6.836 Embodied Intelligence
Final Project:
Tom and Jerry

Gleb Chuvpilo, Jessica Howe
{chuvpilo, howej}@mit.edu
May 15, 2002

Contents

1 Introduction

Tom and Jerry, as they are usually called, are a robotic 'cat and mouse' pair that were built to behave in ways modeled from living cats and mice. Both are implemented as robotic vehicles that are able to move around within their environments, as well as interact with each other and modify behavior based on their current surroundings. The mouse acts as a passive agent, simply moving and wandering through the environment with no sensory feedback, to give the cat something to chase. The cat, on the other hand is the active agent who tracks and chases this mouse around the environment, with a layering of multiple behaviors that are able to take effect at appropriate times.

We had many goals when we began working on this project and the idea for it. First, neither of us had ever built a robot and this was the perfect opportunity to do so. We both wanted to get some experience with building, programming, testing and interacting with a robot.

Second, much of the class is focused on the difficulties and issues that arise when one builds a physical robot, as opposed to working strictly with simulation. In simulation it is usually the case that the environment is not modeled perfectly and physical problems that may arise in building a robot are ignored or overlooked. To test and demonstrate some of these ideas explored we wanted to build a robot of our own.

Finally, the class is called Embodied Intelligence, so it seems fitting to make something that is embodied rather than simulated. In the class we looked at various design architectures, and building a robot would let us discover the pros and cons of different design aspects on our own.

1.1 Basic Design Overview

Both the cat and mouse are built primarily from lego parts and use standard lego motors as their source of locomotion. The processor in the cat is a Handy Board, a board developed by the Media Lab. This board is programmed with Not Quite C, a variation of C that is both much simpler than C and designed to allow an easy interface to both motors and sensors attached to the board. The mouse contains a Cricket, a very simple processor also developed at the Media Lab. This board is programmed in Logo, a language specifically built for the Crickets. As with the Handy Board, this language contains mechanisms for easy interaction with the sensors and motors that may be attached.

The cat has two bump sensors, activated by front left and front right 'whiskers', and three light sensors mounted at the front of the vehicle, facing left, right, and forwards. The bump sensors give a boolean pressed / not pressed value, while the light sensors return integer values between 0 and 255. The mouse, as mentioned, is strictly passive and has no feedback in the form of sensors.

We have chosen to use light as the form in which the cat is able to track and chase the mouse. The mouse has a halogen bulb mounted on top of it which emanates light in all directions. When running the experiments we simply turned off the lights in the room so that the light shining from the mouse was the only light visible to the cat. We chose to use light rather than infrared for mouse-tracking because we are novice roboticists and it is easy to debug light tracking mechanisms when we know the light is either shining or not shining. Using IR, on the other hand, is more difficult to debug simply because we cannot see with the naked eye what the robots see.

1.2 Behavior

The basic behavior of the cat can be broken up into four distinct sections. Each of these behaviors act in a layered fashion one upon another, so that the appropriate action is invoked at the appropriate time, subsuming the behavior of lower levels. The basic action is to wander around the environment searching for the mouse. I say searching but this really means waiting for the light from the mouse to be seen so that a higher subsumption level may be called in. The next layer is this light-following layer, activated strictly by the levels read in by the three light sensors. This allows the cat to turn towards the mouse and move towards it once the two side light sensors read at roughly the same levels. The next level is obstacle avoidance, which is activated by the bump sensors. When an obstacle is hit the cat will back up and turn away from it, after which the lower levels of wander or light-following will resume. The final level is when the cat is playing with the mouse. This is a more complex level that is invoked when the cat is within a certain threshold range of the mouse and involves waiting, stalking, pouncing and freeing the mouse.

I will briefly go through each of the behaviors that are expressed by the cat.

1.2.1 Wander

Wandering is the basic action taken by the cat in lack of stimulus that triggers other actions. The purpose of this action is to give the cat the ability to explore its environment in search of the mouse. While wandering, the cat will repeatedly move forward for a random amount of time and then turn for a random amount of time, pointing it in a new direction. The random length of time is spread between a defined minimum and maximum amount of time for which each of these actions will take place. If in the process of wandering some stimulus is encountered the resulting behavior will take higher precedence over the wander action. When the action is complete wandering will resume.

1.2.2 Light Following

The light following layer is actually quite simple. When the cat is not pointing directly at the light the left and right light sensors will read different values. When these values are more than some defined delta apart the motors will spin to turn the cat in place so that the light is being faced head on. If both side sensors read roughly the same values and the forward sensor reads above some defined threshold (when it actually sees the mouse rather than ambient light levels) it will move forward, correcting its direction if need be.

1.2.3 Obstacle Avoidance

The obstacle avoidance is primarily activated by the 'whiskers' of the cat, or the left and right bump sensors. When an obstacle is hit by either of these bump sensors the cat will slightly back up and turn away from the object that was hit. Occasionally an object will be hit either by the side of the body (as in when it is turning) or in a poor location along the front which does not allow the bump sensors to be compressed. In a case like this the wheels of the cat will keep spinning until either its body shifts enough to activate one of the bump sensors or to free the body from the obstacle, or

when an outside influence such as the mouse approaching causes another action to take over (like turn towards the mouse.)

Although not explicitly coded into the obstacle avoidance (or bump) level, there is also a tendency to turn away from objects based on the input from the light sensors. The tendency of the cat to turn towards the light also causes the cat to turn away from dark objects before they are reached. For example, if the cat is approaching a dark object at an angle, one side light sensor will begin to read a light level lower than the other side. This will cause the 'turn towards the light' action to take over and the cat will in effect turn away from the object before the object is reached. This works best when the lights are on, and the behavior would be very visible if we ever changed mouse tracking based on light to the one based on IR. When run with the lights off this behavior is not always apparent.

1.2.4 Play

The play behavior is a multi-stage behavior in which the cat stalks, pounces upon, and then lets the mouse escape. When the cat gets within a certain distance of the mouse the front light sensor hits a threshold level and the cat begins stalking the mouse. In this stage the cat sits still for a fixed length of time and just watches the mouse, rotating if necessary, but not moving towards it. If the mouse has stayed within the threshold distance of the mouse during this entire stalking time it then pounces upon the mouse: moving towards the mouse at full speed until it hits it. Once it has hit the mouse it briefly backs up and then moves forward to hit it again. This is repeated until the mouse has been hit 3 times. The cat then sits still for a fixed amount of time, ignoring all sensory input, to allow the mouse time to escape. Once this fixed amount of time runs out playing is completed and standard subsumption style behavior is resumed. The play subsumption layer has the highest priority within the subsumption architecture, so while playing all other actions that would be performed by lower subsumption levels are ignored.

