May 1, 1995

Name: Erik de la Iglesia

TA: Scott

Instructor: Dr. K. L. Doty

University of Florida

Department of Electrical Engineering

EEL 5934

Intelligent Machines Design Laboratory

Final Report

R2: Real-Time Contouring

Table of Contents

Abstract

The author investigates a minimal sensor requirement for solid object acquisition and contouring for a mobile autonomous robot. Solid object acquisition is a superset of obstacle avoidance while contouring is a superset of wall-following behaviors requiring the agent to perceive both the displacement from and normal vector to a piece-wise solid surface. For the purpose of this research, a surface is considered to be bounded by a finite perimeter of one surface type. A surface type is defined by having a given nonvarying reflectivity. It is demonstrated that a differential sensor pair and a single tangential sensor are sufficient to realize the aforementioned behaviors. The necessary calculations can be significantly reduced by using imposed linearity upon non-linear sensors with variable emitter currents. Finally, the concept of the partial solutions form of behavior algorithms allowing for highly stable algorithms is discussed as a general method for agent behavior.

Executive Summary

The R2 agent is a Stalt derivative with many advanced features. High precision shaft encoders, extensive sensor suite, and rotating sensor head capability allow for a variety of behaviors and modes of operation. Over twenty sensors of four discrete types allow for proximity and normal detection, battery level sensing, corner detection, head positioning, and shaft encoding. Through effective use of these sensors, the agent can be programmed for straight-line motion and solid face contouring. The objective of the author is to establish a learning base platform from which many levels of software development are possible. With the capabilities of the agent documented, focus shifted towards the specific problem of contouring using a minimal sensor suite.

Contouring involves complex behaviors. Initial attempts showed that any contouring algorithm functions best when some form of memory is utilized. The behavior arbitrator of the agent can normally be thought of as a processor intercepting information from the sensor groups and making decisions based on some criteria that then guide the agent through a given scenario. Such an algorithm can be viewed as purely combinatorial and is prone to instability. If the behavior arbitrator is provided with additional information regarding the current state of the system, more stable behavior is possible. In fact, an optimal algorithm involves a set of partial solutions and a memory table of past solutions.

When implemented, the partial solutions structure eliminates the need for randomness in a behavior system. Repetitive executions which would normally indicate an instability are identified as a series of unreached solutions and a new partial solution is sought. The past solutions table can be implemented as a counter. Every attempt to execute a partial solution incremented the associated counter. A “futility” limit determines if after some number of attempts the execution is impossible and another method is necessary. In practice, few algorithms require the partial solutions approach. However, those that do are inevitably the most critical to the function of the agent.

Introduction

Mobile autonomous agents are explorers. The designer creates a platform capable of some perception in the near environment and then programs some behaviors appropriate to the goal of the agent. This code must compensate for any and all possibilities that might occur in the target environment. The analogy presenting during the first week of class seems highly appropriate here. If an agent were sent to Mars for exploration, commands going to and from the agent would have a certain latency due to the distance from the controller. If a crater were to sudden open up immediately in front of the agent, the image of this impending catastrophe would reach Earth at about the same time that the agent was destroyed as it hit the bottom of the hole. Accurate real-time perception is vital for the survival of the autonomous agent.

In lab, the impending catastrophe may only be bumping into a wall or tipping over. However, one must consider such a failure equally grave to falling into some alien trench. In both cases, the ability of the agent to act autonomously is compromised. The only difference is that one agent that can be picked up, brushed off, reprogrammed, and probably costs less. One must accept that there is no such thing as limited autonomy. Either the agent can achieve the goal it was programmed for, or it can’t. For the purpose of contouring, bumping into an object may not be a critical flaw. Even if the agent requires ten attempts to exit some interior concave surface, the fact that it eventually does indicates that it succeeded in contouring. Even brushing against a surface means little if the agent alignes itself properly after doing so.

The algorithms presented in this paper are goal oriented. Behaviors have a specific purpose which drives the agent to seek a solution through a group of partial solutions. Individual readings of sensors, accomplished through a sensor service routine can be viewed as a group function or singularity. Since the singularity can not itself drive behaviors, some intermediate decision layer must process the singularity information along with the previous singularities, current state variables, and global variables. This layer is referred to as the global variable preprocessor. All variables that are functions of information gathered in a singularity are handled by this preprocessor. A priority arbitrator then compares singularities and devises a partial solution based the desired goal as expressed through system state variables.

This paper focuses on the development of a minimal system to contour a closed surface. The platform built is by no means minimal since it was constructed as a learning base. The premise is that by creating a full-feature agent, an idea of the minimal agent will develop as unused sections are removed in future models. The sensing systems developed include Sharp infra-red sensors for displacement and angle detection, Siemens sensors for shaft encoding, CDS cells for corner detection, and limit switches for head movement. Control algorithms were implemented using Interactive C on an HC11 architecture.

