ME 170C Final Report

March 16

2009

A comprehensive discussion into the design methods, strategies, successes and failures of Robot Design Team 8

1 Final Robot Design

Building a 72 point, cheese grabbing robot is not a one iteration design process. It comes about through multiple design iterations, sampling different mechanical, electrical, and software methods to grasp a better handle on predicting the behavior of autonomous robots. Through the duration of this class, multiple design iterations of mechanical, electrical and software designs were performed to produce the final design of Team 8's robot.

Figure 1shown below is a picture of the final robot for Team 8. It incorporates multiple mechanical, electrical, and software systems to operate. This report documents the development and design paths that led to this robot's creation.

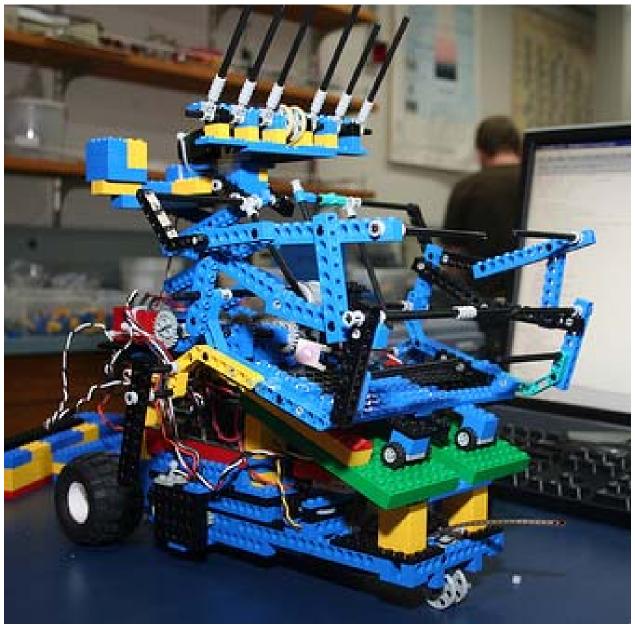

Figure 1 - Final Robot Design

By the end of this report, a vast knowledge of Team 8's design process will be conveyed, the pitfalls and triumphs the team experienced will be described, and some advice for future students will be presented.

2 Mechanical Systems

The mechanical systems of this final design came about through multiple design iterations. There are three important mechanical designs on this robot that will be addressed. These include the drive train, the cheese collection system, and the steering system. These mechanical systems provided the robot the mobility and ability to meet the designer's goals without the intervention of the designer during runtime.

2.1 Drive Train

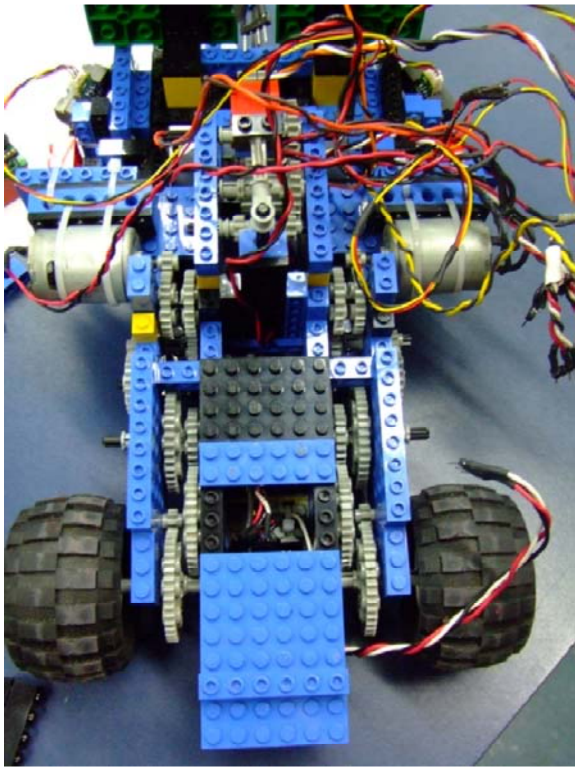
Figure 2 – Final Robot Drive Train, with motor mounts clearly shown near top

The drive train of any robot is a critical. This provides the robot with the mobility to move around and get beyond the starting grid. Without a good drive train, a robot can barely function. A drive train built for speed can overwhelm the electrical feed-back systems of the robot, while a drive train build for power may be too slow to keep up with other robots.

The drive train that our robot used was an independent rear wheel drive system. This system was comprised of a DC motor with a Lego gear attached, mounted onto the robot to act as a drive input to turn the drive train spur gears which turn a wheel. Both rear wheels had their own drive train and motor to operate independently. Two important things that Ryan learned from this drive train design were that correctly mounting a motor and designing a chassis around that motor and drive train is CRITICAL to building a workable robot. Secondly, mounting a motor with a gear attached to it to interface with another gear directly is INCREDIBLY difficult, however, if done correctly, can provide a VERY reliable drive train system. This lesson was learned by rebuilding the chassis around the drive train four times. The final gear ratio of this drive train was approximately 1:375, meaning that for every 375 revolutions of the DC motor, the wheels will make 1 revolution.

The drive train ratio was actually was a design mistake, since it had been determined that a gear ratio of 1:150 to about 1:200 provided a good speed and enough torque for the robot to move around. This design mistake occurred, however because an extra 1:3 gear reduction was put in and build around. By the time the mistake had been realized the competition was too soon to change drive trains! Had this mistake been corrected, the drive train would have run at a 1:125 gear ratio, providing a fast speed, but a dangerously low level of torque.

Determining the “sweet zone” for gear ratios came about through multiple design iterations throughout the quarter, during multiple assignments. Prior to the use of this robot, a prototype robot built for the solo competition had a gear reduction of approximately 1:100. The

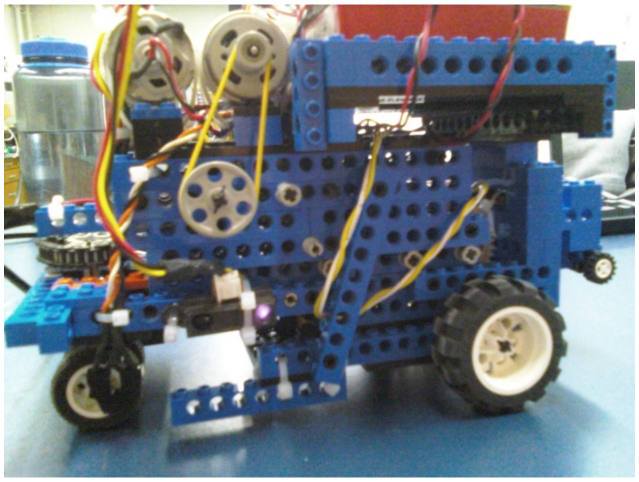
Figure 3 - Wall follower robot with one side of differential coupled motor, motor belt drive, and wall sensor clearly shown

drive setup, which closely resembled the drive setup in Figure 1, except the two big wheels led instead of follow, proved to be very fast, but had little load carrying capability, and had difficult metering the speed to a slow pace. That drive train also suffered from the tail-drag turn, which made the tail of the robot try and spin the robot around, eventually putting it into a hard spin.

