Automatic Chessboard
ESE 350: Final Project
Spring 2011

Brian & James

Introduction

The purpose of this project was to create a physical chessboard that is able to interact with a human player and a computer. The game begins when the player makes a move. The coordinates of the piece's original location and its current location are fed into a computer chess engine after being detected by sensors embedded in the chessboard. The computer then replies with its move, giving the piece's current location and the desired destination. An XY positioning table underneath the board moves to the location of the piece to be moved. A magnet is then raised to attract the above piece and the piece is moved to the desired destination. The magnet is then lowered, and the XY positioner returns to its origin at the center of the board to await the next human move.

General Hardware Description

Hardware and Parts List

- **1 Freescale HCS12 MC9S12C Microcontroller Board**
- **1 Freescale PBMCUSLK Development Board**
- **1 Serial cable**
- **1 USB cable**
- **CodeWarrior Development Studio for HCS12(X)**
- **Epoxy and glue**
- **Wire**

For the Detection Board:

- **64 Reed switches:** http://search.digikey.com/scripts/DkSearch/dksus.dll?lang=en&site=US&WT.z_homepage_link=hp_go_button&KeyWords=HE558-ND&x=0&y=0
- **64 1N4148 Diodes**
- **8 10kΩ resistors**
- **1 2' by 2' total punch board**
- **1 clear Plexiglas plastic sheet for the actual chessboard (2' x 2' x 1/4")**
- **32 (3" by 3") Post-it notes for the squares**
- **1 (0.5") diameter metal pipe**
- **16 small steel disks**
- **16 small weaker refrigerator magnets**
- **1 chess set**

For the XY Table:

- **1 main very flat and heavy wood board to mount everything on (2' x 2' x 7/8")**
- **3 twenty-four inch Drawer Tracks:** http://www.amazon.com/Shop-Fox-D3033-100-Pound-Capacity/dp/B0000DD4AB/ref=sr _1_1?ie=UTF8&qid=1299258131&sr=8-1
- **1 (6' x 4" x .5") piece of pine wood**

- **4 wood screws**
- **2 Bipolar Stepper Motors:** http://www.pololu.com/catalog/product/1200
- **2 Stepper Motor Drivers:** http://www.pololu.com/catalog/product/1201
- **1 servo motor**
- **2 one inch diameter gears:** http://www.vexrobotics.com/products/accessories/motion/276-2250.html
- **2 sets of linear gears:** http://www.vexrobotics.com/276-1957.html
- **6 small Neodymium disc magnets:** http://www.kjmagnetics.com/proddetail.asp?prod=DB2
- **12 10kΩ resistors**

Detection Board Hardware Description

The detection chessboard consisted of a 2 foot by 2 foot, 0.25" inch clear plastic board that was placed on top of a 0.125" thick 2 foot by 2 foot punchboard. The chess pieces were placed on top of the plastic board, and the electrical circuit elements and switches were placed on the punchboard. The result was that the circuit elements were sandwiched between the punchboard and plastic sheet, and not able to be damaged by the magnet underneath. The magnet then had to be able to move pieces with metallic disks attached to them through about 0.5" thickness of material.

The choice of a 2 foot by 2 foot chessboard, which corresponds to 3" by 3" squares was made after testing four key things; the effectiveness of the six neodymium magnets connected together to move the metal disks we chose through 0.5" of material, how close the metal disks could get to each other without being attracted to each other when the magnet was attracting one of them was roughly 1.5", and the distance from being attracted to the smaller weaker refrigerator magnets placed on the human's chess pieces was also about 1.5", the weak magnets were powerful enough to penetrate the 0.25" of plastic board and trigger the magnetic reed switches however not too powerful that they would trigger a neighboring reed switch 3" away. The metal disks were punched out from steel electrical boxes that you'd use in your home to wire light switches or other things, as shown in Figure 1.

Figure 1: weak magnets and metal disks used

To build the detection board we first started by gluing several smaller punchboards into one large 2 foot by 2 foot punch board so that all the circuit elements and sensors could easily be mounted onto it. The board was then wired like a very large keypad. Eight independent column wires separated by 3" were weaved parallel through the punchboard for the length of the punchboard. Also, eight independent row wires separated by 3" were weaved parallel through the punchboard for the length of the punchboard, and perpendicular to the column wires. The 64 magnetic Reed switches were soldered exactly in the center of each 3 inch by 3 inch square. The Reed switches are normally open, meaning that they short when a magnet is brought close to them. A 1N4148 diode was also connected to the row wires in series

with each of the switches. This was to make it so that we could have multiple closed switches at once, rather than only being able to only be able to close one switch at a time. Without the diodes, with three or more pieces there would be ghosting and shorting of wires that not allow us to detect multiple chess piece locations at once since the microcontroller is using an real time interrupt that successively sends a single column high every 4ms. Thus there are no shorts of 5V to the row inputs from other columns when a single column was driven high because the diodes stop current flow in the wrong direction and act as open circuits. For more detail see http://www.dribin.org/dave/keyboard/one_html/. We used 1N4148 diodes because they have a faster switching time than 1N4004 diodes. The entire circuit diagram can be seen in Figure 2. All 8 rows were connected to PORTA