One problem with the play mode as it is is that there is no distinction between hitting the mouse and hitting a wall or obstacle while pouncing. While the cat is pouncing on the mouse it just moves at full speed towards the light until it hits something, which may be an obstacle rather than the mouse. As it is the play mode subsumes all other subsumption level behaviors so there is no way to both use obstacle avoidance and mouse chasing as written in those subsumption levels without writing that code directly into the play functionality. Perhaps one modification that whould be made if we were to redesign the subsumption architecture would be to allow lower subsumption levels to be accessed while playing.

1.3 Basic Architecture Design

When we were first designing our robot we were trying to figure out both what exactly it would do and how it would do those things. We had trouble deciding which type of architecture to implement and how the architecture we chose would be mapped to programming in c. Our main debate was between using subsumption architecture and finite state automata, as in the ants problem set. We figured there would be benefits to both but that there may be trouble turning a complex FSA into reasonable code. The FSA model would be much more complicated to implement so we picked subsumption architecture and went with it.

In the end we decided upon using subsumption architecture as our basic design model. This would be an architecture that we would be able to code up fairly easily, by simply having a global variable for each output value passed by a block within the diagram to another, possibly subsuming another signal. If each of the functions representing diagram blocks are called, the functionality of each can depend on the current values of each of these global signals. In other words the previous signals are maintained during each time loop when each subsumption block calculates its desired output based on these input signals. It seemed like a reasonable approach that was within our grasp and so we took it.

Another reason why we favored subsumption architecture is that we could start easy and build up to a more complex design as time allowed. Seeing that we were both first time robot builders we were both unsure of our abilities and unable to predict exactly how much could be acomplished in the given time frame.

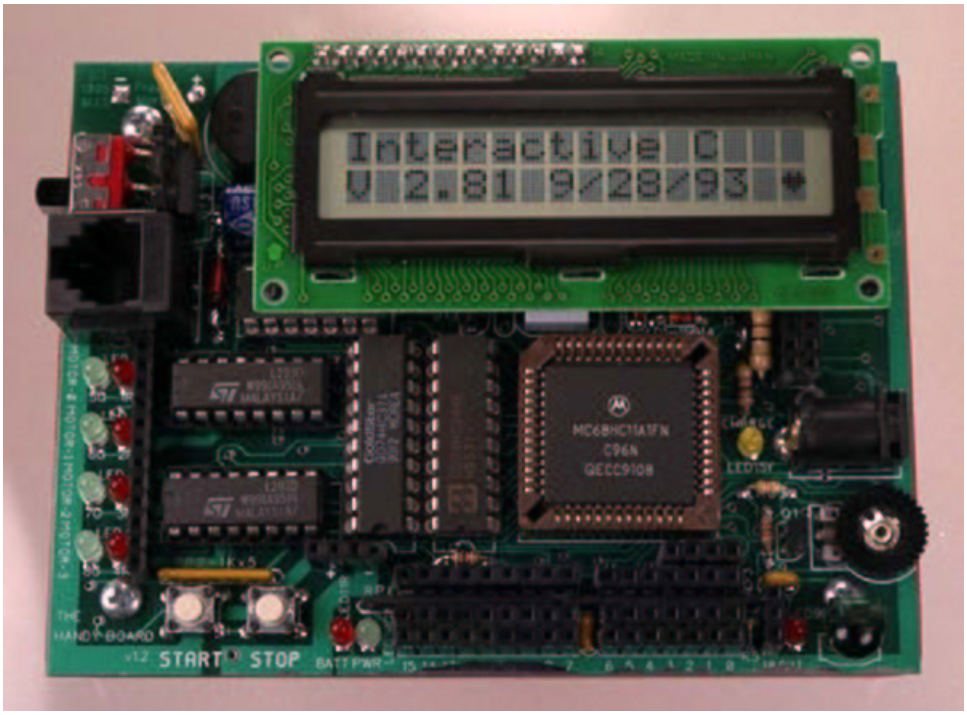

Figure 1. Handy Board.

2 Design

The most important question that we had at the design stage of the project was about finding the right hardware to implement the robots. We knew that a good control is the most important, as well as the most expensive part of robot construction, so we looked at several options, including Lego Mindstorms Kit and Media Lab Handy Board. Luckily, we were able to get our hands on the latter, which turned out to be an awesome piece of engineering. In the following sections we will give an overview of the Handy Board and the Cricket, which is a smaller version of the same controller.

2.1 Handy Board Controller

We have decided to use the Handy Board as the brain of our cat. The Handy Board (Figure 1) was developed at the MIT Media Lab, and it was aimed at making the life of amateur robot builders (like us) easier. The Board is based on a 52-pin Motorola 6811 microprocessor with system clock running at 2 MHz, and 32K of battery-backed CMOS static RAM. The Handy Board has two L293D chips capable of driving four DC motors, powered header inputs for 7 analog sensors and 9 digital sensors, and a 16×2 character LCD screen (see Figure 2 for layout).

The core element of the Handy Board is an 8-bit high-density complementary metal-oxide semiconductor (HCMOS) microprocessor MC68HC11A8 by Motorola (Figure 3) with on-chip peripheral capabilities running with a nominal bus speed of 2 MHz. The HCMOS technology used on the MC68HC11A8 combines smaller size and higher speeds with the low-power and high-noise im-

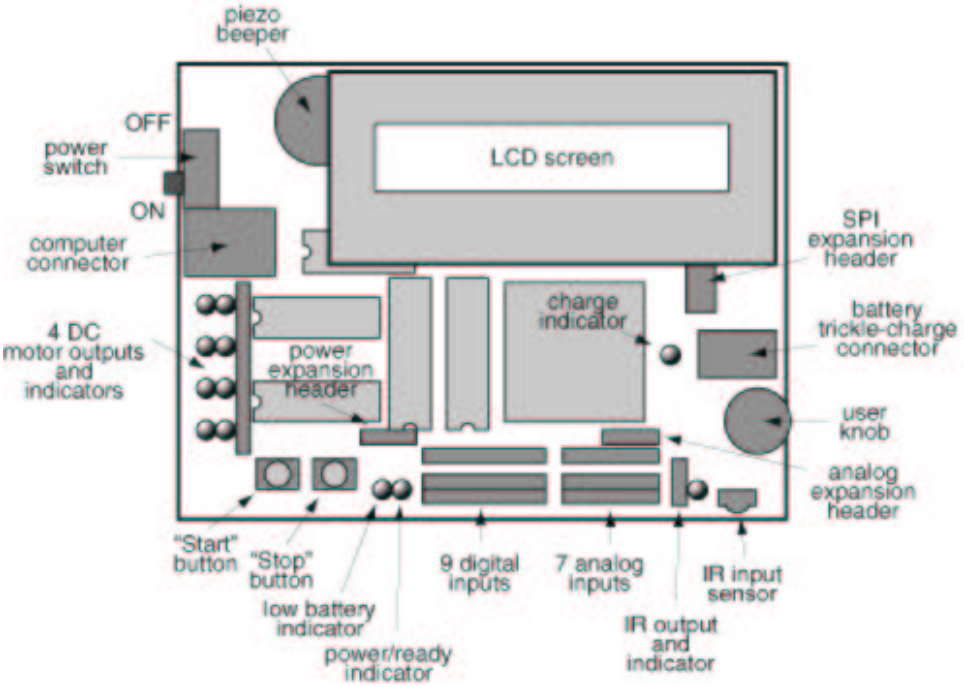

Figure 2. Handy Board layout.

