MECH 423 Final Project Proposal

Liam Foster and Willem Van Dam

40199382 & 33500646

1. Objectives

The objective of our final project is to design and build a physical chess board that moves the opposing side's pieces automatically and plays against the player. It will consist of:

- The 8x8 chess board
- The 32 chess pieces with magnets embedded in the base of each piece
- A 2-axis gantry under the board for moving the pieces
- 2 stepper motors for moving the gantry
- An electromagnet mounted on the gantry to grip the chess piece
- A camera mounted above the board to detect piece position and movement
- A chess AI that plays against the player

2. Rationale

We find the challenge of sensing and moving the chess pieces an interesting and unique design conundrum.

3. List of Functional Requirements

The functional requirements of the chess board include

- Mechanical design of the system, including the chess board itself, the chess pieces, the 2-axis gantry, and the camera mount. The chess pieces and camera mount will be wooden, the board may be wooden or plastic depending on tests, and the gantry will be made from aluminum extrusion.
- Stepper motor control for moving the pieces with the 2-axis gantry. This includes the motor driver circuit and control firmware.
- Electromagnet control for gripping the pieces. This includes the circuit and the code.

- Chess piece position sensing with the camera. This will be accomplished using bitmapping.
- Chess control software and system integration. A chess AI library or API will be found and outputs of computer moves will be converted to XY locations for the electromagnet to move

Table 1 - List of Functional Requirements

Functional Requirements	**% Effort**	**Responsible Person**
Mechanical design (2-axis gantry, chess board, camera mount)	15%	Liam
Motor control of the gantry	20%	Liam
Gripping/engaging pieces	10%	Willem
Sensing piece position w/ camera	25%	Willem
Chess control software & system integration	30%	Liam + Willem

4. Functional Requirement #1: Mechanical Design

The mechanical design of the chess board will include the chess board itself, the magnetic chess pieces, the 2-axis gantry and the camera mount.

4.1. Approach and Design

The objective of the mechanical design is to allow the system to operate smoothly. Friction and complexity should be minimized.
There must be enough space between pieces on the board for the pieces to move between each other, especially for movement of the knights.
The pieces will be uniquely coloured to aid with image processing. This is explained in more detail in section 7.
The gantry extrusions will be screwed to the bottom of the board with L-brackets. The electromagnet will be mounted to the rolling base provided in the lab 3 kit. The rolling base will be actuated with the stepper motors, timing belts and pulleys provided in the lab 3 kit. Longer timing belts will be purchased as necessary

The magnets will be embedded in the base of the chess pieces. A friction reducing material will be glued or taped to the base of the pieces to reduce the magnetic force required to move the piece. Potential materials include PTFE tape and UHMW tape. The camera will be mounted above the board with a custom wooden frame. Wood for the frame will be sourced from discarded pallets.

4.2. Inputs and Outputs

The only real input and output from the mechanical system is the torque required by the stepper motors. This will be affected by the friction between the chess piece and the board, the tension in the belt and the resistance of the bearings.

4.3. Parameters

Parameters to be adjusted in the mechanical design include

- The size of the pieces - this will affect piece movement
- The size of the tiles on the board - this will be adjusted in tandem with piece size to ensure the pieces can pass between each other.
- The thickness/material of the chess board - this will impact the strength of the magnetic field. The thickness of the chess board will be optimized by testing the magnetic field strength through the chess board.
- The material on the base of the chess pieces - this will affect the friction between the pieces and the board, and therefore the torque required by the stepper motors.
- The height of the camera mount from the board - this will determine whether the whole board can be captured by the camera. This will be adjusted by checking the output of the camera while adjusting the height. The frame will be designed with multiple levels of mounting.

4.4. Development Plan

1. Source chess pieces
2. Choose board area and tile size based on chosen chess pieces
3. Test camera field of view and determine camera mount height
4. Make detailed, dimensioned CAD model of board with gantry and camera mount
5. Order additional aluminum extrusion, L-brackets, timing belts, low friction material, fasteners etc.

