Jörmungandr

Team G 2011

24778 Mechatronic Design

Carnegie Mellon University

Project: Connect 4 Player

Anton Galkin
Jon Daneman

Warren Pryde
Greg MacLean

Abstract

The following report describes Team G's experience building a robot that plays Connect Four on a stock-production game board with human and robot opponents. Team G focused design efforts around creating a stable, reliable system and conceived a completely unique mechanism for realizing game play. Despite the high mechanical complexity of the design, Team G succeeded in implementing the robot through iterative design methods and improvements, meanwhile learning about new fabrication methods and design considerations. Team G also experienced positive learning in regard to specifying software systems and in partitioning electronics. Completing our robot proved rewarding as an exercise in team project development and in professional communication, and we feel sufficiently positive about our implementation that we would only consider changes to process and timing if given a second opportunity to undertake this project.

Overall System - Front

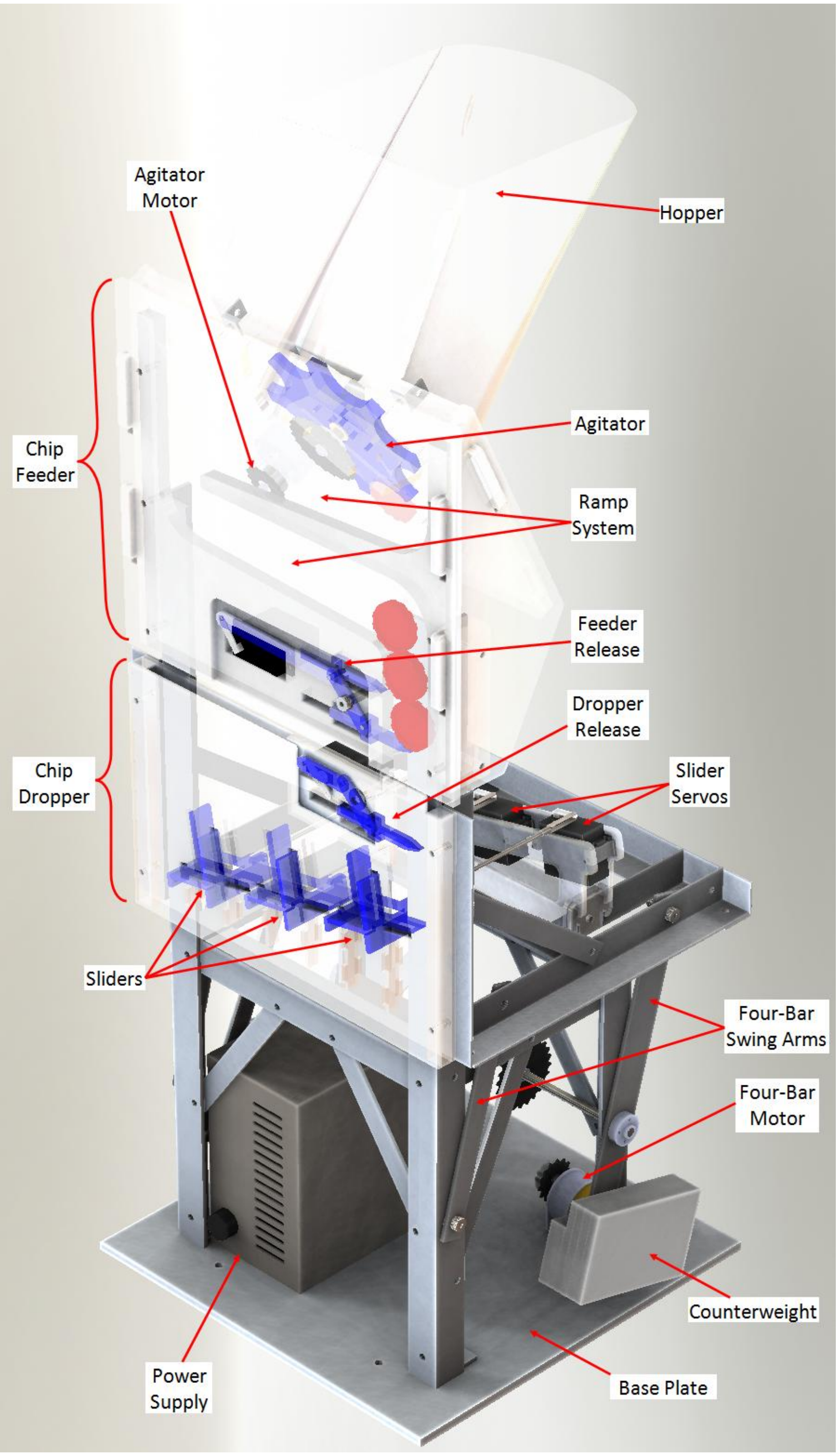


Fig. 1 - Overall System - Front View (Labeled)