munity of CMOS. On-chip memory systems include 8 Kbytes of read-only memory (ROM), 512 bytes of electrically erasable programmable ROM (EEPROM), and 256 bytes of random-access memory (RAM).

The controller of the Handy Board runs assembly, but there exists a convenient interface to the hardware based on a virtual machine programmed in a C-like language called “not-quite-C”. This language is both easy to use and has a set of pre-defined functions to interact with the underlying motors and sensors.

2.2 Cricket Controller

In order to implement the mouse, which requires less functionality, we decided to use the Handy Cricket from the Media Lab (Figure 4). The Cricket is a miniature copy of the Handy Board with a restricted set of functions, but nonetheless powerful enough for our purpose. The Cricket has the following feature set: Microchip PIC microprocessor with built-in Logo interpreter (Figure 5). 4,096 bytes of user program and data memory (this memory preserves program and data even when the Cricket is turned off and batteries are removed). There are outputs for two DC motors, two plugs and one bi-color LED on each output, inputs for two sensors (sensor value may be read as true/false or converted to a number from 0 to 255), two bus ports, which allow the Cricket to interact a large collection of other devices, a built-in infrared transceiver with raw data rate of 50k baud, and a piezo beeper, a program run/stop button, a power LED, and a program run LED.

The Cricket is programmed using the Cricket Logo developed at the MIT Media Lab. This language has the same easy interface to the hardware as the “not-quite-C”, but it was designed

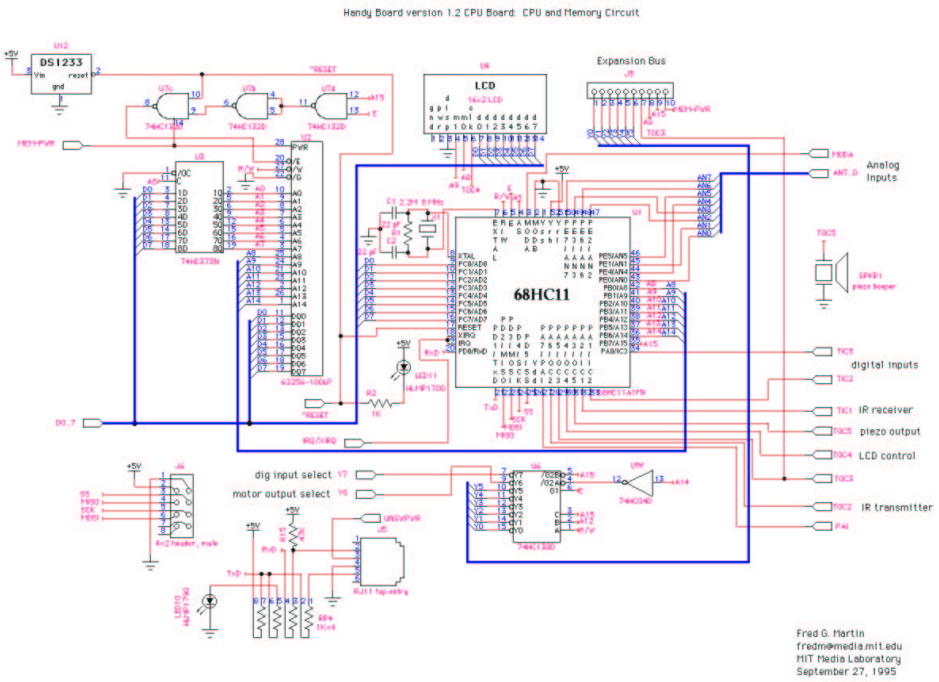

Figure 3. MC68HC11A8 microprocessor.

specifically for kids to program.

2.3 Sensors And Actuators

The next robot part to worry about was its interface to the real world, or, in other words, its sensors and actuators. We decided to give a try to cheap photosensitive Cadmium-Sulphide resistors from the Radioshack store (Figure 6), and bump sensors (Figure 7) and motors (Figure 8) from the Lego Mindstorms Kit. The light sensors return values between 0 and 255 based on the current light levels, but the Lego bump sensors and motors both have boolean on/off settings with values in between. It turned out that the real world is very different from what we were used to in simulations ("Simulations are doomed to succeed!"). Meaning, out of ten light sensors that we bought we only got three having relatively similar readings in identical lighting situations, which was sort of surprising.

2.4 Lego Blocks

After we got the control boards, sensors, and motors working, we had to devise a way to hook them up together so that the construction wouldn't fall apart. We decided to use Legos for attaching parts together. Initially, it seemed like an easy thing, as if we went back fifteen years right in our childhood. Our first cat looked just wonderful – four large wheels mounted on a long body... Then we turned on the power and realized there was a problem... Actually, two problems. First, the robot

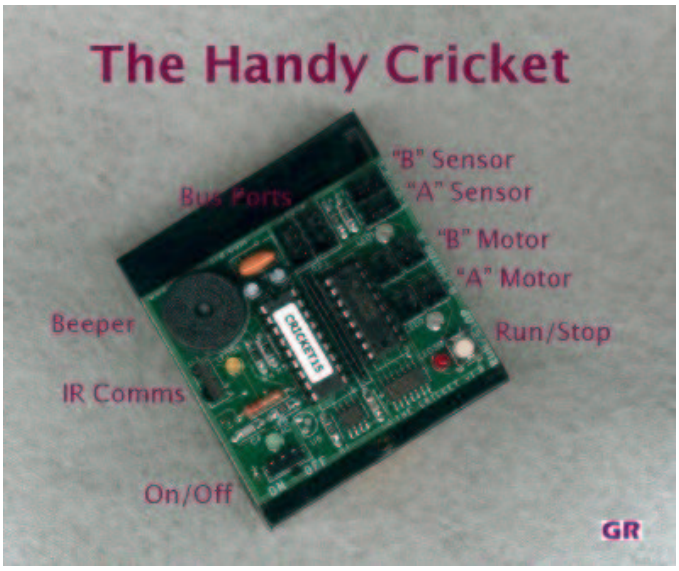

Figure 4. Cricket.