The tail dragger drive system and an independent drive system had been chosen after encountering problems with an older design used for the wall following competition, shown in Figure 3 and Figure 4. The drive train used in this design incorporated a coupled rear wheel drive system, where a differential coupled the two DC motors together, powered a single drive train, and turned a differential connected to the rear wheels. This design required different a different type of steering, discussed in Section 2.3. This drive system was used to explore how effective a differential drive system would be. This drive system worked very well, especially for its purpose of moving quickly around a vacant track, since both motors powered both wheels. The drawbacks noted of this drive system were only a low weight capacity, which Team 8 understood would be critical when designing a cheese grabbing robot. Also, the use of rubber bands to connect the motors to the drive train gave very smooth operation and quite running times, but under heavy loads would force the rubber bands to fly off, giving an unpredictable state to the drive train. Finally, coupling DC brush motors to a free wheeling differential can be disastrous. This is because if only one motor is running, it will “free wheel” the other motor, effectively spinning the other motor backwards instead of turning the differential to move the drive train. This occurs because of the far less resistance to free wheel the other motor. Also, if the motors spun backwards of one another, the differential wouldn’t turn. This meant that “resting” a motor while the other motor operated was out of the question – a considerable design factor that would help extend battery life. It also meant that some knowledge of differentials needed to be present with the user working with this drive train.

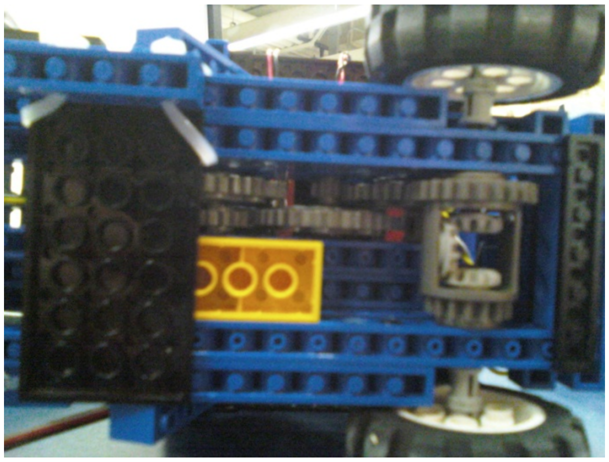
Figure 4 – Undercarriage of Wall Follower Robot with differential drive output clearly shown

Prior to the wall follower design, the robots were driven using the rear-wheel independent drive system. This was done for two reasons. First, almost everyone else was using them, and secondly, they were simple and pretty effective at getting the job done. Most people were focusing on understanding other concepts of the robot at this point in time, so looking into drive trains this far back into the class wasn’t even considered.

The robots designed prior to the wall follower design, did not have Ryan Sass and Rusty Harrington working together on a single team. These two team members were a part of different teams, and joined their knowledge from trying different drive train designs to start work on the wall follower’s coupled drive train.

Prior to Ryan and Rusty coming together, Rusty was working with Group 8’s original team on a robot used for the bump and avoid, as well as the line following assignment that used an independent rear wheel drive system that incorporated chain links to attach the motor to the drive train. Aside from learning the importance of gear ratios for a drive train, which Rusty seemed to discover quite well, Rusty also learned the importance of having a connection, such as a chain link, to connect motor to drive train. The importance of this design did result in extra friction and force to overcome chain slack; however this also meant a simpler method for mounting the motors near the drive train to get them to work.

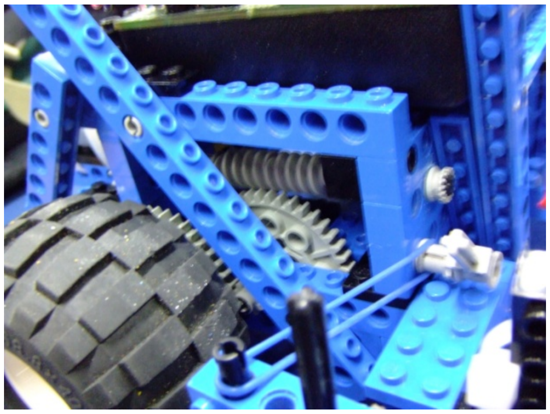
Figure 5 - Drive Train of the bump and avoid robot

Meanwhile, Ryan was a part of Team 5, building first a robot that used a worm gear drive system for his bump and avoid robot. Although warned by the teacher that this system shouldn't be tried, Ryan was interested in why this was, and decided to build a worm gear drive anyway.

From building this drive train, Ryan learned, first, that worm gears are terrible for high-speed motors, since the frictional wear associated with these gears creates a lot of plastic dust. Secondly, because the worm gear could not be mounted perfectly, there was gear slip, making the high torque that the worm gear could produce almost worthless. Finally, he learned that although figuring out things for himself are important for the learning process to occur, it isn't as efficient as listening to the professor.

Ryan then changed to a spur gear drive train but powered by connecting the DC motor to the drive train through a rubber belt. This drive train was used during the line following assignment. This showed Ryan that using rubber bands or different connectors can provide effective ways to attach the motor to the drive train, provided the drive train isn't overloaded, which resulted in the rubber bands flying off.

Through this colorful design history of multiple drive train construction and destruction, Team 8 learned a great deal about what drive trains are effective and reliable. In the end, for our final design, we chose to go with an independent drive train because of its simplicity, proven reliability, and high power capacity. The only suggested improvement to our final drive design would be to remove the 1:3 initial gear reduction, changing the overall drive train reduction from 1:375 to a 1:125 ratio.

2.2 Cheese Collection System

To collect cheese during the final competition, a design using a fork-shaped device on-top of a scissor-lift was used to knock down cheese and a removable basket was employed to catch and then deliver the cheese onto the home wall. No ground cheese was considered to be obtained, because of its low point value, resulting in an undesirable "cost to profit" ratio.

2.2.1 Scissor-Lift

The scissor-lift design started to come to fruition when the strategy to only obtain the hanging cheese was considered. After asking the Professor the balls-to-the-walls question: "Do you grade on performance or effort?", and obtaining the unbelievable but awesome answer "effort", the strategy to design a system to put cheese on the wall meant that any sort of grabbing mechanism needed to fit within a compact space. A scissor lift design was purely chosen because the idea literally popped into Ryan's head one evening while playing with Legos. At first he felt maybe a claw could be designed, but he soon realized a scaffolding type lift could probably be built to put a cheese "swiper" near the hanging cheese to knock them down.

After some initial design changes, including a bulking up of the structure that supported the entire system at its base and at its top, an effective and reliable cheese system had been designed, with only two design iterations. The drive train built to pull the scissor lift in and out had to incorporate a string, which potentially violated the rules, but since the string's purpose served no malicious purpose to any other robot (and the scissor lift looked pretty cool), Ryan decided to shrug his shoulders at the rules and continue with the design. The scissor lift needed a drive train, and since 24 tooth and 8 tooth gears were incredibly abundant, a drive train with a gear reduction of 1:54 was used. This drive train was a WONDERFUL place to use the ancillary DC motor, since the number of gears used created enough friction to prevent back spinning of the motor, and because the lift never needed to be collapsed after its deployment during the competition, it worked out great.

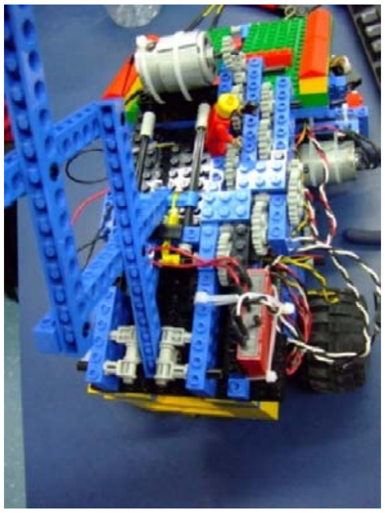
Figure 6 – Scissor Lift base with running track, drive train, ancillary DC motor, and mechanic clearly shown

The entire system was built as a modular device, especially since it was mounted on top of the handy board, and this meant that access to the handy board without a modular device would be tedious and annoying. The only drawback of the system was that the gears needed to be left unengaged (i.e. one cog pulled out) so that any accidental spinning of the motor (like when it switches on) would not force the lift to retract further. This occurred a few times, and literally pulled the Lego pieces apart! What was important about this aspect that I learned a lot from is that designing something to be modular within a complex system makes overall design and day-to-day operation of the device much simpler and easier to work with.