Overall System - Back

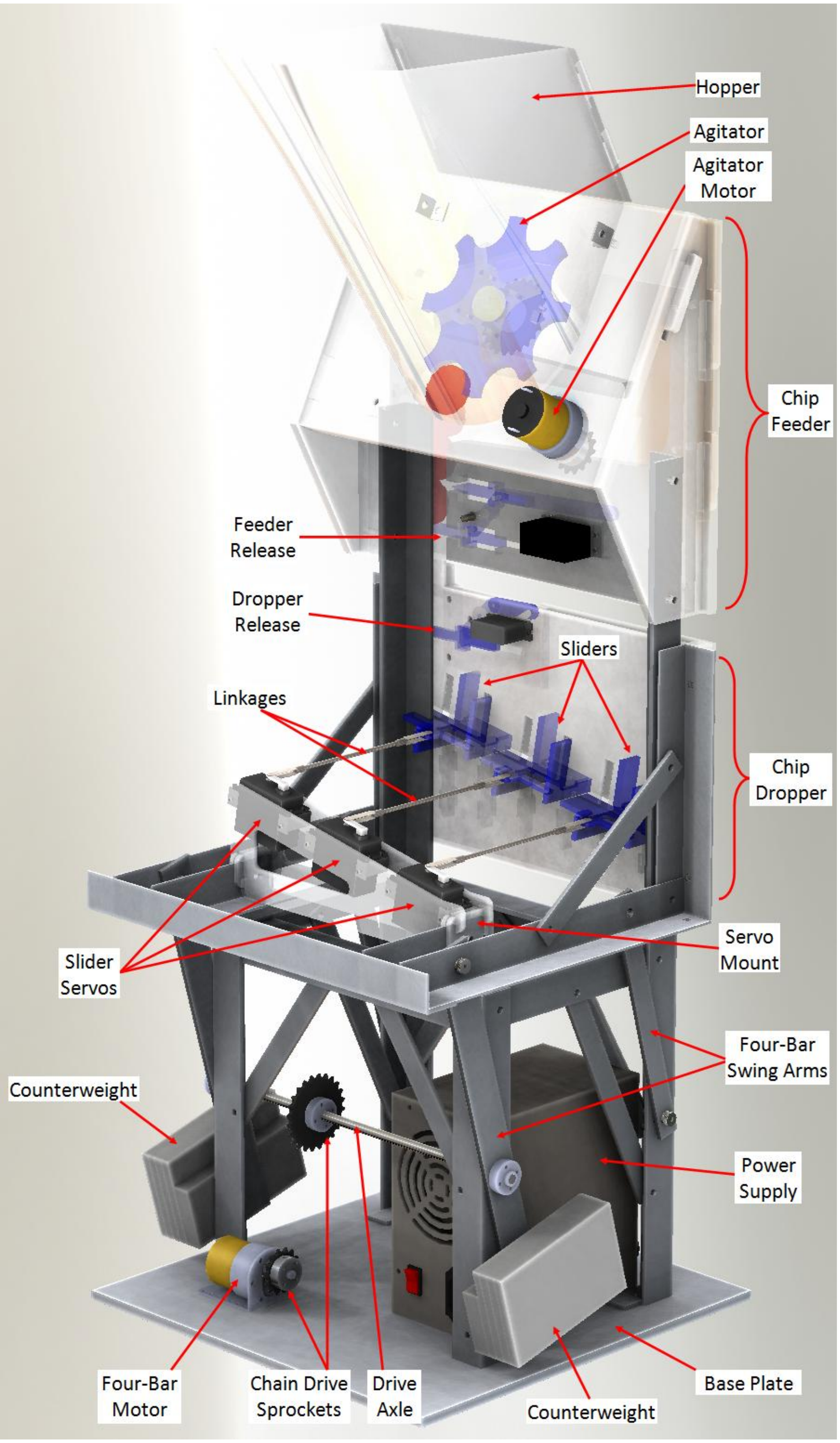


Fig. 2 - Overall System - Back (Labeled)

Overall System - Description

Our central design principal was stability. We strived from the beginning to avoid complicated encoder and feedback schemes for our positioning, instead favoring systems that had physical limitations on their range of motion. This created a system that relied little on extensive calibration to deliver the chips with reasonable reliability. This helped with the cost constraints on the project, allowing us success using the cheaper, class-provided motors and servos without sacrificing accuracy.

Our design also involved construction out of lightweight, and aesthetically pleasing laser-cut acrylic, and standard aluminum angle and bar stock, arranged to allow maximum mechanism visibility. We kept our machine footprint small and also kept the total system very lightweight. It can easily be moved by one person. The relative low-cost of these materials allowed us to iterate through design several times when necessary to optimize part fit and geometry. While this was a new medium for our mechanical engineers, they adapted to it quickly, and ended up rather enjoying the simplicity of the medium, which can move quickly from design through fabrication.

List of Desired Functionality

We were tasked with building a machine which could play Connect Four with another machine. It needed to be able to put a game chip in the right slot at least 9 out of 10 times, and within 20 seconds, and retract from the “neutral area” when it wasn’t its turn. It also needed to play smart and tell its opponent what move it made via serial protocol. We pour the chips into a hopper, press “go” and watch it play; manual intervention would be penalized. We accomplished these feats using a lot of servos, two Arduinos and some clever design.

Playing a game chip into the right column was our first challenge. Instead of grasping a chip and linearly positioning it over the game board, we took our inspiration from coin sorting mechanisms and designed a series of ramps and trap doors to guide each chip along its journey to victory. Three “sliders” form a ramp running the length of the game board, and can be slid back and forth by servos to open a trap door above the appropriate column and dropping the chip into it. If none of the sliders are actuated, the chip rolls all the way down into the furthest column. This subsystem is called the “chip dropper.”

Retracting from the neutral area is accomplished by moving the entire “chip dropper” assembly using a four-bar mechanism. It is extended over the board to make our move, and likewise retracted to allow the opponent to make theirs.

The combination of these two features allowed us to make a move within 2-3 seconds, limited only by the computation time necessary for the game logic Arduino to process which move to make. If we aligned the machine correctly, the kinematics of the mechanism ensured accurate chip dropping. Most of our failures resulted from the four-bar actuating before the chip had fully entered it, or from the chip flinging out the top from the force of impact of the four-bar extending over the game board.

Game intelligence was achieved by building and evaluating game trees to determine which current move was associated with the highest probability of future victory. This probabilistic evaluation was achieved by performing summation of scaled board evaluations within subtrees of the seven potential current moves. The board evaluation function assigns points to each board based on the number of horizontal, vertical, and diagonal win combinations present in the game state.

Serial communication was accomplished by the built-in serial ports on the Arduinos. Simply setting one register allowed us to use the prescribed 8N2 encoding, and by using the unidirectional loop topology that we chose instead of two bidirectional links, we were able to use only the buffered serial ports that the Arduinos provide. Though the Arduino software makes it easy to add software serial ports, we wanted to avoid the issues of not having buffers for those ports. Processing the serial inputs was very simple, requiring only a switch/case selector in the code on each board. Internally we developed a limited instruction set for between the two Arduinos, which is detailed in a later section.

Allowing the user to pour the chips into a hopper was our final engineering challenge, and the one which required the most testing and tweaking. The rotating pinwheel in the hopper agitates the chips poured onto it and when they fall into the cutouts around its edge, it carries them to the top of the hopper where they fall into the ramp system. The pinwheel incorporated encoder segments etched into its surface which allowed us to use a simple sensor to detect jams which prevented it from rotating, and a major challenge was to tune the geometry to the point where the system could clear all such jams itself by reversing the pinwheel rotation.

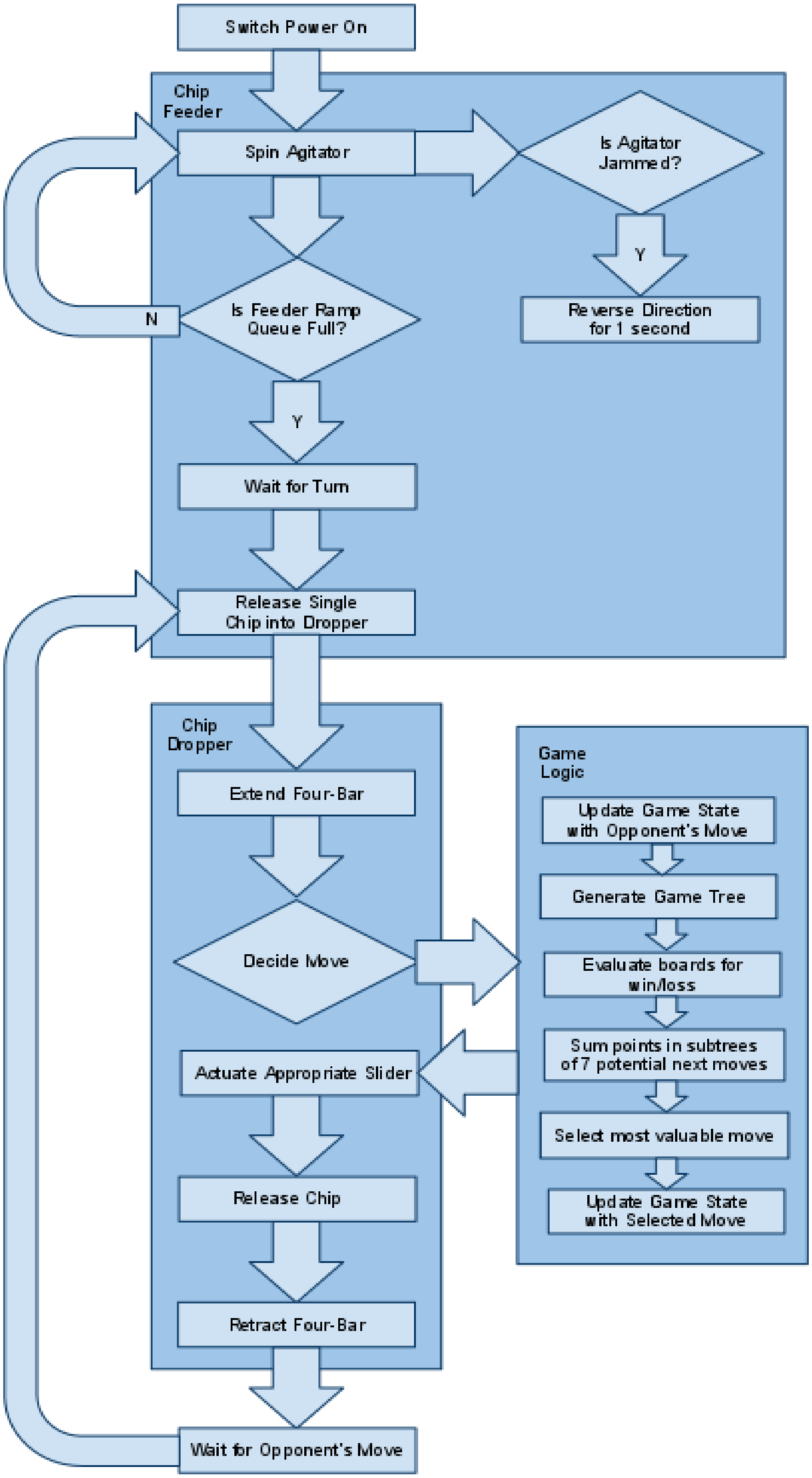
The ramp system included a short but complex ramp and guide system to move the chips from their angled orientation off the pinwheel to a vertical orientation for the queueing ramps. The queuing ramps were simply a series of several angled ramps which could contain approximately 16 chips at a time, terminating in a set of two linked sliders, which allowed us to drop one chip at a time into the chip dropper.

