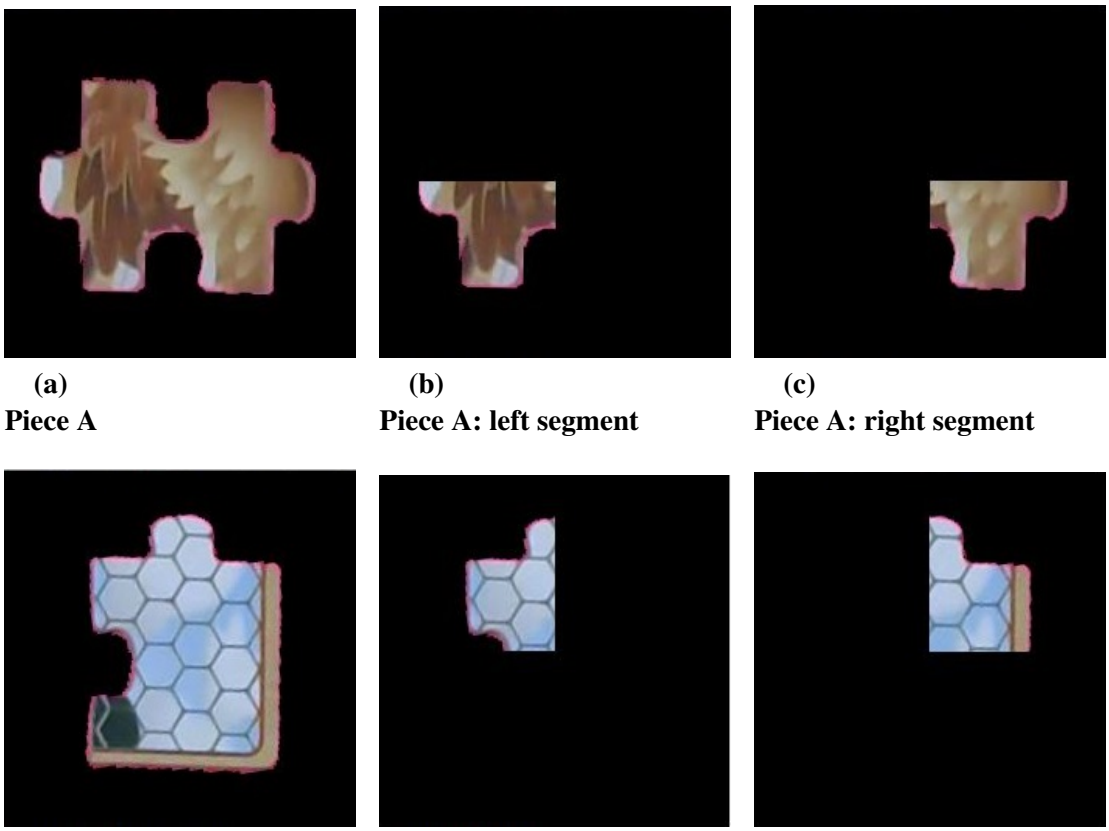
The *Chi-square* and *Hellinger* methods provided the best performance when applied to a few simple test, and as they both provided lower values for greater piece similarities, they could be interchanged easily, which eased the testing process. For testing purposes, both these methods were applied to three 12-piece puzzles in order to identify neighbouring pieces, without any shape-matching. Table 8 below show the success rate of both the methods. Based on these results, the *Hellinger* method was chosen.

Colour Matching Method Comparison		
Puzzle	Hellinger	Chi-squared
1	65%	61%
2	50%	56%
3	76%	64%
Average	**64%**	**60%**

Table 8.
Chi-squared and Hellinger colour-matching comparison

This method can however not be used on its own, as it does not take the shape or orientation of pieces into consideration. To be able to use this method for puzzle-solving, it was adapted to be edge- and orientation-sensitive. This puzzle-solving algorithm segments and masks piece images prior to histogram generation in order to address this problem. The colour-matching algorithm takes two piece images and compared the one's left edge to the other's right, or its top edge to the other's bottom, depending on the parameters passed to it. Additionally, this algorithm compared the pieces' edges in segments rather than as a whole, in order to increase its accuracy. For example, assume the bottom edge of piece A and the top edge of piece B, shown in Figure 25, are to be compared with each other (vertical comparison). Piece A will be segmented and masked into the sub-edges shown in Figure 25b and Figure 25c, and piece B will be segmented and masked into the sub-edges shown in Figure 25e and Figure 25f. The two left segments of both piece A and B will be compared with one another and so will their right segments. Once a result has been obtained for both segment comparisons, they will be averaged and will serve as the similarity value between the two pieces' edges. A similar process is followed when pieces need to be compared side by side (horizontal comparison).

All these steps allow the algorithm to accurately judge how likely two-piece edges are to be neighbours. With shape-matching in place, the colour-matching algorithm increases its performance, not only in accuracy but also in execution time, as the algorithm does not have to consider pieces with non-compatible shapes.



(a) **Piece A**

(b) **Piece A: left segment**

(c) **Piece A: right segment**

(d) **Piece B**

(e) **Piece B: left segment**

(f) **Piece B: right segment**

Figure 25.
Piece colour comparison: segmentation and masking [4]

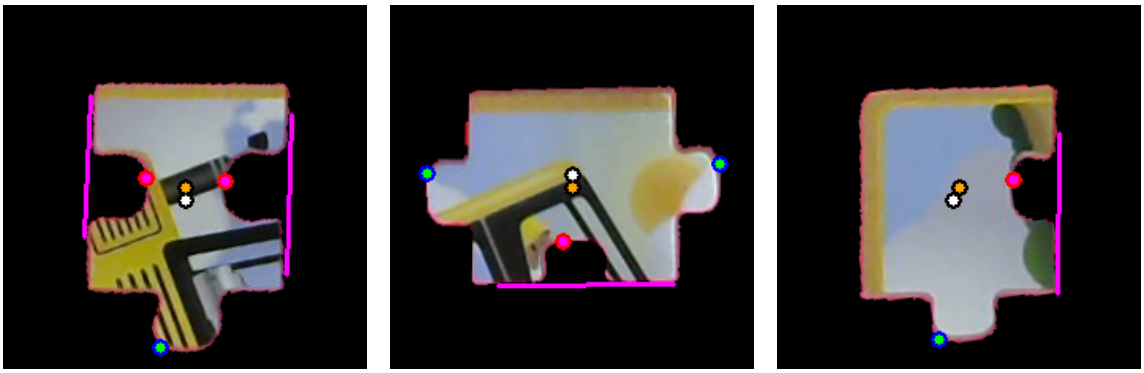
As mentioned above, four candidate puzzle solutions are created by the puzzle-solving algorithm, each constructed from a different initial corner piece. In order for the system to judge the accuracy of these solutions, a method was created to score them. Initially, it was decided take the sum of all the piece colour comparison values, that lead to the solution, as its score. This however, was found to be an extremely inconsistent method, as incorrect solutions would often get a better score than correct ones. This is mainly owing to the fact that colour comparison values largely depend on the colour contents of the puzzle, which is vastly different from piece to piece and is heavily affected by inconsistent lighting anomalies. Also, when an incorrect piece is inserted into a solution early on, the colour matching values for the rest of the solution become very unpredictable and inaccurate. It was thus decided to build an interlocked solution image for each candidate solution. These images can then be analysed in order to determine the accuracy of the solutions they contain, and they serve as a good visual indication of the solution.

An interlocked puzzle solution image is generated by first creating a blank image, the dimensions of which are obtained from the solution pieces' cumulative dimensions. A bitwise *OR* operation is then performed between each piece's image and the blank image, at each piece's corresponding solution location on the blank image. In order to align each piece with its correct solution location on the blank image, the pieces' centre locations are used. If a piece's centre location is not perfectly centred however, the interlocked solution image will contain overlapping piece edges. It is thus very important to have perfectly accurate piece centre locations, to ensure that interlocked solution images are as close to an actual solution image as possible.

The centre location detection method uses a piece's contours to find its centre location, and this creates errors if the piece is not symmetric. For example, if an edge piece, going clockwise from the top, has an edge, an indent, a tab, and, an indent ($\underline{\text{it\_class}} = [0, -1, 1, -1]$), its centre point will be lower than it should be owing to the tab at the bottom extending the piece's contours' centre of mass. So in essence, a tab would skew the centre point towards it, whereas an indent would skew the centre point away from it.

These properties can be used to readjust the centre location. The downside however, is that the exact displacement is very shape-dependant and not easily calibrable. A test was thus applied to all the puzzles being considered for this project and these displacements were measured. After applying the test to eight 12-piece puzzles it was found that an indent offsets the centre by 1.75% of the piece's width, and a tab offsets the centre by 3.5% of the piece's width. With the 720p camera resolution, this translates roughly to 8 and 4 pixels respectively.

A small method was appended to the piece classification algorithm that corrected these errors. Figure 26 below shows this additional algorithm in action. The red dots represent indent extreme points, blue dots represent tab extreme points, *white dots* represent the original centre locations, and orange dots represents the new, corrected, centre locations. In all these figures, it can be seen that the centre point migrates closer to its true position. This would allow for much greater piece-manipulation and puzzle-construction accuracy, and would allow the interlocked solution image to be an accurate representation of the real solution.



(a) **Re-centre piece 1**

(b) **Re-centre piece 2**

(c) **Re-centre piece 3**

Figure 26.
Piece centre location adjustment [4]

There is however, one more thing to do. Recall that the piece class assigns pieces' centre locations based on the middle point of their contours in the overall puzzle image (where the entire puzzle and construction platform is visible), and the centre location that was just adjusted above was for piece images (where only a piece is visible). All the pieces' actual centre locations still need to be adjusted, but the pieces in the overall puzzle image are at different angles than the ones in the piece images (as the piece images have been correctly orientated). Luckily, the angles that piece images have been rotated by are known, and can thus be used to aid the correction process.

Assume that we have a piece image's original centre location $C_{po} = [y_{po}, x_{po}]$, its adjusted centre location $C_{pn} = [y_{pn}, x_{pn}]$, its correction angle θ_p, and its centre location in the overall image $C_o = [y_o, x_o]$. The piece's adjusted centre location in the overall piece image $C_n = [y_n, x_n]$, can then be found as follows:

Convert piece image points to a vector:

$$r = \sqrt{(x_{pn} - x_{po})^2 + (y_{pn} - y_{po})^2}$$
$$\theta = \arctan\left(\frac{y_{pn} - y_{po}}{x_{pn} - x_{po}}\right) \tag{2.8}$$

Adjust for piece image angle:

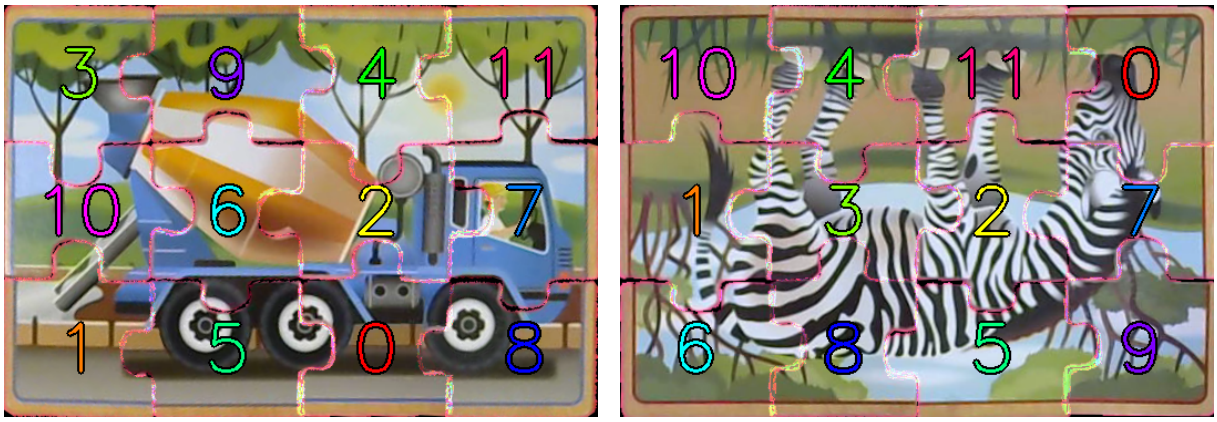
$$\theta_{adj} = \theta_p - \theta \tag{2.9}$$

Find y_n and x_n:

$$y_n = y_n + r \times \sin(\theta_{adj})$$
$$x_n = x_n + r \times \cos(\theta_{adj}) \tag{2.10}$$

Now that the true centre location, and thus shape, of all pieces are known, a more accurate and compact interlocked solution image can be created. Figure 27 below shows two examples of correct interlocked puzzle solutions generated by the system. These accurate, compact, interlocking puzzle solutions make it easy to find the correct destination coordinates and angles

for each piece, and since the current location for each piece has also been adjusted, the source and destination coordinated, and orientation, for each piece is known.



(a)
Blue truck puzzle solution

(b)
Zebra puzzle solution

Figure 27.
Interlocked puzzle solution example [4]

Now that compact interlocking puzzle solution images are available, these images can be used to judge the accuracy of a solution. When a solution is correct, its solution image will contain little-to-no black (background) pixels, whereas an incorrect solution's image will contain a distinguishably larger amount. This can thus be used to judge the likelihood of a solution being correct.

By applying this scoring technique to each of the four available solutions, the algorithm can judge which ones are correct and which are not. This scoring algorithm thus takes shape and colour into consideration, making it more reliable than the initial scoring method.

This scoring algorithm functions by first ignoring a small border, with a width of 10 pixels, around an interlocked solution image. This is necessary owing to the fact that even correct solutions will have a few black pixels around their edges. By ignoring these edges, the algorithm focuses only on areas where puzzle pieces possibly overlap. The puzzle's score thus represents a true version of the puzzle's state.

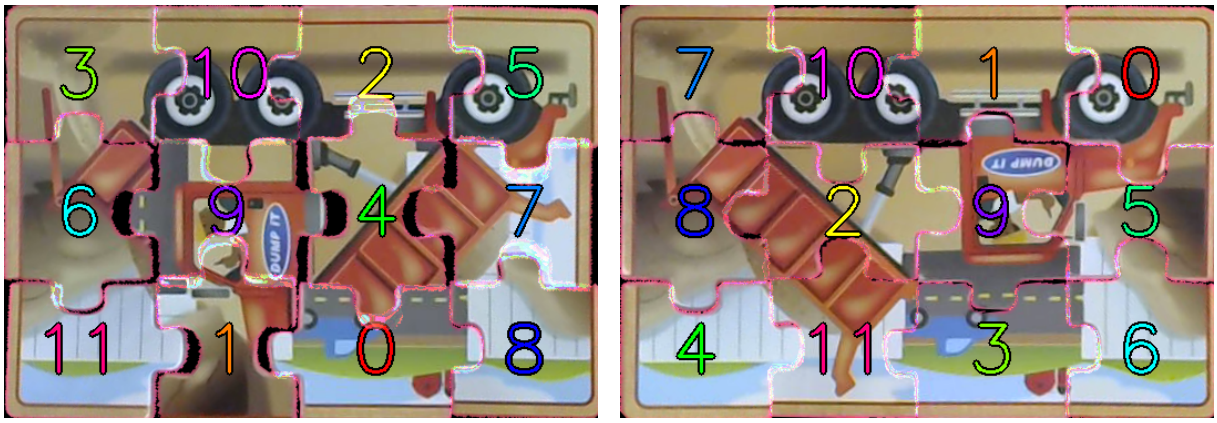
After the border mask has been applied, the algorithm makes use of the numpy.where() function. This function returns an array containing the coordinates that comply with the conditions passed to the function. Once this array is obtained, the shape of it can be used to count the amount of coordinates (pixels) that conform to the provided condition. The *Python* code for this is provided in Listing 2 below.

```
' Width of image solution '
img_width

' Hight of image solution '
img_hight

' Solution image '
img_sol
```

```
9
10 ' Border ignore width '
11 b_width = 10
12
13 ' Apply mask '
14 img_sol = img_sol[b_width:img_width-b_width, b_width:img_hight-b_width]
15
16 ' Count amount of black (background) pixels within border '
17 score = np.where(img_sol == [0,0,0])
18
19 ' Find solution score '
20 score = np.shape(score[0])[0]
```

Listing 2.
Interlocked puzzle solution scoring methood

Figure 28a below shows an incorrect interlocked puzzle solution image. This image contained 34,648 black pixels (not including the image border). Figure 28b below shows a correct interlocked puzzle solution image. This image contained 23,312 black pixels (not including the image border). After applying the algorithm to eight 12-piece puzzles, it was found that correct solutions all scored between **7,500** and **28,000**, where all incorrect solutions scored between **26,000** and **41,000**. Even though there is an overlap in these ranges, it was found that, if both correct and incorrect solutions were present, a correct solution would always score the lowest.



(a)
Incorrect interlocked solution

(b)
Correct interlocked solution

Figure 28.
Interlocked solution scoring method [4]

This technique is thus useful as it allows the colour-matching phase of the puzzle-solving algorithm to not only judge colour compatibility, as in the initial technique, but to also shape compatibility. This provides a greater level of shape matching, which leads to a more compact software solution. A more compact software solution leads to a more accurate physically constructed solution. The new scoring system is thus effective as it increases the overall accuracy of the entire system.

Figure 29 below shows the software flow diagram for the puzzle-solving algorithm. The processes show in red belong to the *Starting Piece Selection* operation, the ones shown in green belong to the *Shape-Matching* operation, and the ones shown in blue belong to the *Colour-Matching* operation.

To summarise this diagram, the algorithm creates four separate puzzle solutions, each starting from a separate corner pieces. For such a solution, the algorithm first inserts its starting corner piece and then proceeds to use shape-matching to find all the corners' possible neighbouring pieces. Once all the possible neighbours have been extracted, they are all correctly orientated to prepare them for colour-matching. Colour-matching is then applied to each candidate's neighbouring edge in order to find the most likely piece. This piece is then inserted and the puzzle's parameters are updated accordingly. The algorithm will continue in this manner until all pieces have been placed. Once all four solutions are complete, the algorithm will select the best one based on its puzzle score.