2.2.2 Fork

The fork's purpose was to force hanging cheese to break its connection with its magnet and fall towards the robot. To do this, a fork system being operated by a servo was used. This was initially considered in order to fit the entire robot within the 1 foot height requirement. The gearing setup for the fork worked well, using rubber bands to connect to the drive train, which was attached to a servo. Several problems resulted from this. First, and most importantly, the servo malfunctioned – ON COMPETITION DAY, of course. Secondly, the rubber band design was not strong enough overcome some of the forces exerted on the fork when swiping at cheese. This resulted in breaking the fork during the competition as well. A re-design of the fork using counter-weights and a back stop instead of a servo not only gave the fork reliability and the ability to compact itself into a 1 foot tall robot when put into a box, but it also lightened up the load on the scissor lift, which already had a little bit of a tilt going to it. The important aspects learned form this design was that servos can't be trusted, and that backup

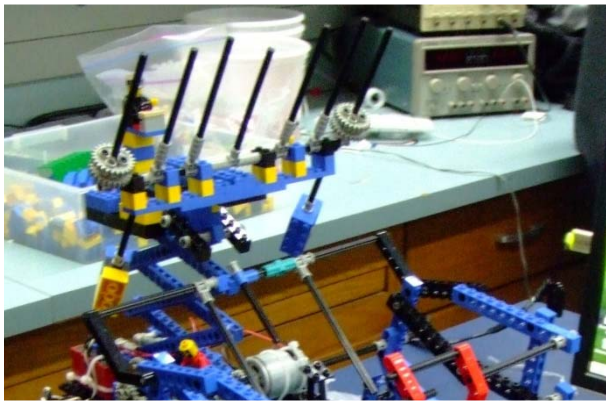
Figure 7 – The fork attachment, located on the top of the scissor-lift

systems should be incorporated into something as critical to a design as a cheese swipe.

2.2.3 Basket

The basket, shown Figure 1 front, was the key point to the entire cheese-collection system working correctly, and this part of the design encountered many problems and flaws. Because the strategy of putting cheese on the wall was chosen, a design using a removable cheese basket was incorporated immediately. The basket, whose design didn't change much, except for a few extra spars here and there to prevent "cheese leaking", used special hanging arms to stop itself from rolling off the table once placed there. The basked was placed on a ramp, shown Figure 8, with wheel chalks that held the basket in place until deployment time. When the basked needed to be deployed, the basked was tilted upward, raising the chalked wheels over the chalks and rolling the entire basket down the ramp, and, hopefully, onto the wall.

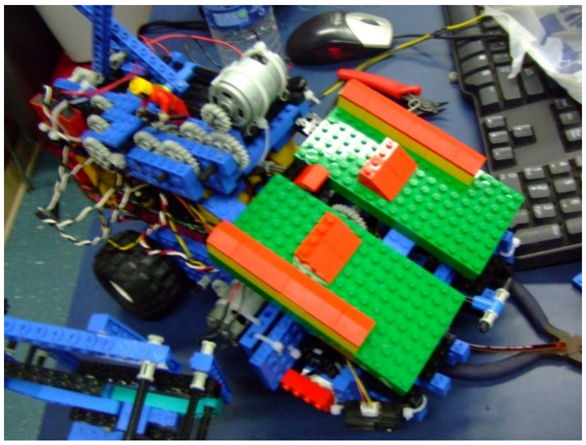

Figure 8 – Basket Ramp

A servo was initially used to deploy the cheese. This was laden with problems, including initializing problems, where it would break Legos because of over-turning of the servo and the tight tolerances it had to fit within. The servos would also malfunction, leading to dumping the basket unexpectedly on the track, when the robot was not ready to dump cheese.

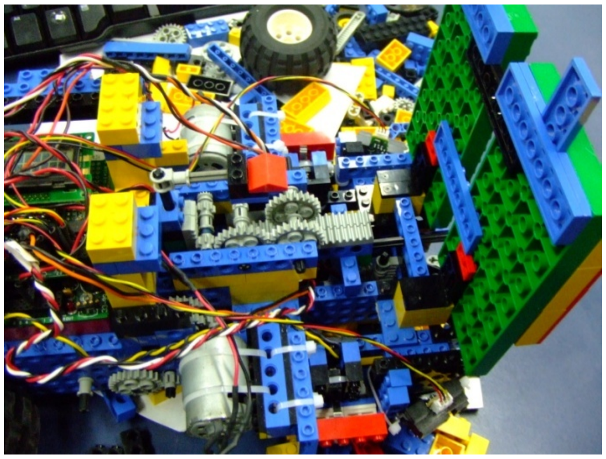

Figure 9 – Basket Ramp lifted up showing the mechanism to translate horizontal motion to vertical motion, which pushes the basket off the ramp

To change this design, an entirely new mechanical design was created to deploy the basket. This mechanical design, shown Figure 9, used the force of the robot running into the wall to move a vertical arm to lift the basket up and off the ramp. The only drawbacks with this system were that any simple collision with anything could throw the basket off the ramp. A design replacing the lifting arm in Figure 9 with a release pin instead would have given the basket more reliability under collision conditions. What I really took away from this part of the design is, again, servos are very unreliable, and things need to be designed to fail in the given conditions you want (i.e. basket attached to the robot) instead of the given conditions you'd expect (i.e. basket on the wall), which can give you unexpected results.

2.3 Servo Steering System

When using the wall following robot, Figure 3, a servo steering system was used because both motors were coupled together and drove the robot. Independent motors were not applied to this robot, so steering could not be performed by varying the speeds of individual motors. Instead, a servo turned a front wheel that influenced the direction the robot traveled. This servo steering system, pictured Figure 10, was interesting and had some quirks. In the end, I can't say I would recommend it for a final competition design; however it was fun to work with for the wall following robot design, because of its light weight design.

One of the most frustrating problems with this design was the resetting of the middle position of the servo. This meant that the chain had to be uncoupled and recoupled every time the servo decided a new position was the center position. If the chain wasn't removed, it would result in incorrect steering. During the wall following competition, the servo worked well and provided enough steering power to navigate the robot around the track successfully. The drawback of the servo was that it wouldn't always turn to the exact same turning degree, even though the same value was given to the servo through the computer. The servo was simpler to code with, too. This was because instead of trying to coordinate two different motors to give a turning radius that you'd like, the servo just needs one value and can easily offer all different types of turning radii. The draw back to this, though, was that tight turning radii was significantly reduced and "turn lock" could occur, where the front wheel turns 90^{O} to the left or right, and the robot will bounce the front wheel in an attempt to continue going straight.

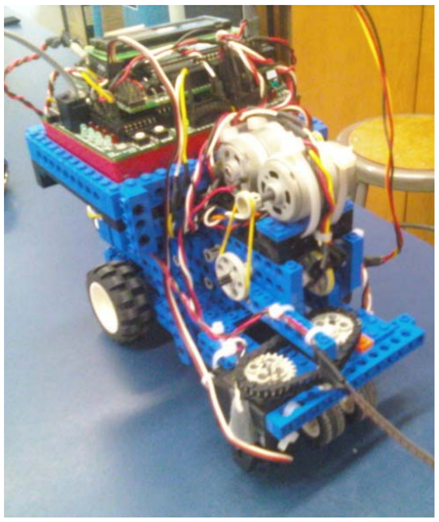
Figure 10 – Wall follower robot with servo steering on front. Servo is mounted beneath left gear on front of robot, wheel is below right gear

3 Electrical Systems

The electrical systems on the robot were incredibly important. Without the electrical systems, the robot could not function within its environment and it could not get the appropriate feedback to react or interact with its surroundings. There were many electrical systems that were used on the final design of Team 8's robot. These will be discussed in the following sections.