Our design has a number of key innovations - namely, the four-bar linkage which is used to extend the chip dropper over the board ensures perfect alignment every time with no sensing required, so long as the mechanism is lined up properly when it’s set up.

Secondly, the acrylic structure lets you see the mechanism as it works, with the appearance of chips virtually floating in air as they move through the feeder ramp queue, and specially tinted blue parts drawing the viewer to focus on the important moving elements of the mechanism.

Other innovations include the electrical system, which kept the system manageable by retaining modularity. All connections for 5V level signals were made through custom-built Molex cables. One end terminates at the sensor or actuator, and the other end at the interconnect board, which was also purpose-built for this project. From the interconnect board, more Molex cables were constructed to connect the devices to the appropriate microcontroller. This was done to allow all parts to be swappable with minimal effort. In the event any servo, sensor, microcontroller, or motor failed, it could be quickly traded with a functional unit in a matter of minutes. The handful of 12V signals were connected via a screw terminal block instead, to handle the thicker-gauge wire necessary for the higher currents of the motors. Also of note is that the controlling PC and the two Arduinos interfaced with each other through a one-way serial loop, instead of bidirectional links between each unit.

System Block Diagram


Switch Power On
Chip Feeder
Spin Agitator
Is Agitator Jammed?
Y
Reverse Direction for 1 second
Is Feeder Ramp Queue Full?
N
Y
Wait for Turn
Release Single Chip into Dropper
Chip Dropper
Extend Four-Bar
Decide Move
Actuate Appropriate Slider
Release Chip
Retract Four-Bar
Wait for Opponent's Move
Game Logic
Update Game State with Opponent's Move
Generate Game Tree
Evaluate boards for win/loss
Sum points in subtrees of 7 potential next moves
Select most valuable move
Update Game State with Selected Move

System Modeling & Analysis

Our design did not require any force modeling - engineering intuition was more than enough to determine where to use 1/8” aluminum angle for strength and where 1/16” would suffice to cut down on weight. However, we made extensive use of CAD modeling to determine proper dimensions of all parts, and ensure proper integration between interlocking mechanisms. Since most parts were laser-cut out of acrylic, the design had to be complete at the time of fabrication - little modification was possible afterwards.

This was especially difficult since different components which had to fit together perfectly were being designed and fabricated by two different engineers. However, frequently exchanging SolidWorks CAD models through email and Google Docs allowed us to keep up to date on what everyone else was doing, and make sure everything fit together well.

On the flip side, it turns out that SolidWorks isn’t magic - it is fundamentally limited by the attentiveness of its user. One difficulty we encountered after our first Chip Dropper subsystem was laser-cut was that 4 out of 7 columns were too narrow to allow a chip to pass through! Even though the columns were perfectly aligned, the spacers separating each column on our machine were much wider than those on the game board.

Overall however, integration wasn’t too difficult, thanks to our extensive preparations during CAD modeling. Sometimes a counter-sink was required to prevent the four-bar jamming, or some wires had to be routed through the mechanism, but it all came together quite nicely.