Platform

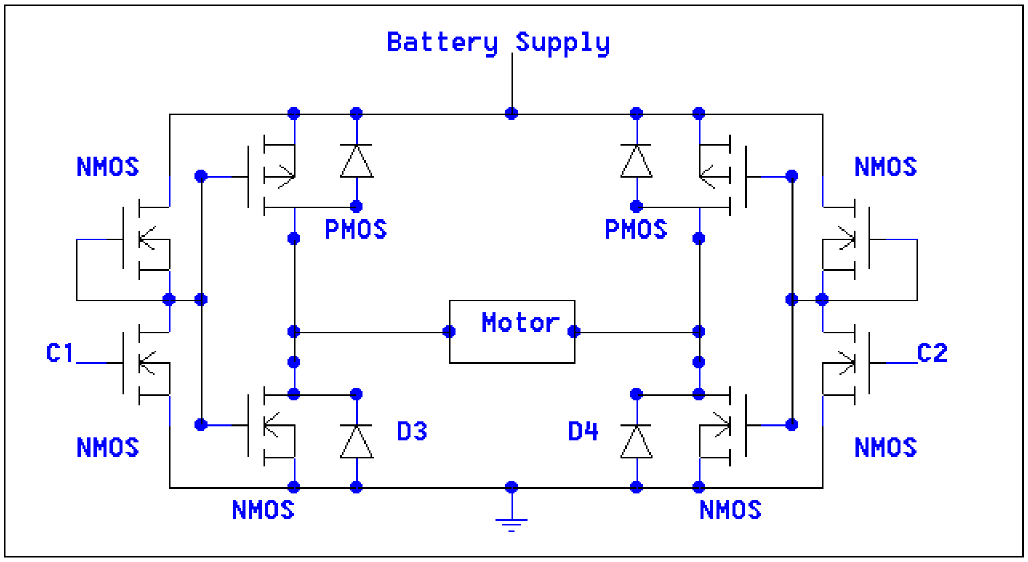
The R2 agent uses a Stalt derived circular wooden platform. The bottom of the main dish houses two converter Futaba FP-S148 servo motors with incorporated shaft encoders. The eight-cell battery pack is housed in a wooden casing which also supports the castor build using Lego components. The wheels are placed directly over the center axis and the battery pack is adjusted so as to provide the necessary rear-weight to prevent tipping over. The main processing board features a variety of subsystems on one Vector board with inverted B1\B2 type pattern. This pattern consists of a distributed ground plane with isolated holes. A component is secured and grounded by simply soldering the ground pin to the board. The board is mounted with one inch standoffs to a wooden board of equal dimensions which slides in place via two vertical posts mounted horizontally over the base. A support beam parallel to the circuitry but one inch higher strengthens the structure and provides a means of securing the rotating head through the upper servo. The longer shaft of the servo is mounted upside down to allow the motor to be placed on the top surface of the head and have the shaft come through the center and lock to the support beam. Some of the head base is removed to better allow the connecting wires to move while the head is rotating into position.

The controlling circuitry is based on a CGN module serving as a host for the HC11 processor. Memory is expanded to 32K using the standard Jones / Flynn method. An AMD PALCE22V10 serves as address decoder to the ram and the six digital ports and display device. A one byte display is implemented using two TI311 chips with latches tied to the PAL control lines. To reduce power consumption, the flash control is tied to the 20% duty cycle of the 40KHz generator implemented using a single 74HC390 decade divider. The analog to digital reference voltages are brought out to potentiometers for fine adjustment and all data lines are brought to test points at the edge of the board. To preserve memory during a battery change, a Dallas battery-backed RAM module is used.

Twin L293 motor drivers control the Futaba drive motors and the head servo. A single 74HC14 hex Schmitt trigger is included to clean any signals which to not meet HC specifications.

Actuation

The two motor groups are controlled independently. All motors are powered directly from the battery via the L293's. Drive motors have the additional capability of speed control achieved by pulse width modulation the enable control line. Head motion is constant at full power. The circuit required to drive a motor device using low current TTL level control lines is called a four quadrant controller, or H-bridge. A schematic of the H-bridge follows using MOS technology parts.



When C1 is high (+5) and C0 low (GND), current flows across the motor from right to left. When the control signals are inverted, current flows left to right. Equivalent signals will cause zero current to flow. The inputs are level shifted from 5V to the battery level via a depletion mode NMOS inverter. Level shifting is necessary to drive the signal high enough to turn off the PMOS devices.