3.1 DC motors

If the handy board is the heart and brain of the robot, the DC motors are the muscles. These brush motors could pull 20W and spin somewhere around 8000 rpm, according to the specification sheet. This meant that these motors provided high speed and VERY low torque. This wasn't a big deal, however, since there was enough gearing to go around reducing these motors to a more reasonable robot speed. The DC motors were mostly reliable. The only noted problems resulted in motors overheating, or draining electrical power from the handy boards. This required some robots, including Team 8's, during some assignments to program in a "rest" function where the robot let's its engines cool so that the engines wouldn't get burnt out.

Although these motors were controlled using pulse-width modulation, I feel that the problems these motors encountered had more to do with the torque they needed to overcome. A loose drive train seemed to operate significantly better than a drive train whose gears were snugged up. Ryan's experience with this came after re-building the final drive train 4 times. One of these drive train setups involved routing a Lego beam, mounted on the DC motor, through a Lego technic piece hole. This was originally done for support of the beam as the motor spun, but what resulted was bending loads subjected on the beam.

This was enough loads on the beam that when turning at a considerable rate, melted the beams and technic pieces. However, what it also did was slow the motors down over a short period of time. A pattern seemed to emerge: upon starting, the motors would run strong, and then slow down and stall. This even occurred when not replacing melted Lego pieces out. Understanding this property taught Team 8 that low torque DC motors MUST be placed under light loads to avoid stalling the motors. In the final design of Team 8's robot, the motors are mounted such that there is little force forcing the motor and initial drive train gear together as shown Figure 2.

3.2 Ancillary DC Motor

The ancillary DC motor as Ryan called it was the 3rd DC motor controlled by an H-bridge switch. This basically used a 5V source to open a switch, which allowed a 9V battery to power a DC motor. This created some interesting problems, however, since this motor could only be used in one direction. The motor could also not be modulated in any way, and ran at full speed. This could be potentially difficult to work with. Luckily for Team 8, a perfect niche for this ancillary DC Motor was found, operating the scissor lift, which needed a one time, one direction, fast spinning motor to lift the scissor lift up. Team 8 learned to not fight the one-way use of the ancillary motor, and instead, use it to complement the design.

3.3 Servos

Servos, or as Ryan likes to refer to them, "the most unpredictable things on the planet", caused many problems for Team 8, as mentioned in section 2.2. The best guess Ryan could come up with regarding why the servos would malfunction was because of current surge while operating the DC motors in conjunction with powering 2 servos. Because servos use feedback loops to ensure its position is kept while in operation, power is consumed even if the servo isn't turning. Only when 2 servos were powered did the servo malfunction occur. Because the handy board was attempting to feed 4 different, high power (for a handy board) sources with power, a surge or drop in current to the servos could cause it to read its position incorrectly, forcing the arm to compensate and turn erratically. What is important to learn from this is that plugging multiple power sinks into a single, somewhat undependable, and low capacity power source will reduce reliability, and that if this power for operation is required, the outside power should be sought at from elsewhere, somewhat like the Ancillary DC motor.

3.4 Wall Sensor

The wall sensor proved to be one of the handiest pieces of electrical equipment to help the robot navigate around the course. The wall sensor, which used IR light to reflect off walls to determine its distance from it, had some quirks to it. It also taught Team 8 a few important things about design feedback controllers.

First, just implementing proportional feedback controllers, something that was taught in ME 155A, was a wonderful hands on learning experience. As a team, we could finally see, hands on, how a proportional feedback controller operated in the real world. Using some fancy mounting procedures, a quazi-derivative function could be added on without the use of any extra code. By mounting the wall sensor 45 degrees off the robot's line of travel, the wall sensor could read up-coming terrain data and have the robot move accordingly (i.e. like a corner). This is how a derivative controller works, to a degree.

The other EXTEREMLY important thing Ryan learned from these wall sensors regarding feedback control was the location of where to mount the sensor. Using the concept of a rear wheel drive car and mounting the sensors close to the back tires, the robot has less feedback, since a radical turn by the front of the robot will result in a small change in the line reader's sensor. On the other hand, placing the sensor as far out away from the back tires (i.e. over the front of the robot), any slight change in the robot's direction will result in a change in the wall sensor. By mounting the sensor further away from the rear wheels, a more stable wall follower was obtained.

Also, the interference from the wall sensor could create tons of problems, including unexpected cheese in the way or IR location beams on the track. This proved to be difficult to work with, especially during the final competition, and is what Ryan believes resulted from his sensors not reading correctly during the final competition day.

Because of calibrations needs, a calibration program was built to calibrate different given distances the robot could be away from the wall. This was important to avoid the frustrating "wall follow hump", where if the wall sensor becomes too close to a wall, the readings will decrease one again, instead of increasing. This was an aspect of wall sensors I was unaware of, and taught me a lot about how to properly work with hardware.

These sensors were incorporated into our final robot design to read off either side of the robot. This was important because the robot needed to wall follow a given distance traveling either direction.

3.5 Line Sensor

The line sensors used in this class were an interesting concept. Most likely a thought about running over the ramp conjures in people's heads, as this was an INCREDIBLY difficult problem to overcome. Of course, one of the simple solutions was to mount the sensor directly behind an axle, so that when the robot travels on the ramp, the distance between the ground and the sensor changes very little, resulting in the ability to maintain the calibration data that was originally set.

Although Ryan didn't enjoy solving the problem of the line sensor traveling over the ramp, Team 8 did learn a lot about robots and sensing from that little problem. It gave a new found respect for robots that can to remote sensing and target tracking. These line sensors still played an invaluable role in traversing the board during final competition, however, because they were used to double-check if the robot was near a given cheese position.

3.6 Light Sensor

The light sensor was used in Team 8's final design; however there was little design background behind it. A calibration program was given to help define the threshold level of light needed to start the robot. One thing Team 8 learned from the light sensor involved accidental light sensing. To combat this, code was written that placed the robot into a "fuse lit" mode, where once you “lit the fuse” by pressing the start button, the robot was ready to accept a “go” light and start the completion. This was helpful during resets, since it allowed anyone to stop the robot and pick it up and place it back at home without the fear of the robot restarting.

3.7 Digital Switches

Digital switches were first used during the bump and avoid assignment. It was soon learned that using digital switches to encounter walls was a poor idea, since by the time the robot understood that a switch was pressed, it had to reverse direction because of its close proximity to an obstacle. What digital switches did work well for, however, was for things like user input. They were used in our final design as kill switches. The also doubled, however as user inputs when the robot was in the “fuse lit” mode. The robot was programmed to determine where it should head next after a reset. If, for some instance that was wrong, the user could use the kill switches to tell the robot to take another direction of travel. It also was used to tell the robot to employ a different strategy if another robot proved to be a greater threat than expected.