Schematics of Electrical Subsystems

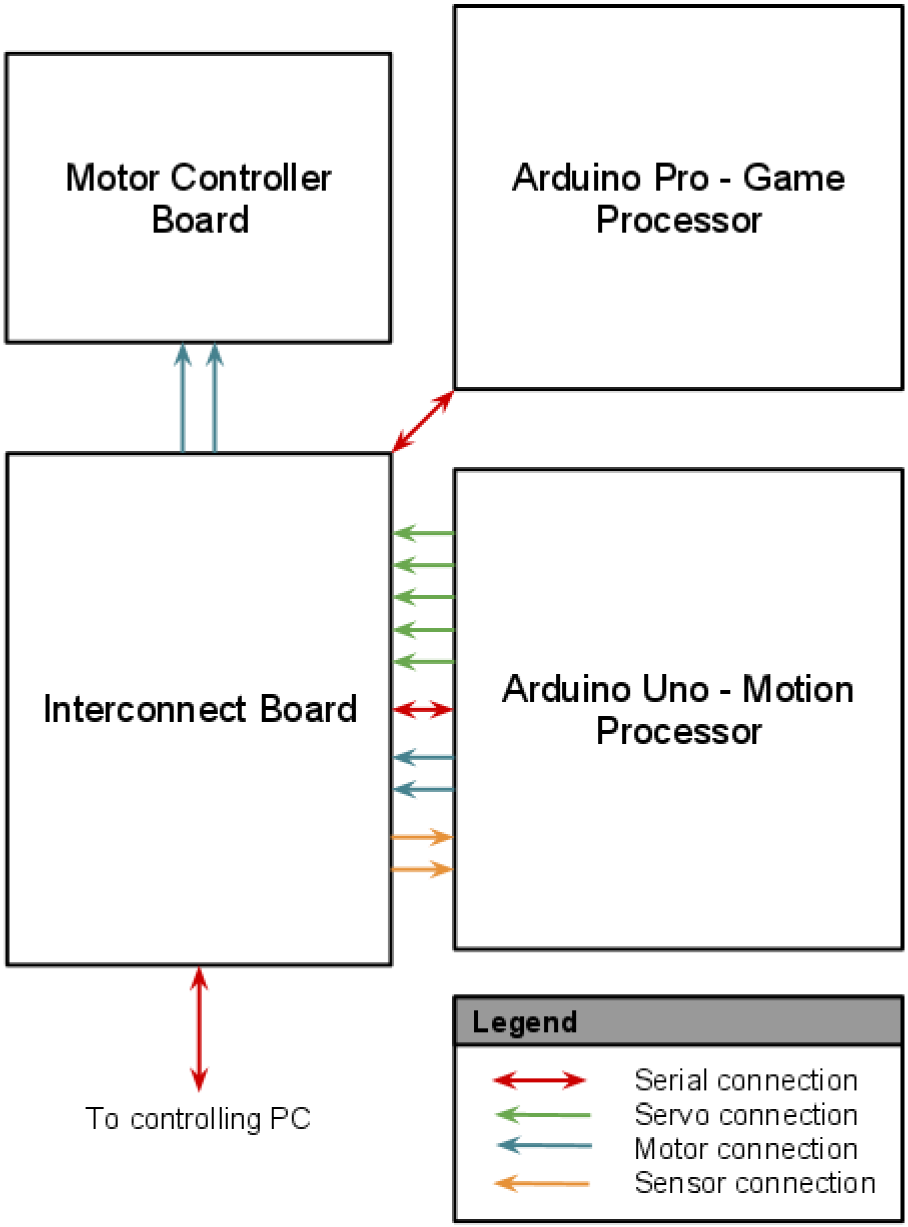


Fig. 4 - Electrical System Diagram

Schematics of Mechanical Subsystems

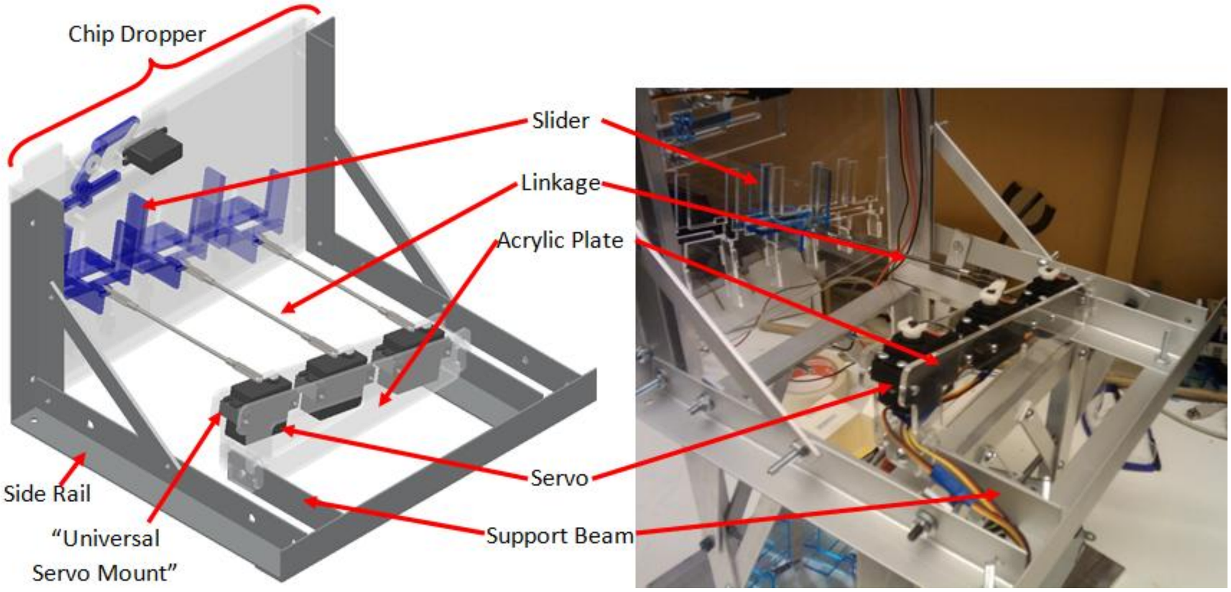


Fig. 5 - Chip Dropper Subsystem

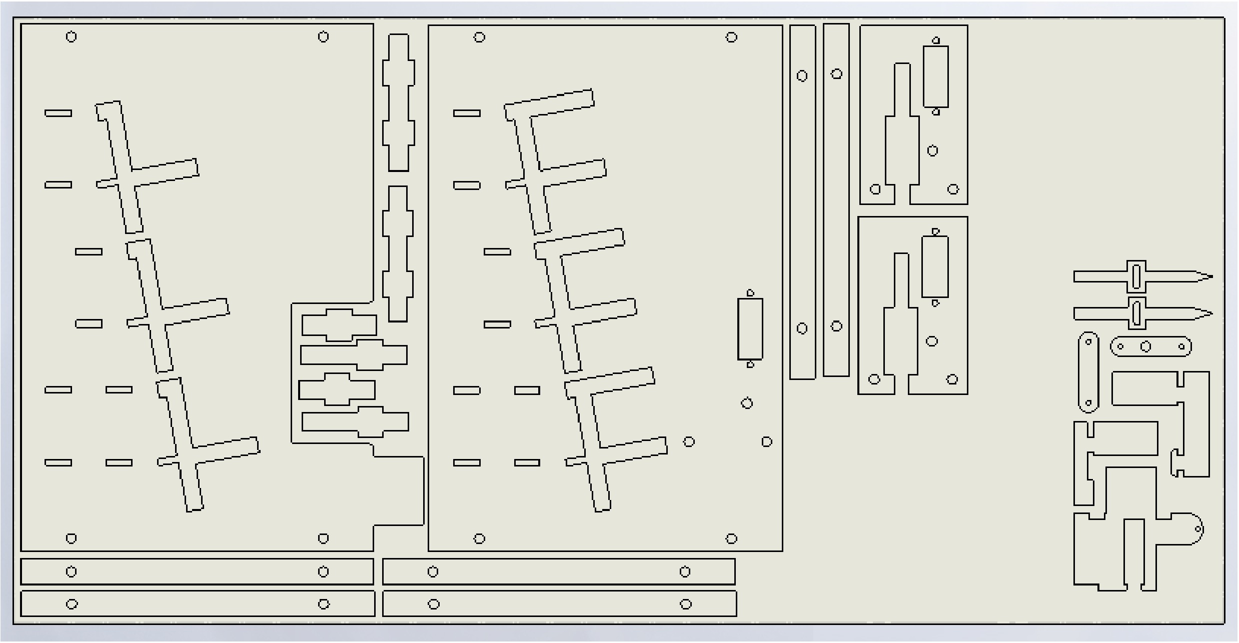


Fig. 6 - Acrylic Chip Dropper Parts Ready to be Laser Cut

The chip dropper subsystem is constructed from aluminum angle and ⅛” acrylic sheet - mostly transparent. Any moving parts are made from ⅛” translucent blue acrylic to signify their importance.

