6.115 Final Project Report

James McCabe

May 15, 2014

1 Introduction

My project is a Connect Four robot that physically plays Connect Four against a human opponent. This means that the human player inserts their game pieces as they usually would on their turn, and the robot inserts its game pieces into the game board on its turn. This project is interesting because it explores artificial intelligence, game strategy, and human-robot interaction. I enjoy projects that highlight areas where technology has surpassed humans in computational ability. In this case, the robot will be able to enumerate the possible game states to some depth and score all of those states in order to try to beat the human player. This is something that the human mind would find difficult and it is why the robot should turn out to be difficult to beat. So combining A.I. with a physical robot is a very in-your-face demonstration of this gap between humans and technology.

2 Mechanical Description

2.1 Overview

At a high level, the robot's mechanics consist of: **(see figure 1)**

- A standard Connect Four game board, which has 6 rows and 7 columns
- A chip dispenser that allows the robot to dispense its game pieces into the game board
- A linear actuator that moves the chip dispenser along the top of the game board
- A control panel for settings

2.2 Linear Actuator

The linear actuator is what moves the chip dispenser along the top of the Connect Four board. This allows the dispenser to dispense a game piece in any one of the 7 columns of the game board. There were a few options available when choosing how to design and build the linear actuator. The econimical and simple solutions almost all involve converting rotational motion into linear motion. A few ways of accomplishing this include using a rack and pinion, a ball screw, or a timing belt and pulley. I chose to go with the latter. The method of using a timing belt and pulley is one that has been very popular in the 3D printer revolution. This makes many of the parts easy to source. It is also an ideal option for quickly moving the relatively light dispenser mechanism.

The linear actuator that I constructed consists of:

- A bipolar stepper motor **(see figure 2)**
- A shaft mountable timing belt pulley **(see figure 2)**

- A timing belt **(see figure 2)**
- A 15in long piece of aluminum c-channel **(see figure 2)**
- An SPDT limit switch **(see figure 3)**
- A timing belt pulley and bearing assembly **(see figure 4)**

A stepper motor is used to provide precise movement of the linear actuator without feedback. This is the simplest and most reliable solution to moving the dispenser a set distance between columns of the game board. The distance between columns is converted into motor steps, which allows the dispenser to essentially have seven distinct positions corresponding to the seven columns of the game board.

However, since there is no positional feedback and all movements of the linear actuator are relative offsets to the previous location, a limit switch is necessary to home the linear actuator upon start-up. It does this by moving the dispenser until it bumps into the limit switch - at which point the actuator knows where the dispenser is located because it knows where the switch is located. All of the relative offsets are calculated from this home location. So after homing, the linear actuator can calculate where the dispenser is located by summing all of the relative offsets that are executed. This is assuming no interference or slipping.

The stepper motor mounts onto the outside of the c-channel such that the shaft sticks inside of the c-channel **(see figure 2)**. This allows the timing belt to be housed inside of the c-channel, which provides it with some protection. The aluminum timing pulley is then mounted onto the shaft of the stepper motor and secured with a set screw.

The other end of the timing belt is wrapped around a similar timing belt pulley that rotates on two bearings **(see figure 4)**. This other timing belt pulley was 3D printed, along with the bearing blocks and the rod that goes through both bearings and the pulley. This assembly allows the timing belt to rotate. The motor spins the timing belt pulley, the pulley pulls on the timing belt, and the bearing assembly rotates freely.

2.3 Dispenser

The dispenser is used by the robot to deposit its game piece into the column that corresponds to the move it wants to make. The design and construction of the dispenser makes heavy use of Solidworks and 3D printing.

The dispenser that I constructed consists of: **(see figure 5)**

- A 9g servo
- A vertical game piece hopper
- A funnel
- A servo mountable game piece extracter
- A slider made for the linear actuator's c-channel

I wanted to design something that required only one actuator to dispense and collect game pieces. I first choose the actuator, a servo. I went with a servo because its position can be precisely controlled. A servo also does not rotate continuously, which I didn't believe was necessary.