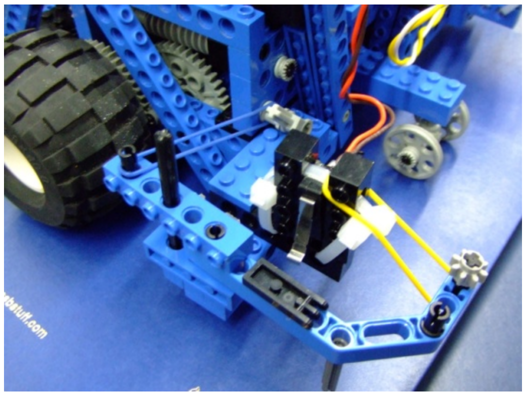
Figure 11 - Wall-encountering switch, used to change the robot's path

3.8 Potentiometer

Potentiometers were useful because of their ability to sense very slight changes. Unlike digital switches, potentiometers were analog and could help determine how far into an obstacle was close to the robot. The potentiometer was used in Team 8’s final design; however, it was used in a more passive role. It was attached with plans to be used, however after being a great “cheese plow” device, its purpose was slightly altered for our robot’s purpose.

4 Software

An autonomous robot is only as autonomous as the software programmer codes it to be. The software a robot runs can make a great robot run terribly and a mediocre robot run very well. The one catch about programming robots, unlike computer programs, is that correcting syntax and logical errors will not guarantee a robot’s function to be predictable, and further, will prove that the same line of code run twice can yield different results.

4.1 Basic Command Lines

Basic command lines were used throughout the entire course. These were critical to the function of the robot, however, as time traversed, many of these basic command lines were buried in functions and not directly referenced or used. Because Ryan had extensive computer programming experience, the important lessons Rusty and Ryan learned were that neat and logical flowing code would allow the other user to interpret what they attempted to code. This also meant that commenting was a critical task.

4.2 Multi-Threading

Multi-threading was a nifty tool that allowed an interrupt command to occur. This was useful in stopping robots, prior to collision. Although the basic and required multi-threading function was constructed for the class, little research went into understanding this concept further because of its lack of perceived usefulness in further code. There was a little design into an attempt to run a second, multi-threaded function in conjunction with the first. This did not produce any results, and the concept was not longer pursued, since most multi-threading tasks could be hard-coded in.

4.3 Advanced Coding and Debugging

Some of the things that helped our robot function wonderfully were some advanced coding techniques and debugging processes. Solving for errors was critical to making a robot run.

Some debugging strategies involved filming robot runs, and stepping through the code while watching the robot run. Motor stops and pauses would be placed in to help determine where and when a code was malfunctioning. Debugging is a critical tool that can change a non-functioning robot to a champion.

Another technique for debugging was using the dial located on the handy board to give the handy board different values to use for different types of code. This help figure out problems quickly and saved lots to time reloading the same program onto the handy-board with one different value changed.

Some of the advanced coding techniques used in our final design were high-level customizable functions and the use of switch-cases. Customizable functions would allow one function to operate multiple I/O devices (on the left side of the robot, for example), and with a different sent value to the function, run the same I/O devices on the opposite side of the robot. This allows for any debugging to occur within one function, instead of multiple functions. It also allows the code to remain much cleaner, making it easier to read.

Switch-cases, an important concept in C and C++, and obviously incorporated in to Interactive C, act like special “if” statements. Except instead of a “true” solution and a “false” outcome, there are an infinite number of outcomes. This allowed our robot to have multiple track routes programmed onboard the robot. To change the, another switch-case would be manipulated to change the programmed route. This allowed an incredible amount of code flexibility with not much increase in the complexity of coding involved.

4.4 Strategy coding

Strategy coding was used in Team 8’s robot. Because our strategy involved going for hanging cheese (a risky strategy), and dumping the cheese on the wall (an even riskier strategy), some thought had to be put into how the robot would run the course.

For starters, the robot needed to change direction. At the start of every new round, the robots had to face left, and by changing direction, a potential collision could be induced, meaning a start over and the ability to place the robot facing any direction. This reset also meant that it could potentially slow down the other team at grabbing cheese.

The basic strategy used by Team 8 was to obtain the three hanging cheese on our side and place them on the wall. This is a 72 point strategy, and producing a reliable code was critical. Of course, Murphy’s Law showed that even reliable code isn’t reliable, as proven during competition day.

Regardless of this lack of total reliability in this code, another route was programmed into our robot, so that it could run around the entire board and attempt to grab all 8 hanging cheese and dump it on the wall. This code was a back-up code, whose purpose was only to be used if the threat of losing to 72 was imminent. Specifically, it was coded in response to Team 9’s killer robot. It also, unfortunately, didn’t work on competition day. Ryan has video documentary of it working correctly, however.

The robot was also designed to determine what route to pick next if a reset had occurred. This strategy involved the three hanging cheese at home and determining which of those three cheese had already been obtained. Luckily, an error checking function programmed into the robot was the ability to correct the robot’s assumption of where to head next. This allowed the user to see what actually happened on the board and correct the robot’s misconceptions about what occurred.

The important concept about this 72 point strategy was that once it had dumped its cheese, it would stop running and wait for the clock to run out, thereby saving batteries and reducing wear on the already over-worked robot.

This 72 point method worked well for the first two rounds of the final competition – we managed to crush the competition. However during a round of competition, a “reset” that should have been declared, wasn’t (i.e. bad call, ref). This resulted in the destruction of Team 8’s robot and its elimination as well. Had Team 8 been more vocal about not having a reset, the outcome of where Team 8 finished may have been potentially different. This was a difficult loss because of the hard work, long hours, and multiple design iterations all being eliminated in an instant.

5 Other Concepts Learned In This Class

There were a few other concepts outside of playing with Legos and building robots that the team members of Team 8 learned. These are described in the following.

5.1 Team dynamics

Team dynamics play a critical role in the performance of a group. High performing teams were those that had a combination of interested and enthusiastic people, people willing to work and learn, and people with a little creativity up their sleeves. Dividing up the work among group members is difficult to do fairly, however the enthusiasm and interest level of those willing to put in the long hours required for this course surely reaped the long term rewards.

5.2 Planning

Construction planning is critical to building a robot correctly and building a robot correctly the first time. Without planning, a robot can be built, however there are many design challenges and flaws that will occur because of this lack of planning. Part of this planning involves prioritizing different parts of the robot to be designed and built. Building a robot based off this strategy is much more likely to build a robot with fewer problems and less reasons to rebuild than one that is haphazardly put together.

5.3 Strategy

Robot strategy, or in a more general sense, clearly defining what a robot intends to and with what parts and tools will significantly narrow the design considerations and requirements down to help more clearly and succinctly build a robot that achieves the required design goals. Strategy goes somewhat hand-in-hand with planning; however, knowing what your robot will be capable of can keep you on track to complete the design.

5.4 Chance

Taking chances and calculated risks can be rewarding. They can also be disastrous. However, the level of creativity and safety allowed in this class taught us that by considering the chances of achieving something, the rewards can propel one to take those chances. For instance, had the professor of this class graded us on cheese capturing ability instead of effort put forth, Team 8 would have never consciously taken a chance at dumping hanging cheese on the wall. Our team would have played a much more conservative route mowing up cheese; which would have provided learning experiences, just not those at the level of excitement, creative freedom, and competitive spirit that made our team that much more receptive to learning these things.

6 Suggestions to futures classes

If a student from a future class were to read this report, there are a few pieces of advice they should walk away from this report knowing. First, is to not be afraid to fail. Team 8 suffered its worst failure during the solo competition. This was a real eye opener in terms of what systems on our robot we malfunctioning and what systems (including the drive train) were just flat out poorly designed. Had our team not failed miserably, we would have never recognized the poor planning and strategy put into designing our robot.

The next piece of advice is to take chances, especially in this class. This class is a great place that allows phenomenal failures given the effort is still put in. In an environment where failure won't result in one's termination, expulsion, or arrest, the time and place to take those high-risk chances is here and now.

Next, any student going through this class should pay close attention to the multi-discipline engineering and multi-discipline collaboration occurring. In a world that is continually demanding physical parts controlled by computers, the demand for such engineers will exist. This cross-disciplinary work is a great way to meet new people, and learn to deal with other types of engineers.