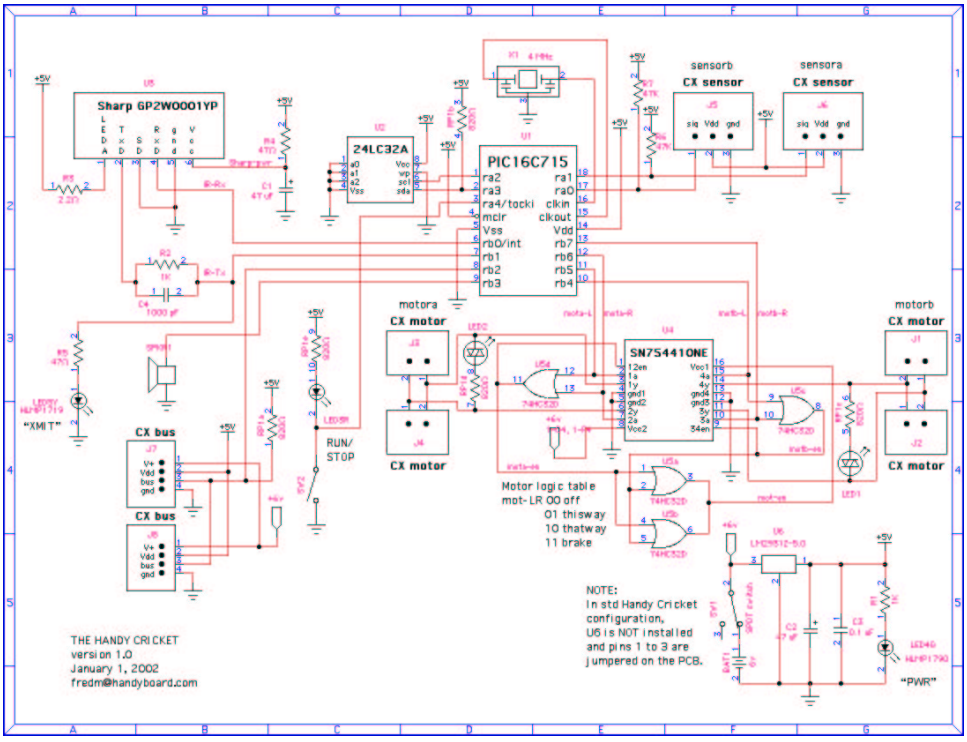

Figure 5. PIC microprocessor.

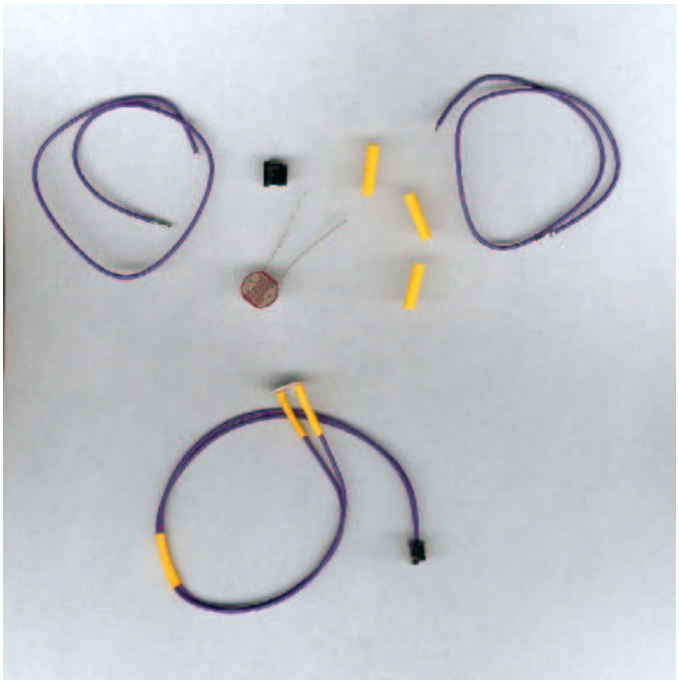

Figure 6. Light sensors.

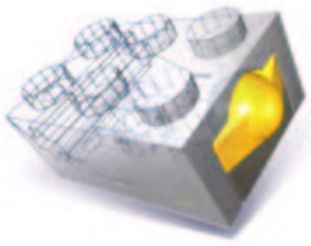

Figure 7. Bump sensors.

Figure 8. Motors.

was too fast (we mounted the wheels directly on motors) – it was accelerating to some enormous speed in just a second, and catching it on its way from the table was quite a feat. Besides, we could't solve this problem even if we pulsed the motors at a very low rate. At the end we decided to use a gearbox.

Second, it couldn't turn. In our first version of the Tom we had four rubber wheels, two of which were passive (in the front of the vehicle). The combination of having four wheels and a long body made it such that turning was nearly impossible. Meaning, it could, but the radius of its turn was closer to that of a jet interceptor rather than an animal! Therefore, the initial construction was later considerably changed to having a third passive wheel with no rubber tire much closer to the center of mass. This is exactly the reason why Legos are the best construction material for people building robots for the first time: designs can be quicky modified if it turns out that things don't work out. We also assume that Legos might be useful even for "professionals" in the field as a quick prototyping instrument, when it is easy to completely rethink and redo the model.

2.5 Putting It All Together: Subsumption Architecture

Okay, we have two creatures, a small one and a big one, remotely reminding us of Tom and Jerry. They look cool (first robot!!), but what so we do next? How do we make them act and play with each other? What is the approach to use? A bunch of "if" statements? A finite state automaton model? An intricate interconnect of functions and gloabal variables? We decided to take advantage of what we learned in the beginning of the class – the approach called "subsumption architecture". Why is it useful? Well, first of all, it is layered bottom to top, which makes it easy to build a complex behavior from scratch. Second, it is easy to draw the picture and structure the thinking around it. And third, it is easy to quickly implement it in C. Therefore, we decided to give it a try, and we liked the approach a lot. Indeed, at the end, we found ourselves in a position of an engineer who is reusing his or her old and working solutions and creating a more complicated and interesting design with less effort.