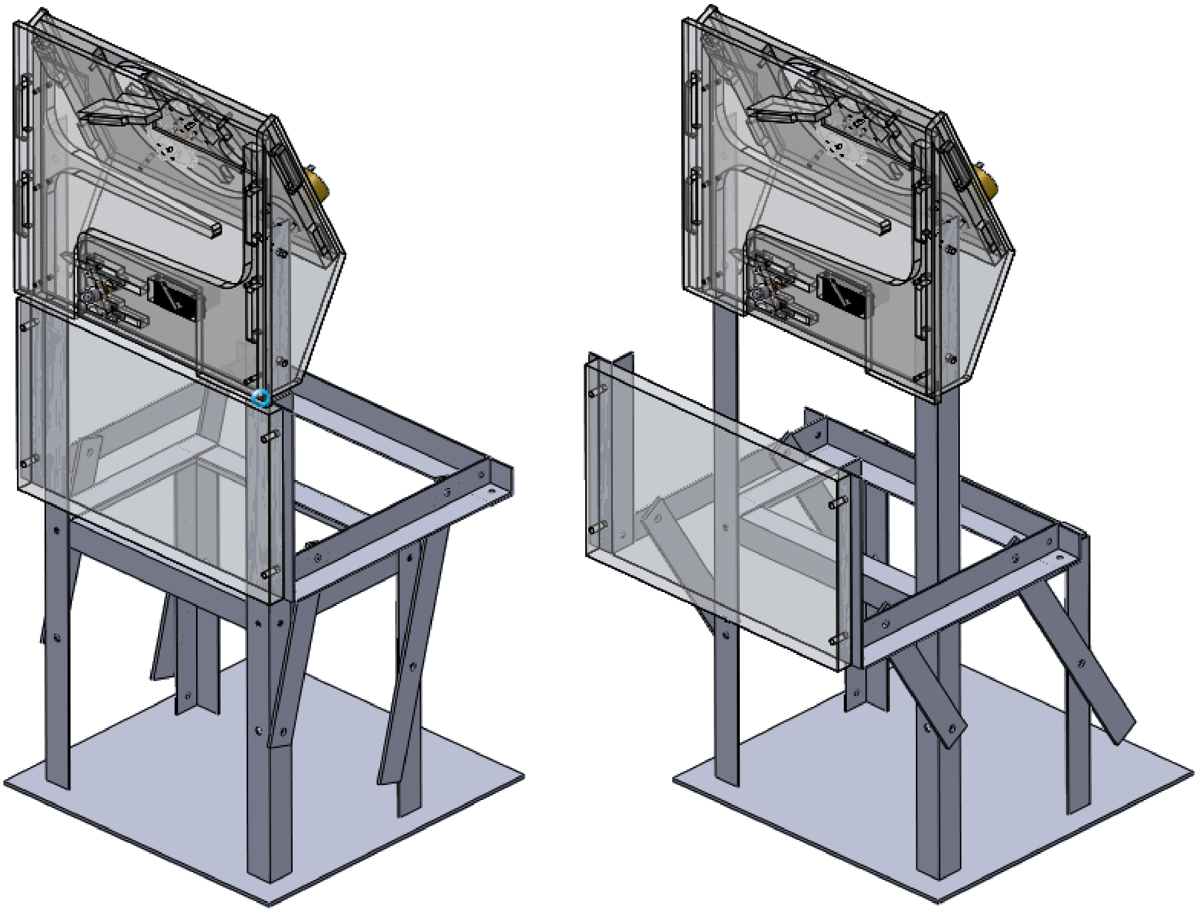


Fig. 7 - Frame Retracted (left) and Extended (right)

The Frame supported the Chip Dropper and extended it over the game board. The stationary portion was constructed from ⅛” thick, 1” aluminum angle with ⅛” thick, ½” and 1” bar stock used for the four-bar linkages. The moving portion, or Rack, which held the Chip Dropper was made from 1/16” thick 1” aluminum angle to save a little weight and hopefully reduce move time. The counterweights (not pictured; see Fig. 1) are scrap metal steel and aluminum plates and bars cut to size. The entire frame is bolted to a 1’x1’ base of ¼” thick aluminum for stability. Between the base plate, the counterweights and the power supply, the entire mechanism was kept quite bottom-heavy despite its great height.