Finally, when taking this class, look for another enthusiastic, willing, and competent student. The greater the collaboration between people in designing something, the better the end result becomes.

7 Summary and Reflection

In short, Team 8 has taken the curious bent approach to design throughout this course to determine what parts worked best. Some of the pitfalls and triumphs that Team 8 encountered have been told, including scoring 72 points during the competition, and our robot being destroyed as we were eliminated. The advice suggested is sewn in hopes that someday it will aide someone. This class has been an incredible journey of learning, coordination, skill, and willpower.

If the competition were to be held again in another few weeks our team would not alter the main strategy of our robot. We would focus our strategy into improving the basket dumping to ensure the basket could not be taken off unless the robot is fully into the wall. Also, we would ensure the robot could run the strategic routes the robot is intended to run. Finally, we would look into methods for sabotage, such as LED interference, or cheese dispersion methods.

8 Appendix – Final Source Code

```
/*
Ryan Sass
Rusty Harrington
Team 8

March 10, 2009

Robot: Scorpion

This robot has pre-programmed routes - one safe, one risky. If run correctly w
ithout interference, safe achieves
72 points, risky achieves >200. It is based on a mechanical cheese basket dump
er.

*/
int TRIGGER=0, SPEEDL=70, SPEEDR=70, CASE=1, PING=1, DEPLOY=1;
float TIME_LEFT=120.0;
//RAISE CHEESE SWIPER COMMAND
void rise(){
    if(!TRIGGER || DEPLOY)
      return;
    set_digital_out(6);
    msleep(1600L);
    if(!TRIGGER){
        CASE=-1;
        return;
    }
    clear_digital_out(6);
    DEPLOY=1;
    msleep(500L);
    return;
}
//GO FORWARD A CERTAIN COUNT COMMAND
void forward(int countgoal){
    int count=0;
    if(!TRIGGER)
      return;
    reset();
    while(TRIGGER && (count<countgoal)){
        count=read_encoder(1);
        yip(1,SPEEDR);
        yip(2,SPEEDL);
    }
    stop();
    return;
}
//FORWARD UNTIL LINE READER OR COUNT COMMAND
void forwards(int countgoal){
    int count=0, pop=0;
    if(!TRIGGER)
      return;
    reset();
    while(TRIGGER && (count<countgoal)){
        if(analog(30)>100){
            pop=1;
            break;
        }
        count=read_encoder(1);
        count=read_encoder(1);
        yip(1,SPEEDR);
        yip(2,SPEEDL);
    }
    if(pop)
      forward(12);
    stop();
    return;
}
//WALL FOLLOWING COMMAND
void wallfollow(int side,int countgoal, int dist, float Kp){
    //LEFT=0
    //RIGHT=1
    int error, data=1, count;
    float P;
    if(!TRIGGER)
      return;
    reset();
    yip(1,SPEEDL);
    yip(2,SPEEDR);
    msleep(250L);
    if(side){//RIGHT WALL SENSOR
        while(TRIGGER){
            count=read_encoder(1);
            error=dist-analog(19);
            P=Kp*(float)error;
            yip(1,(SPEEDL+(int)(P)));
            yip(2,(SPEEDR-(int)(P)));
            if(count>=countgoal)
              break;
            msleep(500L);
        }
    }
    else{//LEFT WALL SENSOR
        while(TRIGGER){
            count=read_encoder(1);
            error=dist-analog(20);
            P=Kp*(float)error;
            yip(1,(SPEEDL-(int)(P)));
            yip(2,(SPEEDR+(int)(P)));
            if(count>=countgoal){
                stop();
                return;
            }
            msleep(500L);
        }
    }
    return;
}
//chill out command
int chill_out(){
    if(!TRIGGER)
      return 0;
    CASE=9;
    TIME_LEFT=TIME_LEFT-seconds();
    reset_system_time();
    while(TRIGGER && TIME_LEFT>=15.0){
        TIME_LEFT=TIME_LEFT-seconds();
        reset_system_time();
        msleep(100L);
    }
    if(!TRIGGER)
      return 0;
    tone(100.0,0.75);
    tone(160.0,0.75);
    tone(135.0,0.75);
    while(TRIGGER && TIME_LEFT>0.0){
        TIME_LEFT=TIME_LEFT-seconds();
        reset_system_time();
        msleep(100L);
    }
    if(!TRIGGER)
      return 0;
    TRIGGER=0;
    return 1;
}
//FUNCTION FOR AFTER CHEESE SHOULD HAVE BENE DUMPED TO PROTECT CHEESE
void post_dump(){
    if(!TRIGGER)
      return;
    yip(1,-60);
    yip(2,-60);
    msleep(500L);
    if(!TRIGGER)
      return;
    turn(1,5);
    stop();
    msleep(750L);
    CASE=9;
    return;
}
/****************************** TURNING FUNCTIONS ***************************
*************************************/
//turn command
void turn(int dir, int countgoal){
    //dir = 0, TURN LEFT
    //dir = 1, TURN RIGHT
    int count=0;
    if(!TRIGGER)
      return;
    stop();
    reset();
    if(dir){
        yip(1,70);
        yip(2,-70);
        while(count<countgoal && TRIGGER)
          count=read_encoder(1);
        stop();
    }
    else{
        yip(1,-70);
        yip(2,70);
        while(count<countgoal && TRIGGER)
          count=read_encoder(1);
        stop();
    }
    return;
}
/****************************** UERLYING FUNCTIONS ***************************
*************************************/
//KILLSWITCH FUNCTION
void killswitch(){
    while (1){
        if (digital(11)||digital(12)){
            alloff();
            clear_digital_out(6);
            TRIGGER=0;
        }
    }
}
//RESETTING FUNCTION
void reset(){
    if(!TRIGGER)
      return;
    reset_encoder(0);
    reset_encoder(1);
}
//MOTOR DRIVING COMMANDS
void yip(int a, int b){
    if(!TRIGGER)
      return;
    if(a == 1){//LEFT MOTOR
        motor(0,b);
        motor(1,b);
    }
    if(a == 2){//RIGHT MOTOR
        motor(2,b);
        motor(3,b);
    }
}
//STOP FUNCTION
void stop(){
    alloff();
}
//ABSOLUTE FUNCTION
int abs(int a){
    if (a>0)
      return a;
    else
      return (a*-1);
}
//REST FUNCTION (NOT CURRENTLY USED)
void rest(){
    if(!TRIGGER)
      return;
    alloff();
    msleep(15000L);
}
/****************************** SAFE FUNCTION ********************************
*************************************/
int safe(){
    int bump=0;
    while(TRIGGER){
        switch(CASE){
            case 0:{
                //RAISE CHEESE RAKE

                break;
            }
            case 1:{
                //HEAD CLOCKWISE AROUND BOARD TO GET CHEESE 1 FOLLOWED BY CHEE
SE 2, THEN 3
                //ASSUME ROBOT FACES LEFT
                turn(1,10);
                forward(25);
                turn(0,3);
                wallfollow(1,2,60,3.0);
                //IF RESET, SENDS BACK TO THIS CASE
                if(!TRIGGER){
                    CASE=0;
                    break;
                }
                //WALL FOLLOWS TO GET CHEESE 1
                stop();
                rise();
                if(!TRIGGER){
                    CASE=0;
                    break;
                }
                wallfollow(1,4,60,3.0);
                forwards(5);
                //TURNS TO START ACROSS LEG (CASE 2, CHEESE 1 OBTAINED)
                turn(0,10);
                //IF RESET HERE, JUST GO FOR OTHER SIDE FOR CHEESE 3 THEN 2
                //PING MEANS CHEESE 2 WAS NOT OBTAINED
                if(!TRIGGER)
                  CASE=3;
                break;
            }
            case 2:{
                //CONTINUE CLOCKWISE AROUND BOARD TO GET CHEESE 2 AND 3
                //forward(3);
                wallfollow(0,3,23,14.0);
                forward(9);
                //CHEESE 2 SHOULD HAVE BEEN OBTAINED BY THIS POINT
                //IF RESET HERE, JUST GO FOR OTHER SIDE FOR CHEESE 3, THEN 2
                if(!TRIGGER){
                    CASE=3;
                    break;
                }
                //THIS IS JUST A CORRECTIONAL TURN
                turn(1,1);
                /*stop();
                            msleep(2000L);*/
                //THIS PUTS ROBOT IN VICINITY FOR CHEESE THREE, SOMETIMES OBTA
INING IT