Figure 9 shows the subsumption architecture for the cat. As you can see, there are five layers overriding one another, which correspond to the behavior examples decribed earlier. The basic behavior is for the cat to move in a straight line. This is subsumed by signals from the explore block when it is time to turn. Likewise, that layer is subsumed by the light following layer, which is subsumed by the obstacle avoidance layer, which is subsumed by the play layer. Light sensor values are fed into play layer and the light following layer, while bump sensor values are fed into the play and the obstacle avoidance layers. The wander layer also has access to a random number generator to set internal timers for turning and moving in a straight line. The lower blocks of the diagram allow for the robot to move in a straight line when lacking input from higher subsumption levels.

Figure **??**fig:sub-mouse) shows the subsumption architecture of the mouse. As described earlier, the mouse has no sensory input and has no internal feedback loops. The same lower block structure as in the cat architecture are present here as well, allowing the mouse to move in a straight line when no signal is received instructing a turn.

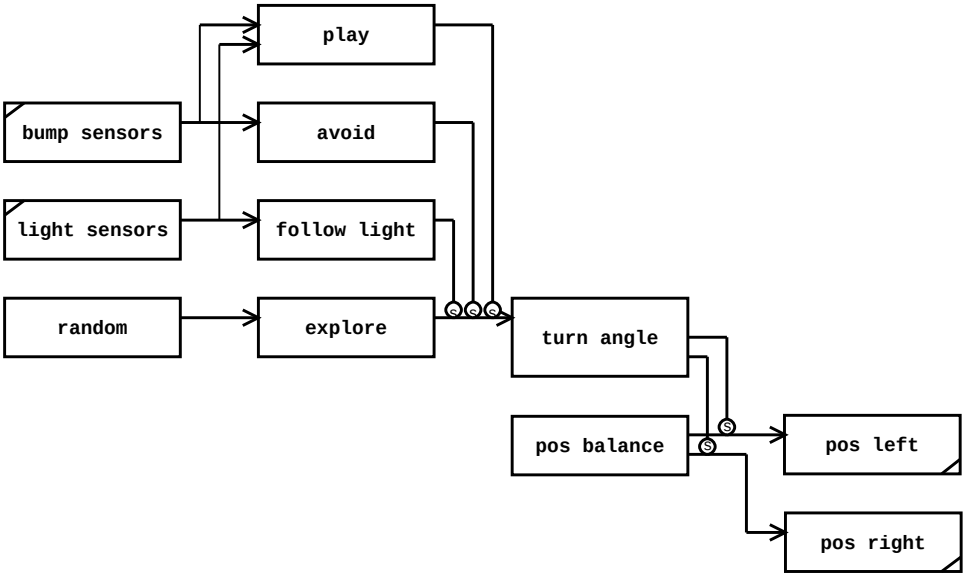

Figure 9. Subsuption for Tom.

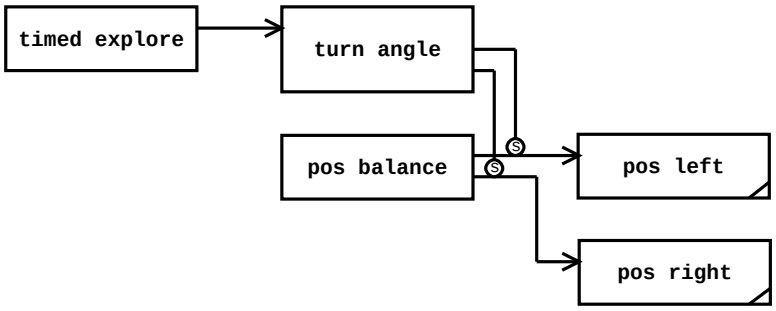

Figure 10. Subsuption for Jerry.

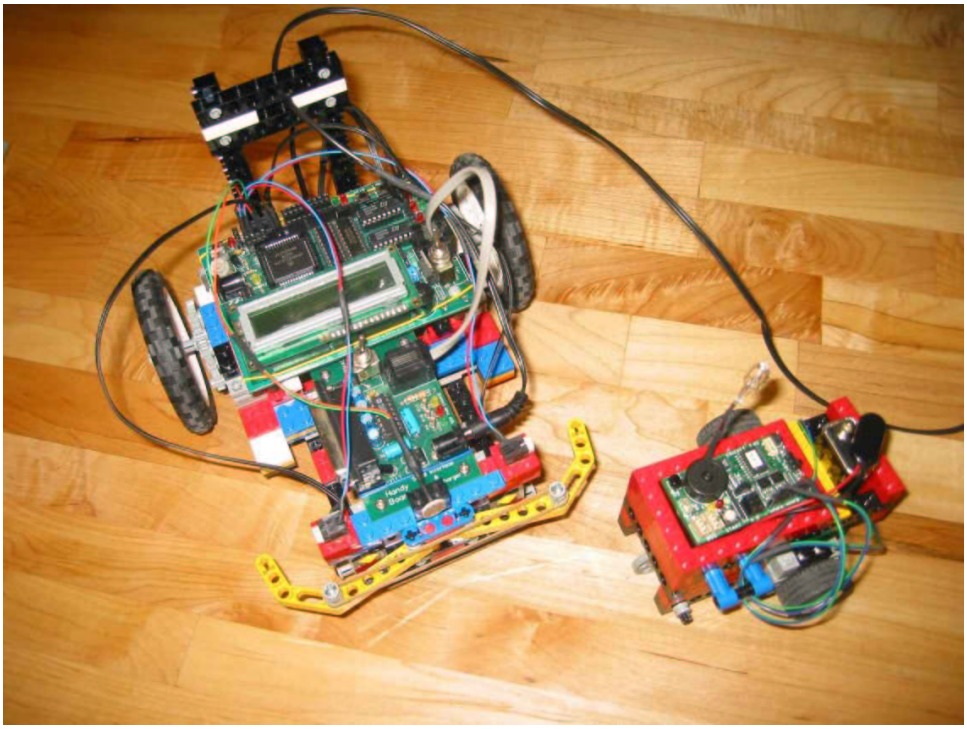

Figure 11. Tom and Jerry.

3 Implementation

3.1 Tom

In this section we will show how we built our two creatures (Figure 11) and how they work. The basic idea is that the mouse has got a light bulb on top of its head, and the cat has got light sensors to track down the mouse. The cat's behavior is to wander around in the darkness while exploring as much space as possible, and go towards the light if there is one. If the cat finds the mouse it begins to play with it, by sitting still in the same place for a while, pouncing on it and batting it a bit, then letting the mouse go away. The mouse can't see the cat, so it just wanders around.