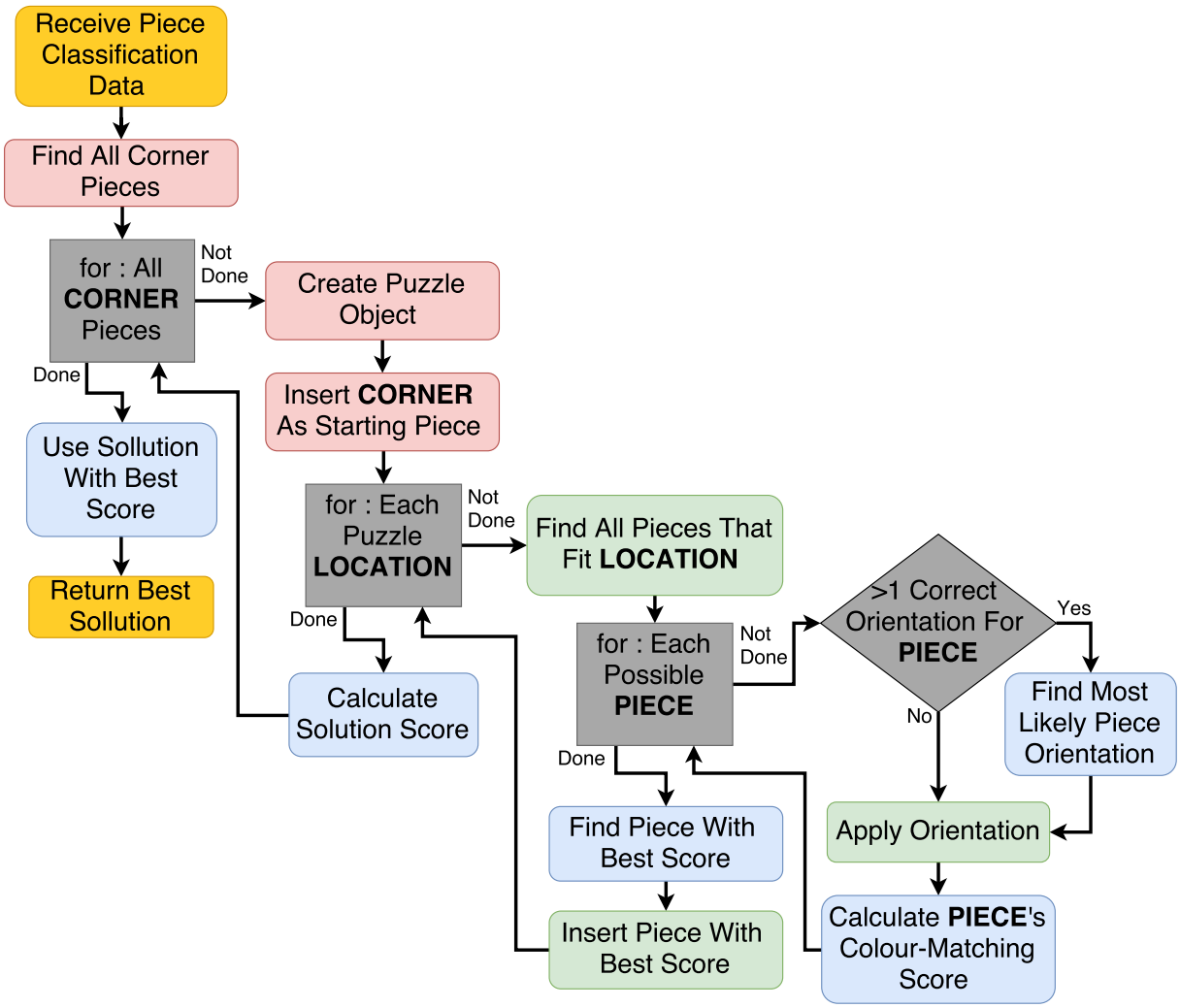
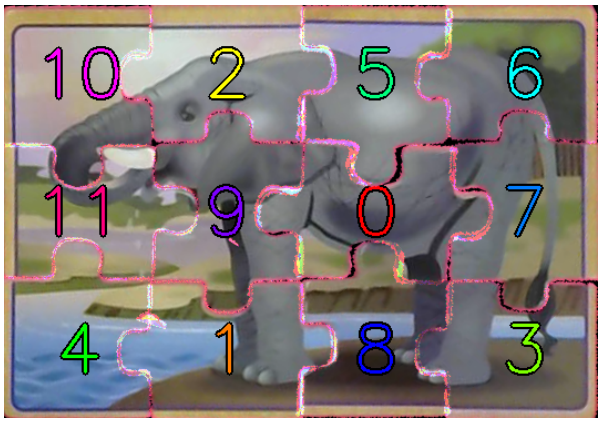
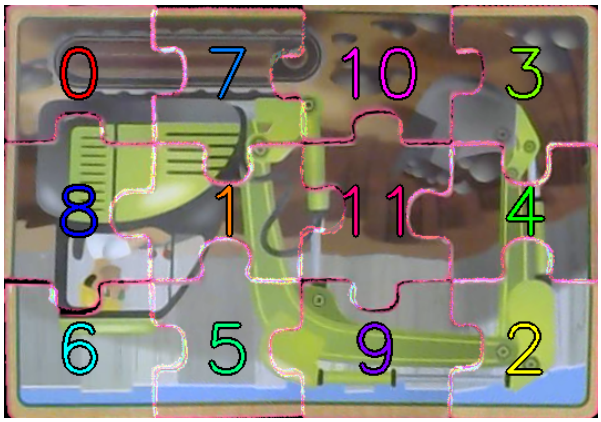


Figure 29.
Puzzle Solving Algorithm Flow Diagram [8]

Figure 30 below shows the algorithm's final solution to four different 12-piece puzzles.

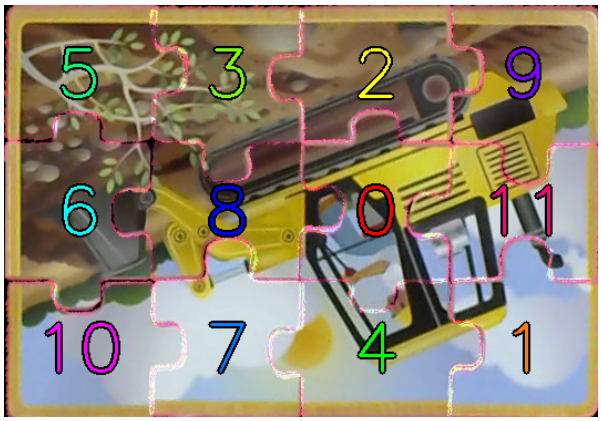


(a)
Elephant puzzle solution



(b)
Green truck puzzle solution

(c)
Lion puzzle solution



(d)
Yellow truck puzzle solution

Figure 30.
Puzzle solution images [4]

System Controller

The system controller regulates when certain functions are called and dictates which actions to perform with their outputs. It functions as a basic state machine, using the results obtained from the functions it conducts to transition between appropriate states.

Figure 31 below illustrates the system controller's flow diagram. Although not explicitly illustrated, this flow diagram functions as a state machine. Once a function has been completed, it will no longer be executed unless the system needs the unit's functionality again (the one exception is the Perform Perspective Transform function, which always runs for graphical display purposes). All the functions surrounded by a **thick black line** are treated as is shown by the sub-flow diagram in Figure 32, which illustrates the state machine behaviour. For example, once all corner points have been found, the Detect Corner Points function will not be applied again, and once the Solve Puzzle function has produced a successful solution, the only function that will still run within the main program loop is the Send Next Construction Command function.

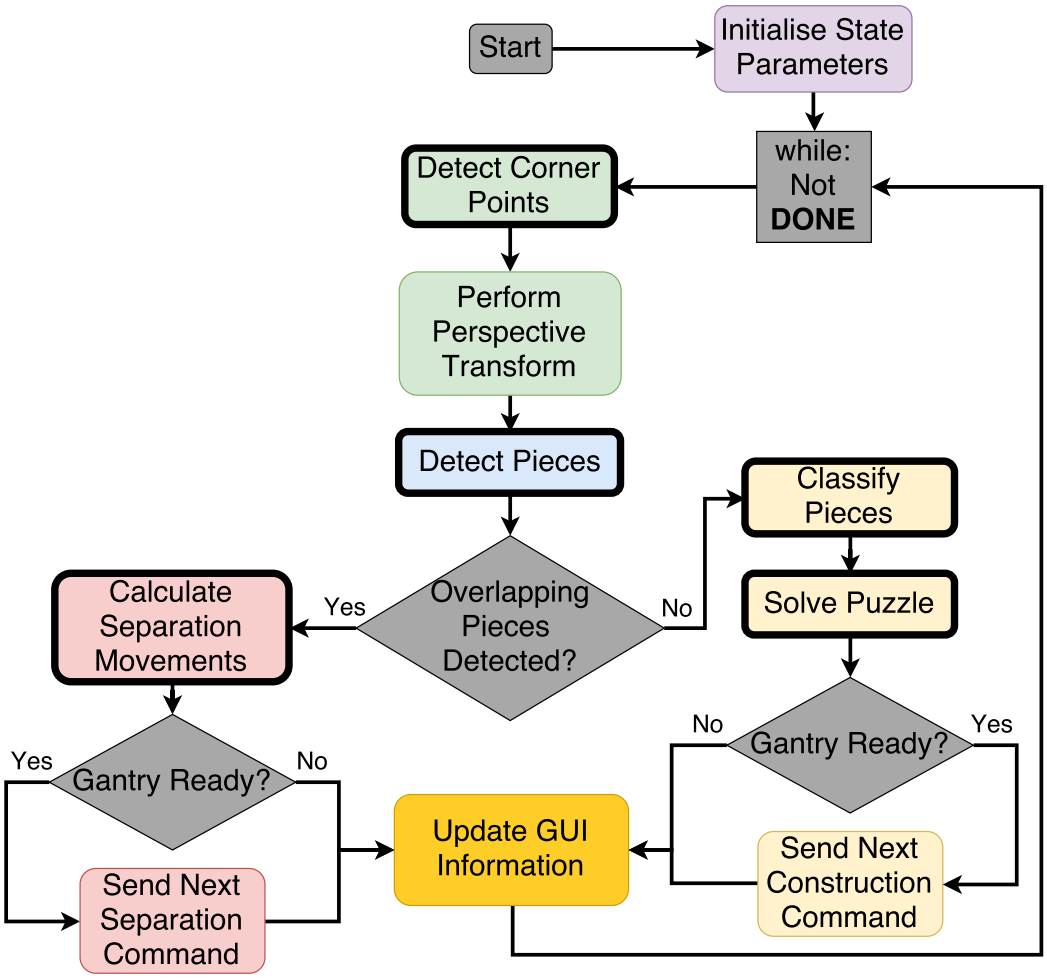


Figure 31.
System controller flow diagram [8]

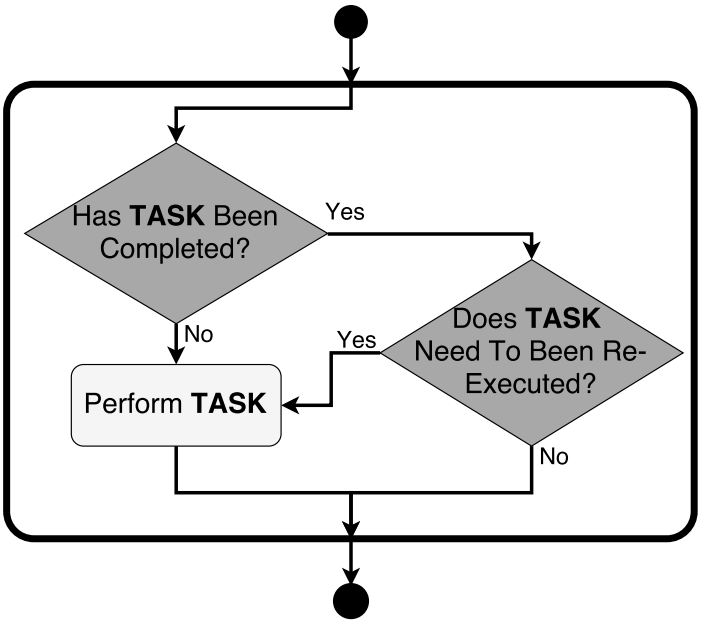


Figure 32.
System controller flow diagram expansion [8]

The most important feature of the system controller, is that it constantly monitors each software component in order to detect and correct errors. It acts as an intelligent feedback control mechanism which allows it to prevent errors from propagating throughout the *PC System*. If any of the *PC System's* components fail to produce a successful output, the system controller will intercept the program flow and return it to the start of the main control loop. The controller will then reactivate all the necessary components by altering their state parameters. This will allow the component that previously failed to reattempt its functionality of a fresh data set.

For example, if the Classify Pieces component fails to gather all the pieces' information, or does so incorrectly, the system controller will detect and intercept the erroneous output, and will prevent the Solve Puzzle component from executing. Instead, the system controller will immediately return to the to of the while loop and reactivate all components that precede the Classify Pieces component. This will allow the system to continue without being affected by the previous error. The right-most query diamond in Figure 32 above illustrates how the reactivation behaviour is handled. This functionality is thus vitally important as it largely contributes to the reliability and robustness of the entire *PC System*.

GUI

The GUI relays information related to the puzzle-solving procedure to users, and streamlines the overall software experience. The system's GUI utilises a minimalistic layout to emphasise only the most important and relevant information, and to simplify the operation of the software. The GUI is dynamic as its height and width can easily be adjusted, however, it was designed to function best in full-screen mode on a 1920×1080 resolution screen, as the camera feed will then be displayed in its full resolution of 1280×960.

Figure 33 to 36 below shows the puzzle-solving system's GUI. The image display window, seen in the middle of the user interface, contains four tabs. These tabs are used to switch between images and camera feeds related to the puzzle-solving process. These tabs contain the following information:

- the Camera tab shows the live camera feed (as seen in Figure 33),
- the Detection tab shows the puzzle piece detection view (as seen in Figure 34),
- the Classification tab shows the puzzle piece classification image (as seen in Figure 35), and
- the Solution tab shows the puzzle solution image (as seen in Figure 36).

The four buttons on the left of the user interface, when pressed, will capture and save the respective image in a PNG format in the *Images* sub folder (located where the puzzle solving program is located).

The top right section of the user interface contains two LCD-style timers for the Software Solution Time [ms] and the Hardware Solution Time [s]. The bottom right section of the user interface contains a LCD-style Piece Counter, which displays the amount of valid pieces the system detects.

The Start button, at the bottom right of the user interface, initiates the puzzle-solving process when pressed. This button becomes unavailable while the puzzle solving system is in progress (as is seen in Figure 33), and becomes available again once the system has completed the puzzle (as seen is Figure 34 to 36).

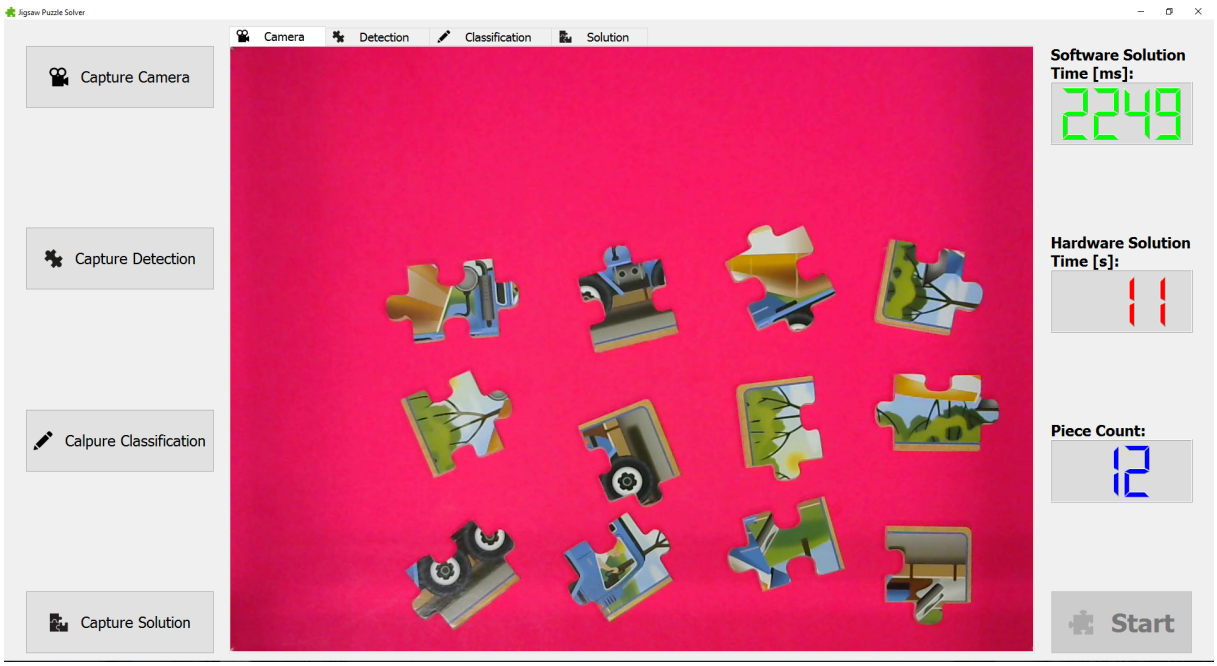


Figure 33.
Puzzle solving system GUI - Camera

Figure 34.
Puzzle solving system GUI - Detection

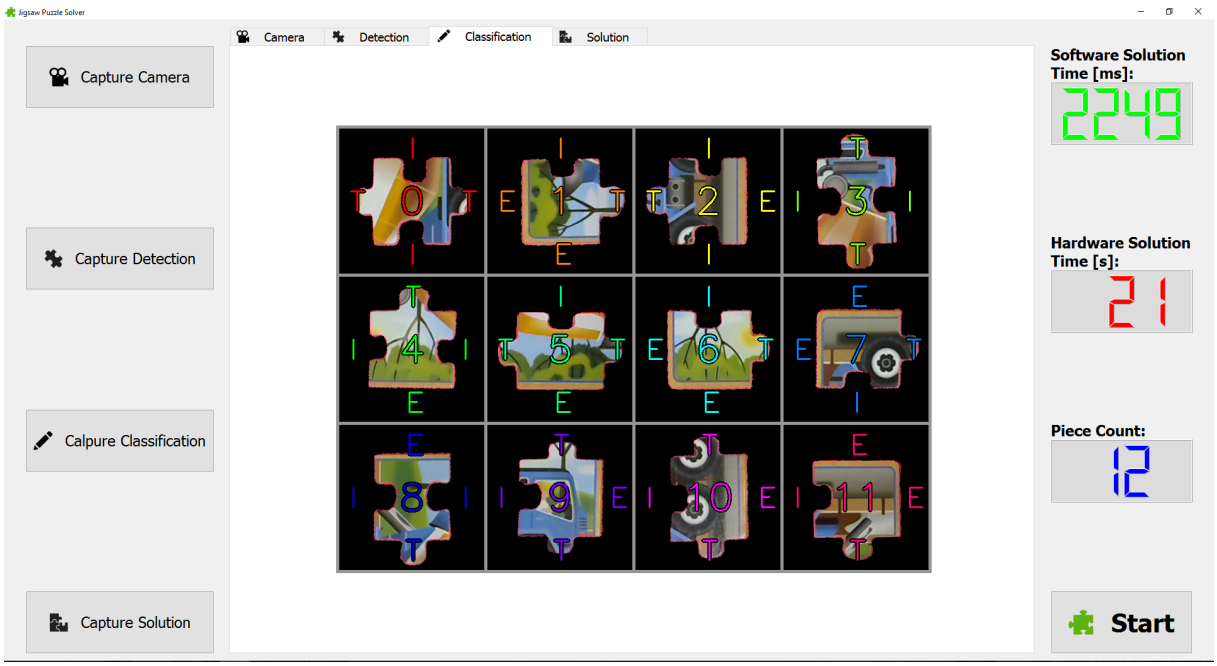


Figure 35.
Puzzle solving system GUI - Classification

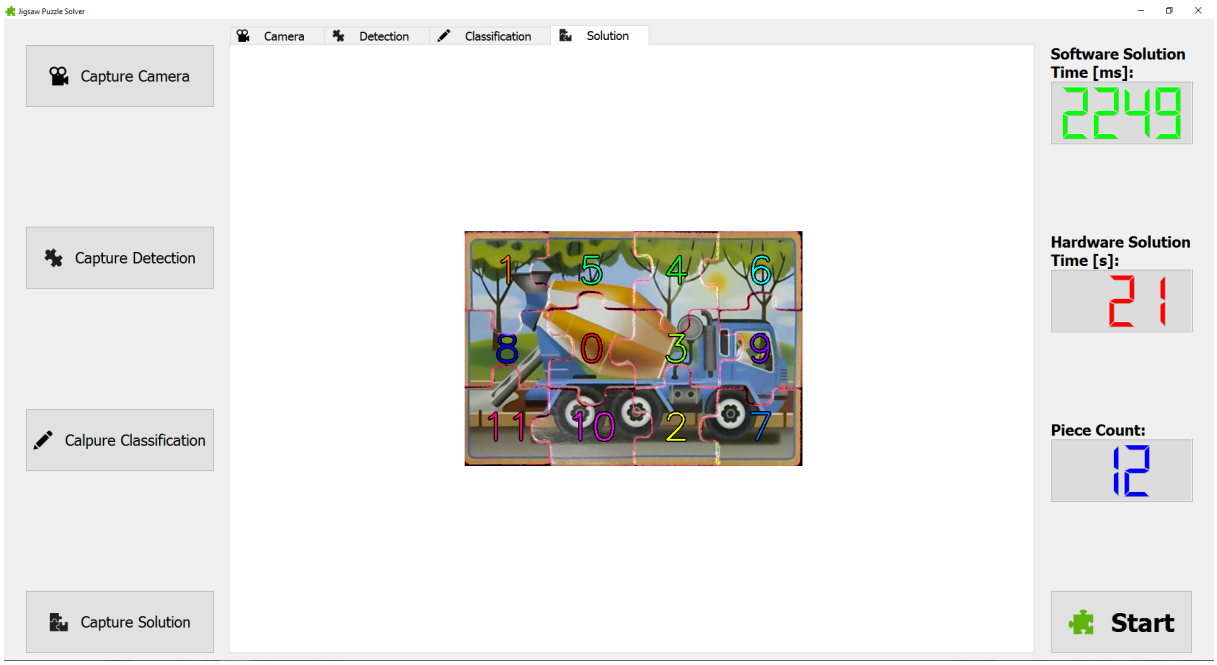


Figure 36.
Puzzle solving system GUI - Solution

Software Simulation Platform

The software simulation platform was designed to test and demonstrate all the functionality of the *PC System's* software components. The simulator was developed similarly to the system controller; the flow diagrams of which can be seen in Figure 31 and 32 above. The main difference between the simulator and the system controller is that the simulator allows users to control all the various software components via the PC's keyboard. Users can activate and deactivate the various software components in order to view their intermediary outputs and results in real time. When the simulator is initialised, it displays the *PC System's* unaltered vanilla camera feed, after which users can toggle the various software components. In order to toggle the components, users can simply press the component's corresponding key on the PC's keyboard. Table 9 below shows the simulator control keys and describes the functionality that they activate/deactivate.