Because the motor is essentially a resistance in series with an inductance, a rapid switching of direction will cause a problem. The field built up in the motor will oppose the voltage forced by the H-bridge. This back-EMF is forced into the battery and causes a massive spike on the power system. The diodes across the transistor terminals attempt to limit this by allowing the current to escape directly. However, the diodes do not completely solve the problem, and some IC versions of this device do not have protection diodes. Software control can help reduce the power spikes by gradually updating motor speed instead of instantaneously switching motor direction.

The reasons for developing a panning head are two-fold. First, using the Choi-Jantz sensor placement standard, (see sensor section) most sensing is done in the forward direction. This means that the agent must be facing the object being scanned. If the agent is a few degrees off an intended course, forward scanning would completely miss an acquire request. However, scanning in a 180 degree field from the right and left of the agents trajectory would detect virtually any object within the sensor range. Second, the minimal approach requires that as few sensors as possible be used to gather the most information possible. Allowing sensors to perceive information in a 180 degree field is certainly better than just a single linear ray.

Sensors

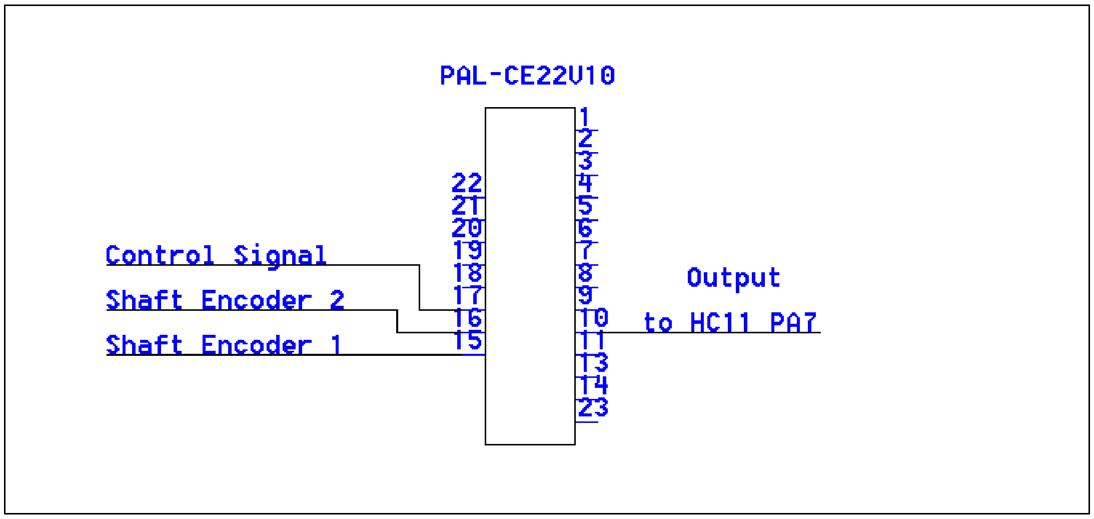
Shaft Encoders

Implementation of shaft encoders would be trivial if the agent could be designed to move very slowly and devote great resources to sampling and correction. Unfortunately, no behaviors are so fundamentally simple that an arbitrarily large time slice can be given to a shaft encoder routine. One can thus form two important guidelines for shaft encoder implementation. First, shaft encoders should provide a great deal of information (number of counts) in the shortest possible time. This allows sampling and correction routines to drain a minimum of processor resources. Second, the implementation should be one such that the processor is relieved of as much of the control of the sensors as possible. Ideally, the processor would simply read the sensors in an infinitely short period of time and correct the motor PWM signals at an infinitely small percentage of the total system time slice. Of course, such a shaft encoding system is not possible. However, one can approach said system by making use of processor-native hardware to motor control using shaft encoders.

The recommended method of implementing shaft encoders appears in the Jones-Flynn book. Shaft encoder signals are read by PA7 and the IC3 input capture pin. An interrupt routine must service every rising edge on the IC3 pin while the PA7 pulse accumulator hardware automatically increments an eight-bit counter upon the edge selected by a control register. Several problems can occur using this method. Since issues relating to the sensor hardware are dealt with later, we will focus first on the immediate problems within the HC11.