With any dispenser design a hopper would be required to store the game pieces. The dispenser must be able to retrieve pieces from the hopper and place them in the game board columns. I went with a vertical hopper design - meaning the game pieces are stacked like poker

chips on top of each other. This is the most space efficient way of storing the game pieces. The hopper rides along with the dispenser on the linear actuator. The hopper I designed holds a maximum of 12 of the 21 game pieces a player has, so it may have to refilled during the course of a game. I didn't want to make the hopper too tall, but it does provide a way of adding an extension to its height, thereby increasing its capacity.

With the game pieces stored on their flat side, there must be a way of rotating them 90° so that they can be inserted into the game board's columns. I designed a funnel to perform this action. A piece is dropped into the top of the funnel and comes out oriented correctly for insertion. The output of the funnel leads directly into a column so that the piece falls in.

Lastly, the game pieces must get out of the hopper and into the funnel. I designed a head for the servo that accomplishes this. The head has a circular cut out that is a bit larger than the game pieces. Initially this circular cutout is under the hopper so that a game piece rests inside of it. The game piece is supported below by another 3D printed part. To dispense a piece, the servo rotates the head from this position, carrying the game piece with it, to over the funnel - at which point the piece falls into the funnel. While this rotation is occuring, the rest of the head rotates below the hopper, preventing the stack of game pieces from shifting downward due to the removal of a piece from the bottom. Only once the circular cutout rotates back to the position below the hopper does the stack of chips shift downward, reloading the head. **(see figure 6)**

The dispensing mechanism must move with the linear actuator. To accomplish this, the dispenser is built on a c-channel slider that was made specifically for the c-channel used in the construction of the linear actuator. The slider is essentially two identical pieces of plastic that slide along the edges of the c-channel. These two pieces of plastic are connected by a few 3D printed pieces. This slider then attaches to the timing belt of the linear actuator, so that the dispenser moves whenever the timing belt does.

2.4 Control Panel

The control panel allows the user to modify the difficulty setting of the robot and ask for a hint. **(see figure 7)**

The hardware of the control panel consists of a 3D printed box, two push buttons, and a 7-segment display. A ribbon cable is used for the electrical connections between the control panel and the robot. More details of the operation of the control panel can be found in the electrical and software descriptions.

3 Electronics Description

The two page schematic is referenced through out this section.

3.1 Overview

The robot's operation is divided between the PSOC and an AT89C2051 (2051) microcontroller. The PSOC and 2051 communicate over a UART serial connection. The robot detects the human's moves by reading the output from photo-interrupters that are placed at the top of each of the Connect Four columns. The robot makes its moves using a stepper motor and a servo as previously described. The difficulty of the robot can be set using a button. A hint can be asked for by also pressing a similar button. Feedback is provided to the user through a 7-segment display.

The chips and other electrical parts used in this project are:

- AT89C2051 - the microcontroller that handles the operation of the robot.
- 82C54 - used to produce the control signal for the servo.
- LM18293 - provides a buffer between the low voltage, low current circuitry and the higher voltage, higher current stepper motor.
- SN74LS47 - provides an easy way of displaying numbers on the 7-segment display.
- GP1A57HRJ00F - photo-interrupters used to detect the user's move.
- 8255 - provides more inputs and outputs for the 2051 so that it can interface with the LM18293, the SN74LS47, the seven GP1A57HRJ00F, the buttons, and the limit switch.
- PSOC - runs the robot's strategy program.
- 74LS14 - inverter used to construct two chip select lines from one 2051 output