Software Simulation Platform Controls	
Key	**Description**
"1"	Detect corner markers.
"2"	Perform perspective transform.
"3"	Detect pieces (stacked and unstacked).
"4"	Separate overlapping pieces.
"5"	Classify pieces.
"6"	Solve puzzle.

Table 9.
Software simulation platform controls

The control keys were numerically assigned so as to mimic the order in which their corresponding functions will be executed in the *PC System*. If these keys and their functions were to be activated sequentially, the program flow of the simulator will represent that of the *PC System* exactly. The simulator thus also serves as a mechanism to test and verify the integration of the various software components.

In order to streamline the user experience, the simulator takes care of all the control sequencing. In other words, if a user presses key *x*, the component associated with *x* inherently needs the functionality of all preceding components. Thus, when the user activates a component, the system will automatically activate all preceding components and deactivate all proceeding components. Similarly, when the user deactivates a component, all proceeding components are automatically deactivated as well. This allows the user to view the output of a pressed key's corresponding component, and only that component, regardless of any previous system inputs.

With all its functionality, the software simulation platform serves as an ideal mechanism to test and validate the functionality and integration of all the *PC System's* main software components. As the simulator runs in real time, it allows a vast variety of conditions and parameters to be tested very easily and rapidly.

2.2.2 Gantry System

In order to physically interact with the real world, the puzzle-solving system utilises electro-mechanical hardware and a microcontroller to control and monitor all of its operations. The microcontroller unit interprets commands from the *PC System* and applies them to the physical puzzle layout, all with the help of discrete electronic components and electro-mechanical hardware units.

The electro-mechanical hardware units of which the system comprises (FU3.3 and FU3.5), along with a brief description of their rolls in the overall system, are listed in Table 10 below:

Electro-Mechanical Hardware Units	
Servo Motors	• Activation and deactivation of the vacuum suction mechanism. • Raising and lowering the piece actuator.
Stepper Motors	• Controlling the *x*-axis movement of the *Gantry System*. • Controlling the *y*-axis movement of the *Gantry System*. • Rotating puzzle pieces.
Micro Switches	• Resetting the gantry to a known position. • Monitoring the *Gantry System* for *x*-axis limit breaches. • Monitoring the *Gantry System* for *y*-axis limit breaches.

Table 10.
Electro-mechanical hardware units

Servo Motors

Servo motors consist of a DC motor and a control circuit. This control circuit allows either the speed or the angular position of the motor to be controlled by a single PWM signal, depending on the mode of operation. In both modes, as per standard, a 50 Hz PWM signal is used to control the motor, which allows for a maximum pulse duration of $(\frac{1}{50})$ $20ms$. Standard servo motors however, only operate when pulse durations range between $0.388ms$ and $2.14ms$.

In the first mode, where the speed of the motor is controlled, the motor will have a maximum speed in one direction when the pulse width is at a minimum of $0.388ms$. As the pulse width increases, the speed of the motor will decrease until it is stationary. The pulse width that results in a stationary motor is known as its "dead zone", and its exact value can be adjusted by altering the motor's internal potentiometer. If the pulse width is increased beyond the motor's "dead zone", it will begin to slowly rotate in the opposite direction. As the pulse width keeps on increasing, up to a maximum of $2.14ms$, so will the motor's speed, but now, in the opposite direction.

The second mode, which allows the angular position of the motor to be controlled, is achieved by attaching the motor's internal potentiometer to its axis. In doing so, the motor will constantly adjust its own "dead zone" as it rotates, and will eventually stop once this "dead zone" is reached. The motor will thus seek out a specific "dead zone" depending on the provided PWM signal's pulse duration. Under standard conditions, a servo motor operating in mode 2 will be at $0°$ with a pulse width of $0.388ms$, $90°$ at $1.264ms$, and $180°$ at $2.14ms$. This linear relationship can be used to calculate the necessary pulse duration for any desired angle.

As it is desired, in both the case of the vacuum suction mechanism and the piece actuator, to have the angular position of the servo motors at a desired location, both mechanisms' servo motors operates in mode 2.

Vacuum Suction Mechanism

Figure 37 below shows the vacuum suction mechanism, which uses a servo motor with $15kg/cm$ of torque. This mechanism operates by moving between three distinct angular locations, namely Release, Rest, and Engage. These positions are illustrated in Figure 38 below. When not in use, the mechanism will remain at its Rest location. To create a vacuum and "suck" a piece, the mechanism will move to its Engage location. Once a piece needs to be released, the mechanism will move to its Release location. Once the piece has been successfully dropped it will return to its Rest location. This strategy will allow the mechanism to always "blow" more air that it "sucks", ensuring that no piece is unintentionally left stuck to its suction cup.

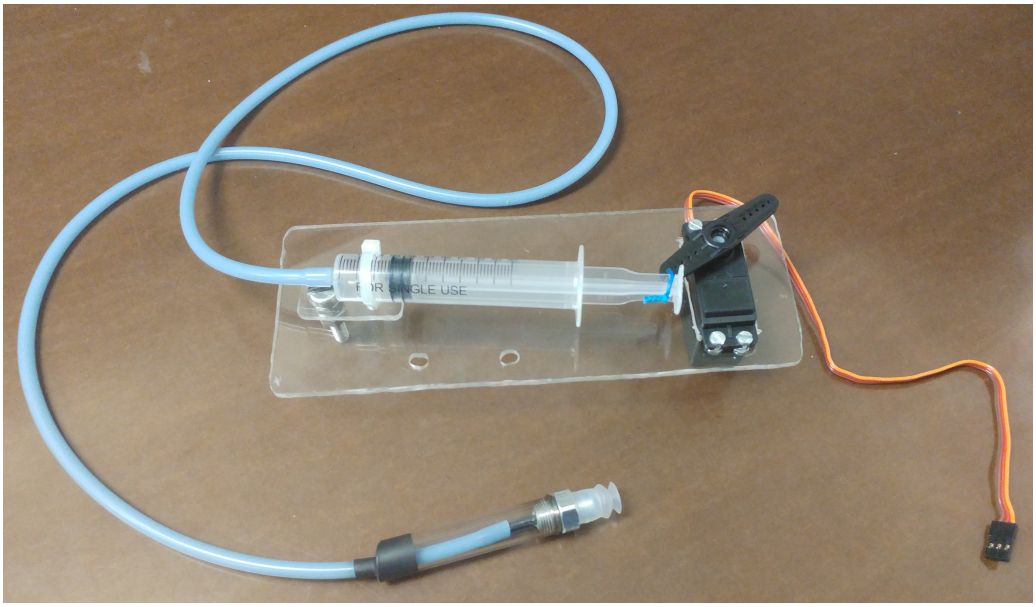


Figure 37.
Vacuum suction mechanism

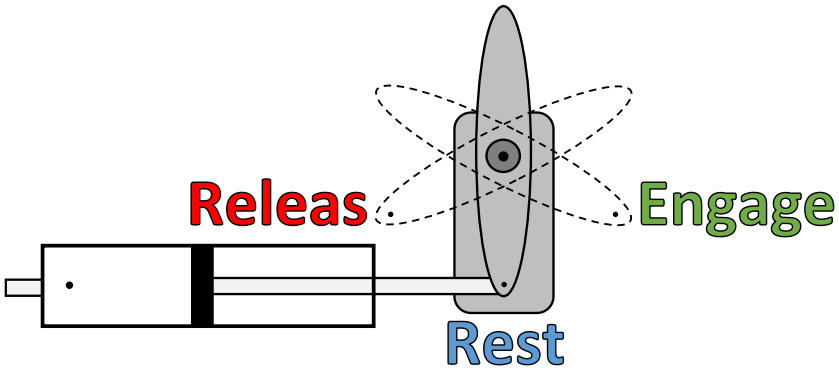
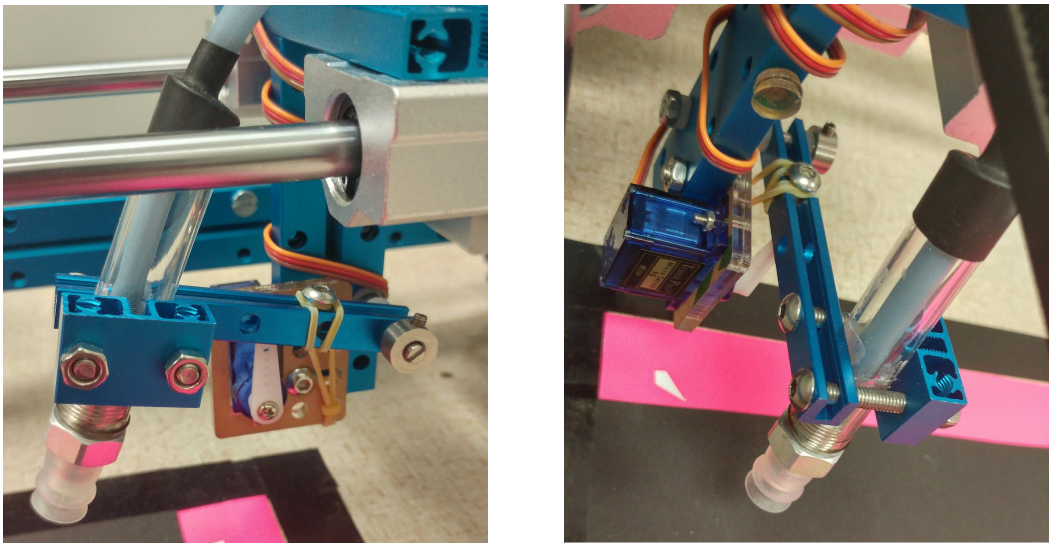


Figure 38.
Vacuum suction angular positions

Piece Actuator

Figure 39 below shows the piece actuator, which makes use of a $90g$ servo motor with $1.3kg/cm$ of torque. The piece actuator needs to handle cases where up to three puzzle pieces are stacked on top of one another. To handle such cases, the actuator has several different depths to which it can lower. This, however, introduces an unwanted linear offset anomaly, which is caused by the actuator's rotary nature. This offset is illustrated in Figure 40 below, as indicated by the green arrow. In essence, if a piece is picked up and dropped at different depths (z-axis), its y-axis gantry coordinate will have an unwanted offset. This offset error is handled by making software alterations to the gantry movement coordinates. By altering the gantry's y-axis coordinates as a function of actuator depth, the linear offset error can be completely avoided.



(a)
Right top view

(b)
Left side view

Figure 39.
Piece actuator

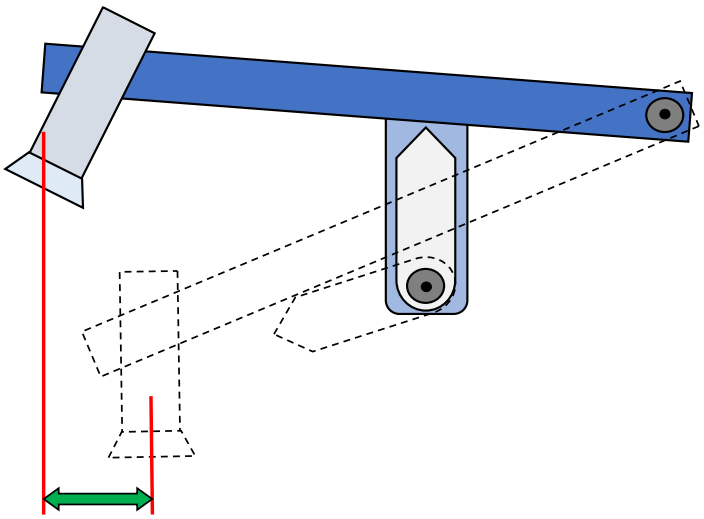


Figure 40.
Linear offset owing to rotary nature of pieces actuator

Stepper Motors and Stepper Motor Drivers

Stepper motors have various armature coils, which allow them to make small, constant steps when current pulses are sent through their coils in a specific sequence. Stepper motor drivers (FU3.3) ease the operation and control of stepper motors as they handle all the pulse sequencing and allows for easy current limitation, so as to protect motors from over-current damage.

The stepper motor driver used for this system is the *DRV8825* high-current stepper motor driver. The wiring diagram for this chip is shown in Figure 41 below. This connection configuration enables the microcontroller to control the motor with only three digital pins, the functionality of which is shown in Table 11 below.

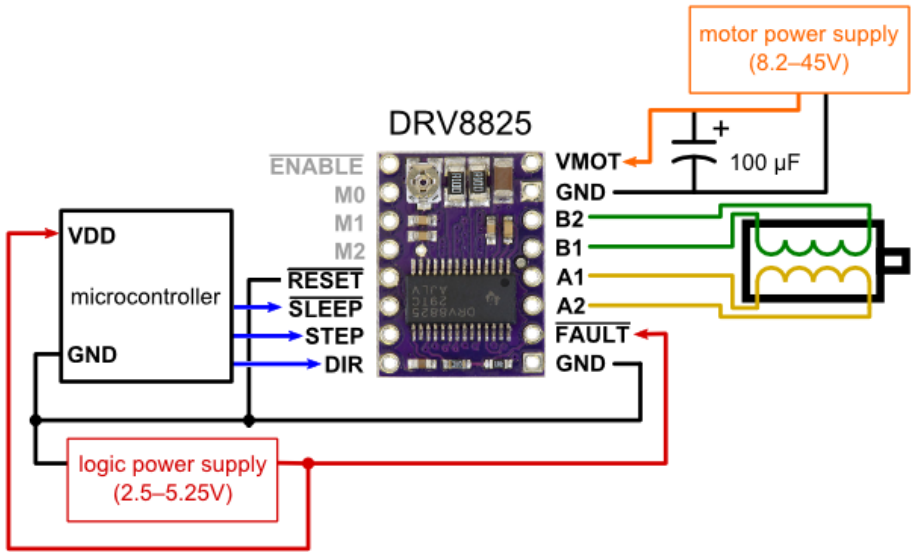


Figure 41.
Wiring diagram for a DRV8825 stepper motor driver (full-step mode)

Stepper Motor Control Pin Description	
Pin	**Description**
$\overline{SLEEP}$	Sets the motor to sleep/awake mode by disabling/allowing current outputs to pin **A1**, **A2**, **B1**, and **B2**.
STEP	A rising edge on this pin will rotate the motor one full step.
DIR	Determines the rotational step direction of the motor.

Table 11.
Stepper motor control pin description

Gantry

Figure 42 below shows all the integrated mechanical and electro-mechanical hardware units of the *Gantry System*. This system utilises two 1.5*A* stepper motors to achieve both *x*- and *y*-axis translations. By keeping track of both its *x*-and *y*-axis steps, the system can numerically control its position.

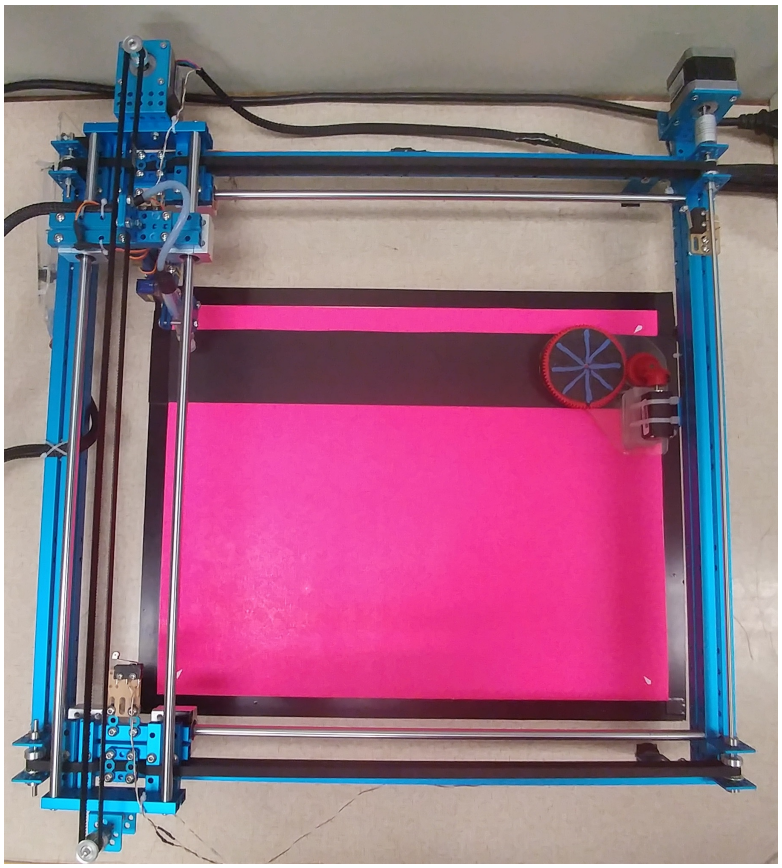


Figure 42.
Gantry system mechanical hardware

If the system wishes to move the actuator from coordinate $[x_{source},\ y_{source}]$ to $[x_{destination},\ y_{destination}]$, it first determines the rotational direction required for each respective motor. This is done by comparing the magnitudes of each coordinate's components. For example, if $x_{destination} > x_{source}$, the x-axis motor will be set to move the actuator right (i.e. clockwise) and vice versa. In cases where $x_{destination} == x_{source}$, no x-axis translation will be required and thus the corresponding motor's direction is irrelevant. The y-axis motor's direction can be determined by applying similar logic.

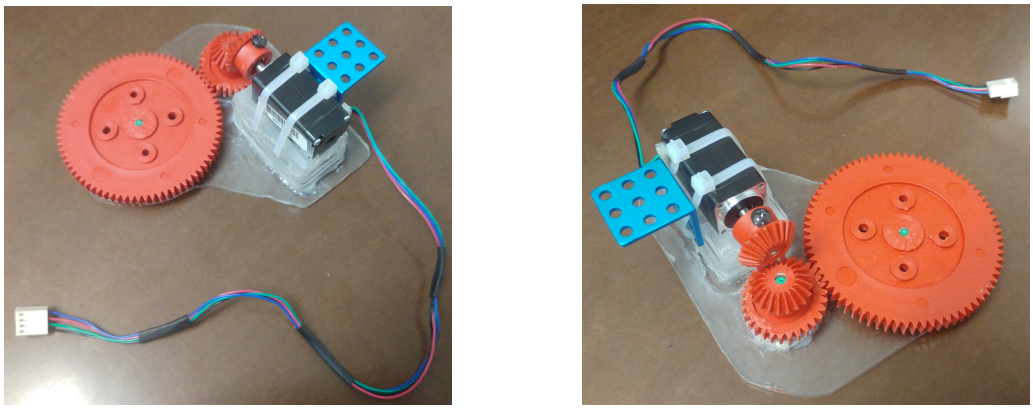
Once the rotational directions for both motors have been determined and set, the amount of steps required for each motor is calculated. This is derived by taking the absolute of differences in coordinate components. The amount of x-axis steps required to reach $[x_{destination},\ y_{destination}]$ from $[x_{source},\ y_{source}]$ is thus simply $|x_{destination} - x_{source}|$. The same logic is applied to determine the required amount of y-axis steps.

In order to achieve translation, the microcontroller sends a $1kHz$ PWM signal to each motor until its desired location is reached. This is achieved by running counters in parallel with the respective PWM signals. Once a counter reaches its motor's desired step count, the corresponding PWM signal is disabled.

The $1kHz$ PWM signal frequency was chosen as it was fast enough to provide fluent motion whilst also being slow enough to minimise start-stop jolt. A few puzzle construction manoeuvres, however, require slower movement. For cases such as these, a PWM signal frequency of $250Hz$ is used.