The implementation of Tom is shown in Figure 12. As you can see, the body of the robot is made out of Lego blocks. Tom has two active wheels in the back and one passive wheel in the front (not seen in the Figure, but it is located right under the serial connector). The reason for having three wheels is simple, and it was explained a little earlier: the main problem of the locomotion design is to avoid friction as much as possible, which means that four equal-sized rubber wheels don't quite work (we tried and failed miserably). Even with this design, the robot needs to rotate its two wheels in opposite directions in order to turn. Turning only one wheel while leaving the other steady is not powerful enough to allow the execution of a turn. However, with short time slicing in the inertial world the motion forward together with occasional turn does the perfect job.

In the front of the cat you can see two whiskers, which press against the two bump sensors when touched. While wandering, when Tom hits an obstacle with its right bump sensor, it backs off a

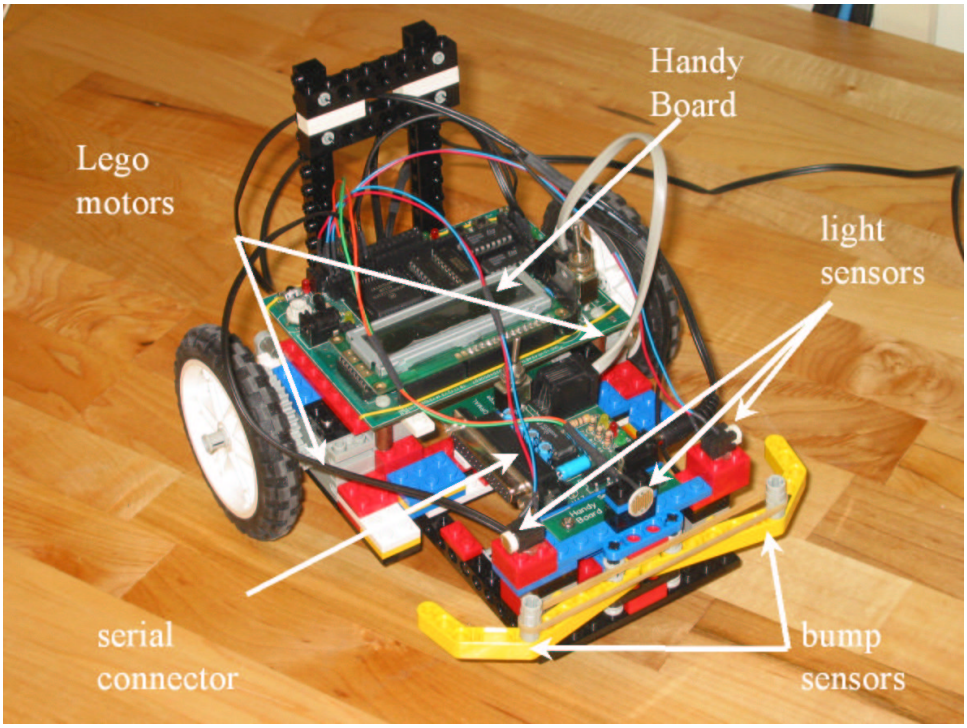

Figure 12. Tom.

little and turns left. The opposite is true for the left sensor. This results in very good obstacle avoidance as compared to a one-bump-sensor robot.

The cat has three eyes: the left and the right light sensors look at +/- 90 degrees from the center line and are used to orient towards the light, while as the front one, looking straight ahead, is used to measure light intensity, which determines the current behavior.

The beauty of the Handy Board is that is runs a C virtual machine. Therefore, Tom's program is written in C, which makes it easy to debug, change, and reuse the code. An example of the high-level code implementing subsumption architecture is shown in Figure 13. The full listing of the program is shown in the appendix.

3.2 Jerry

Figure 14 depicts Jerry, our mouse. As you can see, it is much smaller than Tom, which was exactly our goal. The idea behind the implementation is similar, though. The mouse also has three wheels, two active and one passive. However, a gearbox would be too bulky for the robot of that size, so we decided to do without it and use smaller wheels instead. The mouse has no sensors, and its behavior is completely deterministic. There is no randomness in the time in which it goes forward or turns, as opposed to the cat. On top of the mouse there is a source of bright light (a hallogen lamp) so that the cat would be able to see the mouse from far away. It is worth noting that we had to decouple the system and add another source of power specifically for the light bulb, because when both the cricket and light were run off of the same battery the light drained the

```
while (1) {
  motorR = 0;         /* reset power levels of left and right motors */
  motorL = 0;

  play();             /* implement the layers        */
  avoid_obstacles();  /* of subsumption architecture */
  go_to_light();
  wander();

  set_motors();

  motor(1, motorL);   /* drive the motors */
  motor(2, motorR);
  alloff();           /* turn off all motors */
  sleep(0.025);       /* go to sleep for 0.025 s (pulsing) */
}
```

Figure 13. Example of cat code.

battery very quickly.

You already know that the mouse is controlled by the Cricket, which is programmed in Logo. To give you a feel of how we programmed the mouse please take a look at Figure 15. What does this code do? First, it sets the power of motors to the lowest possible value to compensate for the absence of a gearbox. Second, there is an infinite loop which does the following: go straight for 2 seconds, go back for 0.3 seconds, turn right for 0.3 seconds. This code results in the “random” walk behavior similar to that of the Braitenberg vehicle in Research Assignment 1. Interestingly, this deterministic behavior creates an illusion of an intelligent mouse, which backs off and turns away from a wall if it ever hits one or the cat if it bumps up against it – very nice indeed.