6. Test electromagnet strength with different thicknesses of wood (included in section 6) and choose board thickness
7. Make board and camera frame from abandoned pallet
8. Assemble system once the materials arrive

4.5. Test Plan

1. --
2. Make a mock board out of cardboard to test piece spacing.
3. Establish live feed between the camera and a laptop and then adjust and measure the height above the mock board until the whole board is within the field of view.
4. Double check and compare dimensions of real components to those in CAD model.
5. --
6. See section 6 for the electromagnet test plan. It will consist of testing different board thicknesses and seeing at what thickness the electromagnet can no longer move the chess piece.
7. Make rough assembly between board and camera mount to ensure the right height

5. Functional Requirement #2: Motor Control of Gantry

The stepper motors will control the movement of the gantry, which will in turn control the movement of the AI’s chess pieces. This FR includes the wiring of the motors and the firmware to control the motors.

5.1. Approach and Design

The stepper motors will be controlled using the closed-loop positioning system determined in Lab 3 Exercise 5. One stepper motor will be connected to each motor driver circuit on the Lab 3 PCB. As the board can only control one stepper motor at a time, the x and y position will be controlled sequentially.
A multiple-byte packet structure consisting of lateral and longitudinal positions will control the gantry destination through the UART. The velocity will be constant and set by the firmware. After moving the pieces for the chess AI, the firmware will send a byte

over the UART telling the chess control software that the piece movement is complete. After the move the motors will return the gantry to the origin in the lower left of the board (from the AI's perspective).

5.2. Inputs and Outputs

The inputs of the system are the lateral and longitudinal positions sent by the chess control software through the UART. The packet will consist of a start byte (255), 2 bytes for the x-position, 2 bytes for the y-position, and an escape byte. The x and y position values will be scaled to the dimensions of the board to give a small resolution. This will aid the positioning when the grabber is attempting to engage a chess piece.

5.3. Parameters

The scaling parameters for the x and y positions will be adjusted to the dimensions of the board through testing. The stepper motor speed will be set in the firmware and adjusted through testing so that it's slow enough to move the pieces.

5.4. Development Plan

1. Assemble stepper motors on the Lab 3 gantry while waiting for real board components
2. Create x-y grid corresponding to the Lab 3 gantry with the motor encoders
3. Firmware that will control the motors to step to UART command XY locations
4. UART output command that will tell the C# script that the motion is complete

5.5. Test Plan

1. Test the motor control with the packets to see if the motors turn in the right directions
2. Test that the motor encoders results are converted to accurate XY locations by temporarily sending encoder outputs over UART and inspecting in C# windows form
3. Input expected UART packet and see if motor goes to correct X or Y location
4. Test that correct UART packet is sent when the motor has moved to the correct location

6. Functional Requirement #3: Gripping/Moving Pieces

6.1. Approach and Design

The strategy for gripping and moving the pieces is to use an electromagnet that will engage and disengage with the magnets mounted to the bottom of each piece. This will be done by converting a digital HIGH or LOW signal to a higher current input that turns on the electromagnet. The electromagnet will be mounted to the plate that the 2 axis gantry moves.