Piece Rotator

In order to correctly orientate puzzle pieces, the system utilises a 0.5*A* stepper motor, along with a gearing mechanism to rotate pieces prior to construction. The exact angle that each piece needs to be rotated by is previously calculated in the *Piece Classification* and *Puzzle Solving* steps. Figure 43 below shows the piece rotator from two different angles. The rotator can also be seen fixed to the top-right corner of the *Gantry System* in Figure 42 above. The design of the rotator can be seen in Figure 44 below. In this figure the top view can be seen above the dash-dot line, and the front view can be seen below it. The gears in the image have been colour coded purely for illustrative purposes. Additionally, this figure indicates the gear teeth count for each of the respective gears.



(a)
Right top view

(b)
Left bottom view

Figure 43.
Piece rotator

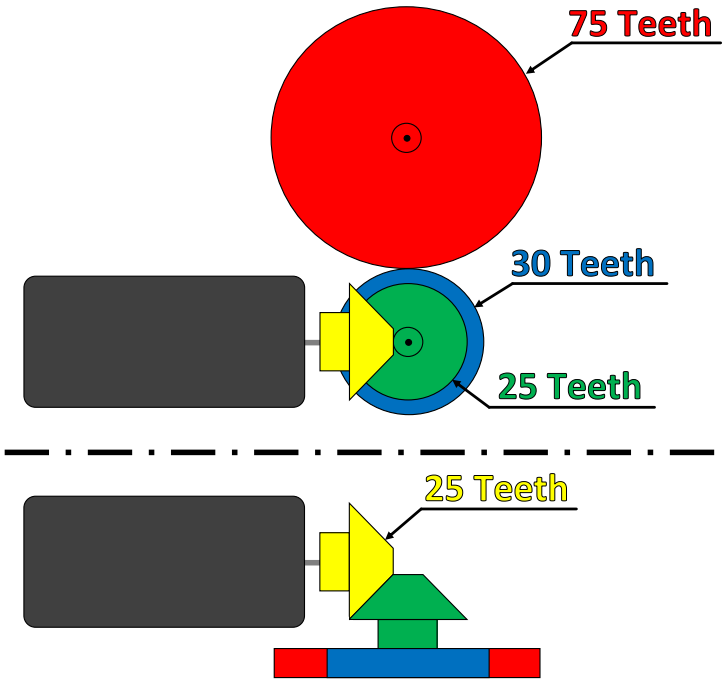


Figure 44.
Piece rotator design

As the green gear is fixed to the blue gear, and as there is a 1:1 gearing ratio between the yellow gear and the green gear ($\frac{25\ teeth}{25\ teeth} : \frac{25\ teeth}{25\ teeth}$), it can be assumed that the axis of the stepper motor is mechanically directly connected to the blue gear, with the addition of a small gearing transmission error. The gearing ratio between the stepper motor and the rotator's platform (red gear) is thus 2:5 ($\frac{30\ teeth}{15} : \frac{75\ teeth}{15}$).

As the stepper motor has a standard full step angular distance of 1.8°, the rotator's platform will have an effective angular displacement of 0.72° ($\frac{5}{2} \times 1.8°$). Equation 2.11 below shows the equation that can be used to calculate the amount of steps the motor has to take ($y_{step\ count}$) to move the rotator's platform x_{angle} degrees. The result is passed through a ceil function, which rounds it upwards, as only an integer amount of steps can be made. This however, could introduce an overshoot error as big as 0.72°, which is small enough to not have a visible effect on the overall solution.

$$y_{step\ count} = \lceil \frac{x_{angle}}{0.72} \rceil \tag{2.11}$$

As the software produces only positive counter-clockwise angles, the motor will be set to rotate counter-clockwise only. This is achieved by grounding the *DIR* pin (as seen in Figure 41) on the rotator's stepper motor driver. Doing so frees up a pin on the microcontroller and slightly reduces the firmware complexity.

After performing a few accuracy tests on the rotator, it was found that gear slippage and reverse gear ratio induced torque often caused an unwanted offset of up to 15°. Luckily, it was found that by applying a feedback monitoring system on piece rotations, this error could largely be avoided. This rotational feedback mechanism is explained in more detail in Section 2.2.3.

Serial Communication

Serial communication (also known as RS-232) is utilised to form the link between the *PC System* and the *Gantry System*, allowing these systems to function symbiotically. This communication link is thus of vital importance as it contributes significantly to the modularity of the problem at hand. It enables each unit to focus solely on its own core tasks, providing the overall system with greater diversification and control.

Standard serial communication allows two devices to sequentially send binary ASCII data to one another. Each device has a transmit (TX) and a receive (RX) pin. The TX pin of one device gets connected to the RX pin of the other device and vice versa. In order to facilitate communication, the two devices have to agree on the same communication parameters. These communication parameters are:

- baudrate (bitrate),
- number of databits,
- parity,
- number of stopbits, and
- flow control.

Once these parameters have been identically initialised on both devices, communication can commence. Table 12 below shows the chosen parameter values, as they have been encoded on both devices. This specific baudrate was chosen as it is shown by the microcontroller's data sheet to provide the lowest error rate out of all the standard baudrates.

Baudrate	Databits	Parity	Stopbits	Flow Control
38400Hz	8 bits	None	1 bit	None

Table 12.
Serial communication parameters

In this system, the serial communication link is between the *PC System* and the *Gantry System*. The communication signal voltage levels of these device, however, have to be identical for communication to be successful. In order to ensure consistent communication signal voltage levels, a *MAX233* line transceiver chip is used. This device acts as a high-speed buffer between the two devices, allowing their communication signals to be converted to the correct serial communication voltage levels.

In order for the two units to work together, they first have to understand each other. This was achieved by designing and implementing a shared communication protocol on both devices. This protocol used a message format that both devices could recognise and interpret, which enabled them to successfully send commands to one another. Such a message consists of an ASCII character string, which contains sub-string components that are separated by white spaces. Each such command string ends on *Microsoft Windows'* newline character, *"\r\n"*, to indicate the end of the command. The format of these sub-string components are shown in Table 13 below.

Communication Protocol Message Format						
Command Tag	X_0	Y_0	X_1	Y_1	$H_{[0:1]}$	Action Tag

Table 13.
Serial communication message format

The *Command Tag* indicates the command type, which dictates how the following components are interpreted. Table 14 below shows all the *Command Tags* that the system can interpret, and describes how the remaining parameters are processed.

Command Tag	Command	Description
"E"	Reset	For a *Reset* command, all the remaining parameters are ignored. This command instructs the gantry to return to coordinate [0, 0] (top-left most position).
"M"	Move	A *Move* command instructs the gantry to move a piece from coordinate [X_0, Y_0] to coordinate [X_1, Y_1]. The $H_{[0]}$ parameter indicates the height that the piece should be picked up from, and the $H_{[1]}$ parameter indicates the height at which it needs to be dropped. The *Action Tag* parameter is ignored for all *Move* commands.
"S"	Separate	A *Separate* command functions similarly to a *Move* command, with the addition of an intermediary overlapping piece separation action. Once a piece has been picked up, it will be raised one height level higher than its initial position. It will then be moved 200 steps up, right, down, or left for an *Action Tag* of 0, 1, 2, or 3 respectively. This will ensure that the actuator does not accidentally flip the top-most piece in an overlapping cluster, should it pick up the bottom-most piece. After this intermediary action, the system will proceed as a *Move* command.
"R"	Rotate	A *Rotate* command functions similarly to a *Move* command, with the addition of an intermediary piece rotation step. Once a piece has been picked up, it will be moved and dropped at the piece rotator. The rotator will then rotate it counter-clockwise by the amount of degrees indicated by the *Action Tag*. Once this is done, the piece will be picked up again and the system will resume normal *Move* operation.
"P"	Place	A *Place* command instructs the system to move to coordinate [X_0, Y_0] and then lower the actuator to height $H_{[0]}$ (without engaging the vacuum mechanism), after which the system will move to coordinate [X_1, Y_1] with the actuator still lowered. This action is used to drag piece along short distances, which is useful for nudging pieces into an interlocking position. The *Action Tag* is ignored for this command.
"D"	Done	The *Done* command is sent from the *Gantry System* to the *PC System* to indicate that it has finished its current action. For this command, all the remaining parameters are ignored.
"T"	Test	The *Test* command is used purely for debugging, accuracy measurement, and calibration purposes. This command instructs the gantry to move to coordinate [X_1, Y_1] from coordinate [X_0, Y_0], and then lower the actuator to height $H_{[0]}$ (without engaging the vacuum mechanism). This allows the actual location of the gantry to be accurately measured. For this command, all the remaining parameters are ignored.

Table 14.
System command tags

Microcontroller and Firmware

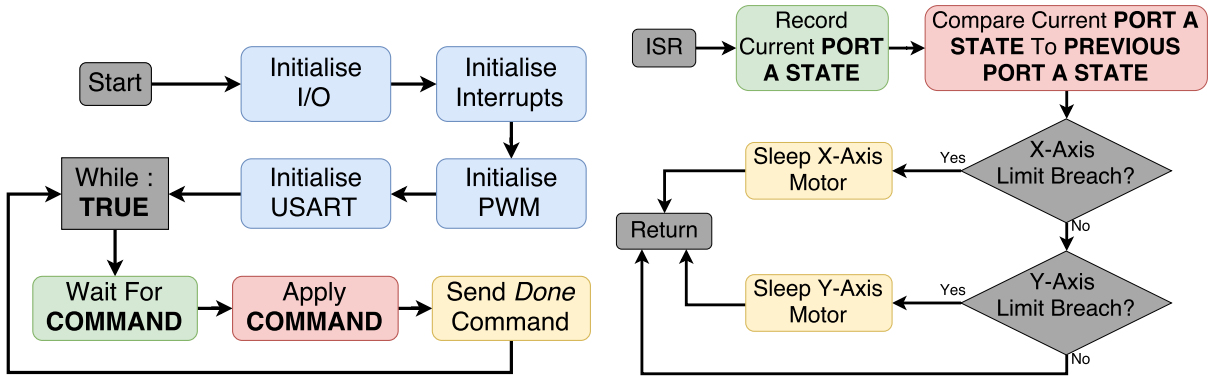
At the heart of the mechanical control unit, lies the *Atmel ATmega1284P-PU* 8-bit microcontroller. This component will house the firmware responsible for directing all the electro-mechanical components, and for communicating with the *PC System*. Table 15 below shows the pins relevant to this system and describes their I/O mode and functionality.

Pin	I/O	Description
PA0 - PA3	I	These pins serve as external interrupt signals from the limit monitoring *x*-and *y*-axis switches. PA0 and PA1 are the left and right *x*-axis switches respectively, and PA2 and PA3 are the top and bottom *y*-axis switches respectively.
PB0 - PB2	O	These pins are responsible for the *Step*, *Direction*, and *Sleep* of the *x*-axis stepper motor respectively. These pins control the motor via the equally named *x*-axis stepper motor driver pins.
PB3 - PB5	O	These pins are responsible for the *Step*, *Direction*, and *Sleep* of the *y*-axis stepper motor respectively. These pins control the motor via the equally named *y*-axis stepper motor driver pins.
PB7 - PB8	O	These pins are responsible for the *Step* and *Sleep* of the piece rotator stepper motor respectively. These pins control the motor via the equally named piece rotator stepper motor driver pins.
PD0	I	This pin is responsible for receiving serial (USART) data (RX).
PD1	O	This pin is responsible for the transmission of serial (USART) data (TX).
PD4 - PD5	O	These pins are responsible for providing the *Vacuum Suction Mechanism* and the *Piece Actuator* with the necessary PWM signals respectively.

Table 15.
Atmel ATmega1284P-PU pin description

Figure 45a below shows the main flow diagram for this device. The Initialisation steps configure all the necessary device parameters, as described in the preceding sub-sub-sections. The Wait For COMMAND step is achieved by polling the microcontroller's USART flag. The Apply COMMAND step first interprets and then applies a received command, given that it is valid. The interpretation and application of such commands, follows the structure described in Table 14 above. The Send *Done* Command step is achieved by sending the ASCII string *"D\r\n"* to the *PC System*, indicating that it has completed its current command.

Figure 45b below shows the flow diagram for the interrupt service routine of the device. As shown in Table 15 above, **PORT A** of the device is connected to the limit monitoring switches.



(a)
Main flow diagram

(b)
ISR flow diagram

Figure 45.
Firmware flow diagram

Figure 46 below shows the integrated system circuit, the design schematic that is illustrated in Figure 47. Table 16 below provides a brief description of the main components and connector pins.

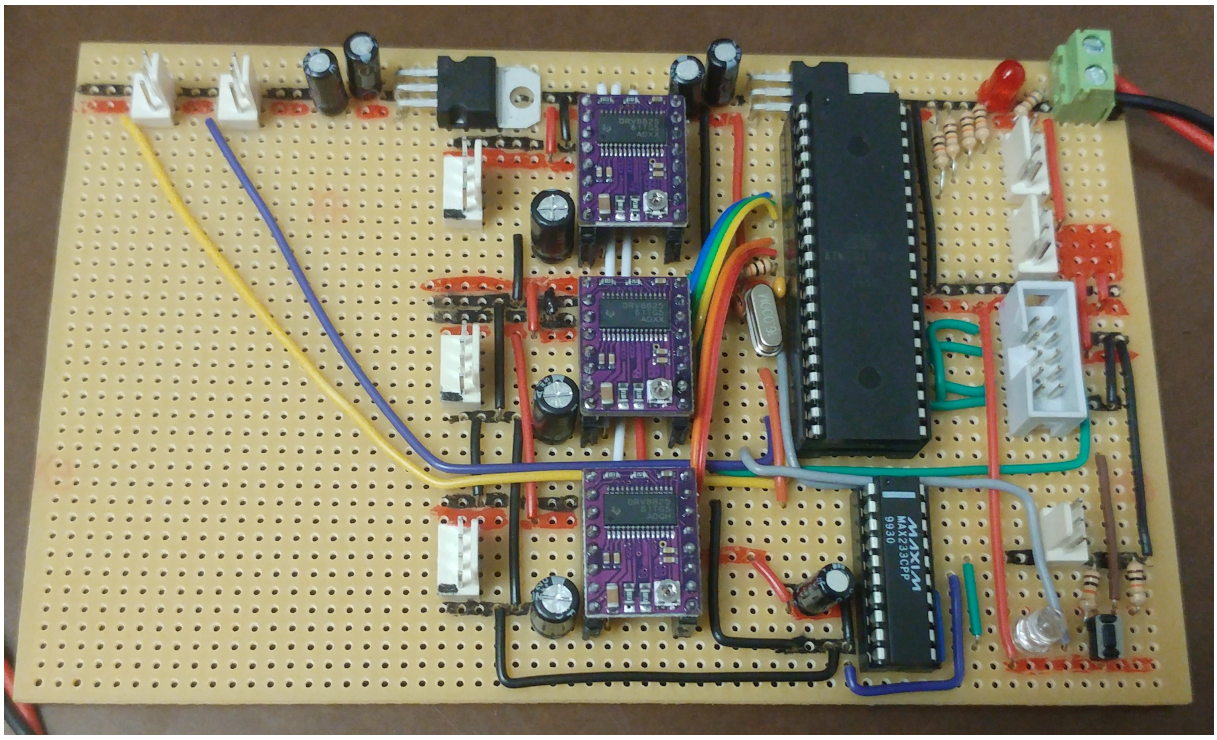


Figure 46.
Control circuit

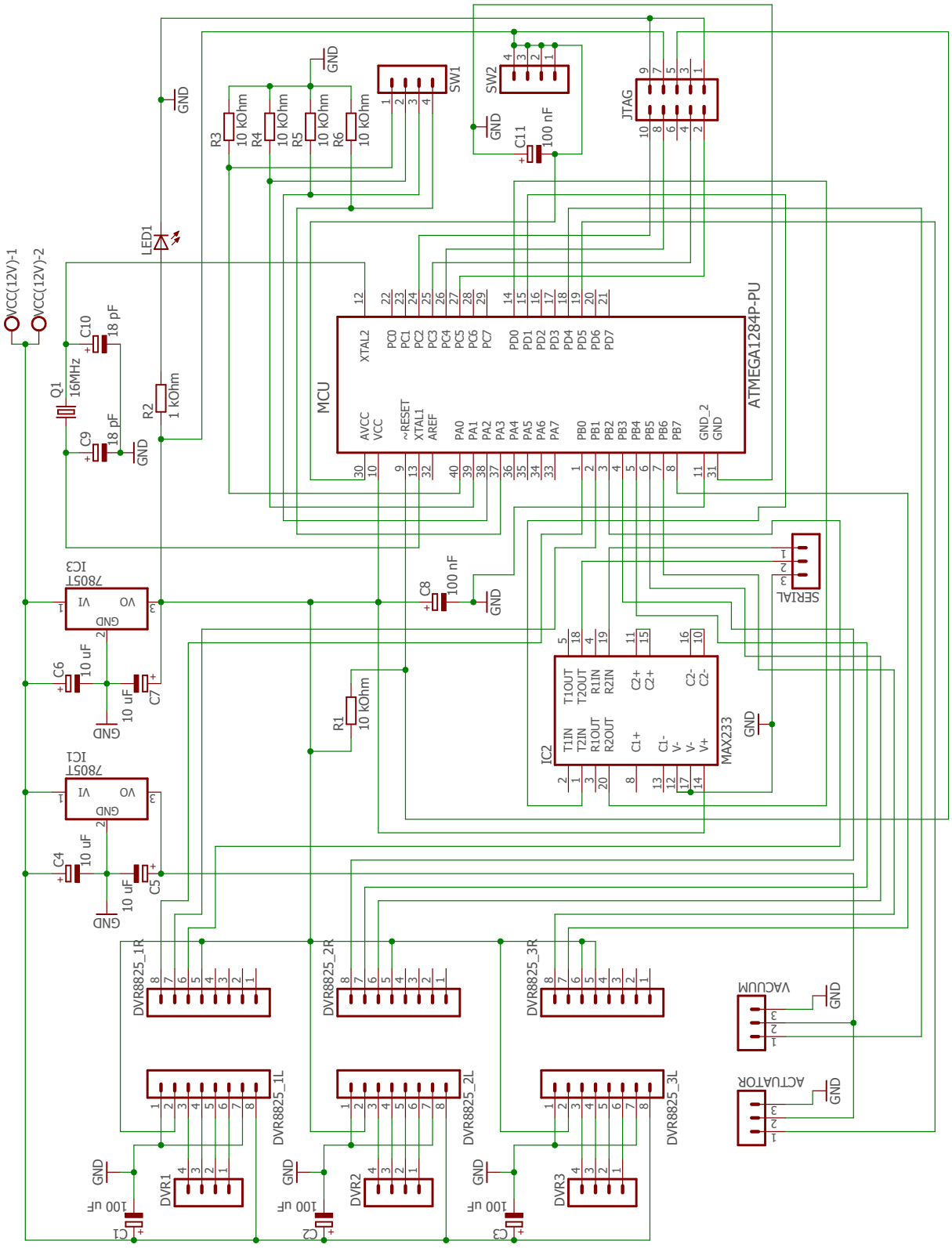


Figure 47.
Control circuit design schematic

Component/ Connector	Description
SW1 - SW2	These male connector pins are used to connect to the limit-monitoring switches. They are connected in a pull-up configuration such that when the switches are pressed, and short-circuited, they pull the corresponding pins' voltage high. When the switches are not pressed, the $10k\Omega$ resistors pull the pins' voltage low.
JTAG	This male connector is used to program the *ATmega1284-PU* microcontroller via a *JTAG ICE-mkII* AVR programmer.
Q1	This ia a $16MHz$ crystal oscillator, which provides the microcontroller with a stable and accurate clock signal.
MCU	This is the *ATmega1284-PU* microcontroller, which houses the firmware that controls the entire electronic system.
7805T	These two chips convert the $12V$ input to $5V$, which is the operational voltage of all the components. The top *7805T* chip (IC3) provides power to all the ICs in the system, and the lower *7805T* chip (IC1) provides power to the two servo motors. These two chips are used separately so as to reduce the current demand and temperature strain from both devices.
MAX233	This chip ensures that the communication voltage levels are at the correct serial communication voltage levels so as to facilitate communication between the two systems adequately.
SERIAL	These male connector pins are connected to the TX (pin 1), RX (pin 2), and GRN (pin 3) pins on the serial cable connected to the *PC System*.
DVR8825_X	These female connector pins are used to connect to the various *DVR8825* stepper motor drivers. $X = 1$ connects to the x-axis driver, $X = 2$ connects to the y-axis driver, and $X = 3$ connects to the rotator driver.
DVRX	These male connector pins are used to connect to the system's stepper motors. $X = 1$ connects to the x-axis motor, $X = 2$ connects to the y-axis motor, and $X = 3$ connects to the rotator motor.
VACUUM	This male connector connects to the PWM signal (pin 1), $5V$ (pin 2), and GND (pin 3) pins on the vacuum suction mechanism.
ACTUATOR	This male connector connects to the PWM signal (pin 1), $5V$ (pin 2), and GND (pin 3) pins on the piece actuator.