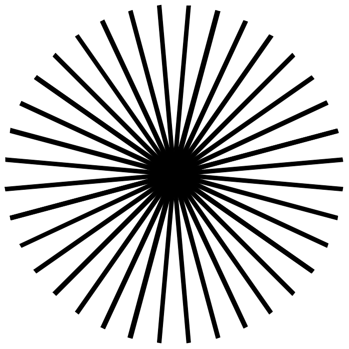
Interactive C (IC) runs as an interrupt driven multi-tasking shell. It is undesirable on a processor of this scale to run many interrupt driven routines simultaneously. This is equivalent to a modern PC running Windows and attempting to multitask several programs each requiring protected mode features and using BIOS routines instead of the INT 21 DOS services. Additionally, the functionality of the IC3 hardware is only 50% exploited as it can capture both rising and falling edges while the pulse accumulator would require a further interrupt routine to swap triggering modes. In short, the system is both software-intensive and hardware-wasteful.

If the HC11 were equipped with two pulse accumulators, it would become trivial to read each shaft encoder signal. The hysteresis buffering of the PA7 pin also makes it ideal for reading a signal which may not necessarily by a perfect square-wave. An easy implementation of this idea, not involving redesign of the processor, is to multiplex the signals into the signal pulse accumulator. If sampling periods are held to a minimum, each sensor could be read after the other giving very accurate readings. Additionally, the software control is minimal. The count register can be zeroed by a single POKE command and then read after an arbitrary delay. This is the solution that the author pursued. The schematic of the shaft encoder signal multiplexer follows:

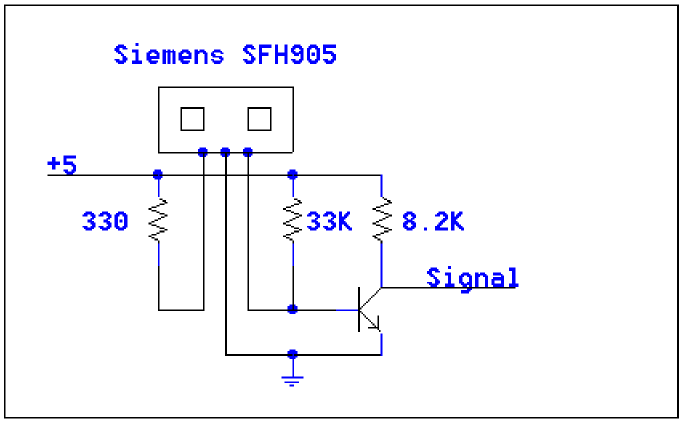


With the additional four pins available at the time of the shaft encoder design, the PAL device becomes the necessary multiplexer. The control line is buffered via a 74HC374A memory-mapped output port located at $7000. By alternately selecting the two signals, the pulse accumulator reads both sensors with no software overhead involved. With a functional software and sampling method implemented, the researcher attempted to perfect the mechanical portion of the shaft encoding system. The suggested method of obtaining pulses from a driving motor was to use a partitioned disk similar to the one shown.

An infrared emitter-detector, such as the Siemens SFH905 that the researcher demonstrate earlier in the semester, would detect each black line as an interruption on the IR reflection and send this signal to the sensing hardware. Naturally, the number of sections directly correlates the number of counts acquired in a given time period. (See diagram to right) The position of the sensor and the location of this disk along the gear-train, among other things, influenced the success of this implementation.

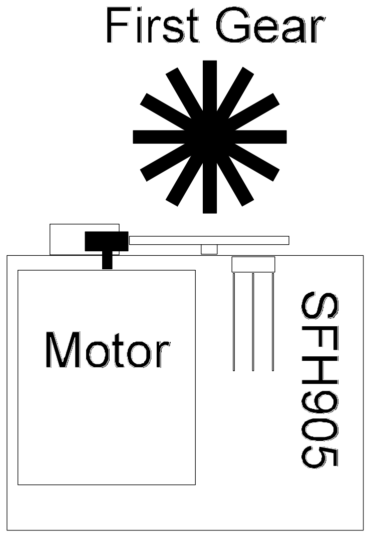


Other students reported problems with signal bounce and level earlier in the semester using the Siemens sensor. The researcher chose to implement a high-speed disk and signal correction circuit to supply the shaft encoder signals to the processor. Below is a diagram of the system designed. Use of a classmate's idea to place the sensory element inside a servo motor was made. Note also that the circuit used does not have the base resistor normally used for a common-emitter amplifier. The 33K pull-up resistor cannot drive the base higher that the Vbe_sat level. All possible bouncing of the signal at higher sensor voltages is eliminated since the signal clamps at about 0.8V. The sensor does not sustain damage from this method.



Even at very high speeds, a nearly perfect square-wave is present at the signal node. To allow for very rapid sampling, the Siemens sensor is placed inside the converted Futaba servo by drilling a hole through the plastic casing and allowing the sensor to reflect off of the primary gear. The above schematic illustrates how this is done.