All 8 columns (or each lettered column line “a” to “h” of a general chessboard) were connected to microcontroller output pins PB0 to PB7, and all 8 rows (or each horizontal numbered row) were connected to microcontroller input pins PA0 to PA7. We inserted 10kΩ pull-down resistors to ground for stability, and so that the 5V input would not be directly shorted to the input. It is effectively a high asserting switch. For someone trying to save space on ports, we recommend using 3 bit multiplexers and decoders so that you only need 6 total inputs and outputs to the board rather than 16.

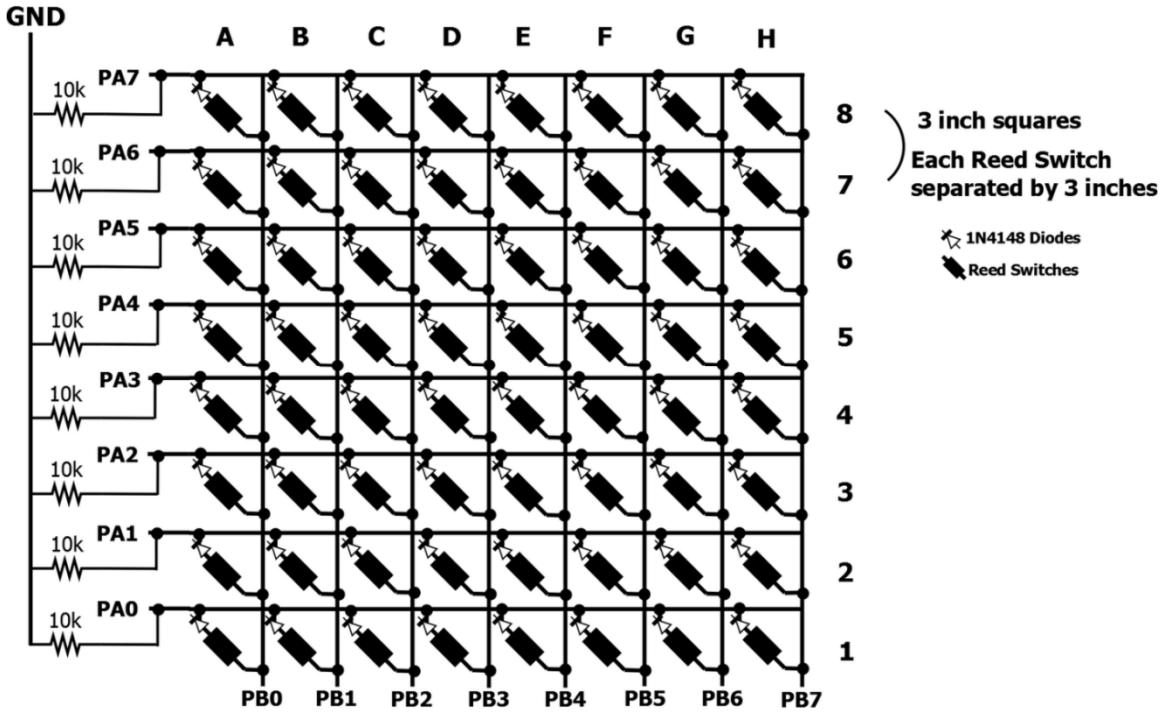


Figure 2: Detection Board Circuit Diagram

Here is a photo of the wired board:

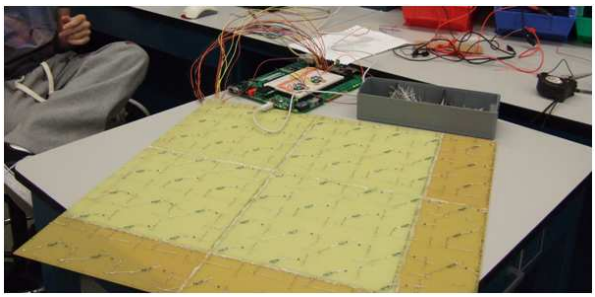


Figure 3: Fully wired detection board

In order to support the board, help keep it from warping, and obtain full range of the XY position system metal poles were mounted on all four sides of the board. Finally, due to the wires being weaved to make the columns and rows, it was necessary to tape printer paper of the thickness of three sheets to the bottom of the punchboard. This ensured the XY position magnet would not get caught on these wires and break off. Also the magnet can then be pressed up flatly against the board to so that the board can still be used even if it is warped. The finished detection board is shown in Figure 4.