3.2 AT89C2051

The AT89C2051 (2051) is the microcontroller used to control the basic operation of the robot. It controls all functions except the game strategy and calculating hints for the user. Unlike the 8051 used through out the course, the 2051 lacks a data bus, address bus, read, write, and chip select pins - all of which are necessary to interface with the 8255 and the 82C54. These buses and pins are emulated on the 2051 through software. Port 1 is used as the data bus. Pins $P3.4$ and $P3.5$ are respectively used as $A0$ and $A1$ to create a 2-bit address line. Two bits is all that is necessary to address the 8255 and 82C54. Pin $P3.3$ is used as the active low read line. Pin $P3.7$ is used as the active low write line. Once you account for the serial communication between the 2051 and PSOC, there is only one I/O pin left, $P3.2$. Pin $P3.2$ is used as the active low chip select line. The 82C54's CS pin is directly connected to $P3.2$. The 8255's CS pin is connected to $P3.2$ through an inverter. This inversion means that only one of these two peripheral chips is selected at once.

The 2051 uses a 11.0592MHz crystal because of the UART serial communication between the 2051 and the PSOC. This crystal value fits evenly with the baud rate calculation.

3.3 82C54

The 82C54 timer chip is used, in conjuntion with the 2051, to create the control signal for the servo.

A servo's angular position is determined by the duration of the high period of the signal on its signal wire. The length of this pulse can range between 1ms-2ms. A 1ms high period corresponds to an angular position of -90°. A 2ms high period corresponds to an angular position of +90°. The angular position does not depend on the duty cycle of this signal - just the time that the signal is high. However, most servos expect a pulse every 20ms - corresponding to a 50Hz signal, but this can vary servo to servo.

The 82C54 is connected to a 10MHz clock. The $OUT0$ of counter 0 is connected to $GATE2$ of counter 2. This allows the necessary servo control signal to be created by running counter 0 in mode 2 and counter 2 in mode 1. The read pin of the 82C54 is pulled high (disabled), because we will not be reading from it.

3.4 SN74LS47

This chip controls the 7-segment display. It takes in a nibble and displays the corresponding decimal number on the 7-segment display. It can display the numbers 0-9. The outputs are active low, so they must be connected to the cathodes of the LEDs inside of the 7-segment display. This means that the 7-segment display that is used must be common anode.

3.5 GP1A57HRJ00F

These are the photo-interrupters used to detect the user's move. There are seven of these on the robot - one over each column of the Connect Four game board. The sensor is a U-shape. It emits infrared light on one side and projects it across the gap of the U-shape to the other side where it is detected. This sensor is active low - meaning the digital output line from the sensor goes from high to low when the gap is blocked. The infrared emitter side of the sensor requires a current limiting resistor to safely operate the device. I chose a value of 330Ω. With a power supply of $5V$ and a typical forward voltage of $1.14V$, this provides $11.6mA$ of current.

In experimenting with these sensors and the game pieces, I found that the IR light tends to go through the center of the game pieces. So when a piece falls through the interrupter, there is a transition from high to low on the thicker edge of the piece, then a transition from low back to high on the center of the piece, and then another transition from high back to low on the edge of the piece, and then finally a transition from low back to high as the piece leaves the gap. The fact that this occurs is not that important because the initial transition from high to low allows for detection of a game piece and the time before another sensor reading is a few seconds.

3.6 LM18293

This buffer drives the stepper motor. The stepper motor is driven with $12V$. At this voltage, it draws $350mA$. These are voltages and currents that cannot be produced by the other circuitry, which is why this chip is necessary. It simply copies the signal on one of its inputs to the corresponding output, but with, in this case, a higher voltage ($12V$) and a higher current capability. One coil of the stepper motor is connected to $OUT1$ and $OUT2$ of the LM18293 and the other coil is connected to $OUT3$ and $OUT4$.

3.7 Buttons and Limit Switch

The difficulty and hint button are housed in the "settings console". The limit switch is mounted on the side of the linear actuator's c-channel. These two buttons and the limit switch are active low. So one side of them is connected to GND and the other side is pulled up to $5V$ via a $10K\Omega$ resistor. The output is taken between the pull-up resistor and the corresponding button/switch terminal.