Table 16.
Control circuit design schematic

Figure 48 below shows the fully-connected electronic system in its enclosure, and Figure 49 below gives an overview of the entire system after it completed a solution construction successfully.

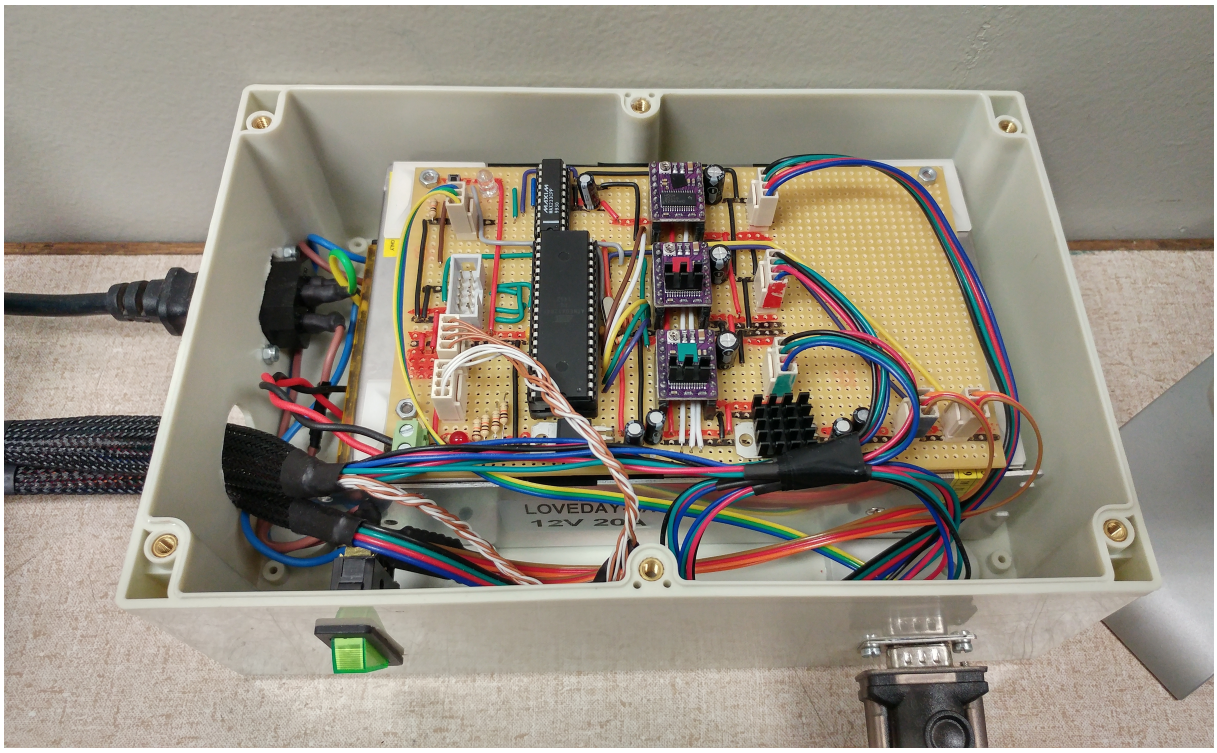


Figure 48.
Control circuit enclosure connection

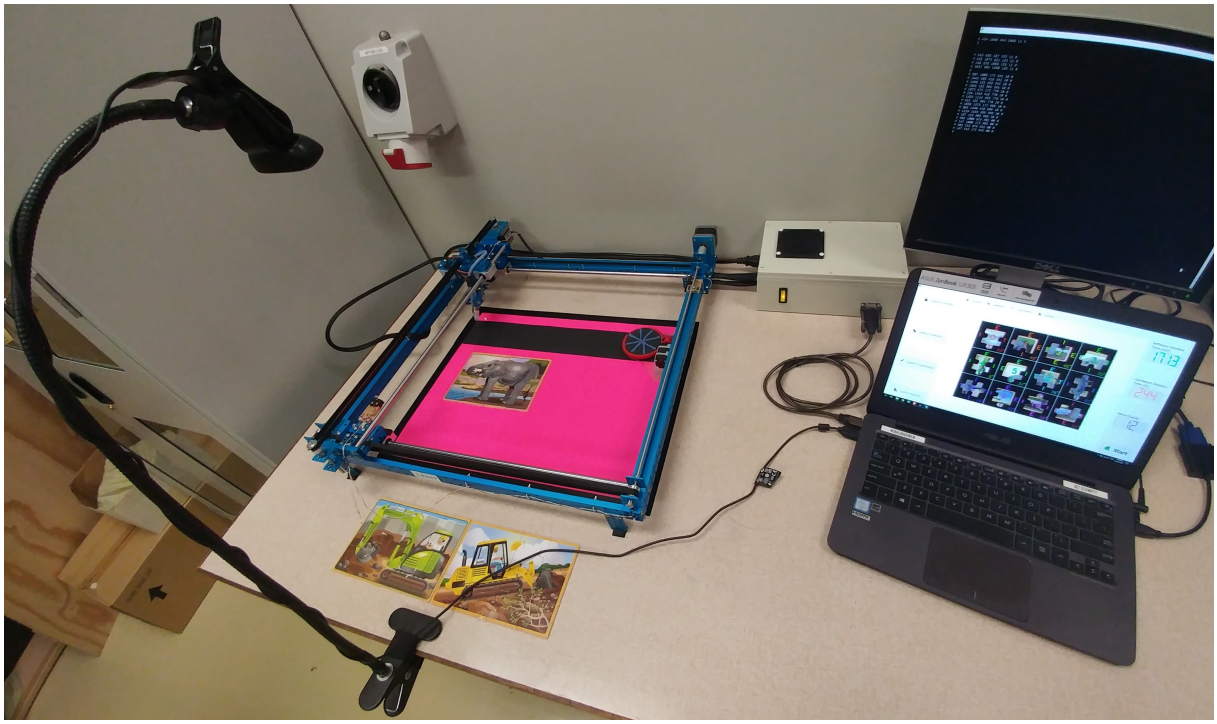


Figure 49.
Entire system overview

2.2.3 Integration

Gantry Accuracy

Once the two systems were fully integrated, a strategy was devised which allowed them to work together to construct a puzzle. Since a software solution and piece classification information was available, it was easy enough to determine the source and destination coordinates for each piece, along with the angle by which it needs to be rotated. The *Gantry System* could move and rotate pieces accurately, and since a known communication protocol was implemented, it could simply use the information provided by the *PC System*. The problem here was that the *Gantry System* performed translation in the form of steps, where the *PC System's* coordinates were all in pixel values.

The initial approach, hereafter referred to as **Movement Approach A**, was simply to take the *PC System's* pixel range ([0 to 1280, 0 to 960]) and convert it linearly to the *Gantry System's* step range ([0 to 2100, 0 to 1650]). By doing so, the following linear relationship was obtained:

x-axis equation:

$$x_{step} = \frac{2100}{1280} \times x_{pixel} \quad \left[\frac{step \times pixle}{pixel} = step\right]$$
$$x_{step} = 1.6406 \times x_{pixel} \quad [step] \tag{2.12}$$

y-axis equation:

$$y_{step} = \frac{1650}{960} \times y_{pixel} \quad \left[\frac{step \times pixle}{pixel} = step\right]$$
$$y_{step} = 1.7188 \times y_{pixel} \quad [step]$$

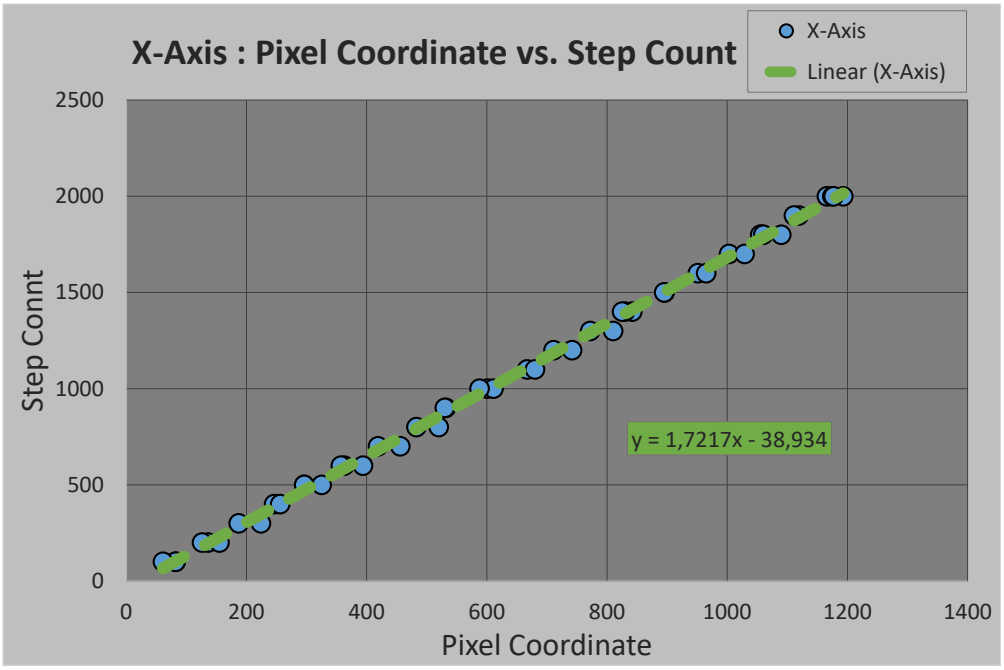
In order to test the accuracy of **Movement Approach A**, a few random pixel locations were chosen. The *Gantry System* was instructed to move to these location once their coordinates were passed through the liner equations shown above. The gantry's deviations from the desired locations were then measured and recorded. After 50 locations were tested with **Movement Approach A**, the following deviations were measured from the gantry's desired destinations to its actual destinations.

- Average deviation: 16*mm* (87 steps)
- Maximum deviation: 28*mm* (152 steps)

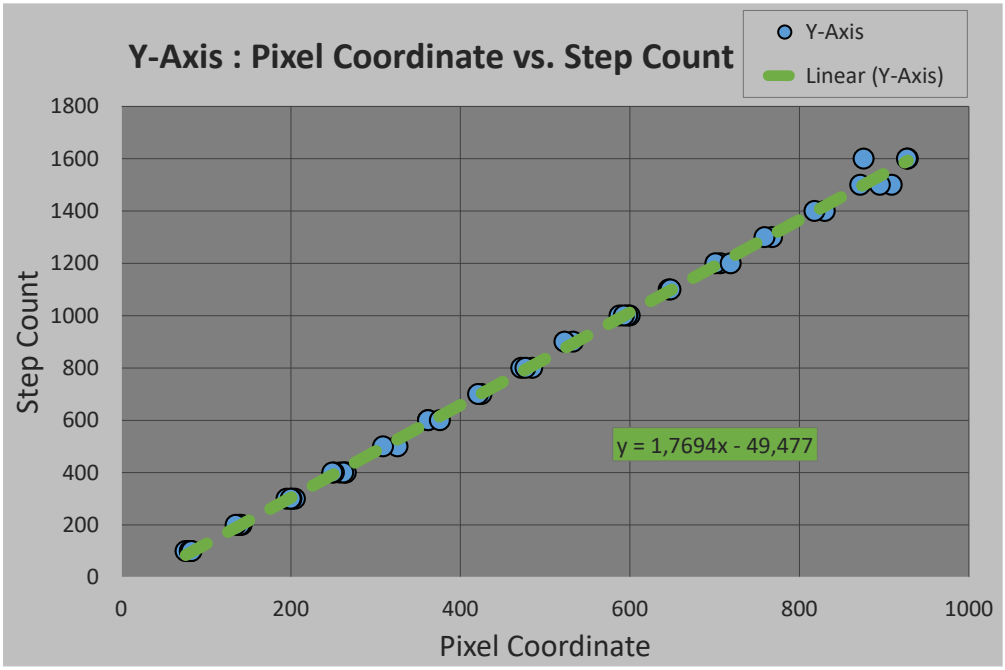
This deviation is too large to achieve accurate piece manipulation, and is larger than the system's error specification. This error is partially owing to the fact that the piece actuator does not start at pixel coordinate [0, 0], which causes an unwanted linear offset error. Another contributing factor could be the start-stop jolt from the stepper motors, causing an unwanted overshoot error.

In order to remedy the errors of **Movement Approach A**, a calibration mechanism was created. The gantry was sent to random step coordinates, and the gantry's corresponding pixel coordinates were recorded. By once again repeating this 50 times, a large set of step versus pixel coordinate

data was obtained. With the help of a trend line function, a best fit-line formula for these coordinates was calculated. Figure 50 below shows the initial calibration graphs for both the *x*- and *y*-axis data respectively. The trend line formula for each graph is visible in the green boxes. These formulas were then encoded onto the *Gantry System* to form **Movement Approach B**.



(a)
Initial calibration: x-axis



(b)
Initial calibration: y-axis

Figure 50.
Initial movement calibration

Prior to applying any accuracy tests, numerous coordinates can be seen deviating from the fit-lines in the calibration graphs above. When implemented, this will cause the system to have unwanted overshoot/undershoot errors. The linear offset shown in the fit-line formulas above confirm that the actuator does indeed not start at pixel coordinate [0, 0]. As these calibration graphs inherently take the linear offsets into account, the coordinate deviations cannot be attributed to this offset.

By once again applying the random location test, as described above, and by closely analysing the movements of the system, the main problem became clear. For **Movement Approach A** and **Movement Approach B**, the stepper motors were put into sleep mode whilst not in use (i.e. not moving). The problem with this approach is that when the motors are in sleep mode, current is prevented from flowing through their armature coils and, as a result, there is no electromagnetic force being applied to the axis of the motors, which leaves it free to rotate unsupervised. This motor sleep scheme thus causes the following two unwanted anomalies:

1. as soon as a motor reaches its destination, it is immediately put into sleep mode, which leaves its momentum free to rotate the motor's axis beyond than which had been intended, and
2. when the motors are brought out of sleep mode, the resulting force from the sudden armature current influx causes an unwanted jolt in the motors, which could cause small, yet significant, linear deviations.

The impact of these anomalies becomes clear when reviewing the results of the accuracy test. After 50 test locations, **Movement Approach B** was found to have the following results.

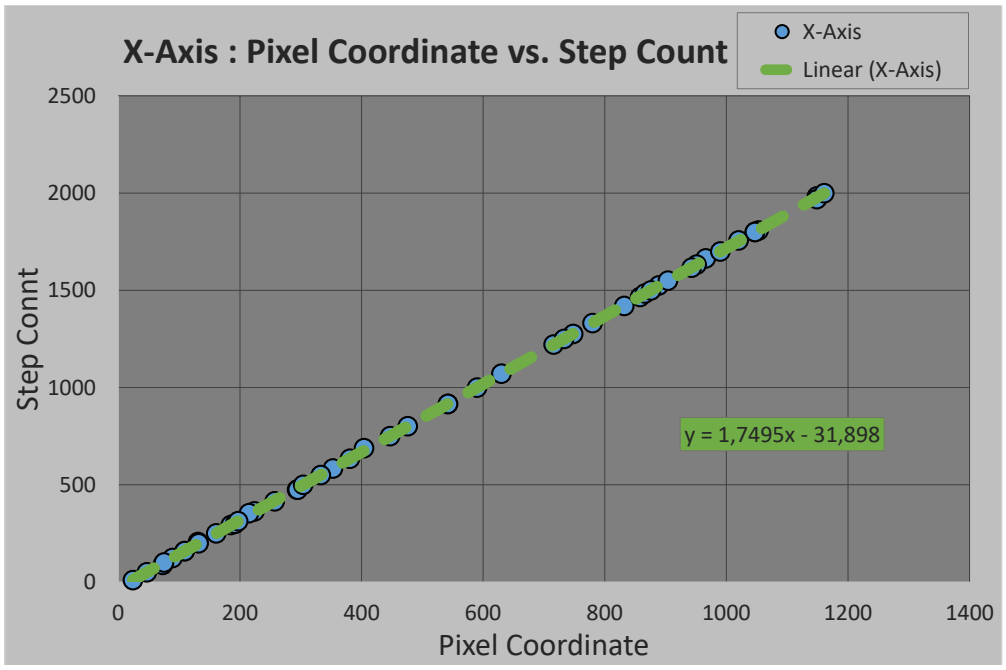
- Average deviation: 5*mm* (27 steps)
- Maximum deviation: 12*mm* (65 steps)

Although better than that of **Movement Approach A**, the accuracy of **Movement Approach B** is still not sufficient enough to meet the system requirements. To solve this problem, it was decided to only put the motors into sleep mode once all puzzle-solving operations had been completed. This grants the system greater accuracy during puzzle construction, whilst still allowing the motors to sleep when the system is idle, which reduces the overall strain on the motors and motor driver hardware.

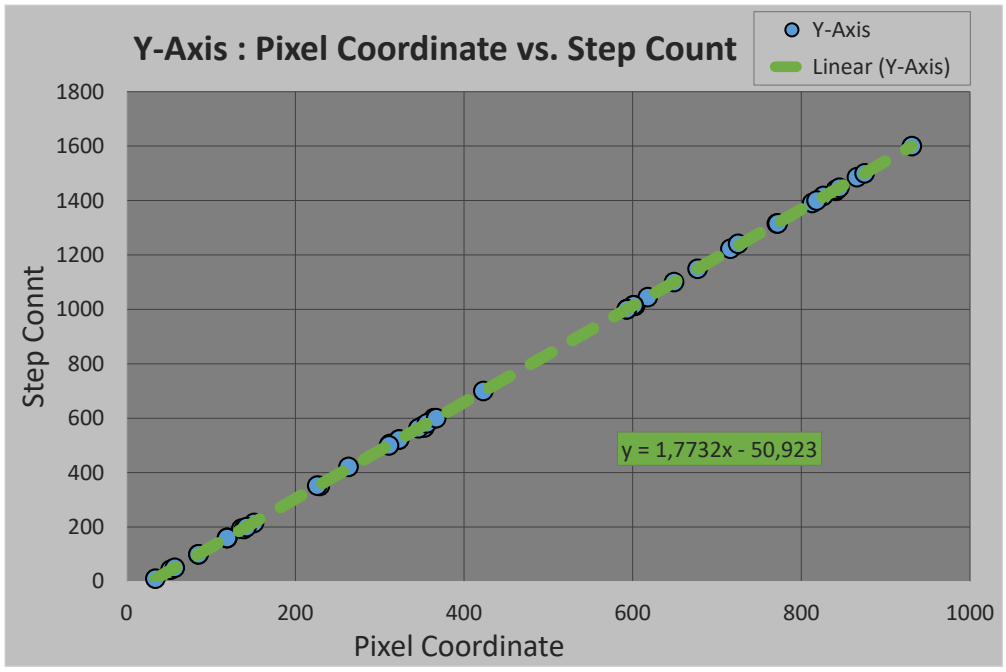
After the abovementioned changes were made to the firmware, the calibration mechanism was applied to the system once more. The results of this second calibration is shown in Figure 51 below. The results of this second calibration, along with the new motor sleep pattern, forms **Movement Approach C**. These graphs show a clear improvement from **Movement Approach B**, as little to no coordinate deviation is visible. After applying the accuracy test to **Movement Approach C**, the following results were obtained.

- Average deviation: 1*mm* (5 steps)
- Maximum deviation: 4*mm* (22 steps)

These measures are within the system's specifications and accurate enough to successfully manipulate puzzle pieces.