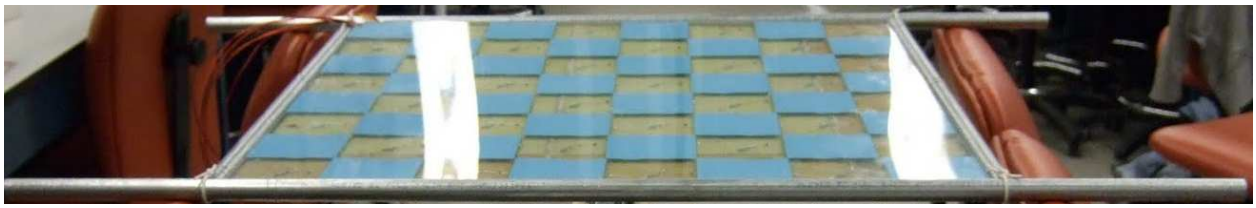
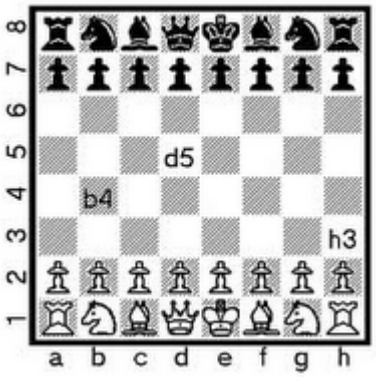


Figure 4: Finished Detection Board

As a final note on the detection board, the human's chess pieces have the weak refrigerator magnets glued to the bottom of them, and the computer's pieces have the metal disks glued to the bottom of them. This is because it is only necessary to detect the movement of the user's pieces, not the computer's pieces. In addition the human pieces could also therefore have sticky tack attached to them since the XY positioner doesn't need to move them and so that the strong neodymium magnet on the XY positioner does not attract them.

Detection Board Software Description

A general chessboard's layout is labeled as in the following diagram.



This is therefore the type of coordinates that the computer chess algorithm that we found outputted. Thus we needed to perform some conversions to interface with the chess program. We needed to convert the numerical move coordinates of the human's chess move detected by the detection board into a (letter, number) to (letter, number) coordinate.

Recall that the circuit diagram for the detection board is shown in Figure 2. By successively driving a single column line high, and the other columns low, the microcontroller could then read the input ports to see whether any of them are high. A coincidence of a row and column line being high means that a human's piece is occupying that particular square. If the input to the microcontroller is low when the column is driven high, this means there is no piece on that particular square.

The scanning of the column lines was done using a real-time interrupt. About 4 ms of low output was spent on each column. Every 4 ms the RTI interrupt is called, and the next column is sent high. The next column to send high is determined by the variable rti_cnt which ranges from 0 to 7. When rti_cnt is 0 this means column "a" (or 0 numerically), (corresponding to PB0) is low, when rti_cnt is 1 this means column "b" (or 1) is low, etc.

An 8 by 8 matrix stores the current state of the board. For example when the game first starts out the program will obtain the following board state matrix:

```
{ {1, 1, 1, 1, 1, 1, 1, 1}
  {1, 1, 1, 1, 1, 1, 1, 1}
  {0, 0, 0, 0, 0, 0, 0, 0}
  {0, 0, 0, 0, 0, 0, 0, 0}
  {0, 0, 0, 0, 0, 0, 0, 0}
  {0, 0, 0, 0, 0, 0, 0, 0}
  {0, 0, 0, 0, 0, 0, 0, 0}
  {0, 0, 0, 0, 0, 0, 0, 0} }
```

A '1' represents a piece has been detected, and a '0' represents an empty square. Note only the player's pieces will be detected since the computer pieces do not have magnets on them. 64 bytes may seems

like a waste of space when only 64 bits are needed, but C does not have a bool type and bit-manipulation can be difficult.

Note that the state matrix won't look exactly like the chessboard. It will look like a flipped mirrored image because the matrix starts at (0,0) at the top left, whereas the chessboard starts at (A,1). This is ok, because the program just needs to extract the start coordinates and end coordinates of the piece the human moves. It will be a direct map from the board to this matrix.