3.8 8255

The 8255 provides extra input and output pins for the 2051, which has no left over I/O pins after all of the pins have been dedicated to communication. The 8255 must then read the inputs from the seven GP1A57HRJ00F, read the buttons and limit switch, control the 7-segment display through the SN74LS47, and control the stepper motor through the LM18293. The output of the seven GP1A57HRJ00F sensors are connected, from left column to right, to $PA0 - PA6$ of the 8255. The control nibble for the SN74LS47 is connected to the high nibble ($PC4 - PC7$) of port C on the 8255. The LM18293 is controlled with the lower nibble ($PC0 - PC3$) of port C. The output of the limit switch, help button, and difficulty button are respectively connected to $PB.0$, $PB.1$, and $PB.2$. So port A and port B are set in software as inputs, and port C is set as an output.

The active low CS pin of the 8255 is connected to $P3.2$ of the 2051 through an inverter. So the chip is selected when $P3.2$ is set high.

4 2051 Firmware Description

The 2051 controls every operation of the robot except the game strategy and hint generation. A high level of the software operation is given by the following flowchart. The actual code use can be found in **Appendix A** of the code packet. The following is a description of some parts of the firmware.

4.1 Communication with the 8255 and 82C54

The 2051 communicates with the 8255 and 82C54 through the data bus, read, write, and chip select pins that are described in the hardware description.

To write to the 8255 or 82C54, first the address of the internal register that is going to be written to must be set on the address lines ($P3.4$ and $P3.3$ of the 2051). Then the CS pin ($P3.2$ of the 2051) is enabled. For the 82C54 this means writing a low to $P3.2$, and for the 8255 this means writing a high to $P3.2$. Now the data is put on the data bus, $P1$ of the 2051. Then the write line, $P3.7$ of the 2051, is pulled low and then raised back high. This creates a short write pulse. Then the CS pin can be disabled by flipping its value. However, this simply selects the other chip because one chip is always selected. This is not a problem as long as the read and write lines are disabled (i.e. high).

Reading data from the 8255 is done in a similar way to writing data. First the address of the port on the 8255 that is to be read to must be selected (ports A and ports B are inpus). However, now we must pull up $P1$ of the 2051 by writting $0FFh$ to it, because it is going to be used to input data rather than output data. Then the CS pin is enabled by pulling $P3.2$ high. Then the read line is enabled by clearing $P3.3$. Then the data is read in from $P1$ of the 2051 before disabling the read line by seting $P3.2$ high. Lastly the CS pin is disabled.

4.2 Communication between 2051 and PSOC

The communication between the 2051 and PSOC is done over UART serial at a BAUD of 9600. Every transmition first sends an "action character" and then possibly a "parameter". The action character represents what type of action is either to be made or has been made.

The action characters are:

- 'M' : This represents that the human has made a move when sent from the 2051 to the PSOC. This represents that the robot is going to make a non-winning move when sent from the PSOC to the 2051. The parameter that follows is an integer from 1-7, representing a column on the game board from left to right.
- 'H' : This represents that the human has asked for a hint when sent from the 2051 to the PSOC. It represents that the PSOC is responding to a hint request when sent from the PSOC to the 2051. No parameter follows this action character when sent from the 2051 to the PSOC. However, when the PSOC sends this action character, the parameter that follows is the hint. The hint is an integer from 1-7, representing a column on the game board from left to right. The hint is displayed on the 7-segment display.
- 'L': This is only ever sent from the PSOC to the 2051. It represents that the robot is about to make a winning move. The parameter that follows is the same as the one for the 'M' action character.
- 'W': This is only ever sent from the PSOC to the 2051. It represents that the human has won the game. No parameter follows this.

- 'D': This is only ever sent from the 2051 to the PSOC. It represents that the human has changed the difficulty. The parameter that follows this character is the difficulty, which is an integer from 1-4.