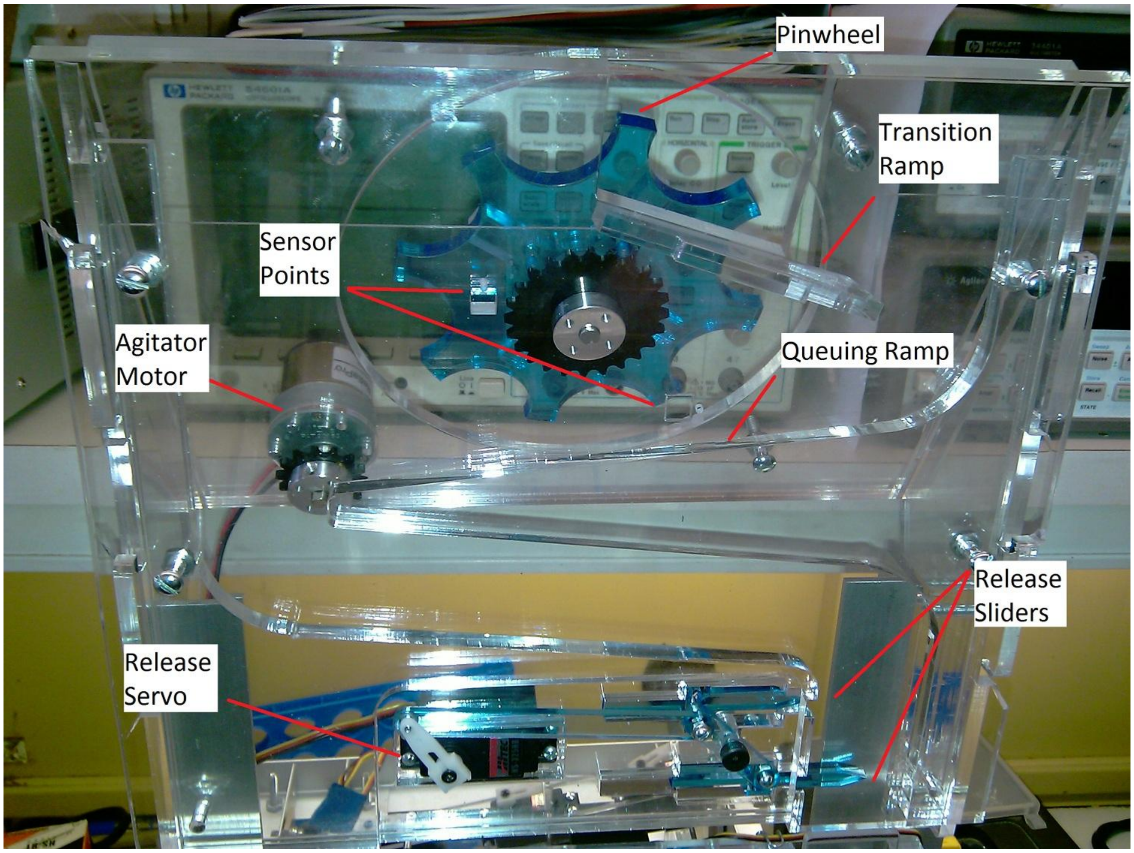


Fig. 8 - Feeder Subsystem

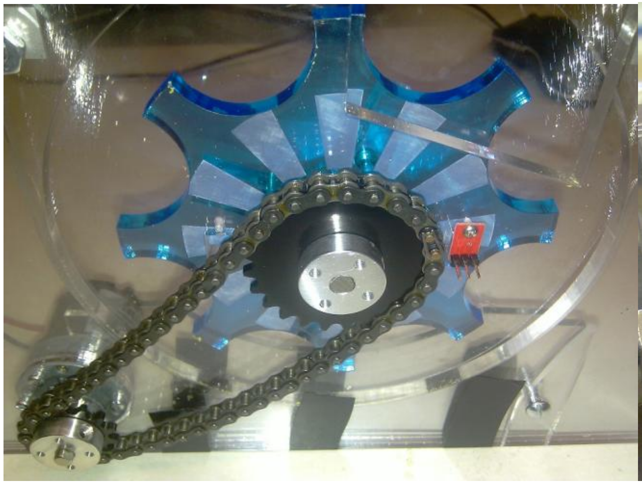


Fig. 9 - Encoder Segments and Drive Chain

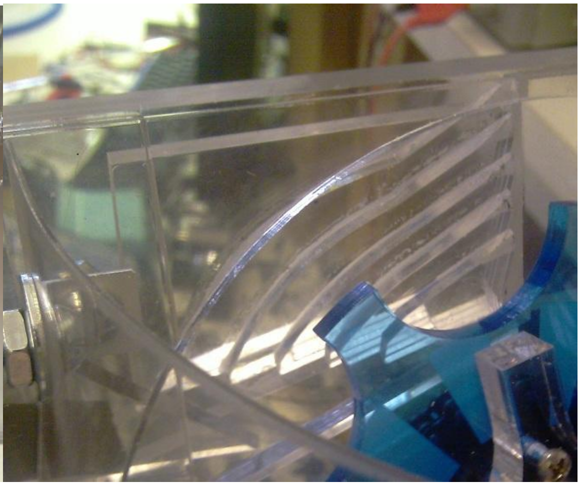


Fig. 10 - Transition Ramp

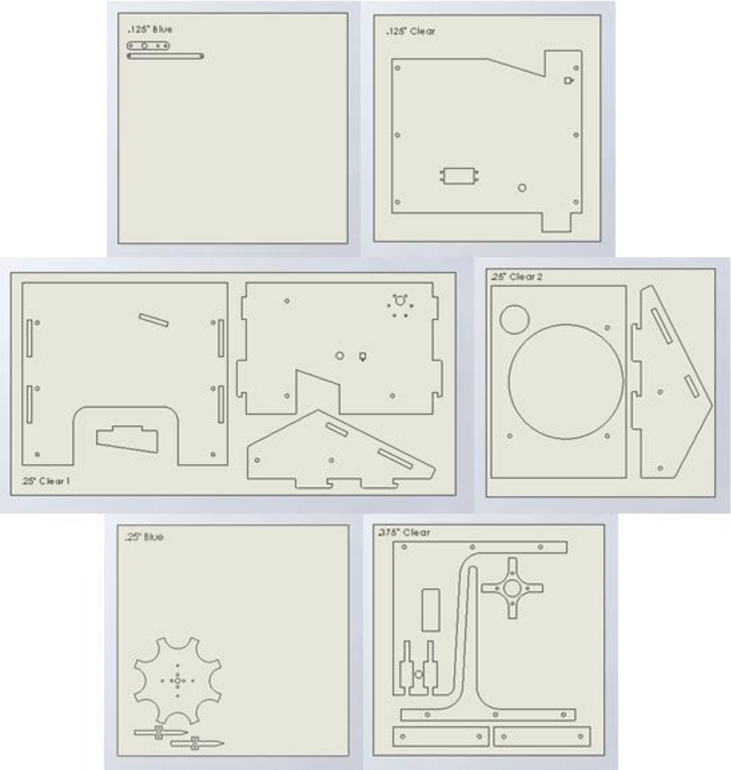


Fig. 11 - Acrylic Chip Feeder Parts Ready to be Laser Cut

The feeder was cut from acrylic of various thicknesses, depending on structural requirements and spacing requirements for chip clearance. Most component designs were relatively straightforward, including interlocking the various pieces to provide fastener-free assembly. The area that required the most work, including repeated production iterations, was the transition ramp geometry shown in Fig. 9. This had several requirements, as the system only had capability to clear jams that remained within reach of the pinwheel. Preventing jams from occurring in the transition ramp itself, where they would be inaccessible and unclearable, was the goal. This included preventing individual chips from falling flat rather than transitioning to vertical, preventing multiple chips from lodging in the area side by side, and preventing partially jammed in the ramp from jamming when the pinwheel reverses. This required a rather complex shape, which was fabricated by modeling the ramp as solid 3D geometry and cutting it as a series of sections. These, along with a small guide at the front, formed the single-chip plane change passage needed.

Flowchart or Pseudocode of Microcontroller Software

The game processor selects a column to play by dynamically building and evaluating a game tree using pre-order traversal. Following evaluation, the current move that is associated with the most valuable game sub-tree is selected as the chosen move for it's likeliness of achieving a future, if not immediate, win. There are five structures associated with move selection:
(1) A bit-board style array of integers used to store the robot's current chip positions
(2) A bit-board style array of integers used to store the opponent's current chip positions
(3) An array of seven integers for storing the height of chips in each column
(4) An array of seven integers (longs) for storing points associated with moving in a particular column
(5) An N-sized array of game boards, used for building the game tree.

The array of heights is simply an optimization used to reduce repetitive computation.

When it is the robot's turn to select a move, the following steps are performed:

```
For each of the seven next moves col:
  -Generate the board associated with moving in col
  -Store this board in index 0 of the board array
  -Evaluate the board and add point value to index col of values array
  -For each of the seven potential next moves from this state:
                -Generate the board associated with making this move
                -Store this board in index 1 of the board array
                -Evaluate the board and add point value to index col of values array
                -For each of the seven potential next moves from this state:
                             -Generate the board associated with making this move
                             -Store this board in index 2 of the board array
                             -Evaluate the board and add points to index col of values array
```

Once the subtrees of the 7 potential moves are evaluated (through scaling and summation of board evaluations in embedded loops above), the values array is then scanned to find the most valuable column, which corresponds to a move selection with the greatest probability of leading to a win. This column is selected as the move to play, communicated to the control system, and used to update the game state. This process is illustrated in the figure below. Note that only one branch of boards can be stored and evaluated at one time. Also note that evaluations are scaled by a multiplier, such that wins/losses at a nearer game state are weighted more heavily then wins/losses deeper in the tree.

One can continue embedding loops to examine deeper game states; the number of embedded loops directly corresponds to the depth of game tree evaluation. We successfully embedded 11 loops, but only completed testing on 8 embedded loops, for we realized the Arduino could not handle deeper than 5 levels of game tree evaluation using this scheme.