(a)
Movement calibration: x-axis



(b)
Movement calibration: y-axis

Figure 51.
Movement calibration

Puzzle Construction

Since all puzzle piece information, including that of the solution, is made available by the *PC System*, the *PC System* simply needs to supply the *Gantry System* with a construction strategy. As

an initial construction strategy, hereafter known as **Construction Approach A**, it was decided to build the puzzle starting from the top left-most corner, working down to the bottom right-most corner. To illustrate, the indexed solution image, as generated by the *PC System*, shown in Figure 52 below will be constructed in in the order $5 \rightarrow 3 \rightarrow 0 \rightarrow 1 \rightarrow 4 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 11 \rightarrow 10 \rightarrow 2$.

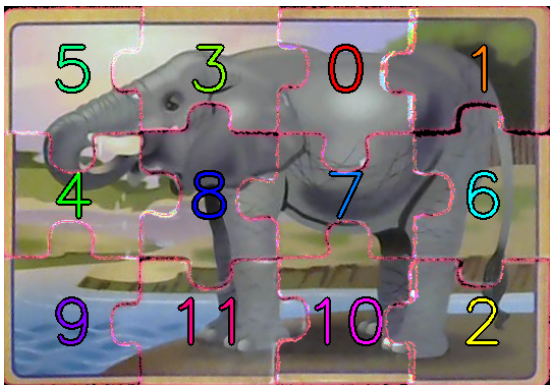


Figure 52.
Indexed software solution [4]

Before construction could commence, however, ample space first had to be made available. In order to do so, pieces would first be stacked in piles on the black temporary holding strip, as can be seen in Figure 53 below. The red squares indicate the four temporary pieces holding location, which would each hold three pieces. These pieces would be placed from left to right in the reversed construction order, such that the next piece needed for construction would never be underneath another piece. Once all pieces had been stacked, the system proceeded to build the puzzle in the construction area, indicated by the blue rectangle in Figure 53. Before a piece was placed in the building area, it would be rotated correctly by the piece rotator, shown in the top right corner in Figure 53.

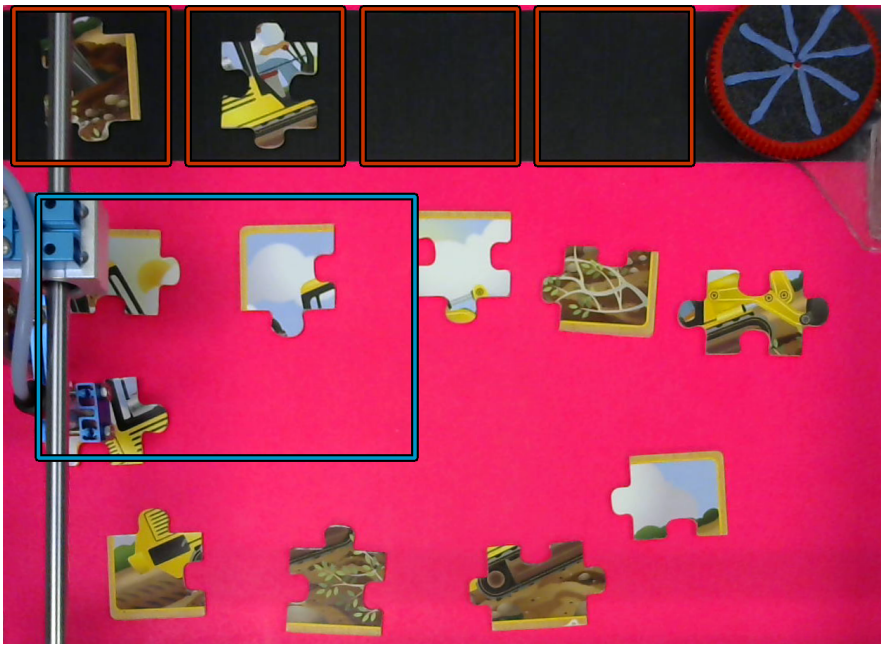


Figure 53.
Puzzle construction and stacking locations [4]

In order to test the accuracy of a construction strategy, a solution scoring system was devised. For each piece that is in the correct position and perfectly interlocked, the solution gains one point. After a solution is completed, its total accumulated score is then divided by its puzzle size in order to get an accuracy percentage. If a 12-piece puzzle is built perfectly, it will score 12 points and thus 100%. If a 12-piece puzzle solution has 9 pieces correctly placed and interlocked, it will score 75%. Additionally, the construction time of each solution is also recorded. To ensure that the results are not selective towards a specific puzzle, three different 12-piece puzzles were used interchangeably whilst scoring solutions. After 50 solutions were built, the following results were obtained for **Construction Approach A**.

- Average solution score: 78.33%
- Average solution time: 264.12 seconds
- Total dropped pieces: 7

Considering that a solution consists of 36 moves on average, and that 50 solutions were built, 7 dropped pieces out of 1,800 moves (36×50) gives a piece-drop likelihood of only 0.38%, which is low enough for the overall success of the system. The average solution time of 264.12 seconds is well within the 15-minute time constraint. The average solution score of 78.33%, however, needed improvement.

The main inefficiency of **Construction Approach A** was that it moved each piece to a temporary stack location irrespective of its initial position. If a piece is not in a spot that would hinder the construction of the puzzle, it does not need to be moved to a temporary location. As each piece's centre coordinates are already made available by the *PC System*, it is easy enough to check if a piece is within the puzzle construction area or not.

As a new construction strategy, **Construction Approach B**, only pieces that are within, or near, the puzzle construction area will be temporarily stacked, whilst the other pieces will remain in their initial positions until they need to be rotated or placed. The rest of the strategy will be executed as the initial strategy mentioned above. After 50 solutions were scored, the following results were obtained for **Construction Approach B**.

- Average solution score: 86,67%
- Average solution time: 187,78 seconds
- Total dropped pieces: 5

As **Construction Approach B** used fewer moves on average than **Construction Approach A**, the average solution time improvement, as shown by the results, is to be expected. It was observed that each piece manipulation had the possibility to created a small offset, owing to the $1mm$ average gantry deviation. After multiple manipulations, these minor offsets would add up, which caused a greater overall offset. It is thus expected that when fewer moves are performed, the average solution accuracy would increase, as there would be fewer cumulative deviations. This is supported by the increase in average solution accuracy from **Construction Approach A** to **Construction Approach B**, as shown by the results.

Despite the overall improvements, there were still a few minor defects in the system. The main cause of these defects was a combination of the cumulative gantry deviations and the piece actuator linear offset (the latter as described in Sub-sub-section 2.2.2). Moving pieces to and from

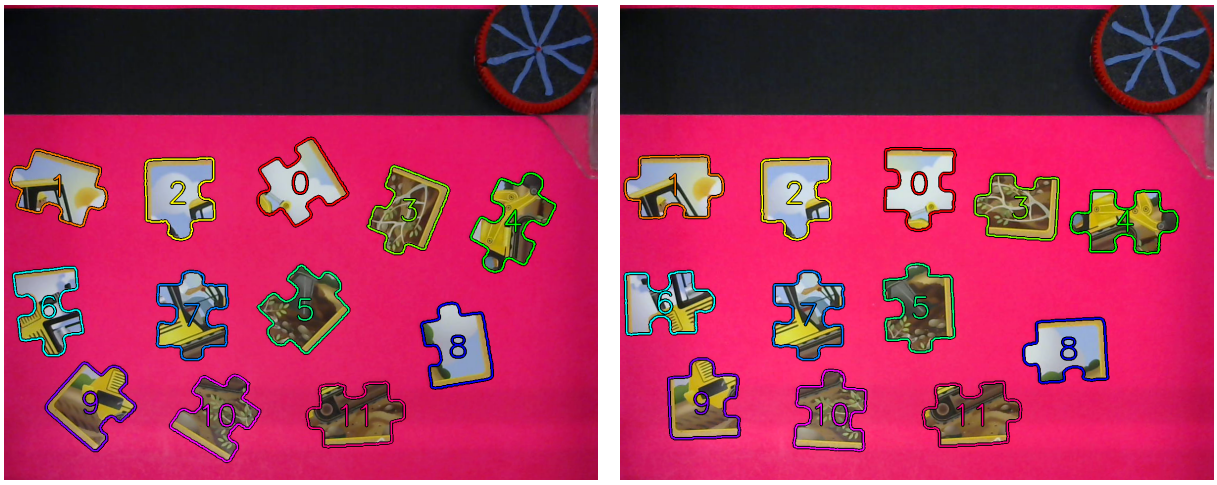
the rotator had the biggest linear offset effect on pieces, mainly because it was approximately the height of five stacked pieces. Additionally, after a few measurements, the piece rotator was found to have the flowing deviations.

- Average deviation: 3°
- Maximum deviation: 15°

These factors contributed to pieces being picked up or rotated incorrectly, which introduced small errors to puzzle solutions. To remedy this, it was decided to introduce a feedback system, using the *PC System* already in place, to analyse and correct piece rotations. When puzzle construction starts, pieces will be picked up, rotated, and then returned to their initial locations. After each piece has been rotated and returned to its original position, the gantry will return to coordinate [0, 0], which will also remove any built up gantry deviation. The *PC System* will then recalculate each piece's angle and centre location, by using techniques very similar to the piece classification method. This will accomplish the following two things.

1. By recalculating each piece's centre location, any offset errors introduces by piece rotations are completely removed.
2. By recalculating each piece's angle, any piece that is not correctly rotated can be identified and re-rotated.

Figure 54 below shows the piece detection before and after the piece rotation phase. To further reduce the likelihood of dropping a piece, the gantry will reset prior to rotating a piece with a narrow centre (such as piece 4, 6, and 7 below). This will increase the accuracy during the piece rotation phase, as the reset command removes any built up offset from the gantry.



(a)
Piece detection before rotation

(b)
Piece detection after rotation

Figure 54.
Piece rotation feedback [4]

Figure 55 below show an example of all the command messages sent from the *PC System* to the *Gantry System* during one entire puzzle construction cycle. Figure 55a shows the piece separation

and piece rotation commands. After these commands have been completed, the system applies the rotation feedback algorithm, as described above. Figure 55a shows the temporary piece stacking, piece movement, and piece placement commands.

```
S 1355 1017 1927 1368 11 1
E

S 1186 969 1534 1080 11 1
E

R 890 1008 890 1008 11 37
R 1445 687 1445 687 11 70
E
R 1039 600 1039 600 11 50
R 253 637 253 637 11 48
E
R 1877 674 1877 674 11 98
R 248 1341 248 1341 11 289
E
R 1453 1109 1453 1109 11 351
R 454 1054 454 1054 11 346
R 1870 1366 1870 1366 11 255
R 811 1440 811 1440 11 52
E
R 1233 1430 1233 1430 11 29
R 652 667 652 667 11 45
E
```

(a)
Separation and rotation commands

```
M 645 665 187 153 11 0
M 453 1073 621 153 11 0
M 246 635 1056 153 11 0
M 1037 603 1490 153 11 0
E
M 887 1009 173 541 10 0
M 1443 685 416 541 10 0
M 1490 153 659 541 10 0
M 1056 153 902 541 10 0
M 1873 673 173 770 10 0
M 239 1366 416 770 10 0
M 1453 1114 659 770 10 0
M 621 153 902 770 10 0
M 1858 1375 173 999 10 0
M 803 1440 416 999 10 0
M 1229 1435 659 999 10 0
M 187 153 902 999 10 0
P 902 1008 876 981 00 0
P 147 1008 173 981 00 0
P 902 515 876 541 00 0
P 147 515 173 541 00 0
E
```

(b)
Movement and placement commands

Figure 55.
Command message example

This construction method, along with the rotation feedback control, forms **Construction Approach C**. This approach will perform more moves on average than **Construction Approach B**, but with improved accuracy. After 50 solutions were scored, the following results were obtained for **Construction Approach C**:

- Average solution score: 97,44%
- Average solution time: 210,46 seconds
- Total dropped pieces: 1

As can be seen in Figure 55b, there were only 16 puzzle construction moves (i.e. movements and placements) after the second-last reset. There were thus far fewer moves to contribute to the cumulative offset error, and hence the much improved accuracy of **Construction Approach C**. This exact anomaly also contributed to the lowered piece-drop rate. The rotation feedback implemented in **Construction Approach C** did however slow it down, which caused it to be slightly slower than **Construction Approach B**, yet still fast enough to meet the system's specifications.

2.3 DESIGN SUMMARY

Table 17 lists all the tasks that were completed for this project and describes how they were implemented.

Task	Implementation	Status
Design and develop an algorithm capable of detecting stacked and unstacked puzzle pieces.	The algorithm was implemented in *Python*, partially utilising a contour detection algorithm. The majority of this task, and the main functionality of the algorithm, was designed and developed from first principles.	Completed
Design and develop an algorithm capable of classifying puzzle pieces.	The algorithm was implemented in *Python*, partially utilising the *Hough Line Transform* and a contour defect algorithm. The *Hough Line Transform* was adapted to fulfil the needs of the algorithm. The majority of this task, and the main functionality of the algorithm, was designed and developed from first principles.	Completed
Design and develop an algorithm capable of creating a software solution for an unsolved jigsaw puzzle that the algorithm has no prior knowledge of.	The algorithm was implemented in *Python*, partially utilising the *Chi-square Histogram Comparison*. The *Chi-square Histogram Comparison* was adapted to fulfil the needs of the algorithm. The algorithm contains a starting piece selection, shape-matching, and colour-matching component, all of which were designed and developed from first principles. All the auxiliary methods created to aid the algorithm, such as the centre-correction method and the interlocked solution-generation method, were also designed and developed from first principles.	Completed
Design and develop a GUI that can relay all relevant system information to users.	The GUI was designed in *QtDesigner* and converted to *Python* code with *PyQt*. The GUI's functionality was designed and developed from first principles.	Completed
Design and develop a software control algorithm that can integrate and direct all the system's software components.	The algorithm was fully implemented from first principles in *Python*.	Completed
Design and develop a software simulation platform capable of demonstrating all the software component's functionality.	The simulator was implemented in *Python*, partially utilising the *OpenCV* graphical display tools. The majority of this task and the main functionality of the simulator was designed and developed from first principles.	Completed

Design and construct a piece actuator, vacuum suction mechanism, and piece rotator capable of manipulating and rotating puzzle pieces respectively.	These mechanisms were created using discrete electro-mechanical and mechanical components. These mechanisms were designed and constructed from first principles.	Completed
Design and construct a gantry robot capable of transporting puzzle pieces.	The robot was created using discrete electro-mechanical and mechanical components. The design of the robot and its components were taken off the shelf.	Completed
Design and develop a communication mechanism between the *PC System* and the *Gantry System*.	RS-232 was used to facilitate physical communication, the infrastructure of which was constructed from discrete electronic and software components. The command message protocol was designed and developed from first principles.	Completed
Design and construct an integrated electronic system capable of controlling the system's electro-mechanical units.	A microcontroller, motor drivers, and discrete electronic components were used to design and develop the integrated electronic system. The circuit layout was designed in *Eagle PCB Designer* but was ultimately built on veroboard. This system was mostly designed and constructed from first principles with the exception of the microcontroller and the motor drivers. The power supply for this system was taken off the shelf.	Completed
Design and develop firmware capable of controlling the system's microcontroller.	The firmware was designed and developed using the microcontroller's libraries and discrete software components. The majority of this task and the main functionality of the firmware was designed and developed from first principles in *C* code.	Completed
Design and develop a puzzle construction scheme capable of translating a software solution into a physical solution.	A spreadsheet was created to perform all calibration calculations. The movement scheme, which includes partial feedback control, was implemented in *Python*, and was designed and developed from first principles.	Completed

Table 17.
Project task description

3. Results

3.1 SUMMARY OF RESULTS ACHIEVED

Description of requirement or specification (intended outcome)	Actual outcome	Location in report
Mission requirements of the product		
The system should be able to solve and build a physical, kindergarten level, jigsaw puzzle within 15 minutes.	The final system could construct a 12-piece jigsaw puzzle in 210 seconds (3 minutes 30 seconds) on average.	Sub-section 3.7
The system should be able to handle a minimum of 12 standard jigsaw puzzle pieces, each ranging between $40 \times 40 \times 1mm$ and $80 \times 80 \times 5mm$ in size with a maximum tab diameter of $40mm$.	The final system could successfully manipulate pieces that fit within the size specification, and had a piece-drop likelihood of only 0.04% per piece manipulation.	Sub-section 3.3 Sub-section 3.7
The system should be able to handle cases where pieces in the initial puzzle layout overlap, provided that such overlapping clusters consist of no more than three pieces.	The final system could successfully separate overlapping piece clusters, consisting of up to three pieces, and could still construct a successful solution. The system had a piece separation likelihood of 85.33%.	Sub-section 3.3 Sub-section 3.7
The system should operate with a maximum displacement error of $10mm$.	The final system had an average displacement error of $1mm$ and a maximum displacement error of $4mm$.	Sub-section 3.7
Field conditions		
The system will be designed, tested and operated in controlled laboratory conditions with ambient lighting ranging between 300 and 500 lux.	The system was designed in the ISG undergraduate labs at the University of Pretoria, which complies with the operational conditions. The system could successfully construct puzzles in this environment.	Sub-section 3.7
The system will handle all puzzle pieces on a controlled, and uni-coloured platform.	All physical puzzle operations were performed on a pink uni-colour platform. The system could successfully construct puzzles in this environment.	Sub-section 3.7

Specifications		
The piece detection algorithm (FU2.1) should be able to process images of the initial puzzle layout and identify all the puzzle pieces provided that they are all uniform, valid, undamaged, and present.	The system could successfully detect pieces in initial layouts with a success rate of 99.33%. The system could also successfully classify puzzle pieces with a success rate of 95.17%.	Sub-section 3.2
The puzzle-solving algorithm (FU2.2) should be able to solve the puzzle by manipulating virtual pieces in software.	The system could create successful software puzzle solutions with a success rate of 96%.	Sub-section 3.5
The movement calculation unit (FU2.3) should be able to determine the most optimal route to deliver pieces to their correct location, given that a software solution exists.	The system could successfully translate software solutions into physical solutions with an accuracy of 97.44%.	Sub-section 3.7
The system controller (FU2.4) must control and connect all the main software components.	The system controller succeeded at integrating and directing all the various software components.	Sub-section 3.2 Sub-section 3.3 Sub-section 3.4 Sub-section 3.5 Sub-section 3.7
The GUI (FU2.5) will display all software and hardware operations in real time, on a graphical user displays.	The GUI could successfully relay all relevant system information to users, and allowed users to operate the system easily.	Sub-section 3.7
The communication unit (FU3.1) must facilitate communication between the *PC System* and the *Gantry System* at a rate that will satisfy the time limitations.	The communication mechanism allowed the *Gantry System* and the *PC System* to communicate with one another successfully, and had a 100% success rate.	Sub-section 3.3 Sub-section 3.6 Sub-section 3.7
The microprocessing unit (FU3.2) must control the entire *Gantry System* by translating commands received from the movement calculation unit (FU2.3) into destination coordinates.	The *Gantry System* could successfully monitor and control the entire integrated electronic system, and could accurately apply all commands received from the *PC System*.	Sub-section 3.3 Sub-section 3.6 Sub-section 3.7
The *Gantry System* (FU3.4, FU3.5, and FU4) will be capable of moving a piece within the $0.4 \times 0.4 \times 0.05m$ problem space. The robot's movements have to be fluent enough to ensure the piece manipulator has constant control over puzzle pieces whilst in transit.	The *Gantry System* could successfully manipulate pieces within the construction area, and had a piece-drop likelihood of only 0.04% per manipulation.	Sub-section 3.3 Sub-section 3.7