                //BUT NOT FOR SURE
                forwards(25);
                //IF RESET HERE, ONLY NEED TO GET CHEESE 3
                if(!TRIGGER){
                    CASE=3;
                    PING=0;
                }
                break;
            }
            case 3:{
                //CONTINUE CLOCKWISE AROUND BOARD TO GET CHEESE 3 AND RETURN
                //TURN TO FACE FINAL WALL
                turn(0,7);
                //IF RESET HERE, STILL NEED TO GET CHEESE 3
                if(!TRIGGER){
                    CASE=3;
                    PING=0;
                    break;
                }
                //CHEESE 3 SHOULD HAVE BEEN OBTAINED AT THIS POINT
                //HEAD DOWN WALL
                forward(4);
                wallfollow(1,5,51,3.0);
                forward(3);
                //RAM FORWARD TO DUMP OFF BASKET
                turn(0,3);
                forward(15);
                turn(1,4);
                forward(50);
                //GUARD CHEESE FUNCTION
                post_dump();
                //IF RESET NOW, JUST RE-DUMP TO ENSURE CHEESE IS OFF AND MADE
IT
                bump=chill_out();
                CASE=9;
                break;
            }
            case 4:{
                /***************************DOUBLE CHECK - FUNCTIONALITY PROVE
N W/ PING=0&1**********************///////////
                //CHEESE 1 OBTAINED, CHEESE 3 AND POTENTIALLY 2 ARE NOT OBTAIN
ED
                //ASSUMES ROBOT FACES RIGHT
                //HEATS FOR CHEESE
                turn(1,3);
                forward(25);
                turn(1,3);
                //IF RESET, SENDS BACK TO THIS CASE
                if(!TRIGGER){
                    CASE=3;
                    break;
                }
                //WALL FOLLOWS TO GET CHEESE 1
                wallfollow(0,17,60,3.0);
                forwards(4);
                //TURNS TO START ACROSS LEG (CASE 2, CHEESE 1 OBTAINED)

                //IF PING=0, CHEESE 2 WAS OBTAINED.
                //IF PING=1, CHEESE 2 WAS NOT OBTAINED
                if(PING){
                    if(!TRIGGER){
                        CASE=3;
                        break;
                    }
                    turn(1,9);
                    //CHEESE 3 SHOULD HAVE BEEN OBTAINED BY HERE
                    //HEAD TO CHEESE 2
                    wallfollow(1,12,23,14.0);
                    forward(6);
                    //CHEESE 2 SHOULD HAVE BEEN OBTAINED BY THIS POINT
                    //IF RESET HERE, JUST GO FOR OTHER SIDE FOR CHEESE 3, THEN
2
                    if(!TRIGGER){
                        CASE=3;
                        break;
                    }
                    //THIS IS JUST A CORRECTIONAL TURN
                    turn(0,1);
                    if(!TRIGGER){
                        CASE=4;
                        break;
                    }
                    //CHEESE 2 SHOULD HAVE BEEN OBTAINED BY HERE
                    //TURN LEFT 90 TO HEAD HOME
                    turn(1,7);
                    //GO HOME AND DUMP
                    wallfollow(1,9,31,7.0);
                    forward(50);
                    //GUARD CHEESE
                    post_dump();
                    bump=chill_out();
                    //IF RESET HERE, ALL CHEESE OBTAINED, JUST NEED TO DUMP
                    if(!TRIGGER){
                        CASE=9;
                        break;
                    }
                }
                else{
                    //CHEESE 2 WAS PREVIOUSLY OBTAINED, JUST GET CHEESE 3, WHI
CH OCCURED NOW
                    //U TURN TO HEAD HOME
                    turn(0,14);
                    //WALLFOLLOW HOME
                    wallfollow(1,9,51,3.0);
                    //RAM WALL TO DUMP CHEESE
                    forward(50);
                    //GUARD CHEESE
                    post_dump();
                    //IF RESET HERE, JUST DUMP CHEESE
                    bump=chill_out();
                    CASE=9;
                }
                break;
            }
            case 5:{
                /****************************DOUBLE CHECK - FUNCTIONALITY PROVE
N!!!!!!!!**********************///////////                           //CHEESE
1 AND 3 WAS OBTAINED, JUST ATTEMPT TO GET CHEESE 2
                //ASSUME CAR FACING FORWARD
                //WALL FOLLOW STRAIT TO GET CHEESE
                wallfollow(0,19,31,7.0);
                forward(19);
                //IF RESET HERE, CHEESE 2 WAS NOT OBTAINED, TRY AGAIN
                if(!TRIGGER){
                    CASE=4;
                    break;
                }
                //CHEESE 2 WAS OBTAINED AT THIS POINT
                //UTURN TO HEAD HOME
                turn(0,14);
                //WALLFOLLOW HOME
                wallfollow(0,6,31,7.0);
                forward(50);
                //GUARD CHEESE
                post_dump();
                bump=chill_out();
                break;
            }
            default:{
                //CAR HAS ALL CHEESE AND WILL JUST RE-DUMP CHEESE TO ENSURE CH
EESE WAS DUMPED
                forward(14);
                //GUARD CHEESE
                post_dump();
                bump=chill_out();
                break;
            }
        }
        //CASE ADVANCES, HOW CASE WORKS
        CASE=CASE+1;
        //THIS KEEPS TRACK OF HOW MUCH TIME LEFT
        TIME_LEFT=TIME_LEFT-seconds();
        reset_system_time();
        //IF THERE IS LESS THAN 15 SECONDS, JUST DUMP WHATEVER CHEESE YOU HAVE
        if(TIME_LEFT<=15.0)
          CASE=9;
    }
    return bump;
}
/****************************** RISKY FUNCTION ******************************
***************************************/
int risky(){
    int bump=0;
    while(TRIGGER){
        switch(CASE){
            case 0:{
                //RAISE CHEESE RAKE
                rise();
                break;
            }
            case 1:{
                //HEAD CLOCKWISE AROUND BOARD TO GET CHEESE 1 FOLLOWED BY CHEE
SE 2, THEN 3
                //ASSUME ROBOT FACES LEFT
                turn(1,3);
                forward(25);
                turn(1,3);
                //IF RESET. SENDS BACK TO THIS CASE
                if(!TRIGGER){
                    CASE=0;
                    break;
                }
                //WALL FOLLOWS TO GET CHEESE 1
                wallfollow(0,3,65,4.0);
                forward(12);
                wallfollow(0,3,65,4.0);
                forward(6);
                turn(0,1);
                forward(6);
                wallfollow(1,25,100,1.5);
                wallfollow(0,25,65,4.0);
                forwards(23);


                if(!TRIGGER){
                    CASE=3;
                }
                break;
            }
            case 2:{
                turn(1,7);

                forward(18);
                //wallfollow(0,9,23,14.0);


                if(!TRIGGER){
                    CASE=3;
                    break;
                }
                //THIS IS JUST A CORRECTIONAL TURN
                turn(0,1);
                /*stop();
                            msleep(2000L);*/
                //THIS PUTS ROBOT IN VICINITY FOR CHEESE THREE, SOMETIMES OBTA
INING IT
                //BUT NOT FOR SURE
                forward(15);
                //IF RESET HERE, ONLY NEED TO GET CHEESE 3
                if(!TRIGGER)
                  CASE=3;
                break;
            }
            case 3:{
                //CONTINUE CLOCKWISE AROUND BOARD TO GET CHEESE 3 AND RETURN
                //TURN TO FACE FINAL WALL
                turn(1,7);
                //IF RESET HERE, STILL NEED TO GET CHEESE 3
                if(!TRIGGER){
                    CASE=3;
                    break;
                }
                //CHEESE 3 SHOULD HAVE BEEN OBTAINED AT THIS POINT