It is important to note that simply painting the primary gear in partitions is not enough. The researcher found that the plastic gear is actually partially translucent to IR light. Initially, an unpainted section will not reflect sufficient IR to trigger the sensor. After repeated use however, the buildup of grease on the opposing side will make the unpainted areas partially reflective. The only solution to this dilemma is to use a reflective paint over the remaining sections of the gear. Testors brand silver and flat black paints are ideal for this. It is also vital to completely cover the wheel surface. Border regions should be handled by applying the second paint over the first thus eliminating any possibility of the old plastic surface showing.



A finite bounce still occurs using this system. Since any high frequency bouncing that is seen through the multiplexer is corrected as actual signal by the PAL, it is necessary to filter the incoming signal to eliminate said bouncing. Experimentally, anywhere from two to seven bounces per active signal edge can be seen. The immediate solution, which works very well, was first discovered by a classmate and used here to a much larger extent. A large capacitor is placed directly at the input of the pulse accumulator. The reason for this placement is two-fold. First, the PAL input is much more sensitive that an HC11 input and should not be “modified” unless absolutely necessary. Second, the inbuilt hysteresis correction of the pulse accumulator input makes such modification possible and acceptable. Experimentally, a value greater than 1.0uf over the pulse accumulator input will eliminate the bouncing and properly condition the signal.

If constructed well, the aforementioned configuration can give about thirty counts per 50ms sampling time. This number depends greatly on the number of sections painted on the first gear. The researcher feels that eight sections is the most physically allowable due to the nature of the sensor dimensions. In the agent designed, six sections are used.

Sharps

The Sharp hack provided an easy and inexpensive method of distance measurement using a pulsed 40KHz Infra-red LED and a modified sensor. This hack was demonstrate during the first two weeks of class and had been well established as the predominant distance measuring technology. However, a significant flaw existed. The analog signal recovered from the sensor had a significant rise and fall time making successive measurements very slow or nearly impossible at acceptable speeds. An analysis conducted by David Novick confirmed these characteristics. Rise and fall times around 100ms could be expected when using the sensor. Although certain situations allowed for times only one quarter of this, reliable measurements could only be taken if the software conforms to this hardware limitation.

After extensive experimentation by this researcher, a solution to this dilemma presented itself. The analog signal taken as part of the Sharp hack was forced over a surface mount capacitor of value 0.1uf. Removing this capacitor should shorten the rise and fall times. After confirming that the surface mount capacitor was indeed integrating the signal, the researcher cut the capacitor and re-evaluated the output signal. Unfortunately, the resulting signal was far too unstable to be of any use.

Theoretically, some value of capacitance below 0.1uf should have been able to correct the instability. Several values tested showed interesting results. A value less than 500pf produced unstable output. However, a value of 0.01uf produced a stable signal with rise and fall times of only 2-3ms. After demonstrating this innovation to several coworkers, the researcher presented the findings to Dr. Doty who subsequently confirmed them. The new hack showed sufficient promise to warrant incorporation into the sensors of future Sharp user. Another significant advantage of the new hack was its preservation of the digital output. This, in theory, would allow digital communication and analog sensory measurement using the same sensor, although not simultaneously.

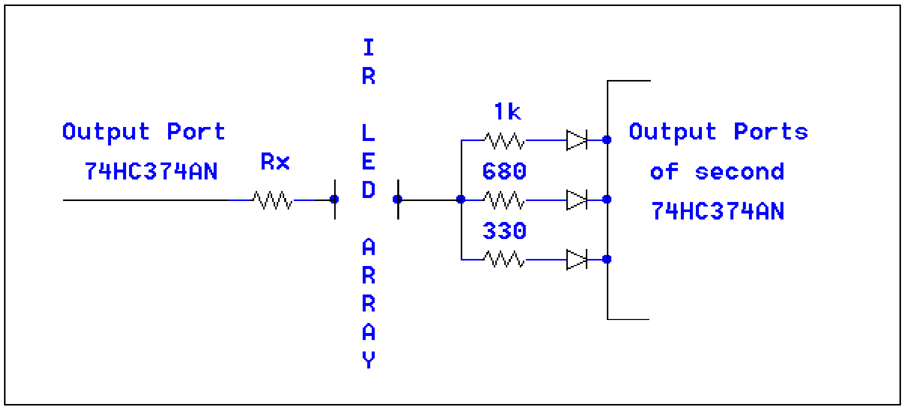
Unfortunately, the arrival of the sensors to be used for this years class brought an unexpected surprise. The newer sensors were redesigned containing fewer surface mount components. This researcher set out to characterize the newer sensors and document the functionality of the old hack and the recent improvement on the new sensors. The initial results showed that the hack was equally effective on the new sensors and that rise and fall times remained in the 2-3ms range using the 0.01uf replacement capacitor. Several problems soon became apparent however.