Deliverables		
Jigsaw puzzles to construct during demonstration.	The system had access to 12 12-piece jigsaw puzzles that fell within the piece size specification of the system.	—
A digital camera capable of capturing images of the problem domain (FU1).	The system utilised a *Logitec C310* 720p web camera, which contained a camera-to-USB interface.	—
A digital camera interface capable of supplying the image-processing software with a data format it can interpret (FU1).	The image acquisition unit functioned as specified.	Sub-section 3.2 Sub-section 3.4 Sub-section 3.5 Sub-section 3.7
A software component that can recognise puzzle pieces and store relevant information about each piece in an appropriate format (FU2.1).	The piece detection and piece classification algorithms could successfully detect and classify pieces with a success rate of 99.33% and 95.17% respectively.	Sub-section 3.2 Sub-section 3.4
A software component capable of solving a digital version of the puzzle when provided with the necessary puzzle piece information (FU2.2).	The puzzle solving algorithm could successfully create software solutions with a success rate of 96%.	Sub-section 3.5
A software component that can determine appropriate puzzle piece manipulations that will lead to a physical puzzle solution, if provided with a digital puzzle solution (FU2.3).	The movement calculation algorithm could successfully translate software piece separations and software solutions into movement commands.	Sub-section 3.3 Sub-section 3.7
A software component capable of controlling and integrating various software components in order to derive a physical puzzle solution strategy from an aquittal image of an initial puzzle layout (FU2.4).	The system controller succeeded at integrating and directing all the various software components.	Sub-section 3.2 Sub-section 3.3 Sub-section 3.4 Sub-section 3.5 Sub-section 3.7
A graphical user interface to display the software component's functionality and processing (FU2.5).	The GUI could successfully relay all relevant system information to users, and allowed users to operate the system easily.	Sub-section 3.7
Class and flow diagrams to describe and illustrate the interconnections and operation of the various software components.	All class and flow diagrams appear either in this report or in the technical documentation section.	Sub-section 2.1 Sub-section 2.2 Part 5
A software simulation platform that can verify the software components functionality independently.	The software simulation platform could illustrate the functionality of all the main software components, and allowed users to control and observe the program sequence.	Sub-section 3.2 Sub-section 3.4 Sub-section 3.5

Communication hardware that can facilitate communication between the software unit and the mechanical control unit (FU3.1).	The communication mechanism allowed the *Gantry System* and the *PC System* to communicate with one another successfully, and had a 100% success rate.	Sub-section 3.3 Sub-section 3.6 Sub-section 3.7
A microcontroller to run the firmware and interconnect the various mechanical control unit modules (FU3.2).	The microcontroller could successfully monitor and control the entire integrated electronic system.	Sub-section 3.3 Sub-section 3.6 Sub-section 3.7
Motor drivers that will convert microcontroller signals to adequate high-voltage and high-current outputs by means of a power supply unit (FU3.3).	The motor drivers successfully allowed the microcontroller to control all the system's stepper motors.	Sub-section 3.3 Sub-section 3.6 Sub-section 3.7
Electric motors capable of driving all the gantry mechanisms (FU3.4).	All the system's motors successfully facilitated piece manipulations, translations, and rotations.	Sub-section 3.3 Sub-section 3.6 Sub-section 3.7
Switches capable of monitoring the gantry system's movements, so as to prevent limit breaches (FU3.5).	The edge switches were successfully used to reset the gantry and to prevent limit breaching movements that would damage the system's hardware.	Sub-section 3.3 Sub-section 3.6 Sub-section 3.7
A power supply unit that can convert power from mains to the correct low voltage levels required by the electronic components (FU3.6).	The power supply unit could supply enough power to run all the electronic and electro-mechanical components.	Sub-section 3.3 Sub-section 3.6 Sub-section 3.7
Circuit diagrams of all the electronic hardware units.	All circuit diagrams appear either in this report or in the technical documentation section.	Sub-section 2.1 Sub-section 2.2 Part 5
An electronic module, capable of connecting and integrating all the electro-mechanical hardware units, constructed from discrete components.	The integrated electronic circuit could successfully monitor and control all the system's electro-mechanical components.	Sub-section 3.3 Sub-section 3.6 Sub-section 3.7
Firmware capable of monitoring and controlling all the electro-mechanical units.	The firmware successfully enabled the microcontroller to monitor and control all the system's electronic and electro-mechanical components.	Sub-section 3.3 Sub-section 3.6 Sub-section 3.7
A mechanical gantry system and puzzle piece manipulator (FU4).	All the piece manipulation hardware successfully facilitated puzzle piece manipulations, translations, and rotations. The final *Gantry System* had a piece-drop rate of only 0.04% per move.	Sub-section 3.3 Sub-section 3.6 Sub-section 3.7

Table 18.
Functional unit specifications

3.2 QUALIFICATION TEST 1: PIECE DETECTION

3.2.1 Qualification Test

Objective

The main objective of this test was to analyse the accuracy, robustness, and speed of the piece detection software component. Additionally, this test also also verified the functionality of the image acquisition unit (FU1). This test *did not* consider any overlapping piece configurations.

Equipment Used

The following equipment was used to perform this test:

- digital camera,
- PC,
- eight distinct 12-piece jigsaw puzzles, and
- software simulation platform.

Experimental Procedure and Parameters

The software simulation platform was used to monitor the performance of the piece detection component. This component was presented with 50 unique initial puzzle layouts containing the following:

- slightly varying lighting conditions,
- varying initial piece layouts, and
- foreign object present on the construction surface.

The accuracy was measured by tallying the amount of pieces that the system failed to detect. Conditions that caused the algorithm to fail were recorded, and the component's execution time for each initial layout was measured. A spreadsheet was created to track all results.

3.2.2 Results and Observations

Measurements

After the 50 initial puzzle layouts and conditions, the following measurements were made.

- Amount of pieces the component failed to detect: 4 pieces
- Average component execution time: 22.58ms

Statistical Analysis

As each initial layout contained 12 pieces and as there were 50 initial layouts, the 4 pieces which the system failed to detect, indicated a piece detection accuracy of 99.33% ($\frac{50\times12-4}{50\times12}\times100$).

Description of Results

The component had an exceedingly high level of accuracy, and could function in a wide verity of environmental conditions. The component could mostly detect and ignore foreign objects. The speed of the algorithm allowed the simulation platform to output results at a frame rate of 44 FPS ($\frac{1}{22.58}\times10^3$).

Observations

When foreign objects had dimensions very similar to that of a puzzle piece, the component would incorrectly detect them as puzzle pieces. When pieces were slightly outside of the puzzle construction boundary, the system would fail to detect them. In extremely low and high lighting conditions, the system would fail to detect pieces completely.

3.3 QUALIFICATION TEST 2: PIECE SEPARATION

3.3.1 Qualification Test

Objective

The main objective of this test was to analyse the efficiency of the piece separation strategy, and the various system components involved. This test aimed to verify the *PC System* and the *Gantry System's* ability to successfully separate overlapping piece clusters such that they can be individually detected. This test thus inherently also verifies the communication mechanism between the *Gantry System* and the *PC System*, and verifies the functionality of the integrated electronic system. Additionally, this test also measured the average speed to the piece separation process.

Equipment Used

The following equipment was used to perform this test:

- digital camera,
- PC,
- eight distinct 12-piece jigsaw puzzles,
- RS-232 serial-to-USB cable, and
- *PC System*.
- *Gantry System*.

Experimental Procedure and Parameters

Multiple different initial puzzle layouts, containing overlapping piece clusters, were presented to the system, with some initial layouts containing more than one overlapping piece cluster. These clusters consisted of 2 to 3 pieces and were arranged in a multitude of ways. As the piece separation scheme makes use of a feedback mechanism, it was decided to measure the amount of attempts the system needed to separate each cluster. Additionally, all failures were measured and the circumstances that lead to them were recorded. In total, 50 initial puzzle layouts were presented to the system. In total, these layouts cumulatively contained 75 overlapping piece clusters, of which 25 consisted of 3 pieces. Additionally, each communication failure was recorded, and the average separation time for each cluster was also recorded. A spreadsheet was created to track all results.

3.3.2 Results and Observations

Measurements

After the 50 cluster layouts were processed by the system, the following measurements were made.

- Average amount of attempts needed to separate a cluster: 1.49 moves
- Average cluster separation time: 9.33s
- Amount of times the *PC System* failed to find adequate space: 7 failures
- Amount of times the *Gantry System* completely failed to separate a cluster: 4 failures
- Amount of communication failures: 0 failures

Statistical Analysis

In total, the system failed on 11 $(7+4)$ attempts for a total of 75 clusters, which gives the system a separation likelihood of 85.33% ($\frac{75-11}{75} \times 100$). The system was also shown to have a flawless communication success rate, and a reasonable separation speed.

Description of Results

This component had reasonable accuracy, especially considering that it involved a lot of physical mechanical manipulations. Considering that 25 out of the 75 clusters consisted out of 3 pieces, a minimum of 100 moves were required to separate all the clusters. Thus the average number of attempts needed to separate pieces (1.49) is not unreasonable, considering some clusters needed 2 moves to be separated. The 100% communication accuracy clearly validates the success of the communication mechanism, and the overall functionality of the separation process also verifies the functionality of the integrated electronic system. The average cluster separation time of 9.33 seconds was low enough to comply with the system's time specification.

Observation

Sometimes when an initial layout contained two or more overlapping piece clusters, the system would separate and place a piece from the first cluster in such a way that it occupied all the free space, leaving no free space to which the remaining clusters could be separated. A similar case would seldom arise when separating the first piece in a cluster containing three pieces, leaving no room to which the second piece could be separated. Also, when two pieces were stacked on-top of each other such that the bottom piece was mostly hidden from the system, the algorithm would fail to detect the overlapping cluster entirely.

3.4 QUALIFICATION TEST 3: PIECE CLASSIFICATION

3.4.1 Qualification Test

Objective

This test aimed to verify the functionality and speed of the piece classification component. Additionally, this test would validate the successful integration of all preceding software sub-components. This test *did not* consider any overlapping piece configurations.

Equipment Used

The following equipment was used to perform this test:

- digital camera,
- PC,
- eight distinct 12-piece jigsaw puzzles, and
- software simulation platform.

Experimental Procedure and Parameters

The software simulation platform was used to monitor the performance of the piece classification component. The system was provided with 50 distinct initial puzzle layouts. The amount of misclassified pieces were tracked, along with the circumstances that caused them. Complete classification failures were also monitored, and the time it took to classify each of the initial puzzle layouts fully was recorded. A spreadsheet was created to track all results.

3.4.2 Results and Observations

Measurements

After the 50 initial layouts were processed by the system, the following measurements were made.

- Amount of misclassified pieces: 5 pieces
- Amount of failed classification attempts: 2 attempts
- Average component execution time: 218.10*ms*

Statistical Analysis

The system succeeded at correctly classifying 95.17% ($\frac{50\times12-2\times12-5}{50\times12}\times100$) of pieces, and only failed at 4% ($\frac{2}{50}\times100$) of its overall classification attempts.

Description of Results

When pieces were in very close proximity to each other, yet not overlapping, the system seldom failed to classify them correctly. This is owing to segments of these pieces making it into each other's piece images, which causes errors in the indent/tab/edge detection algorithm. This anomaly would sometimes cause the algorithm to detect too many or too few total straight edges, which caused the algorithm to fail. Additionally, the average execution time was extremely quick and complied with the system's time constraints.

Observations

As only two out of the total seven errors were complete classification failures, it can be deduced that the component is reasonably robust. Additionally, it can be concluded that all preceding software components function correctly, and were successfully integrated. It is also important to mention that the classification time includes that of the piece detection component, and thus represents the speed of the cumulative system up to, and including, the piece classification component.

3.5 QUALIFICATION TEST 4: SOFTWARE SOLUTION

3.5.1 Qualification Test

Objective

This test aimed to verify the accuracy, robustness, and speed of the software puzzle-solving component. This test also serves as a verification mechanism for the successful integration of all preceding software components.

Equipment Used

The following equipment was used to perform this test:

- digital camera,
- PC,
- eight distinct 12-piece jigsaw puzzles, and
- software simulation platform.

Experimental Procedure and Parameters

Once again, 50 distinct initial puzzle layouts were presented to the software simulation platform. The correctness and execution time of each solution was recorded. It is important to note that the initial layouts presented to the system in this test were not as severe as those presented to previous tests. A spreadsheet was created to track all results.

3.5.2 Results and Observations

Measurements

After the 50 initial layouts were processed by the system, the following measurements were made.

- Amount of correct solutions: 48 solutions
- Average component execution time: 584.95ms

Statistical Analysis

The system was found to have a software solution accuracy of 96% ($\frac{48}{50} \times 100$).

Description of Results

As the initial layouts did not contain as many extreme cases as they did in previous tests, the accuracy of preceding components were a lot higher and as a result, did not affect the accuracy of the puzzle-solving system. This test thus thoroughly assesses the puzzle-solving component and its integration with its preceding software components. The execution time of the system was acceptable as it falls well within the system's time constraints.

Observations

Some solutions were built with the puzzle's image upside-down. In order to correct this, advanced feature extraction techniques are required, which fall well outside the scope of a graduate engineering project. Owing to this, upside-down solutions were also considered successful. It is also important to mention that the software solution time includes that of the of the piece detection and piece classification component, and thus represents the speed of the entire *PC System*.

3.6 QUALIFICATION TEST 5: GANTRY ACCURACY

3.6.1 Qualification Test

Objective

The objective of this test is to asses the accuracy of the *Gantry System*. This test will also inherently validate the integrated functionality of the communication mechanism between the *Python* software and the *Gantry System*. Additionally, this test will also confirm that the integrated electronic system functions as intended.

Equipment Used

The following equipment was used to perform this test:

- digital camera,
- PC,
- *Gantry System*, and
- RS-232 serial-to-USB cable.

Experimental Procedure and Parameters

In order to test the accuracy of the *Gantry System*, 50 random pixel coordinates were chosen. A *Python* script was created, which instructed the *Gantry System* to move to these locations after their coordinates were passed through the linear conversion equations (pixel to step-count conversion equations) of **Movement Approach C** (as shown in Sub-sub-section 2.2.3). The script utilised the *Test* command, as described in Table 14, to move the gantry. The gantry's deviations from the desired pixel coordinates were then measured and recorded.

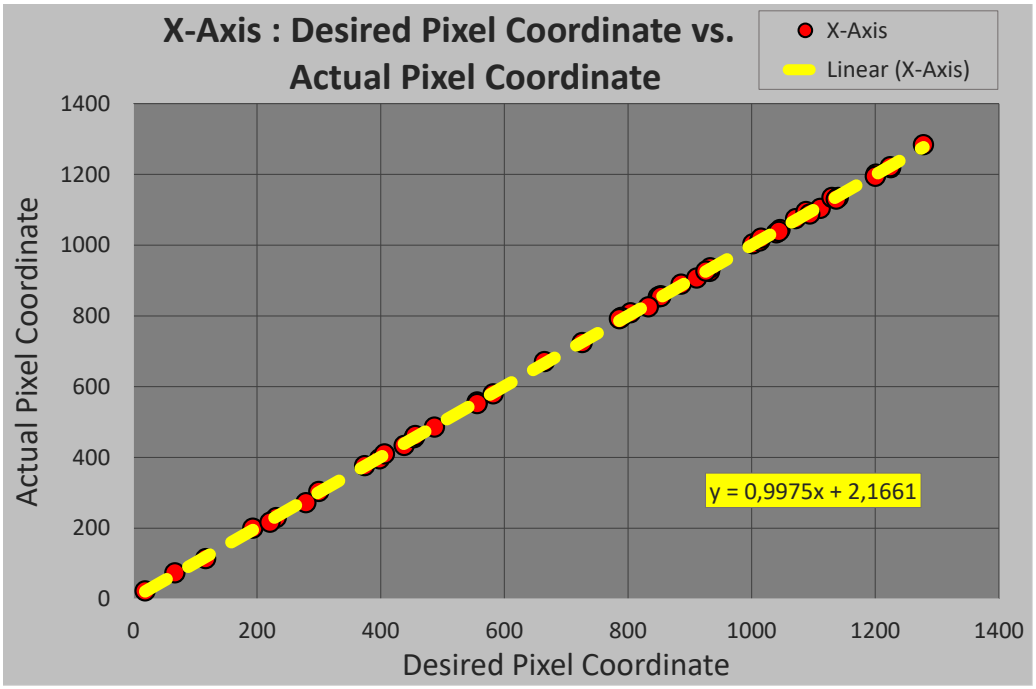
3.6.2 Results and Observations

Measurements and Graphs

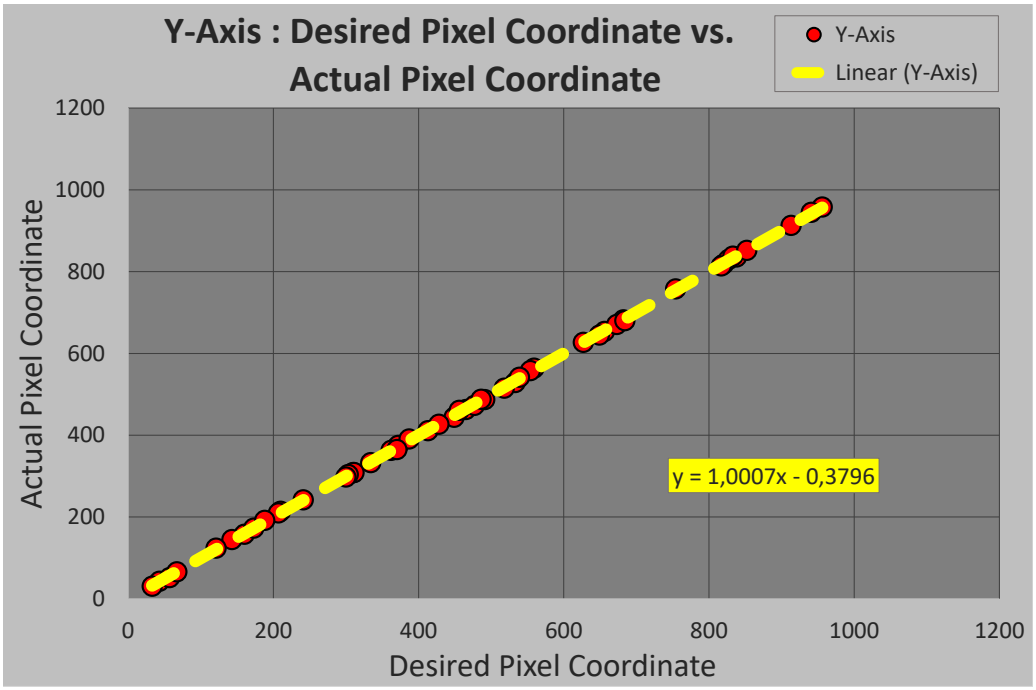
After 50 locations were tested with the linear conversion equations of **Movement Approach C**, the following deviations were measured from the gantry's desired pixel coordinates to its actual pixel coordinates.

- Average x-axis deviation: 3.9 pixels
- Average y-axis deviation: 2.72 pixels
- Maximum x-axis deviation: 12 pixels
- Maximum y-axis deviation: 10 pixels

Figure 56 below shows graphs of the actual pixel coordinates versus the desired pixel coordinates for the *x*- and *y*-axis of the system. The trend line formula for each graph is shown in the yellow boxes.



(a)
Gantry accuracy: x-axis



(b)
Gantry accuracy: y-axis

Figure 56.
Gantry accuracy

Statistical Analysis

By using the inverse of the linear conversion equations of **Movement Approach C**, the pixel deviations were converted to step-count deviations, which allowed this test's results to be compared to that of the initial accuracy test (as described in Sun-sub-section 2.2.3). After applying this process, the following results were obtained.

- Average x-axis deviation: 6.8 steps (3.9×1.7495)
- Average y-axis deviation: 4.8 steps (2.72×1.7732)
- Maximum x-axis deviation: 20.9 steps (12×1.7495)
- Maximum y-axis deviation: 17.7 steps (10×1.7732)

This lead to the following average results.

- Average deviation: 5.8 steps ($1.1mm$)
- Maximum deviation: 19.3 steps ($3.6mm$)

When comparing these results to that of the initial accuracy test for **Movement Approach C** (average deviation: 5 steps, maximum deviation: 22 steps), it can be seen that these results are all within 15% of each other. This further strengthens the results of this test.

Description of Results

Ideally, the trend line formulas show in Figure 56 should tend towards $y = x$. This would indicate that there are no deviations from the system's actual coordinates to its desired coordinates. The actual formulas, however, were shown to be:

x-axis equation:

$$y = 0.9978 \times x + 2.1661 \tag{3.1}$$

y-axis equation:

$$y = 1.0007 \times x - 0.37961$$

The gradient of each of these formulas were within 0.22% of the ideal unity gradient, which once again confirms that the calibration mechanism was extremely successful. The formulas each had an unwanted linear offset of 2.1661 and −0.3796 pixels respectively. If these numbers were significantly large, one could adjust the linear offset of the linear conversion equations in order to correct this error. The linear offsets, however, are extremely small and would have no visible effect on the system's accuracy. It can thus be concluded that the overall system's accuracy is well within its specification, and that the various system components all function together as intended.