                //HEAD DOWN WALL
                forward(3);
                wallfollow(0,12,70,2.8);
                wallfollow(1,28,110,1.5);
                forward(7);
                wallfollow(0,10,65,3.0);
                forward(8);
                wallfollow(0,10,65,3.0);
                if(!TRIGGER){
                    CASE=3;
                    break;
                }
                forward(10);
                turn(1,3);
                forward(15);
                turn(0,3);
                forward(50);
                //RAM FORWARD TO DUMP OFF BASKET
                //forward(50);
                //GUARD CHEESE FUNCTION
                post_dump();
                //IF RESET NOW, JUST RE-DUMP TO ENSURE CHEESE IS OFF AND MADE
IT
                bump=chill_out();
                CASE=4;
                break;
            }
            case 4:{
                /****************************DOUBLE CHECK - FUNCTIONALITY PROVE
N W/ PING=0&1***********************///////////
                //CHEESE 1 OBTAINED, CHEESE 3 AND POTENTIALLY 2 ARE NOT OBTAIN
ED
                //ASSUMES ROBOT FACES RIGHT
                //HEATS FOR CHEESE
                turn(0,3);
                forward(25);
                turn(0,3);
                //IF RESET HERE, NEITHER CHEESE 3 OR 2 WERE OBTAINED, RESET TO
HERE
                if(!TRIGGER){
                    CASE=3;
                    break;
                }
                //FUNCTION TAKES IT TO CHEESE 3
                wallfollow(1,9,60,3.0);
                forward(12);
                //IF PING=0, CHEESE 2 WAS OBTAINED.
                //IF PING=1, CHEESE 2 WAS NOT OBTAINED

                if(!TRIGGER){
                    CASE=3;
                    break;
                }
                turn(0,9);
                //CHEESE 3 SHOULD HAVE BEEN OBTAINED BY HERE
                //HEAD TO CHEESE 2
                wallfollow(0,12,23,14.0);
                forward(6);
                //turn(1,1);
                forward(6);
                if(!TRIGGER){
                    CASE=4;
                    break;
                }
                //CHEESE 2 SHOULD HAVE BEEN OBTAINED BY HERE
                //TURN LEFT 90 TO HEAD HOME
                turn(0,7);
                //GO HOME AND DUMP
                wallfollow(0,9,31,7.0);
                forward(50);
                //GUARD CHEESE
                post_dump();
                bump=chill_out();
                //IF RESET HERE, ALL CHEESE OBTAINED, JUST NEED TO DUMP
                if(!TRIGGER){
                    CASE=9;
                    break;
                }
                break;
            }
            case 5:{
                /***************************DOUBLE CHECK - FUNCTIONALITY PROVE
N!!!!!!!!**********************///////////                          //CHEESE
1 AND 3 WAS OBTAINED, JUST ATTEMPT TO GET CHEESE 2
                //ASSUME CAR FACING FORWARD
                //WALL FOLLOW STRAIT TO GET CHEESE
                wallfollow(0,19,31,7.0);
                forward(19);
                //IF RESET HERE, CHEESE 2 WAS NOT OBTAINED, TRY AGAIN
                if(!TRIGGER){
                    CASE=4;
                    break;
                }
                //CHEESE 2 WAS OBTAINED AT THIS POINT
                //UTURN TO HEAD HOME
                turn(0,14);
                //WALLFOLLOW HOME
                wallfollow(0,6,31,7.0);
                forward(50);
                //GUARD CHEESE
                post_dump();
                bump=chill_out();
                break;
            }
            default:{
                //CAR HAS ALL CHEESE AND WILL JUST RE-DUMP CHEESE TO ENSURE CH
EESE WAS DUMPED
                forward(14);
                //GUARD CHEESE
                post_dump();
                bump=chill_out();
                break;
            }
        }
        //CASE ADVANCES, HOW CASE WORKS
        CASE=CASE+1;
        //THIS KEEPS TRACK OF HOW MUCH TIME LEFT
        TIME_LEFT=TIME_LEFT-seconds();
        TIME_LEFT=TIME_LEFT-seconds();
        reset_system_time();
        //IF THERE IS LESS THAN 15 SECONDS, JUST DUMP WHATEVER CHEESE YOU HAVE
        if(TIME_LEFT<=15.0)
          CASE=9;
    }
    return bump;
}
/**************************** MAIN FUNCTION ******************************
*************************************/
void main(){
    int bump=0,select=1;
    start_process(killswitch());
    msleep(1500L);
    enable_encoder(0);
    enable_encoder(1);
    while(1){
        if(start_button()){
            msleep(1000L);
            while(1){
                if(start_button() || (analog(22)<150)){
                    TRIGGER=1;
                    reset_system_time();
                }
                if(TRIGGER){
                    if(!select)
                      bump=safe();
                    else
                      bump=risky();
                }
                if(bump)
                  break;
                //BEEP FUNCTION TO TELL USER WHICH WAY TO FACE ROBOT
                //ONE BEEP MEANS FACE LEFT
                //TWO BEEPS MEANS FACE RIGHT
                //THREE BEEPS MEANS FACE CENTER
                //LOW PITCH BEEP MEANS FACE WALL
                if(stop_button()){
                    if(digital(11)){
                        switch(CASE){
                            case 0:{
                                tone(40.0,0.7);
                                break;
                            }
                            case 1:{
                                CASE=9;
                                break;
                            }
                            case 4:{
                                CASE=5;
                                break;
                            }
                            case 5:{
                                CASE=1;
                                break;
                            }
                            default:{
                                CASE=4;
                                break;
                            }
                        }
                    }
                    if(digital(12)){
                        msleep(1000L);
                        if(select==0){
                            select=1;
                            tone(320.0,1.0);
                            msleep(1000L);
                        }
                        else{
                            select=0;
                            tone(100.0,1.0);
                            msleep(1000L);
                        }
                        if(DEPLOY)
                          CASE=1;
                    }
                }
                switch(CASE){
                    case 0:{//FACE ROBOT LEFT
                        beep();
                        msleep(500L);
                        break;
                    }
                    case 1:{//FACE ROBOT LEFT
                        beep();
                        msleep(500L);
                        break;
                    }
                    case 4:{//FACE ROBOT
                        if(!select)//LEFT IF RUNNING SAFE
                          beep();
                        else{//RIGHT IF RUNNING RISKY
                            beep();
                            beep();
                        }
                        msleep(500L);
                        break;
                    }
                    case 5:{//FACE ROBOT CENTER
                        beep();
                        beep();
                        beep();
                        msleep(500L);
                        break;
                    }
                    default://FACE ROBOT AT WALL
                      tone(60.0,0.7);
                    break;
                }
            }
        }
    }
}
```