For example, say we move a pawn from C2 to C3 on the physical chessboard. The RTI system sees this as moving from (1,2) to (2,2). Note that chess labeling is not (row, column) like matrix notation, it's (column, row). The program will catch the change from 0 to 1 in the board state matrix at coordinate (1,2), (we incorporated some debouncing here). The program has a four character long string and change the first two characters to "C2 ". Then the program will catch the change from 1 to 0 in the board state matrix at coordinate (2,2), (again some debouncing). The character string will then be updated to "C2C3". This move of "C2C3" will then be passed into the computer program. The computer will then output its move.

The following is the correspondence table. The row number + 1 corresponds to the second coordinate, whereas the column number corresponds to a letter which is the first coordinate:

Row	→	Second Coordinate
0		1
1		2
2		3
3		4
4		5
5		6
6		7
7		8

Column	→	First Coordinate
0		A
1		B
2		C
3		D
4		E
5		F
6		G
7		H

Figure 5: Converting Human Piece Moves to Computer Input

Lines 213 to 272 perform the human move detection in the main function. The accompanying functions are getPositions(), arrayCopy(), and arrayCompare(). The getPositions() function is the function that goes through each of the 64 chess squares and writes a 1 if the Reed switch is closed (therefore a piece present) and otherwise a 0 for no piece present. This function relies upon the RTI to bit rotate PORTB and rti_cnt to increase. The arrayCopy() function copies the contents of one 8x8 array to another 8x8 array. This then later allows us to compare the present board state to a previous board state to determine which square a piece moved from and to which square it went to. The arrayCompare() function essentially just compares two 8x8 arrays to see if they are different, so that we can see if a human made a move, and then where it went to. It takes the address of two arrays. Then column by column, row by row each element is compared. If there are no differences then 0 is returned, otherwise 1 is returned and the two variables at the memory address of Xchange and Ychange are changed to reflect the difference.

After the start and destination coordinates are obtained they are output through serial as the human's move.

XY Table Hardware Description

The XY table positioner needed a very flat and heavy base on which to mount everything. We chose a relatively cheap particle board that was 2' x 2' x 7/8", which looked like it might be used on a countertop. The first step was to mount two drawer tracks exactly parallel to each other, one on each side of the board. These tracks serve as the X-axis direction of motion, and they had to be perfectly parallel. If the bottom tracks are not mounted perfectly parallel, the stepper motor would only be able to move in the X-axis direction a limited distance before stopping at some arbitrary location instead of going the full two feet desired range of motion.

Next, the circular gears were glued onto the stepper motors, which involved drilling a hole through the gears that was large enough so that the gear would fit the axle. Then, linear gears were glued onto a 2 foot long piece of pine wood in a perfectly straight line, and the wood was glued right next to the track. This effectively raised the linear gears up because the 1" circular gear was too small to reach the gears if they were glued directly on the flat board next to the drawer rack. Thus after this, the motor had to be mounted at a specific height using an 8 inch piece of the pine wood and then the pine wood glued onto the track with the circular gear mated with the linear gear. This was tedious as the gear needed to be able to run along the entire length of the two feet of linear gears. After this another 8 inch piece of pine wood needed to be mounted in the same place on the track directly across from the first support. These two pieces of wood will then serve to support the Y-axis motor. Here is a photo of the X-axis motor mounted. Note that the linear gears are not glued in place yet, but the glue is still drying:

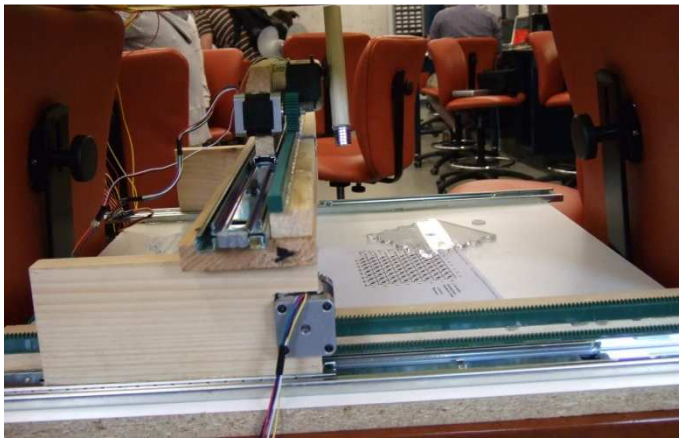
Figure 6: X-axis of the XY table

The next step was to mount the Y-axis drawer track. First a two foot piece of pine was mounted directly on top of the 8 inch pine supports shown in Figure 6. It was mounted with 4 wood screws so that it was perfectly perpendicular to the X-axis drawer tracks. Then, one drawer track was mounted on this piece of wood. Then, similar to the X-axis, linear gears were raised and placed on a two foot long piece of pine wood and placed directly

along the drawer track. The Y-axis motor was also mounted on another 8 inch board to raise the motor to have the circular gear mate with the linear gears. This motor was not mounted as stable as the X-axis motor due to the lack of a cross support, so a counterweight was needed on the pine wood to help the gear stay in contact with the linear gears. Note that the motors were mounted in the same direction such that we could have an intuitive positive direction in the X-axis and Y-axis corresponding to the same rotation direction of the motor axles.