Primarily, the digital signal was thought to be undisturbed by the analog tap. This proved incorrect. As soon as the analog tap was loaded, even with only the 20pf scope input capacitance, the digital output lost its 50% duty cycle and became a pulse width modulated waveform. When the original sensors showed similar destruction of the digital signal, it became apparent that digital communication through the “analog” sensor might be impossible. Dr. Doty theorized that it might be possible to rectify the analog signal into the original digital waveform using a Schmitt trigger. This solution has yet to be tested.

As soon as the digital loading problem became apparent, the researcher set up another experiment to extensively test the new sensor. This experiment supplied the emitter resistor and sensor capacitor to a breadboard for easy altering of parameters. Testing began on 2-10-95 and produced immediate interesting data. The newer sensor lacked the distance resolution of the earlier version.

Tae Choi tested his parameters which were successful on his class robot the previous semester and found that the range was half that expected. The “blind spot” immediately in front of the emitter was still the same, (roughly two inches) but the range dropped from over thirty six inches to about fourteen inches. By varying the focal distance, a term used here to describe the distance at which the sensor first reaches saturation, many different ranges were observed. Range seemed to increase by about six inches per one inch of focal distance compromise. Tests continued and consistently demonstrated the new sensors’ inferiority to the older model.

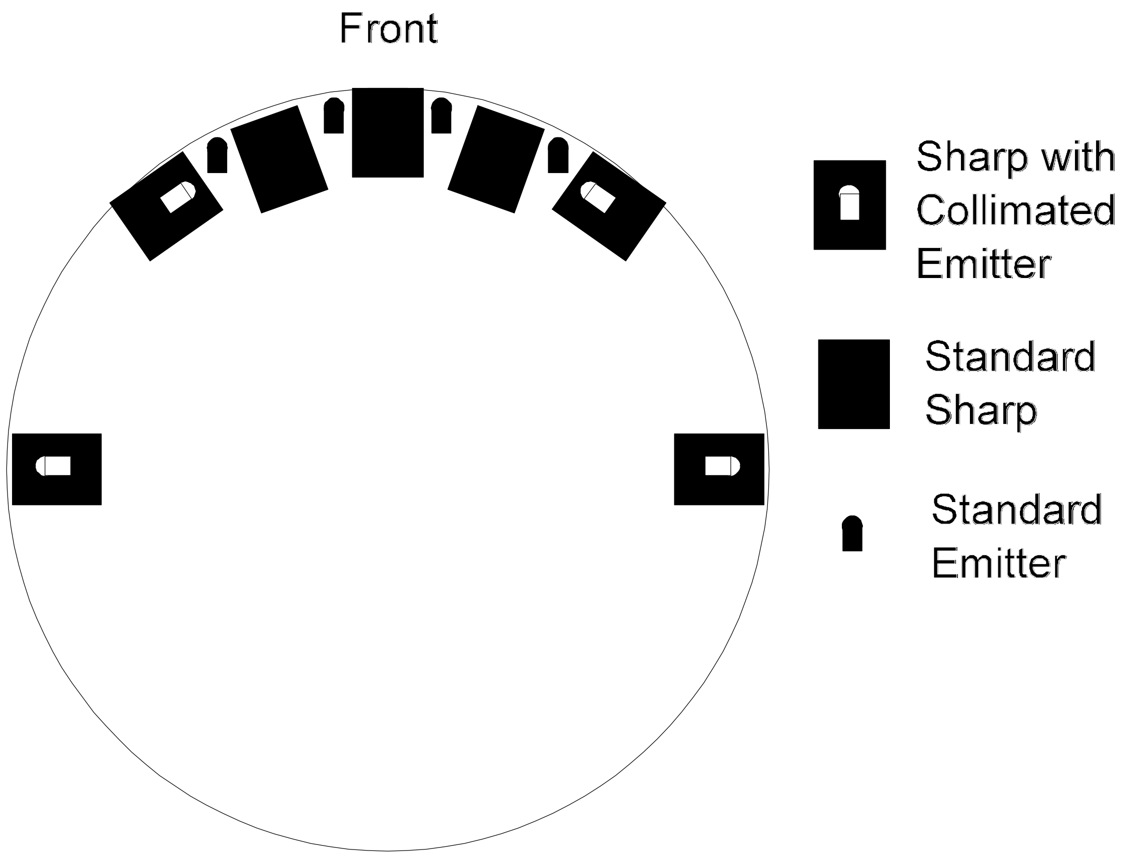
Additional experiments were initiated to examine the feasibility of providing software control over the IR emission intensity. Since a standard D/A chip cannot handle the necessary current levels, discussion focused on creating an effective D/A with higher loading capability. Current levels up to about 35mA had to be available. Tai Choi and Scott Jantz collaborated on a novel method of producing such a device. The researcher was able to test and verify the performance of the design as well as extend it to allow the first quantization of multi-level IR distance measurement. The schematic of the design devised from the original Choi / Jantz design follows:



The IR LED Array can be a single LED is a series string if LED’s. Since the voltage drop across the LED’s is over 1V however, the resistances used must also decrease and no more than three LED’s should ever be used in series. The initial current limiter Rx can be a short if desired. The researcher in general prefers to place resistances on all digital outputs to avoid a short-circuit situation. Since all resistances can be socketed using high profile sockets, the initial resistance Rx can be shorted after the assembly of the secondary resistance branches.

The range of the Sharp sensors can now be modeled as the complete range from the saturation of the low current setting to the first detection of the high current setting. Experimentally, the low current setting has a full deflection range covering about 12 inches with a saturation length within the perimeter of the agent. The high current setting can detect reflections at distances greater than 30 inches. Adaptive algorithms can request a data value by selecting the high current setting, and dropping to a lower level if saturation occurs. With the modification of the reference voltages of the A/D converter, six bits of precision are available over the eight ranges.

The issue of sensor placement was not thoroughly research. Instead, the Tae Choi model was combined the Scott Jantz zonal distribution method to develop the following sensor suite. On the following page, the black boxes are the Sharp sensors whereas the smaller black and white objects are noncollimated and collimated IR emitters respectively.

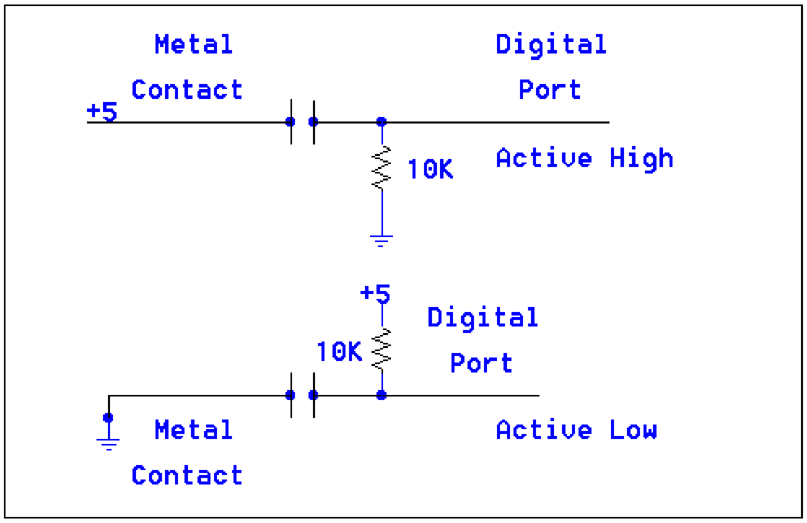


Since the agent has the capability to pan its head one quarter revolution to each side, the two lateral sensors become front sensors when the head is completely turned. The possibility of adding uncollimated emitters to these sensors is being explored, since any objects in the agents way must be picked up by these sensors alone when in pan mode.

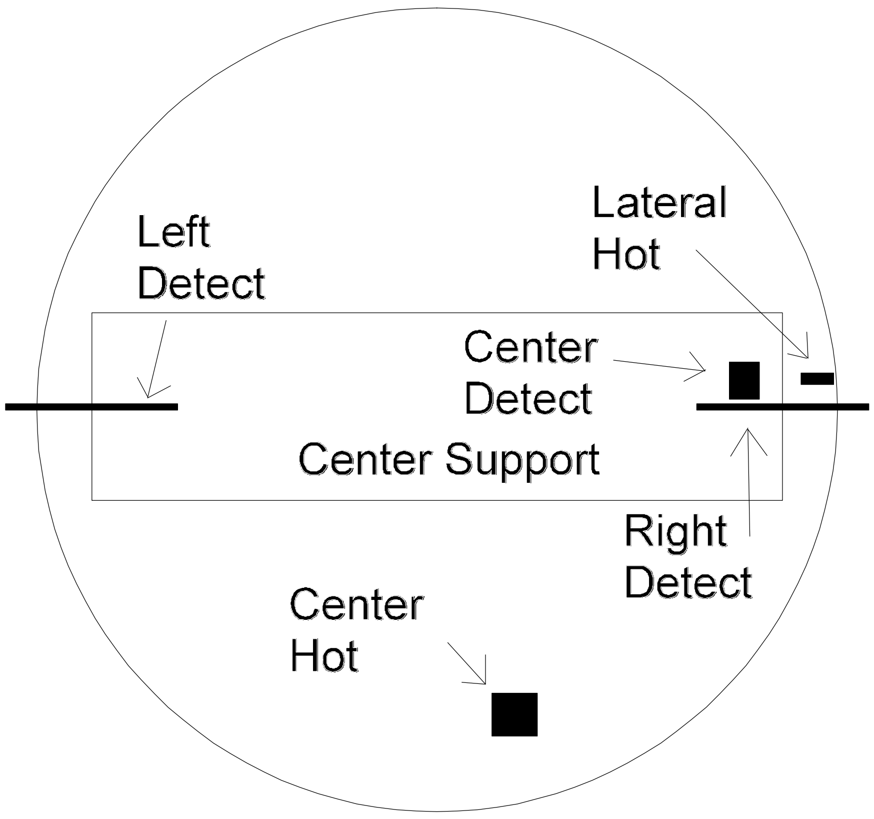
Since a vast majority of sensing capability is concentrated towards the front of the head, constant use of the panning feature must be used in order to fully analyze an environment. It is expected that the panning feature will allow for high speed wall following and "passing analysis" of objects.