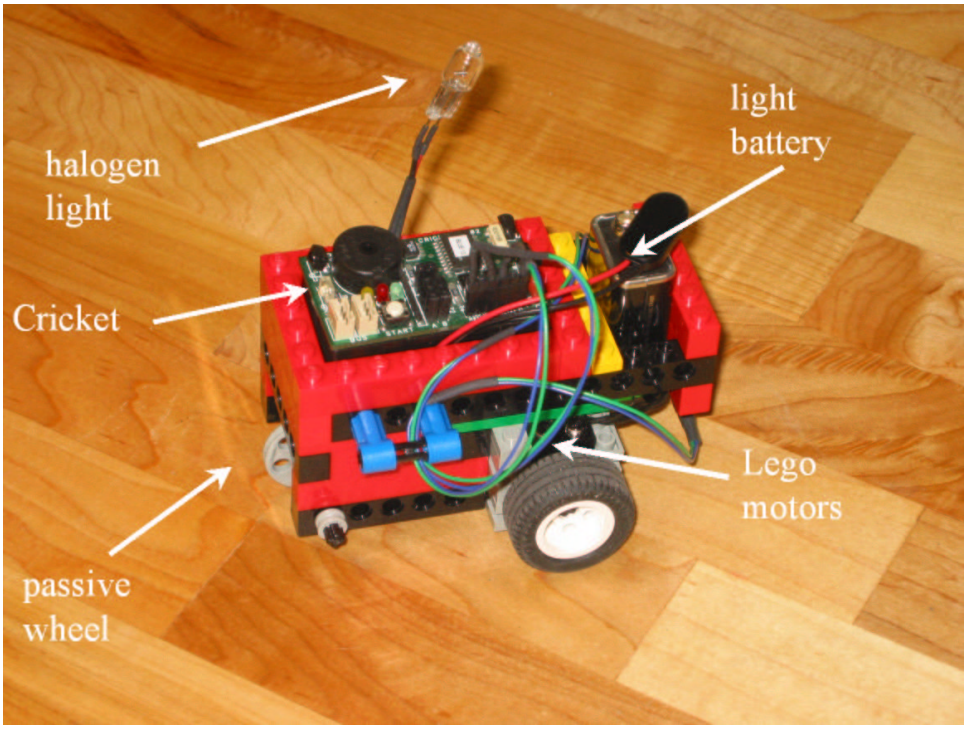

Figure 14. Jerry

```
to wander
        setpower 1
        loop[
             ab, repeat 20 [onfor 1 wait 1]
             ab, rd
             ab, repeat 3 [onfor 1 wait 1]
             b, rd
             ab, repeat 3 [onfor 1 wait 1]
             a, rd
        ]
end
```

Figure 15. Example of mouse code.

4 Results

4.1 Behaviors of Subsumption Levels

Overall each of the subsumption levels alone perform very well. Without a doubt the base behaviors of wander and obstacle avoidance work very well. When these two levels are run alone we see exactly what we expect to see, the cat moves throughout the environment, turning slightly away from things if something is run into and from time to time turning in some random direction after going straight for a long time. Obviously, the light levels play no part in the outcome behavior of the cat during these tests. The mouse is ignored completely unless it is bumped into, in which case the cat will act the same as if it had run into a wall: it will back up and turn away from the bump target.

When the light chasing level is added in we see that the cat almost always behaves as expected. It does in fact notice and turn towards the light when the mouse is within visible range, and then charges towards it. It also keeps the obstacle avoidance functionality when something is bumped into and the wandering functionality when sight of the mouse is lost. But sometimes we see that this action behaves somewhat unlike what an observer would expect. For instance, sometimes the cat will decide to turn towards the wall rather than towards the mouse. Due both to the particular placement of the light sensors on the cat and the reflection levels on the walls, sometimes the reflection off the wall is at an angle that is visible while the light itself is not. When we think about this logically we can figure out why it makes sense and why this type of action takes place, but to the observer watching the cat and mouse drive around together all she sees is the cat turning towards the wall when the mouse is behind it, which is somewhat confusing.

With the light chasing level added we also see that there are fewer cases of when the cat gets stuck in its environment. We know that it is possible for the cat to turn when near an obstacle so that the wheel gets caught on the corner, with the wheels spinning but the bump sensor not being activated. Another example is when the cat gets in a repetitive loop in a corner bumping into a wall, backing up, turning away and then bumping in to the perpendicular wall and repeating its back up and turn away loop towards the original wall. With the light chasing level added in we add new ways for the cat to free itself from these situations by noticing and turning towards the light rather than continuing on in its static original loop.

When the play level is added the behavior is changed drastically. The way in which the cat plays with the mouse are as follows: it chases the mouse through normal light following subsumption levles and when it gets close enough (when the sensor values are at a certain threshold level) it stops and watches the mouse for a timed period. During this time the cat may turn to reorient itself and keep facing towards the mouse but does not move forward. If the mouse moves too far away during this period then the playing is cut short and the cat resumes normal light following until within a suitable range again. After the cat has sat and watched the mouse for the timed period it charges at a faster than normal speed and rams into the mouse, chasing it until a bump sensor is activated. The cat then backs up a short ways and repeats the ramming three times. After this the cat in effect 'closes its eyes' and lets the mouse escape for a short period so that it may resume chasing it again.

When the cat is moving around in standard form it moves at about the same speed as the mouse, maybe a little slower. But when the cat is pouncing on the mouse the speed is almost doubled,

mimicing an attack or leap by a cat in an attempt to catch prey.

To the observer it is sometimes confusing as to whether the cat is sitting still because it is stalking the mouse or if it is in the phase in which it sits still to let it escape. The one way in which we know the cat is pouncing on the mouse and not simply orienting and moving towards the light in standard form is the fact that pounces are done with this heightened speed level. Sometimes it can be confusing, because if the cat is stalking the mouse and the mouse moves such that the cat is now out of threshold light range the cat will resume standard light following behavior until it is within this range again. To the observer the cat is moving towards the mouse, sitting still while stalking, then moving towards it again but not pouncing as the observer would expect but just moving towards it like normal, then stalking again. Sometimes these phases can be repeated numerous times in which the cat stalks but then has to catch back up, then stalk again, etc.

It seems like part of the problem is that there is no visual distinction between the different behavior phases executed by the cat other than the speed of the pounce, which makes it difficult to distinguish in some cases. For instance, if the cat had some sort of paw that it could use to hit the mouse around we would be able to tell without a doubt that the cat is now playing with the mouse. Another idea is a tail that it could wag in certain ways when certain actions are taking place. All of these are extra pieces that would not really modify the behavior per se but would make it clearer for the observer.

4.2 Physical Creature Interaction

In many ways the robots behave differently from what would be expected simply because these are actual physical robots and it is easy to forget that the laws of physics and logic still hold. For instance, when the cat runs into the mouse at full speed the mouse gets pushed around and sometimes will be lifted up a bit. Sometimes the wires on the mouse get tangled in the whiskers of the cat and need to be freed by someone (like me) so they don't pull apart and break each other. It is very easy for the two robots to run into each other and get themselves stuck against a wall with no way to free themselves. When either of the robots need to escape from a pinned situation the standard response is to turn away from the side that had its bump sensor compressed, but sometime the wheel itself is pressed against the wall and is in the way of executing a successful turn.

As noted before, the cat has a tendency to get stuck in corners while wandering. When the mouse comes within visible distance of the cat, the cat can exit this repetitive loop in the corner by turning to chase the mouse. The problem with the cat in the corner is that all behaviors are based on local current state of the sensors, so the cat has no way of knowing if it is stuck in the corner. Perhaps there could be an internal counter that marks the sequence of left and right bumps recorded by the bump sensors and act accordingly if a cornerlike sequence is encountered, but there is also a possibility that this sequence could be encountered by luck alone.