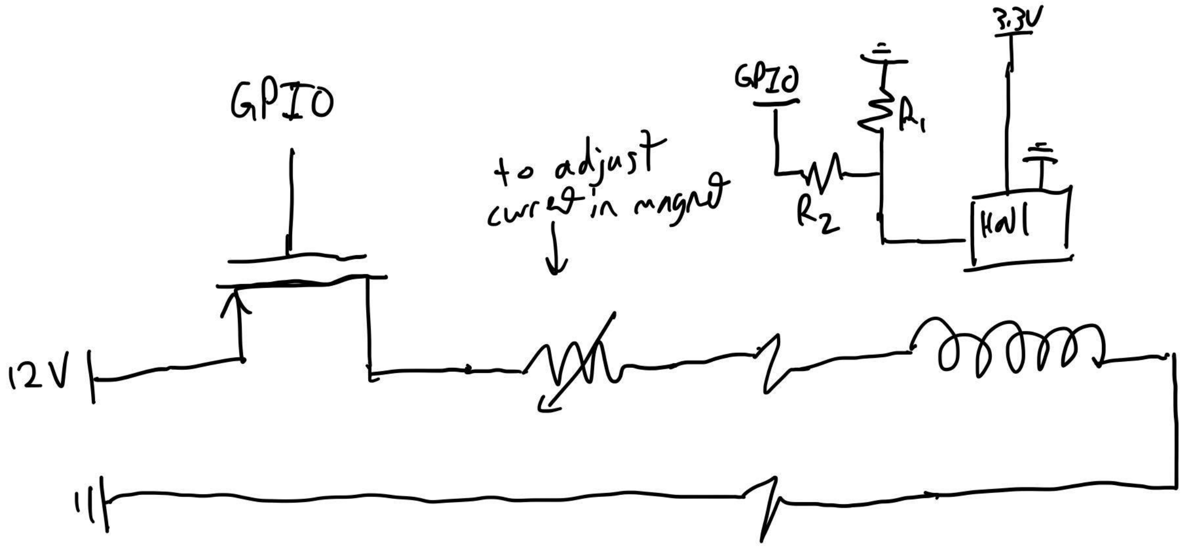

Figure 1. Simple Circuit Diagram for Electromagnet

The circuit diagram will look something like the Figure above with the electromagnet being the inductor. The electromagnet will have a ferromagnetic core to amplify the magnetic force and will be activated using the digital GPIO from the MSP430 on the gate pin of the mosfet. A hall sensor will be used to detect when the electromagnet has locked in with a permanent magnet on the bottom of the chess piece. This will allow us to know whether we have correctly located a piece or not. The R1 and R2 resistances will be chosen so that the difference in magnetic field from the electromagnet without a PM attached and the magnetic field of the electromagnet with the PM attached will drop the voltage below the threshold for the digital input of the GPIO.

6.2. Inputs and Outputs

The inputs for the electromagnet system are the 12V and 3.3V power supply, and the GPIO pin that switches the mosfet. The input GPIO pin will be turned HIGH when the magnet is where the piece is expected to be and stays high until the OUTPUT of the system (the Hall Sensor GPIO in pin changes indicating locking in with the magnet piece. The motor control will also need to move around a threshold of values in the X and Y direction until the piece locks in.