The next step was to mount the servo motor that would flip up the magnet. The servo was placed very close to the Y-axis stepper motor so that we could get the full range of motion from the entire system. Then the magnet was glued to one end of a wooden dowel rod that was about 6 inches long. This dowel rod was then glued to the servo motor when it was in the raised position so that we could make it be perfectly flat facing upward when in the on position. The dowel rod had to be this long because the magnet needed to be at least 4 inches away from the board when it was lowered so that it wouldn't trigger any Reed switches when the board was to detecting moves.

This then completes the construction of the XY table. It was rather tedious to build, and I recommend anyone that is building one to take their time to get all the measurements exactly correct so that you won't have to end up taking it apart and rebuild it. We were pretty lucky in that it worked perfectly after the first try of building it.



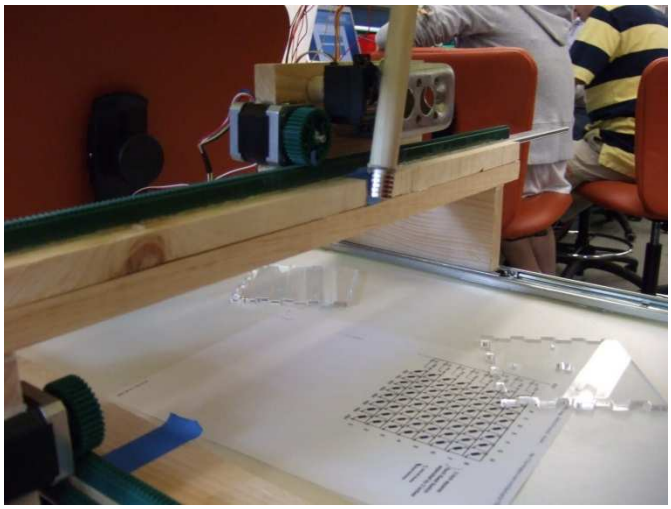


Figure 7: Completed XY Table

The final part of construction on the XY table involved wiring up the stepper motors. A motor driver was needed for each stepper motor.

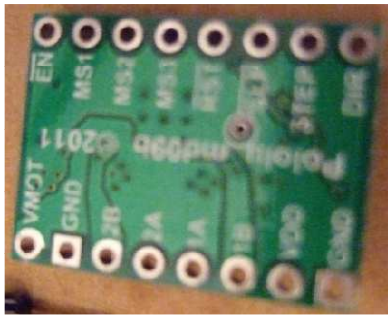


Figure 8: Stepper Motor Driver

The motor we chose can be either unipolar or bipolar and has 6 leads. We chose to use it as bipolar since it only required the use of four of the leads. Bipolar steppers have a single coil per phase. The three inputs we had that were controlled by the microcontroller were DIR, STEP, and $\overline{\text{SLP}}$. The DIR determines the direction of rotation of the motor. When low, the direction will be clockwise and when high, counterclockwise. A low-to-high transition on the STEP input sequences the translator and advances the motor one increment. The size of the increment is determined by the combined state of inputs MS1, MS2, and MS3. We tied MS1 to high (5V) to get half-steps. The Sleep input minimizes power consumption when the motor is not on. It was necessary to use this because the motors draw 1 to 1.2A constantly if they are not disabled. If left on, the drivers could become overheated and burn. This happened to one of our drivers, and luckily we had an extra one.

The VMOT input controls the voltage and current input to the motor. The motors were rated at 4V and 1.2A. The minimum voltage needed by the board at VMOT is 8V. For stepper motors, the voltage is not as important as the current, so we could set this to 8V with no problem. However, the current limit to the motors had to be set. The driver has a built in current limiter that can be changed for different size motors. There is a small circular Vref pin in the center of the driver board. If everything is connected up correctly and the logical voltage and motor voltage is applied, this voltage can be measured. The current limit is then Vref/(8*.05). The value of Vref can be changed by adjusting the potentiometer on the back of the board with a screwdriver. The current limit was adjusted to 1.2A for these motors.