Observations

The y-axis was shown to be slightly more accurate than the x-axis. This could most likely be attributed to the fact that the y-axis is 33% shorter than the x-axis, and thus requires fewer total displacements. Throughout this test, all the *Gantry System's* electronics remained at safe temperatures, and no dangerous electronic events (such as shorts) occurred.

3.7 QUALIFICATION TEST 6: PUZZLE CONSTRUCTION

3.7.1 Qualification Test

Objective

This test aimed to assess the overall system's puzzle construction speed, accuracy, and robustness. Additionally, this test will inherently validate the integrated functionality of the *PC System* and the *Gantry System*, which, most importantly, verifies the entire system's functionality. This test thus serves as the final verification mechanism for all the system's mission critical specifications. (Note: this test and its results have already briefly been described in Sub-sub-section 2.2.3)

Equipment Used

The following equipment was used to perform this test:

- digital camera,
- PC,
- three distinct 12-piece jigsaw puzzles,
- RS-232 serial-to-USB cable,
- *Gantry System*, and
- *PC System*.

Experimental Procedure and Parameters

In order to assess the accuracy of a constructed puzzle, a solution-scoring system was created. For each piece that was in the correct position and perfectly interlocked, the solution gained one point. After a solution is completed, its total accumulated score was then divided by its puzzle size in order to get an accuracy percentage. If a 12-piece puzzle was built perfectly, it would score 12 points and thus 100%. If a 12-piece puzzle solution had 9 pieces correctly placed and interlocked, it would score 75%. Additionally, the construction time of each solution was recorded. To ensure that the results were not selective towards a specific puzzle, three different 12-piece puzzles were used interchangeably whilst scoring solutions. The system was once again provided with 50 initial puzzle layouts. Ten of these layouts contained overlapping piece clusters consisting out of two pieces each, and five of these layouts contained overlapping piece clusters consisting out of three pieces each. The number of moves each puzzle construction required was recorded (it is important to note that each piece rotation manoeuvre consisted of two moves). Additionally, the average construction time for each initial layout was measured via the system's GUI. A spreadsheet was created to track all results.

3.7.2 Results and Observations

Measurements

After 50 initial puzzle layouts were constructed and scored, the following results were obtained.

- Average solution score: 97.44%
- Average solution time: 210.46s
- Average amount of moves per solution: 48.15 moves
- Amount of perfect solution constructions: 37 constructions
- Total amount of dropped pieces: 1 piece

Statistical Analysis

As there were 50 solution constructions, which each performed 48.15 moves on average, the entire test cumulatively performed 2,407.50 moves. Because only one piece was dropped during the entirety of this test, it can be seen that the system had a piece drop likelihood of 0.04% ($\frac{1}{2407.5} \times 100$) per move, and 1.9% ($0.04\% \times 48.15$) per solution construction.

Description of Results

The average solution time of the system was well within that of the mission critical specification. It can be seen that the system would construct a perfect solution 74% ($\frac{37}{50} \times 100$) of the time, and solutions would on average contain 11.69 ($\frac{97.44}{100} \times 12$) correctly interlocked pieces. This complies with the accuracy requirements of the mission critical specification.

Observations

The system managed to separate all the overlapping piece clusters that were presented to it successfully, and still managed to construct successful solutions despite thereof. This test thus verified the system's overlapping piece separation capabilities, its most unique aspect.

4. Discussion

4.1 INTERPRETATION OF RESULTS

The system's main objective was to solve and build a 12-piece, traditionally shaped, jigsaw puzzle, the initial jumbled layout of which contained overlapping pieces. Various quantification experiments were designed to test all of the system's main components, and to quantify their performance. Overall, the final system and all of its core components performed exceptionally well. All the results indicated that the system's specifications were met, and that all its sub-components were successfully integrated.

Despite some of the system's components displaying minor errors, it is important to recall that the majority of these components (especially the software components) are controlled by a feedback mechanism. The software system controller directs all the various software components, such that if a component should produce an unsuccessful output, the controller would simply reacquire a new image and reattempt all its processes until all of its components produce successful outputs. Similarly, the piece-separation and the piece-rotation components also partially make use of feedback control, such that if these components would fail, they will simply repeat their functionality until they are successful. Thus, if any of the abovementioned components should fail, the system would be able to correct itself and still produce a successful output. This robust design choice thus greatly contributed to the overall accuracy and consistency of the system.

4.1.1 Piece Detection

The key results that were ascertained from the piece-detection test are listed below.

- Piece detection likelihood: 99.33%
- Average component execution time: 22.58ms

The piece detection component, and the functions pertaining to it, worked well as a whole, as it could quickly, accurately, and reliably detect physical puzzle pieces. This component could function in varying lighting conditions and could accurately detect and ignore foreign objects on the puzzle-construction surface. The use of the HSV colour space and a thorough background colour calibration technique, allowed the piece-detection component to be extremely robust and accurate. Choosing to use the HSV colour space over the RGB colour space was thus a good design choice. All the components that depend on the piece detection component will strongly benefit from its accuracy and robustness.

The piece detection component required the software simulation platform and the digital camera to function correctly. The successful results of this test thus inherently verified the functionality of the software simulation platform and the digital camera.

4.1.2 Piece Separation

The main results obtained from the piece separation test are listed below.

- Piece separation likelihood: 85.33%
- Average cluster separation time: 9.33*s*

This component performed well and had reasonable accuracy and speed, especially considering that it involved a lot of physical (mechanical) piece-manipulations. It could separate overlapping piece clusters with good speed and reliability. This is mainly owing to the effectiveness of the gantry calibration mechanism and **Movement Approach C** (as discussed in Sub-sub-section 2.2.3). The utilisation of the separation command, as described in Table 14 in Sub-sub-section 2.2.2, allowed this component to separate piece clusters successfully even when the actuator grabbed the bottom piece in the cluster. The addition of this separation command into the original command set was thus a good design choice.

The ability to handle initial puzzle layouts, which contain overlapping piece clusters, is one of this project's main focuses. The accuracy and robustness of this component thus contributes greatly to the overall success of the system. The piece-separation component required the *Gantry System* and the *PC System* to function correctly. The successful results of this test thus inherently verified the functionality of the *Gantry System* and the *PC System*, as well as that of the communication mechanism between them.

4.1.3 Piece Classification

The most relevant results obtained from this test are listed below.

- Piece classification likelihood: 95.17%
- Average component execution time: 218.10*ms*

This component proved to perform extremely well, as it could quickly, accurately, and reliably detect and classify physical puzzle pieces. This was partially owing to the success of all preceding software components. There were however a few minor errors, which were mainly caused by puzzle pieces being too close to one another in the initial jumbled puzzle layout. This would cause segments of these pieces to make it into each other's piece images, which could cause errors in the indent/tab/edge detection algorithm. Luckily, the piece separation component could find adequate spacing most of the time, which reduced the likelihood and severity of these errors.

This component plays a vital part in the puzzle-solving process, as it gathers and stores information about puzzle pieces. If this component was to classify an aspect of a piece incorrectly, the puzzle-solving component would be supplied with false information. It is thus very important that this component functions as well as possible as it has a direct influence on the final goal of the system.

The successful results of this test also inherently verifies the successful integration between the piece detection and the piece classification components.

4.1.4 Software Solution

The key results that were ascertained from the software solution test are listed below.

- Average solution accuracy: 96%
- Average component execution time: 584.95*ms*

This component performed very well as it could provide a quick and reliable software solution to a jumbled puzzle layout. This test assesses most of the system's software components as this component uses the cumulative outputs from most preceding components. This test thus shows that the entire system's software components have been successfully integrated, with the exception of the movement calculation component. The success of this component can thus partially be attributed to the success and accuracy of preceding software components. The decision to use the alternate solution-scoring technique over the initial one (as discussed in Sub-sub-section 2.2.1) also added to the success of this component and can thus be seen as a good design choice.

As mechanical robotic systems are generally not extremely accurate, it is very important to ensure that the software component is as accurate as possible. If software errors are present, it would only contribute to even larger mechanical errors. As this component provides extremely accurate solutions, it will aid the accuracy of not only the *Gantry System*, but the entire system as a whole.

4.1.5 Gantry Accuracy

The most important results pertaining to the gantry accuracy are listed below.

- Average deviation: 5.8 steps (1.1*mm*)
- Maximum deviation: 19.3 steps (3.6*mm*)

The *Gantry System* had reasonable accuracy, but was not perfect. The main cause of these errors was found to be minor start-stop jolts caused by the gantry's movements. An alternative strategy, that could partially remedy this error, would be to vary the speed of the *Gantry System* based on the distance it has travelled and the distance to its destination. To better illustrate this, Figure 57 below shows the gantry's speed as a function of the distance to its destination; here D_0 is the starting distance from its destination.

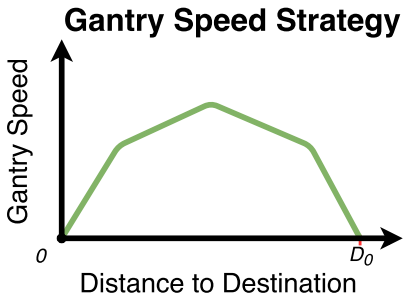


Figure 57.
Alternate gantry speed strategy

This approach would thus greatly decrease start-stop jolt and, as a result, increase the *Gantry System's* accuracy. The current accuracy of the *Gantry System*, however, was found to be adequate enough to facilitate puzzle-construction. This, along with time and laboratory access constraints, lead to this alternate approach not being implemented.

The accuracy of the gantry system inherently verifies the integrated functionality of the entire electronic system and its various components. Throughout this test, all the *Gantry System's* electronics remained at safe temperatures, and no dangerous electronic events (such as shorts) occurred.

4.1.6 Puzzle Construction

The main results obtained from the puzzle construction test are listed below.

- Average solution score: 97.44%
- Flawless solution likelihood: 74%
- Average solution time: 210.46*s*

These results are extremely good, especially since they verify that the entire system's main goal has been achieved. The results of this test imply that roughly three out of every four puzzles will be solved and built perfectly, and those that are not perfect will most likely only have one piece that is not perfectly interlocked. The average solution time of the system is well within that of the mission critical specification, which can be attributed to the speed of *PC System's* and the agility of the *Gantry System*.

The average solution time is also well within the system's specification. This test thus verifies the success of the overall system, and its various components.

4.2 IMPROVABLE ASPECTS

Despite the overall system's success, there are still a few factors which can be improved. As it stands, all the background and piece shape parameters have been carefully measured and hard coded into the system. As such, the system can only function in a specific environment and with specific piece dimensions, without the need for software alterations. To fix this, an automatic background and piece dimension calibration mechanism can be added. This will allow the system to be even more robust.

One of the system's strong points is that it can detect and correct errors from many of its components by means of feedback control. There are, however, a few components that are not monitored by feedback control, and as such, cannot be corrected by the system. The system has no way of monitoring the gantry's position, and it cannot detect when the gantry incorrectly drops pieces. By utilising the system's camera or electro-mechanical components, a feedback mechanism for these aspects can be created, which would greatly improve the reliability of the system. A similar feedback mechanism can also be created to verify and correct final puzzle solutions.

As mentioned in Sub-sub-section 4.1.5 above, the *Gantry System's* accuracy can be improved by using an acceleration-based movement strategy as opposed to the current constant speed strategy. This would decrease start-stop jolt, which causes minor deviations in the system.

4.3 STRONG POINTS

Overall, the system performed very well. It could solve and build a jumbled puzzle with good speed and accuracy, which can be attributed to some of its strong points. As mentioned above, the system utilises a few feedback control mechanisms to detect and correct errors from many of its components. This enables the system to mitigate errors, which adds to its robustness and reliability.

The system's GUI, despite not being a key component, offers and excellently streamlined software experience. It is simple to use and clearly illustrates all the system's operations in a convenient manner.

4.4 SYSTEM FAIL CONDITIONS

A few of the system's components are not monitored by feedback control, and as such, cannot be corrected by the system. After the piece rotation phase, no more feedback control is applied to the system; thus, if errors or inaccuracies were to occur after this phase, the system would have no way of detecting and correcting them. If the system accidentally dropped a piece during the temporary stacking or construction phase, it would remain unaware of this. If such a piece were to fall on top of another piece or inside the construction area, it would cause uncorrectable errors down the line.

Even though feedback control is applied during the piece rotation phase, the system would not be able to detect if a piece fell on top of the piece rotator. This would cause all other piece rotations to possibly suffer from this, which could lead to a unwanted cascade of undetectable and uncorrectable errors.

4.5 DESIGN ERGONOMICS

All the system's electronics are sealed in a compact electronic enclosure. This enclosure is neatly connected to all of the *Gantry System's* components, but can be easily disconnected and reconnected at the user's leisure; which makes it hassle-free for users to interact with the unit. This enclosure has connector ports for a power cord (kettle cord) and for a serial cable, which makes it easy to set up and connect to the *PC System*. The enclosure has a power switch which allows users to control the power to the system without having to connect or disconnect the power cable.

The GUI and the *PC System* can be opened by simply double clicking on the main *Python* file, thus the system does not require any installation. The GUI offers and excellently streamlined software experience. It is simple to use and clearly illustrates all the system's operations in a convenient manner. The GUI's buttons, displays, and tabs are all laid out and labelled logically, such that the system can be controlled and monitored without the need for user tutorials.

The various electronic and hardware components of the system can easily be interchanged if they should fail. The component sockets are laid out such that connectors cannot be connected the wrong way around, which makes it less confusing and frustrating for users to maintain the system.

4.6 HEALTH AND SAFETY

The enclosure seals all the systems electronics and is partially water-resistant. This protects the components from environmental damage but, most importantly, prevents users from coming into contact with dangerous components. This protects users from possible burns or electrical shocks. All the system's electronics are grounded to the the power supply, which provides the entire system with earth leakage protection. The enclosure also has an air vent with a mesh filter. This gives the power supply's fan access to cool air whilst still protecting the system from dust and other small particles.

The gantry's movements are limited by software and by the limit-monitoring switches. These two mechanisms prevent the gantry from hitting the edges, which protects its hardware. That being said, the gantry has no way of detecting foreign obstructions, such as a user's limbs. Care must thus be taken to ensure that users are aware of these dangers.

4.7 SOCIAL AND LEGAL IMPACT

This project focused on robotic automation by means of CNC and image-processing techniques. These techniques can be adapted to fit industrial problems. Large-scale industrial automation has many benefits in terms of efficiency and profit, but also leads to job loss. The social and legal impacts of this thus have to be carefully considered.

If the puzzle-solving robot were to be packaged and sold as a toy, it would be profitable and it would have an educational impact on its consumers. In the technology-driven era that we live in, it is always good to get people interested in technology and science from a young age. This product could thus serve a fun and entertaining way to spur interest in technology. Before the product can be sold, however, great care must be taken to ensure its customers' safety. The legal implications of such a product also needs to be investigated in order to protect manufacturers and customers best.

4.8 ENVIRONMENTAL IMPACT

As with most electronic systems, this system is not easily recyclable. That being said, this system only contains a few electronic components. These components operate under safe and stress-free condition, which would increase their lifespan and reduce the overall product's environmental footprint. Most of the system's electronics are modular and can be individually replaced. This means that if a certain component should fail, only that component needs to be replaced, which also reduces the overall product's environmental footprint. The system draws at most 40 Watts of power, and is thus reasonably energy efficient.

5. Conclusion

By analysing the results and performance of the final system, it can be concluded that the design and implementation choices made throughout this project indicate sound engineering logic and reason. The system and its sub-components were designed and build efficiently and effectively, which contributed to the overall success of the system.

5.1 SUMMARY OF WORK

The main goal of this project was to design and implement an intelligent CNC robotic system, capable of autonomously solving and building a 12-piece, traditionally shaped, jigsaw puzzle, even when the jumbled puzzle contained overlapping pieces.

The core tasks pertaining to this project are listed below.

- A literature survey on existing automated puzzle-solving techniques was completed.
- An algorithm was developed from first principles in *Python*, which could autonomously detect and classify jumbled jigsaw puzzle pieces.
- An algorithm was developed from first principles in *Python*, which could autonomously create a software solution to a jumbled jigsaw puzzle.
- A software system controller was developed from first principles in *Python*, which could detect and correct errors within the software system.
- A software simulation platform was developed from first principles in *Python*, which could test and verify the functionality of the system's main software components.
- A user interface was developed in *Python* with the aide of *QtDesigner*. The user interface's functionality was designed and developed from first principles.
- The system's hardware, which housed a microcontroller at its core, was designed and constructed from first principles.
- *C* code for the microcontroller was developed from first principles.
- An algorithm was developed from first principles in *Python*, which could translate a software puzzle solution into into physical puzzle solution. This was achieved by utilising a serial communication link between the systems's software and hardware units.

5.2 SUMMARY OF OBSERVATIONS AND FINDINGS

The final system achieved its main goal with exceptional accuracy and consistency. All the system's auxiliary specifications were met, and each functional unit performed as intended.

The key observations about the system and its results are listed below.

- The system could autonomously solve and construct a 12-piece jigsaw puzzle in 210 seconds on average, with an accuracy of 97.44%.
- The system could separate overlapping piece clusters, consisting of up to three pieces, and still construct a successful solution. The system had a piece separation likelihood of 85.33%.
- The final system had an average displacement error of 1mm and a maximum displacement error of 4mm.

5.3 SUGGESTIONS FOR FUTURE WORK

Image feature detection is a useful and interesting topic worth appending to this project. Once a software solution image is available, an image feature detection technique can be applied to it in order to detect features within the puzzle's picture. This will allow the system to correctly orient each solution it constructs with regards to the puzzle's picture, which the current system cannot do.

Image feature detection by itself can be extremely complex. To remedy this, the system could limit itself to only a few key features (such as an animal, building, or vehicle) and the orientations thereof. This will make the task of feature detection more manageable and realistic in terms of the overall scope of the project.

The following lists two possible approaches to feature detection.

1. The system could make use of an artificially intelligent technique (such as a neural network or a hidden Markov model) to detect features. This will add a machine-learning component to the project.
2. The system could make use of a web-query technique (such as Google image search) to detect features. This will add a web-integration component to the project.

These approaches are merely suggestions, but they show that feature detection could add useful and interesting components to into the project's scope.

6. References

[1] M. R. Maire, "Contour detection & image segmentation," Ph.D. dissertation, University of California, Berkeley, 2009.

[2] G. Paikin and A. Tal, "Solving multiple square jigsaw puzzles with missing pieces," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015, pp. 4832–4839.

[3] D. J. Hoff and P. J. Olver, "Automatic solution of jigsaw puzzles," *Journal of mathematical imaging and vision*, vol. 49, no. 1, pp. 234–250, 2014.

[4] N. Erell and R. Nagar, "Robotic jigsaw puzzle solver," Ben-Gurion University of the Negev, Tech. Rep., 2012.

[5] N. Aggarwal and W. C. Karl, "Line detection in images through regularized hough transform," *IEEE transactions on image processing*, vol. 15, no. 3, pp. 582–591, 2006.

[6] D. Pomeranz, M. Shemesh, and O. Ben-Shahar, "A fully automated greedy square jigsaw puzzle solver," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 9–16.

[7] D. J. Hoff and P. J. Olver, "Extensions of invariant signatures for object recognition," *Journal of mathematical imaging and vision*, vol. 45, no. 2, pp. 176–185, 2013.

[8] Draw.io. (2015) version 5.0.3.7. 20-22 Wenlock Road, London, UK. [Online: accessed 21-July-2016]. [Online]. Available: https://www.draw.io/

[9] Wikipedia. (2016) HSL and HSV — Wikipedia, The Free Encyclopedia. [Online: accessed 21-July-2016]. [Online]. Available: https://en.wikipedia.org/wiki/HSL_and_HSV

[10] ——. (2016) Image Moment — Wikipedia, The Free Encyclopedia. [Online: accessed 26-July-2016]. [Online]. Available: https://en.wikipedia.org/wiki/Image_moment

[11] OpenCV. (2016) Hough Line Transform. [Online: accessed 26-July-2016]. [Online]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

Part 5. Technical Documentation

This main report is supplemented with technical documentation. This provides more detail on the software that was used in the experiments, including program listings, a user guide and circuit designs. This section appears on the DVD that accompanies this report.

The DVD is organised into the directories listed below:

Author
Software
Firmware
Flow Diagrams
Class Diagrams
Project Photos
Photos Videos
Datasheets