6.3. Parameters

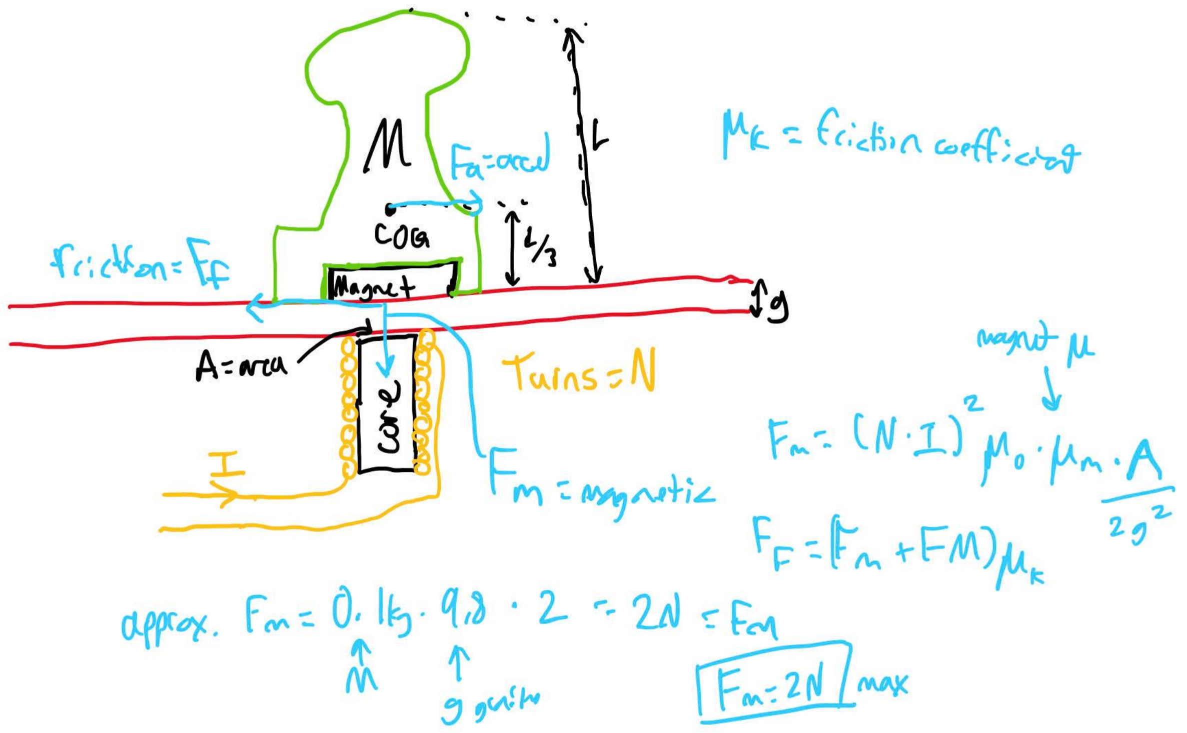

Figure 2. Rough Force Calculations for Electromagnet

Based on the rough force calculations the number of turns and current can be calculated using the 2N magnetic force. The 2N was approximated by assuming it would take only 2g to hold the piece down from tipping while moving it across the board. This can be easily adjusted later by either modifying the turns in the electromagnet, modifying the friction between the piece and the board, or changing the value of the potentiometer in the circuit design above to change the current supplied to the electromagnet. The values for the turns, current, and area of the electromagnet can be determined once the thickness of the board and the material for the magnets are chosen.

The parameters for the hall sensor will be determined after the magnetic field can be calculated. When purchasing the hall sensor it needs to be specced for the max magnetic field generated by the electromagnet and the voltage divider values can be calculated to make the field with the magnet engaged and the field with the magnet not engaged difference around the threshold of the digital input of the pin. As a contingency if this range is too small to be picked up by the digital input pin an amplifier can be used to increase the difference so that it can be detected by the input pin. If this ends up being too expensive or too complicated the analog input can be used for the output of the hall sensor instead of the voltage divider and the threshold can be adjusted according to that input instead.

6.4. Development Plan

1. Calculate the electromagnet specs after the mechanical design has been finalized. Also estimate the hall sensor specs
2. Build the activation circuit for the electromagnet and validate that it can engage with the piece on the other side of the board.
3. Test the output of the hall sensor using an oscilloscope when engaged and disengaged with the magnet
4. Spec the resistors in the voltage divider to use digital input for hall sensor input. If this does not work use the analog input or an amplifier
5. Write the firmware to move the electromagnet in the x and y directions until the magnet has been engaged

6.5. Test Plan

1. Calculations and part datasheets should validate whether the numbers seem accurate
2. Test switching on and off the electromagnet. Then test engaging the piece on the other side of the board material, can slide electromagnet with hand if 2 axis gantry isn’t finished yet. Use accelerometer (or eyeball) the accelerations capable with different potentiometer values to see if they are acceptable, if not modify the windings of the electromagnet or the friction and thickness of the board.
3. Take oscilloscope measurements of hall sensor output while moving over magnet on other side of the board.
4. Check if difference between outputs in step 3 are enough to safely go past a transition of a digital input pin. If that works then choose a voltage divider. If it doesn’t use an analog input to get better resolution.
5. Test firmware to move throughout entire area of square until hall sensor triggers. Make sure that current location is still being tracked. It should be able to find a piece anywhere within the playable square and stop once it has engaged.

7. Functional Requirement #4: Sensing Piece Position w/ Camera

7.1. Approach and Design

The general design for this element of the product is to use a webcam mounted above the board that can take images to bitmaps of the board. In addition to the webcam there needs to be a UI interface and a button for input used to indicate when the player is done taking their turn. The general flow of this component will be as follows.