The mouse too has a tendency to get stuck against obstacles in the environment, but this has more to do with the fact that there are no sensory input mechanisms in place on the mouse to let it know when it is in contact with something. The mouse repeats a loop in which it goes forward for a fixed amount of time, goes backwards for a small amount of time, and then turns either left or right for a fixed medium length amount of time. Many times the mouse will just go up to a wall and keep spinning its wheels attempting to go forwards. Sometimes after this it will briefly back up and then attempt to turn, only to find itself very close to the wall without enough clearance room to actually perform the turn. When this happens the mouse in effect stands still against the

wall, unable to move forward or to turn. These types of situations don't tend to last too long, as the mouse eventually frees itself from its position. Corners are still a problem for the mouse though because even after the time of random turning there is still a good chance that the direction it ends up facing will be towards a wall and it will be stuck again.

When the two robots are together and pinned against a wall or stuck in a corner it ussually takes a while for them to become free. The way that the cat plays with the mouse is to pounce on it and then sit still and let it escape. If the two robots are touching each other then the cat must be pouncing (or else it would have stopped to track the mouses position) and will allow the mouse to escape by sitting still. But if the cat is pinning the mouse up against a wall then there is either no way or a narrow way for the mouse to escape. Seeing that the mouse has no sensory input the freedom of the mouse depends solely on luck to direct it towards this possibly nonexistant escape route. In other words, when the two robots become pinned together next to a wall it is often the case that they will stay there stuck for quite some.

4.3 External Influences on Performance

There are many elements of the environment that influence how the cat and mouse interact. One of the most obvious influences is the type of floor that the robots are run on. In all of our experiments the robots were run on tile and the parameters of the system have been set based on this setting. If we were to move our robots to a carpeted room they may not work at all. Perhaps the robots would be unable to turn at all on a carpeted surface, or would just move at a different speed. Whatever the differences may be in physical movement of each robot in the new environment, the interaction between the two will definitely change. This is because the parameter values set beforehand allowed the robots to be able to do things such as turn a particular angle in a particular time frame. For example, if instead of backing up and turning the roughly 30 degrees that the cat turns when a bump sensor is hit but instead turning 10 degrees, the turn will not be enough to clear the obstacle when the cat continues forward.

If the ambient light level in the room is very high (meaning not completely dark) then the cat will only require a small amount of light from the mouse to reach the threshold light levels that instigate high level behavior. If the lights in the room are on then the mouse will continually think it is very near to the mouse and will attempt to play: stop and stalk the ambient light and then pounce forward until it hits something, most likely the wall, even if the mouse is nowhere in sight. Unfortunately this is not very smart behavior, but we built our system to react solely based on the light levels read, and that is what is happening.

We also notice that interesting behavior can be seen when there is a high level of reflection off of the walls. At certain angles the cat will be able to see the reflection of mouse light when the mouse itself is not visible, and will turn towards the wall in an attempt to chase the mouse.

For all of our experiments we made our walls and obstacles be a height that would guarantee that a bump sensor would be hit if the cat ran into it head on. With shorter obstacles it would be possible that the cat could hit it head on but not compress one of the bump sensors.

4.4 Videos

We have included four videos along with our report but I will briefly explain them here.

The first video is shot with the lights on in the room and demonstrates the wander, obstacle avoidance, and light balancing subsumption levels. We see the random wandering taking place and correct turning when an obstacle is bumped into. As well as this we see that when a dark object is held up in front of one of the side light sensors the cat turns away. The dark object makes the light sensors imbalanced and the light leveling subsumption layer makes the cat turn away from this dark object before it bumps into it. In the background we can see the mouse executing its standard wander.

The second and third videos are very similar to each other. These are shot in the dark with same behaviors as in video one, but also with the behavior that allows movement towards a light source. The play behavior is not included in either of these videos. We can see the cat successfully wandering, bumping into objects and turning away from them, and orienting towards and chasing the mouse when close enough. We can also see examples of the subsumption layering in these videos: even when the cat is chasing the light the standard obstacle avoidance behaviors take over when an object is run into. After the obstacle avoidance sequence is completed the light chasing is resumed.

The fourth video shows all three of the above subsumption behavior layers as well as a first version of the play layer as well. When the cat gets very close to the mouse after chasing it, it stops and tracks it for a few moments. The cat then moves forward to pounce on the mouse, moving directly towards the light until the mouse is hit. It then sits stll for a few moments and lets the mouse escape.

Note that this is an early version of our play routine and not the final version that we accomplished. The pounce towards the mouse is done at standard speed rather than at full speed, the movement is still a bit jerky, and the mouse is only pounced upon one time rather than three. Unfortunately after we got the improved play behavior working the serial connector of the cat refused to download any more and left our cat temporarily immobilized. And just our luck this happened right as we were preparing to film the new and improved behavior. This just goes to show that when working with robots sometimes things break and it is very different from the ivory world of simulation. Serial connectors don't just stop working in simulation. ;)

5 Conclusion

So what have we learned by building our robots? Lots of things. It took us a very long time to get anything to work right on the robots. Aside from figuring out how to get all of the digital pieces to talk to one another, just building a successful structural design and getting a simple code loop working took a long time. A lot of time was spent on exploring the capabilities of the motors and sensors: what type of motion was possible and what types of information could be delivered to us by the sensors. We also found that many little tweaks of the code were needed to get successful behavior from the robots.

Sometimes it was difficult to tell exactly what or why the robot was doing what it was doing, and it took crawling through the code to parse the logic we had implemented to find where the problem was. For example, sometimes the cat would just spin in place and we had no idea why this was happening; the rest of the time it behaved just fime. We found that there was a glitch in our implementation of the random number generator, but that this glitch was only shown in particular circumstances.

Part of the difficulty of not being able to tell what exactly was going on originated with the fact that there was no output from our robots, no log that we could analyze. We made use of other ways that allowed us to get a peek into the internal state of our robots, such as lights flashing or beeps beeping at certain times in the code. As mentioned earlier, the internal state of the robots had to be deduced from the behaviors exhibited, and sometimes these behaviors were confusing. If we were to add other visual aids such as a tail that would wag or an arm that would play with the mouse, the actual behavior wouldn't change that much but it would be easier for the observer to tell what was going on.

We also learned that things don't always work as you think they would. Less than a third of our light sensors worked properly and one might suppose that it takes a long time when building a larger robot to test the individual components. Physical environmental influences such as friction and reflection played a much larger role than we first imagined.

Generally, we learned that building robots is tough, but it is fun to see the birth of a creature out of a pile of components. Even though it was very frustrating at times we both found the reward to be much greater than this small amount of pain. We wanna build more robots!

A Code