Whenever the PSOC sends an action character that gets followed with a parameter, it waits for the 2051 to ask for the parameter before sending it. This ensures that no data gets lost. The 2051 asks for the parameter by sending a 'P' to the PSOC when it is ready to accept the parameter.

4.3 Operational Details

These are some details about how the robot operates.

Whenever the robot is turned on it must home the linear actuator. It does this by moving the dispenser towards the limit switch until it is pressed. It then moves a few steps away from the limit switch in order to get over the first column. The robot homes after every move it makes unless it wins. The user must wait until it is done homing after every move before they can move, else the robot will not detect their move as it is not reading the photo-interrupters during homing.

The difficulty of the robot is set using the red button on the "settings console." The difficulty ranges between 1-4. The difficulty is displayed on the 7-segment display before the first move is made and anytime the difficulty button is pressed. While the difficulty can be changed during the middle of a game, the PSOC does not actually change the difficulty of its algorithm. So the difficulty must be set before the user has made their first move for the PSOC to acknowledge it.

The user can ask for hint by pressing the green button on the "settings console." The 7-segment display will display a '9' to acknowledge that you asked for a hint. After a few seconds the hint will be displayed on the 7-segment display. The hint is a number from 1-7, representing a column from left to right on the Connect Four board.

5 PSOC Game Strategy Software Description

The PSOC runs the game strategy that is behind the robot. The same algorithm that runs the game strategy is also used to generate hints for the user if asked to do so. The code for this can be found in **Appendix B**.

5.1 Negamax algorithm

The algorithm used is called Negamax. It is a variant of the popular Minimax algorithm and is used when the score of one player is the negation of the score of the other player. The Negamax algorithm searches the game tree to a specific depth. It does this recursively by calling the Negamax algorithm on the children of each node in the tree. The children of a node is simply the columns on the Connect Four board that are not full. It then calculates the score of each board state using a heuristic function. The score returned by the heuristic function is relative to the robot. This means that a large positive score is good for the robot, and a large negative score is good for the human. So if the current node that the Negamax algorithm is looking at is for the robot's move then it chooses the maximum score of all of its children. If the current node is for the human's move then it chooses the minimum score of all of its children. These best scores and best moves propagate up the recursion tree until the best move is returned.

5.2 Search Depth

The search depth of the Negamax algorithm is not constant. It depends on the difficulty setting and the number of possible moves left in the game. A higher difficulty setting increases the

search depth for all possible number of possible moves left. The search depth depends on the number of possible moves left in the game because as this number decreases, the algorithm can search deeper in the same amount of time. For example, if the robot is set to a difficulty of 4. Then the search depth is initially 4. There are initially 7 possible moves because all 7 columns are empty. That equals $7^4 = 2501$ game states to explore. The PSOC does this in a few seconds. When the number of possible moves drops to 4, then the depth can up increased to 5, which results in $4^5 = 1024$ game states to explore.

The fact that the depth is influenced by the difficulty means that any hint generated by the program will only be as good as the robot.

5.3 Heuristic Function

The heuristic function is what computes a score for a given board state. This is necessary because most board states do not have a winner or a loser. So the Negamax algorithm needs some way of telling which move is best even when no moves at the given depth produce a winner.

The heuristic function used is one that counts, out of all of the ways of producing a '4 in a row', how many times each player has a '3 in a row', '2 in a row', and '1 in a row'. A '4 in a row' is win. 'Row' in this case does not mean actual row. It counts columns and diagonals as well. In order for a piece to be added to one of these counts, it must fall in some stretch of 4 locations that does not contain a game piece of the other player. If the other player's game piece falls in the same stretch of 4 locations then you are essentially blocked on that stretch.

The robot score is the weighted sum of these counts for the robot, and the human score is the weighted sum of these counts for the human. The actual score returned is the difference between these, so that the score is relative to the robot.