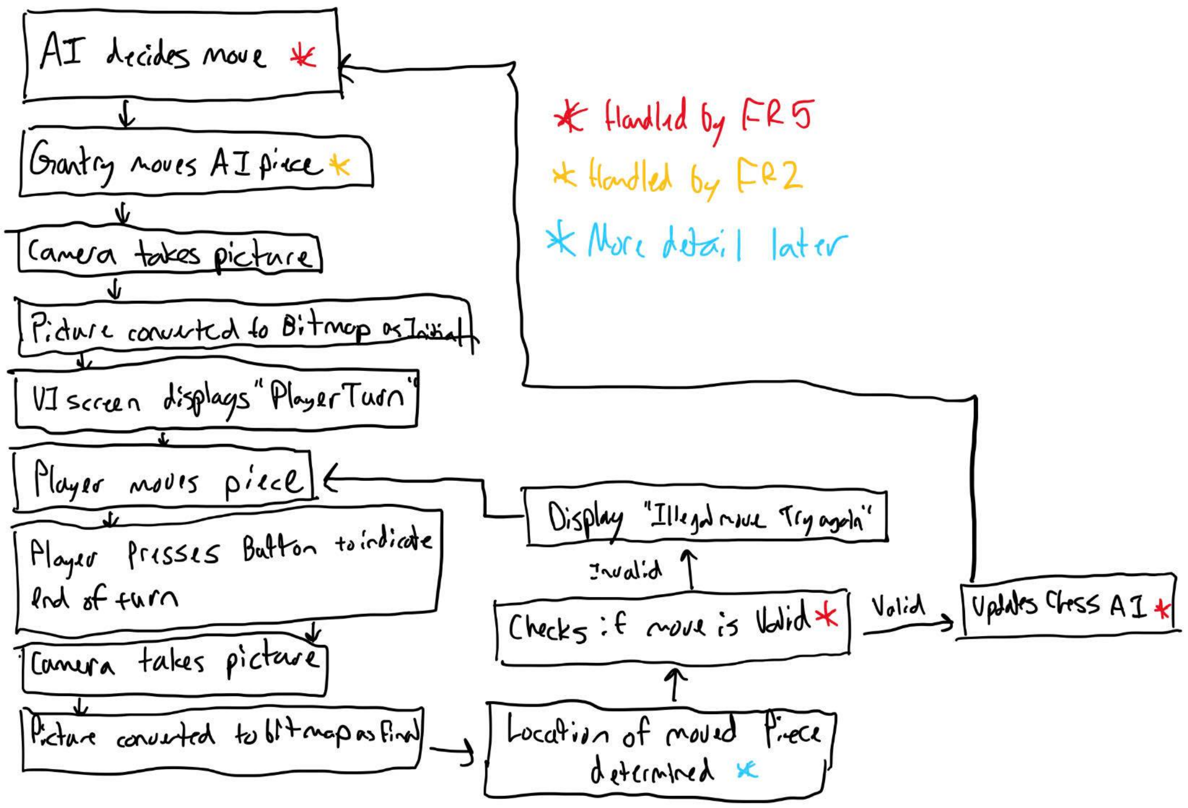
AI decides move *
Gantry moves AI piece *
Camera takes picture
Picture converted to Bitmap as Initial
UI screen displays "Player Turn"
Player moves piece
Player presses Button to indicate end of turn
Camera takes picture
Picture converted to bitmap as final
Location of moved piece determined *
Checks if move is Valid *
Invalid
Display "Illegal move Try again"
Valid
Updates Chess AI *
* Handled by FR5
* Handled by FR2
* More detail later

Figure 3. Function Diagram of Camera and UI Actions

The function diagram above explains most of the actions that the code will go through during the gameplay. This code will be run on a computer running a C# windows form to run the image processing and the software level chess decisions. The button input will be a simple push button with pull up resistor connected to a GPIO of the MSP430 to indicate the turn has been taken. This response then needs to be sent over UART to the computer. There will be a windows form to show the UI of the board. The UI will look something like the diagram below.