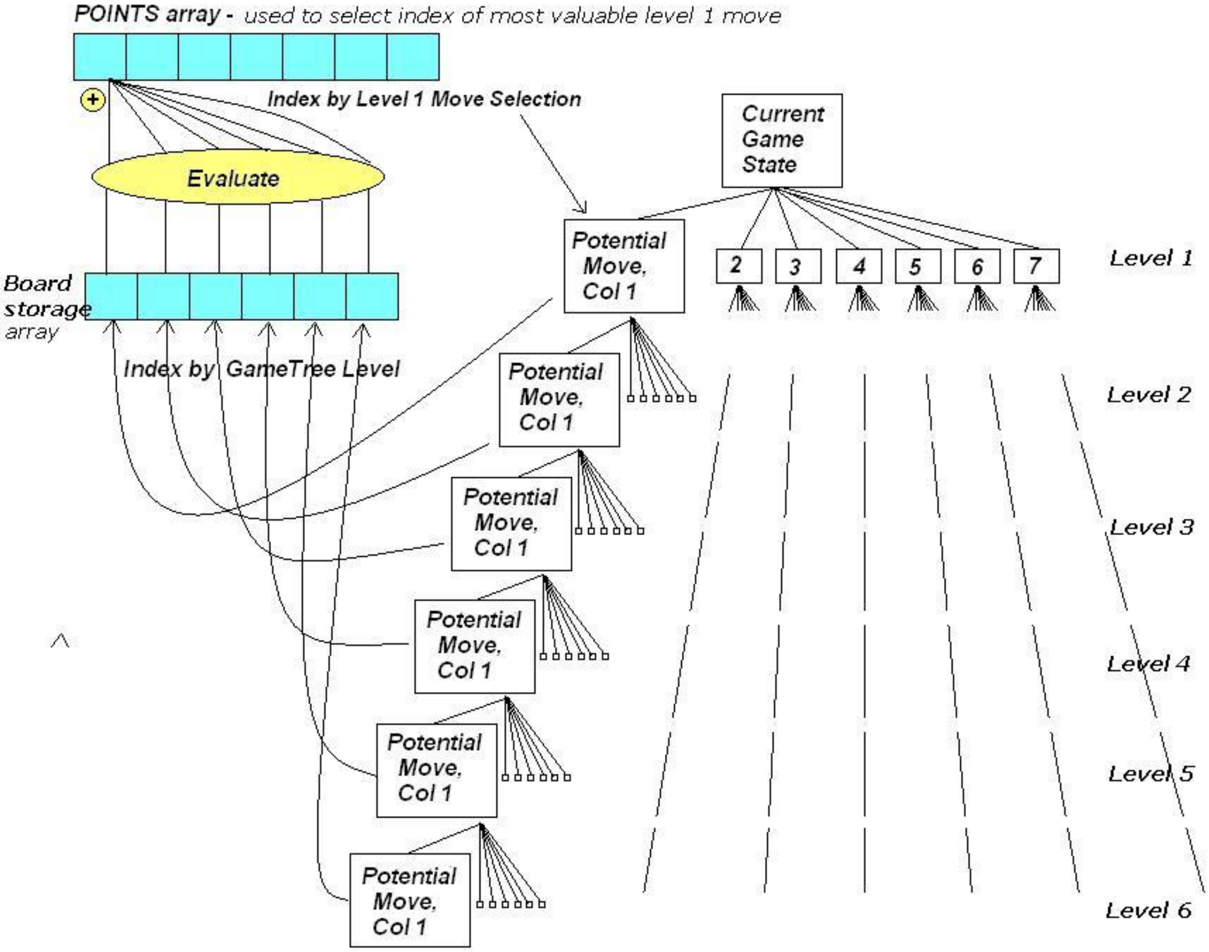


Fig. 12 - Illustration of game tree

The motion microcontroller had a fairly straightforward program flow. At the beginning of each loop iteration, the Arduino polls the reflectance sensor on the agitator wheel and compares it to the previous value. If it detects that the agitator is stalled, it reverses motor 2 for one second before resuming the original direction to clear any jam. The method that handles that also checks the value of gameBegun to determine whether the agitator should be active, and checks the value of chips to see if it is okay to stop the agitator after the game has begun. Following that subroutine, the Arduino checks to see if there is any available data in the buffer from the serial input. If there is, it reads in the characters one at a time and acts accordingly to whatever command it received, using the command set described in Table 1 below.

The other major subroutines are makeMove() and nzc(). The makeMove() function takes an integer input and activates the proper servo to open the corresponding column selector to place the chip in the appropriate column, as determined by the game AI. The nzc() function simply takes 1 or 0 as input to determine which direction to move the four bar linkage. It signals motor 1 in the proper direction for one second before braking the motor.

Command	Action
!	haltAll()
1	makeMove(1)
2	makeMove(2)
3	makeMove(3)
4	makeMove(4)
5	makeMove(5)
6	makeMove(6)
7	makeMove(7)
s	onSetup = 1; gameBegun = 1;
S	onSetup = 0;
T	nzc(1);
.	gameOver()

Table 1 - Motion microprocessor serial command set

Parts List & Cost

Source	Part #	Description	QTY	Unit Price	Total Price
McMaster	4630T15	anodized aluminum angle, 1/16" thick, 1"x1" legs, 6' long	1	12.68	12.68
	8560K257	1/8" 12"x24" clear cast acrylic sheet	1	13.47	13.47
	85635K473	1/8" (0.118") 12"x24" flourescent blue cast acrylic sheet	1	11.89	11.89
	9246K13	12"x12"x.25" 6061 Plate	1	19	19
	88805K48	1"x1" 1/8" Aluminum angle, 8'	1	18.54	18.54
	9414T6	1/4" collar	4	0.6	2.4
	1327K68	1/4"x24" shaft	1	12.04	12.04
	6261K282	ANSI25 chain, 2ft	1	7.28	7.28
	6261K108	ANSI25 connecting link	1	0.89	0.89
	89755K23	6061 bar, 1/8"x1"x6'	1	7.99	7.99
	91259A535	Shoulder Screw, 1/4"x3/8"	6	0.98	5.88
	91259A539	Shoulder Screw, 1/4"x5/8"	1	1.01	1.01
	6261K281	ANSI25 chain, 1ft	1	3.64	3.64
	6261K108	ANSI25 connecting link	1	0.89	0.89
	90275A247	10-24 flat-head machine screws (100 pk)	1	5.04	5.04
	6338K411	1/4" ID, 3/8" OD, 1/4" long flanged sleeve bearing	2	0.6	1.2
Servocity	BTR-7006	90 Degree Universal Servo Mount	3	1.95	5.85
	GPMQ3795	4-40 Metal Clevis Linkage (pack of 12)	1	6.99	6.99
	98847A005	4-40 12" Stainless Steel Threaded Rod	1	1.95	1.95
	RHS-250-14	1/4" 14t hub mount sprocket	2	5.48	10.96
	RHS-250-24	1/4" 24t hub mount sprocket	2	6.28	12.56
	3463H	1/4" set screw hub	4	4.99	19.96
	3472H	6mm set screw hub	2	4.99	9.98
	AGB-A2-8	1/4"x1/4" bushing	3	1.45	4.35
Delvie's Plastics		1/4" clear cast acrylic, 12"x24"	2	13.75	27.5
		3/8" clear cast acrylic, 12"x24"	1	16.75	16.75
		1/8" clear cast acrylic, 12"x24"	2	7	14
		1/4" Transparent blue cast acrylic, 12"x12"	1	7.75	7.75
Sparkfun	DEV-09219	Arduino Pro 328 - 5V/16MHz	1	19.95	19.95
	DEV-09716	FTDI Basic Breakout - 5V	1	14.95	14.95
Salvaged	HP-P2507F3CP	Dell ATX 250W Power Supply	1		
		Perfboard	1		
Provided		Various nuts & bolts, wiring, MOLEX connectors	0		
		JameCo ReliaPro DC Motor w/ gearbox	2		
	HS-325	HiTec Servo	4		
	HS-81	HiTec Mini Servo	1		
		3 shared packages + 2 packages		Shipping	35
				TOTAL	332.34

Table 2: Parts List & Cost

Performance Evaluation

Our systems did not always perform as well as we expected, so we tweaked and adjusted them until they did! Our metrics for system performance are usually quite binary: either it works, or it needs to work better. Actually, a lot of systems worked better than expected the first time:

- the chip release mechanisms in the chip feeder and chip dropper
- the servo mount
- the sliders and their linkages
- the four-bar mechanism
- the chain drive for the four-bar
- ...and many more.

Some systems required more tweaking than others:

- the hopper enclosure was too small, allowing chips to fly out of it
- the hopper enclosure was too vertical, creating a wedge where chips could easily jam
- the agitator was too small for the final hopper enclosure configuration, allowing chips (often just one chip) to rest at the bottom of the hopper, untouched
- the feeder transition ramp would often get jammed with chips
- the feeder ramps were not steep enough, allowing chips to get stuck on the ramp
- the walls of the sliders were too close together, leaving a small gap for chips to roll through
- 4/7 of the chip dropper columns were too narrow, not allowing chips to pass through
- some bolts sticking out of the frame interfered with the four-bar linkage
- ...and so on

During the competition, our mechanism performed quite well - the only failures were due to an over-eager four-bar linkage which flung one or two chips out of the machine, and a few bolts which had come loose from vibration during transport.