The $\overline{\text{EN}}$ enable pin was constantly tied to ground so that the motors would stay enabled. Also, the $\overline{\text{RST}}$ reset was tied to 5V so that the motor wouldn't reset. VDD was the logic voltage and had to be tied to 5V. The entire circuit is shown in Figure 9.

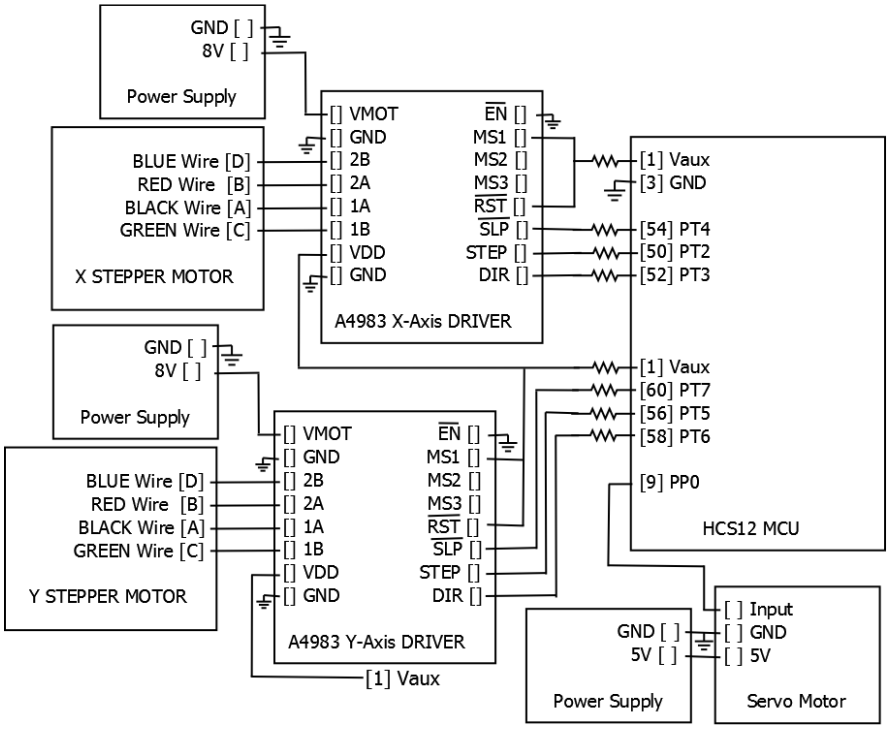


Figure 9: Stepper and Servo Motor Wiring

The blue wire of the stepper was connected to 2B, the red wire to 2A, the black wire to 1A, and the green wire to 1B. PT2 and PT5 of the microcontroller were utilized as output compare to control the X-axis step and Y-axis step respectively. PT3, PT4, PT6, and PT7 were used as GPIO output pins to control DIR and $\overline{\text{SLP}}$ since all of PORTA and PORTB were used by the detection board.

Figure 9 also contains the wiring for the servo motor as well. We used PP0 as input to the servo, which is a pulse-width modulation port.

XY Table Software Description

Lines 273 to 456 control the XY table movements of the computer's move. The segment of the code dealing with XY table movement begins with the getMove() function on line 273. This function obtains the computer's move over serial and places it in a length 4 array called computerMove.

We needed to have another conversion of coordinates to be able to interface the XY table movement with the computer's coordinate outputs. We needed to convert the computer's letter, number coordinates to numerical coordinates for the XY table to perform its task.

The moves will come in the form (letter,number,letter,number). For example a move could be C7C6, which would mean move the piece at square C7 to square C6. Again recall that the computer outputs in (column, row) coordinates instead of matrix coordinates (row, column). Thus we do an inverse coordinate transform as that shown in Figure 5 to get the correct numerical coordinates. Thus C7 would correspond to (6,2), and C6 would correspond to (5,2).

We designed the XY magnet system so that it can transport the pieces along the lines in between the squares so that the pieces do not collide. This is another reason why the board squares needed to be a large 3" by 3". Thus the XY magnet system needed to have a finer coordinate system so that it can both go to the location of the piece and go in between other pieces. The following grid shows the coordinate system of the XY magnet system.