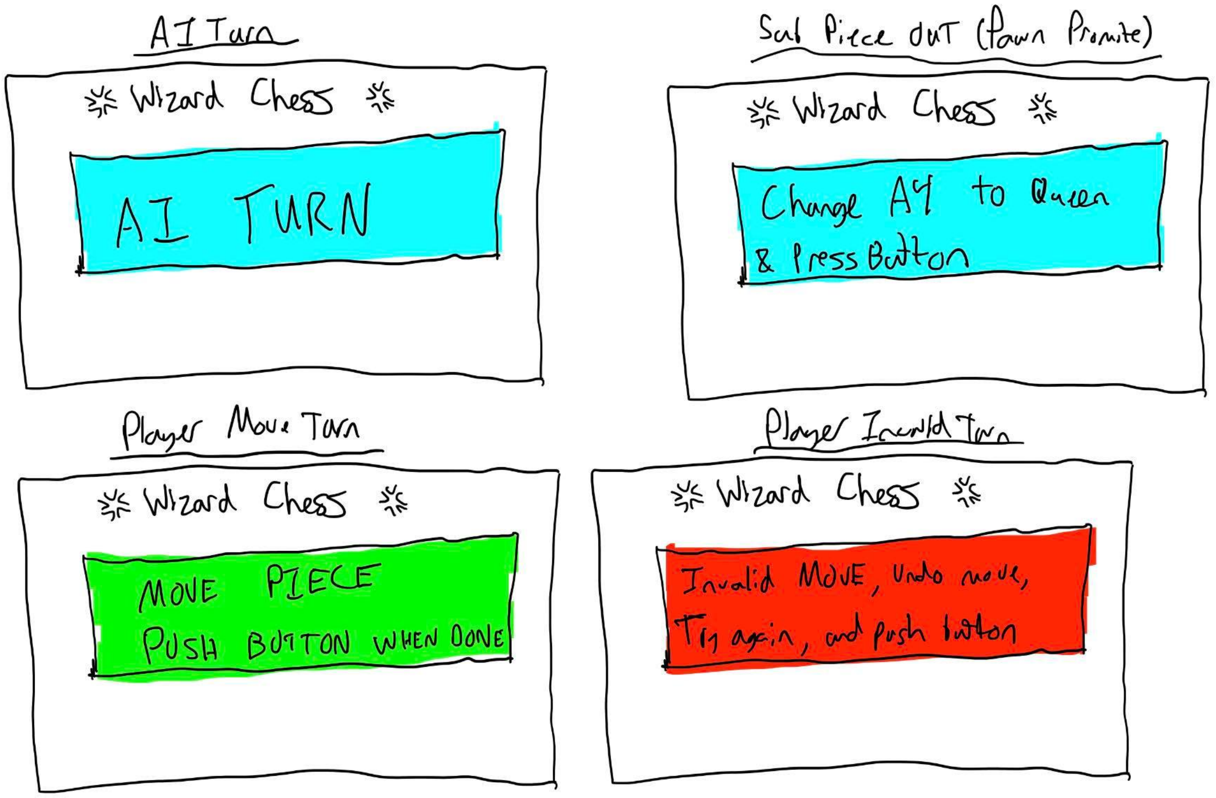

Figure 4. UI Diagram

The image processing will be done by taking the final image bitmap and subtracting the values of the initial image bitmap. Each piece will be painted a different color. This will result in a large difference in the two squares where a piece moved from and to. The rest of the squares will end up with a value of zero as they are the same before and after. Then I will search through the matrix of the bitmap to find which locations and colors the pieces moved from. By checking the values of the RGB values I can figure out what piece moved where. This is then sent to the Chess AI side which updates the board and decides the next move.