Our AI did not perform competitively, but it was written by our own team and embedded in a microprocessor, as specified in the course requirements, and we’re proud of it! We robustly tested and verified 8 levels of game tree evaluation. We began expanding the logic to embed 11 levels of evaluation but halted upon realizing the Arduino could only handle 5 levels within the time specification. The Arduino selects a move in an approximately 6 seconds when considering 5 levels of evaluation. The logic successfully blocks 100% of threats and plays 100% of wins.

Problem areas leading to innovation

Setting aside simple fixes like a counter-sink or support bracket, the main area of innovation is probably the Chip Feeder subsystem. Every step of the way was a challenge:

- preventing the agitator from jamming the moment 21 chips are poured into the hopper required careful design of the hopper enclosure, which consisted of a 0.05” thin sheet of PTFE bent into a curve, designed using the SolidWorks sheet metal toolkit. The angle where the two met at the bottom was of critical importance.

- The transitional geometry linking the hopper to the ramp system presented a unique challenge: chips needed to be oriented in an orderly way so they could fit into the ramp entrance and not jam up against the edge of an acrylic plate. They came in at a ~20° angle to the vertical and needed to end up completely vertical, but their tendency of course is to fall over horizontal. Thus, we created a 3D geometry which left a very small gap through which the chips could fit, a path which used the kinetic energy imparted to them by the spinning agitator to force them upright. If they did not fit, they simply bounced out.
- The chip dropper column spacing was an issue that came up after the first iteration of that subsystem was fabricated. It turned out that our column spacers were so wide, that conforming to the other dimensional constraints of the game board resulted in most of the columns being too narrow for a game chip to fit through! After trying some different options, the best one seemed to be leaving a gap below the column spacers and aligning the center-most column only. This way, even though the outer columns were misaligned by a fraction of an inch, there was room for the chip to bounce a little left or right and consistently make it into the correct column, but not enough space that it could bounce into an adjacent column.
- The design of the “sliders” was especially challenging, considering the very small space they needed to fit into: there was no room for mechanical fasteners or standard components of any kind! Nevertheless, they had to hold together firmly and slide smoothly.

  The sliding was accomplished by taking advantage of the smoothness of laser-cut cast acrylic: part of the slider structure was simply used as railings, while an appropriately sized gap was cut into the acrylic plate supporting these sliders to act as the sleeve of a sliding bearing.

  The assembly was accomplished using pressure-fit components; previously glue was used, to ill effect - pieces would eventually come apart from chip impact. Each interlocking part was designed with a small amount of overlap at joints, which made for a snug fit between all components that was enough to hold them together. This allowed assembly of the sliders without any intrusive fasteners that might interfere with chips rolling down the ramp.

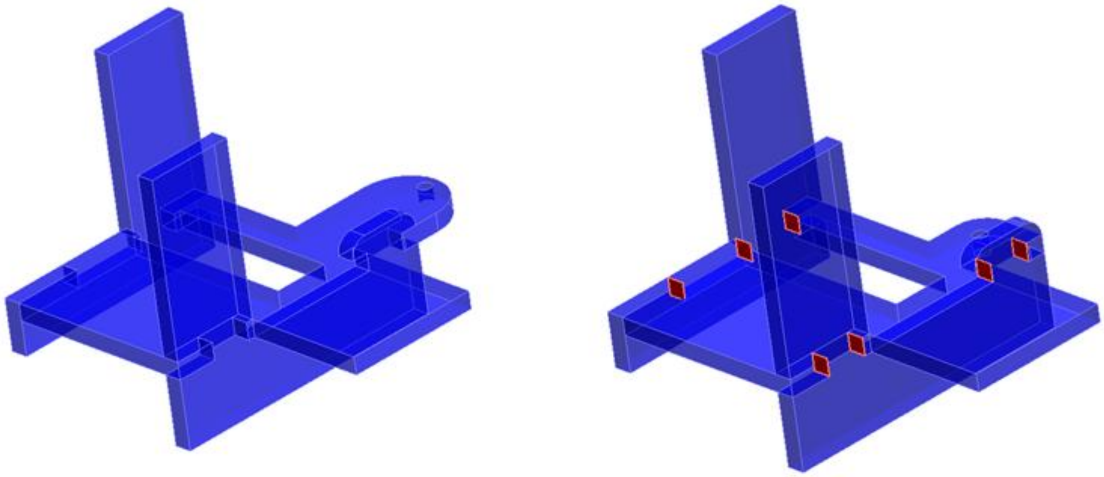


Fig. 13 - Slider design CAD, old and new; pressure-fit interference marked in red

Implementing the game design was straightforward, but there were two major constraints on achieving higher levels of intelligence: available memory and computation time. We initially considered pre-computing game boards, but the amount of game board variations far exceeds the storage capabilities of the microcontroller. We soon found that the available memory for storing small trees of boards was limited, and we decided to move forward with the dynamic algorithm previously described. Though the AI implementation could be scaled to evaluate any specified number of levels, the speed of computation was limiting. Our algorithm could successfully evaluate 5 levels within the time constraint, however, deeper evaluations required significantly more time.

The remainder of the electrical system also provided few problems. The main limitation we reached was with the number of available digital output pins on our Arduino. After we calculated our needs in this respect, we found we were already one over the number we had available. We dealt with this by purchasing an additional board to share the load. The second board ended up with only two connections, but combined with a reduction in the number of pins needed, made everything work better. This also ended up helping by letting us use one board to run the AI and the other to run the control system. It is likely that if this hadn't been done, we wouldn't have been able to look as deep into the game tree, and we also wouldn't have been able to poll our sensors often enough to control the agitator properly as the game logic would have consumed all resources for several seconds at a time.

Conclusions

Overall, we would not do much different if we had to do it a second time - we'd probably just make the same mechanism, with some refinements and additional capabilities like game state sensing and display.

We learned a lot over the course of this project... certainly how to design and fabricate acrylic components! Coding the AI provided a substantial review of C programming and game theory. It also provided a good opportunity to fully specify and develop a code project without starter material. The Arduino was easy and fun to work with and seeing the mechanism finally come to life built confidence in our ability as engineers to make something from nothing.

It's hard to quantify the difficulty rating of design and implementation, but we tried (see Table 3).

	Anton	Greg	Jon	Warren	Average
Design	9.5	8	7	8	8.125
Implementation	8	8	9	8	8.25

Table 3: Difficulty Rating of Design and Implementation

Recommendations for Future Work

We would not recommend starting a business that manufactures and sells connect-4 playing machines, really don't think there's a market out there for that sort of thing.

However, if we were to engage in this endeavour, some future improvements might include:

- more robust hopper & agitator operation, including sophisticated routines that would adjust speed to help get that last chip from the hopper to the ramp system.
- potentiometer sensing and PID control of four-bar linkage to reduce force of impact at the end of each extension & retraction motion
- use a small computer motherboard for game logic processing to allow for deeper game tree exploration and more optimal move selection
- allocate more time for testing game play on the boards to explore microcontroller constraints and optimizations
- implement game tree pruning and try parallelize computations, perhaps by building and evaluating distinct subtrees in parallel during move selection.

References

N/A