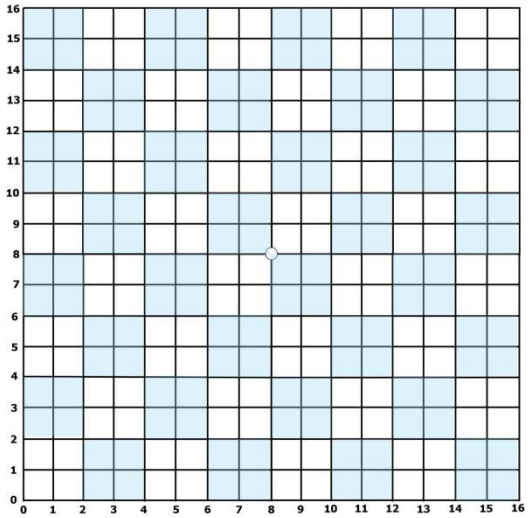


Figure 10: XY Positioner Coordinate System

The coordinates now range from 0 to 16 in both X and Y directions. The actual chess squares are colored so that the coordinate system can be visualized overlaying the actual chessboard. The chess pieces can only be located at coordinates where both the X and Y coordinates are both odd.

Note that the transformed numerical move coordinates will again need to be changed to fit the granularity of the XY system coordinate system. This involves multiplying each coordinate by two and adding 1: (2x+1,2y+1). Thus if the instructions are move piece at (6,2) to space (5,2) are received, this would correspond to the XY system needing to move the piece at (13,5) to (11, 5). This is done on lines 277 to 280.

The XY system starts at the center of the board at the start of each computer move, which is position (8,8). The XY system is then returned to (8,8) after each move so that it can maintain a constant reference point. Luckily the stepper motors are extremely accurate and no recalibration was needed to prevent error. Even after moving a piece from one corner of the board to the opposite corner, the motors would return back to the exact center after completion.

So when the origin coordinates are received, we subtract (8,8) from the coordinates. For example, (13,5) – (8,8) = (5,-3). The X stepper motor then shifts up five coordinate steps; each coordinate step of 1.5" has a specified number of motor steps that was determined through testing. The number of steps is 255 and defined as SET_COUNT_X and SET_COUNT_Y just in case the X-axis and Y-axis motors happened to be different, but they weren't. The Y stepper motor then shifts left 3 coordinate steps.

Note that the stepping is done using the stepSpace(motorselect,numsteps) function. This function steps either the X or Y axis stepper motor the distance of numsteps coordinate steps (each coordinate step is 1.5 inches). The direction is changed accordingly if numsteps is negative or positive. When motorSelect=0, the X-axis motor will step. When motorSelect=1, the Y-axis motor will step. When numsteps<0 then DIR=0; the direction is clockwise (set PT3 or PT6 to 0). When numsteps>0 then DIR=1; the direction is counterclockwise (set PT3 or PT6 to 1). The corresponding motorselect motor is first woken up, the motor then turns the specified number of turns corresponding to the value of numsteps, and then it is put back to sleep.

After the XY magnet positioner arrives to the start coordinates, the magnet is then flipped up with the servo motor to attract the piece (line 299). Some delay ensures that the magnet has time to attract the piece.

Then the current coordinates are subtracted from the next coordinates. For example, (11,5) – (13,5) = (-2,0). Thus the X stepper motor will need to shift down 2 coordinate steps. The Y stepper motor will not need to shift in this case. However, to avoid piece collision we want to travel along the lines instead of through the squares. Thus, each time we are moving a piece, we first shift up and right 1 coordinate step. So we'd go from (13,5) to (14,6). Then we'd do the (-2,0) shift, thus going from (14,6) to (12,6). Then we'd cancel out the offset shift by going down and left one coordinate, thus going from (12,6) to (11,5). The piece has then arrived at the destination coordinates.

Lines 309 to 351 take care of the offset. For most coordinates we simply apply the generic offset as just explained, so that it is no longer centered on the square but on the grid line. The piece is then moved to wherever it needs to go. This is not the shortest route but it does get the piece there. However, since our board has a metal pipe border we could not have the magnet anywhere near the border, as it would get attracted to the pipe and the range of motion can't go beyond the border anyway. Thus a lot of the code in lines 309 adds exceptions to certain moves. For example, if we want to move a piece from the top row, we now shift down instead of up when offsetting initially. And, if the destination is a bottom square on the right side of the board, we shift in from the top left or bottom left of the square instead of the top right.

To finish the move, the magnet is flipped down, and the stepper motors move the system back to the center (8,8) coordinate by subtracting the current coordinates from (8,8). For example, (8,8) – (11,5) = (-3, 3). The loop then starts over again and the computer then waits for the next human movement on the detection board

Chess Algorithm Software Description

The chess engine communicates to the MCU via a RS232 interface (serial port). The chess engine runs on a computer and is fed chess moves via coordinates (C3C4), the engine replies to these moves by generating its own move and outputting its own coordinates. Originally, the engine sent and received data from the terminal, however to interface with the MCU we needed an easy and compatible method of communication.