7.2. Inputs and Outputs

The Inputs of the system are as follows:

- Images of the chess board (taken by the camera and converted to bitmaps by the computer)
- Chess board tracked by chess API (used to predict what the board was before the player movement to help determine piece moved)
- Button input (to indicate player moved piece)
- Valid Move (software boolean that is determined based on players move)

The outputs of the system are as follows

- Predicted piece move (calculated by the image processing and sent to the chess AI to determine if it is valid or to continue the flow of the game)
- UI screen instructions (for player to know whether to substitute piece for the AI or if the move is valid or not)

7.3. Parameters

The colors of each piece will need to be chosen to produce the most obvious differences between before and after bitmaps on both black and white squares of the chess board. This will mean choosing colors as far apart from each other as possible on the color wheel but also choosing slight shades of the colors to determine whether a color moved away from a white square is different from the opposite color moved onto a black square. The colors of the squares will probably be black and white but could be changed to green and white or some other colors as a backup if the image processing is not working.

The colors of the pieces will determine the side that the piece is on and the function of the piece. This should be enough determination to figure out when a piece moves and what it moved to including if it took another opponent piece.

Another parameter that needs to be determined will be the distance the camera is from the board. Since the camera will always be mounted in the same spot the board can be determined during development so that each square of the chess board corresponds to a certain range in the bitmap, making it easier to determine where the grid is after subtracting the two bitmaps. The distance of the camera from the board will determine the resolution we have of each piece.

7.4. Development Plan

1. Setup importing of still images from webcam to bitmap on the raspberry pi
2. Setup button response that is transmitted over UART and triggers an event in C# for user input
3. Design the UI with different screens and messages for parts of the game
4. Choose colors and coding for the chess pieces
5. Setup matrix calculations for subtracting bitmaps and identifying changes in board
6. Make algorithm to identify colors in changed board and predict what piece moved where
7. Add functionality to the algorithm to correctly identify taking pieces, castling, and pawn promotion

8. Send output of the predicted move to the Chess AI to check if the move is valid. If this doesn't work then make a ruleset myself that checks the moves based on the piece.

7.5. Test Plan

1. Lookup scripts for this and test that a command can be sent from the windows form to take a picture from the webcam and convert it to a bitmap file saved to the computer.
2. Setup UART TX from the MSP430 to send button press interrupt to the computer. Validate that computer RX receives it properly. Then have this signal trigger an event in the C# script and validate that it triggers appropriately
3. Windows form design make sure it can switch screens depending on state in C# code
4. Test by investigating bitmaps of each piece on a white square or a black square and subtract it from empty black and white squares to make sure resulting values are different enough from each other for analyzing which piece was moved.
5. This is somewhat validated by doing the tests on development plan step 4. Otherwise test different piece movements after taking pictures and see if it the matrix changes as expected (mostly zeros with positive and negative RGB values in squares that have changed)
6. Test algorithm that combs through the subtracted bitmap and identify significant areas of change using thresholds of the RBG values and the affected areas around it. Then check if it is possible to predict what color has moved.
7. Same validation as above but with the more advanced move rules
8. Send the piece movement in a format the Chess API can understand and receive a valid or invalid move. Ensure that the C# form state changes accordingly

8. Functional Requirement #5: Chess Control Software & System Integration

8.1. Approach and Design

The components of this functional requirement involve a chess AI API (or library) that can take a player chess move as input and output the next computer move. This output computer move then needs to be translated into a series of X and Y movements that are sent over UART to the MSP430 so that the gantry can move the piece accordingly. Ideally the library or API can check if the move is valid, but if not then a rule table for different piece moves will need to be implemented by ourselves.

The output of the Chess AI will first send the predicted location in the XY plane that the piece that needs to move will be. The control is then handed off to the function in the Gripping/Moving Pieces functional requirement that moves around until the piece is engaged. Then the actual position of the piece is sent over UART to the C# software again and the new X and Y position will be calculated and sent over UART back to the MSP430 for the piece to move to. If this needs to be done in multiple steps (maneuvering a knight around pieces that it is “jumping” over) then the C# software will send a series of XY commands to fully complete the move. The last command will be to disengage the electromagnet.