There was a simple library for serial communication included with the IDE for the MCU, so we decided to use serial communication. That meant that the computer chess engine written in C would need to be modified to send and receive data over serial. By incorporating a PC serial library we were able to do this. Since the library was initially intended for a windows or Linux environment a few modifications had to occur.

When the computer chess engine is waiting for user input it, it actually waits for data to be received over serial. The program takes the first four bytes (four characters for the coordinates) of the serial buffer and interprets these as a move. The chess engine outputs its moves.

Conclusion

By the end we had accomplished all that we set out to do; build a chessboard that can play physical chess against a computer. The result was that it had the look and feel of playing against an actual opponent in real life rather than playing a computer game on the computer. There were some small problems that we could have fixed with more time. The metal disks we chose for the computer's pieces were a little bit too big, and sometimes when the magnet was moving in between the other computer pieces they would attract and either switch pieces or drag multiple pieces at once. This effect was somewhat alleviated by putting some cardboard "bumpers" on each of the pieces in order to stop a

strong attraction. Also, the board we used to place the pieces on was not perfectly flat. Therefore some parts of the board, particularly the edges, were higher and the magnet sometimes couldn't attract and move these pieces. Another problem was that the human pieces couldn't be placed directly in the center of the Reed switches because at that location for some reason the Reed switch wouldn't close. Thus the piece had to be placed slightly off-center. The result was that sometimes if the human didn't place it correctly the computer wouldn't register a move taking place and the program would say illegal move. We didn't program a back button, so every time an illegal move occurred we would have to restart the entire program. In addition, we didn't have the capability to castle. At one point the computer tried to castle, and we just had to help it by moving the rook to the other side of the king for it. If we were to re-design the program we could make it so that if the king moves as if to castle, we would also pick up the corresponding rook and move it to the other side. Finally, the biggest problem was combining the detection board together with the XY table. This was difficult because the board had to be mounted at a specific height above the XY table magnet so that the magnet was flush against the bottom of the detection board. You can see from the photos that we used some chairs to prop it up on since they were adjustable to any height. The result is that it takes a while to get the board aligned correctly, but after that it works great as long as nothing moves.

We are proud of our accomplishment of finishing a complicated and very lengthy project in such a short amount of time. At many points throughout the project we were uncertain that it would ever be possible to do this. We thought it was very interesting that we used a lot of the subjects we learned in the labs to complete this project, and the whole project used the HCS12.

Please see the videos on our blog at: www.acboard.blogspot.com

Our final Demo video: http://youtu.be/2r53uC6XEgA (it was too large to upload to the file)

The code (for both the hcs12 and chess algorithm) is in the same folder as this report

Our presentation slides.

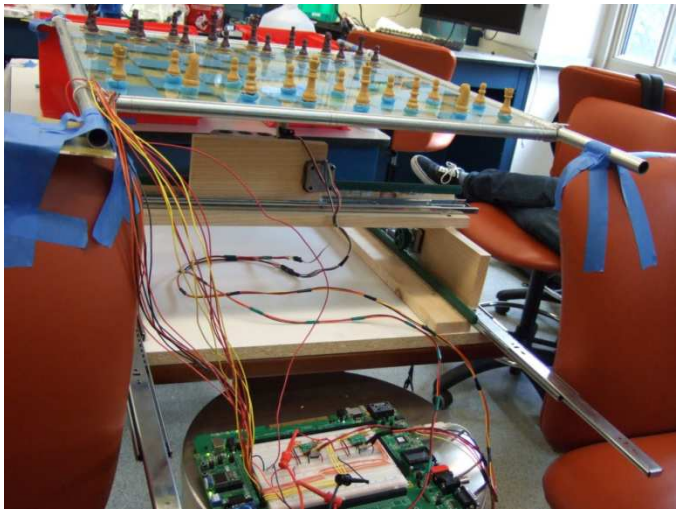


Figure 11: The Finished Project

References

For figuring out how to get multiple key presses: http://www.dribin.org/dave/keyboard/one_html/.

We used this website for some initial guidance on how to build the XY table. From this we got the idea of using drawer tracks on a flat board in combination with the linear and circular gears to get the X-Y motion, and a servo for a z-axis. We also got the idea of using a computer to play chess against but we didn't use or even look at any of the code on this website at all. We made all the code ourselves. And, everything else we did was different. http://www.instructables.com/id/How-to-Build-an-Arduino-Powered-Chess-Playing-Robo/

The stepper motor driver datasheet: http://www.pololu.com/file/download/a4983_DMOS_microstepping_driver_with_translator.pdf?file_id=0J199