8.2. Inputs and Outputs

The inputs of this functional requirement are as follows:

- Player piece moved (sent from the Sensing Piece Position FR) and inputted into the Chess AI
- Signal that the ElectroMagnet has been engaged (UART command sent from MSP430 to Computer) signals that piece moving can begin
- Signal that the AI turn is done (sent over UART after electromagnet has disengaged to indicate that the player move can start)

The outputs of this functional requirement are as follows:

- (Internal Output) Chess AI move which is translated internally to motion control locations
- Piece initial location (sent to the MSP430 to start finding the piece)
- Piece movement XY positions (sent to the MSP430 in order to move the piece safely to the final position)

8.3. Parameters

The sensor position resolution for the XY locations will affect the pathing of the pieces because they will need to be fine enough to move into the spaces between the pieces even after being potentially off of the normal grid from the Gripping Moving Pieces piece finding firmware.
Another parameter is whether both the X and Y stepper motors can move at the same time. If they can then diagonal path planning would be ok (for everything but the knight) but if not then linear movements in the spaces between pieces would need to be done.

8.4. Development Plan

1. Find Chess AI that works in C# and can take the player moves as an input and output computer moves
2. Make algorithm to send position commands from computer move to location on board of the initial piece location
3. Make algorithm that breaks down piece actual initial location to final location into single commands of XY locations (may need intermediate locations)
4. Make UART communication of XY locations to MSP430 for motor controls to take over
5. Make receiver for piece engaged signal from MSP430 (over UART)
6. Make MSP430 command that sends “motion finished” command over UART to the computer to indicate that the piece has moved and the program can enter players turn

8.5. Test Plan

1. Test integrating the Chess AI library or API into simple user text box inputs using whatever format the library needs. Confirm that the outputs are reacting properly to the inputs
2. Create lookup table for XY locations of the square the piece to move should be in and send initial XY value to MSP430 along with command indicating that electromagnet should turn on. Validate by sending expected UART signals to board to move to the correct location and that the electromagnet turns on. Then ensure that control is handed off to the Gripping/Moving pieces firmware executes to find the actual piece
3. Come up with method of first moving to edge of square, then move to Y position, then move to X position etc. Validate different distances and directions (diagonal, L shaped, straight) movements work and send expected XY intermediate locations
4. Check that outputs from previous step are converted into UART signals and sent to the board correctly (use logic analyzer on TX and RX pins)
5. Check that UART signal from Gripping/Moving firmware indicating piece is engaged is sent to the C# form and successfully starts the final location path finding function
6. Validate that after encoder readings are the XY locations that are the final position a UART signal is sent to the C# software that triggers the player turn state

9. Most Critical Module

The most critical module of our system is the chess control software. This is a completely new concept and not taught in any of our courses so it will require more research. Without the chess control software the board will be unable to play against the player.

To test feasibility, we will first find a chess strategy AI online that is compatible with C# and then determine if its methods will accomplish what we need given our firmware and hardware constraints.

The following tests will be performed to test the functionality:

1. Test whether the AI can keep track of the movement of the pieces in the game on a virtual board.
2. Test whether the AI can understand the opponent's moves and make its own moves strategically.
3. Make a grid in software simulating the real-life board and test whether the chess control software can accurately translate chess notation from the board to x-y coordinates in the grid.

10. Risks and Countermeasures

Potential risks could include

- Parts shipping too late
- Too much vibration from the stepper motors for the board to function
- Hall sensor signal doesn't work or is too much to implement in time constraints

Countermeasures for these risks

- Create list of backup parts in case 1st choice parts ship too late
- Implement optional damping between the board and the gantry
- Use Image Processing to check if piece is moving as expected, or assume that pieces are perfectly centered. Alternatively we can comb over the entire area of the square every time and assume the piece is picked up at some point during that process

11. Learning Objectives

Our learning objectives for this project are

- Learn how a strategic chess AI works
- Learn how to implement image processing using a webcam
- Gain experience with using electromagnets
- Gain more experience with data packet structure communication
- Practice simple pathfinding techniques to move pieces without crashing into other pieces
- Gain experience with more advanced state machines