Limit Switches

One major obstacle to the design of an effecting panning head was the position sensing of the head. The servo-pot used to mechanically turn the head does not have a standard center-tap potentiometer and so did not allow for a single analog reading of position. About twenty combinations of the seven taps available were fully tested with very limited success. In each case, the nonlinear nature of the device gave higher resolution on one turn side. The slip of the device also did not allow for precise positioning. The researcher determined that a digital means of position detection should be implemented. The standard limit switch configuration is shown. Because of the extended ground-plane present of the agents main board, the active high solution was implemented. A diagram of the specific implementation chosen follows on the next page.



Pan Control Limit Switches



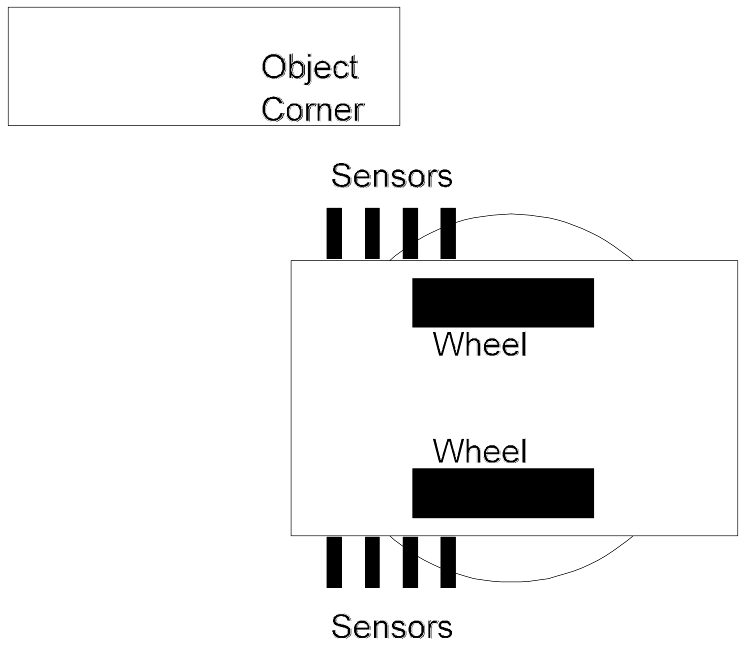
The hot contracts are tied directly to Vcc while each detect line is tied to an HC11 input port with a resistance to ground. Left and right detect pods are made using metal pins and the hot contact using tempered "blue steel" plates. The rigidity of the plates serves two purposes. First, the plates are strong enough to absorb the inertia of the head when hitting a limit contact, thus allowing for more precise positioning. Second, the nature of the material allows the material to be bent along its grain into shapes which are held indefinitely. This becomes critical when construction the center hot tap. The switch must give a clear signal but not interfere with the free rotation from either extreme to the other. By bending the steel into a miniature parabola and using a third pin head to "scratch" the surface, virtually no interference occurs during a full sweep.

CDS Cells

CDS cells are basically variable resistors controlled by ambient light. For application to the agent, the cells will be colimated and used to give a digital response. There are two reasons for this decision. The range or responses given depend greatly on the ambient room light. Thus, any algorithm using the cell as an analog input must account for a variety of possible environments. Having the sensor trigger a threshold device to give a digital signal is possible if a high intensity light source parallel to the colimation chamber is reflected off a close object. In short, the objective of this researcher is to create a functional equivalent of the digital Sharp sensor at a fraction of the price.

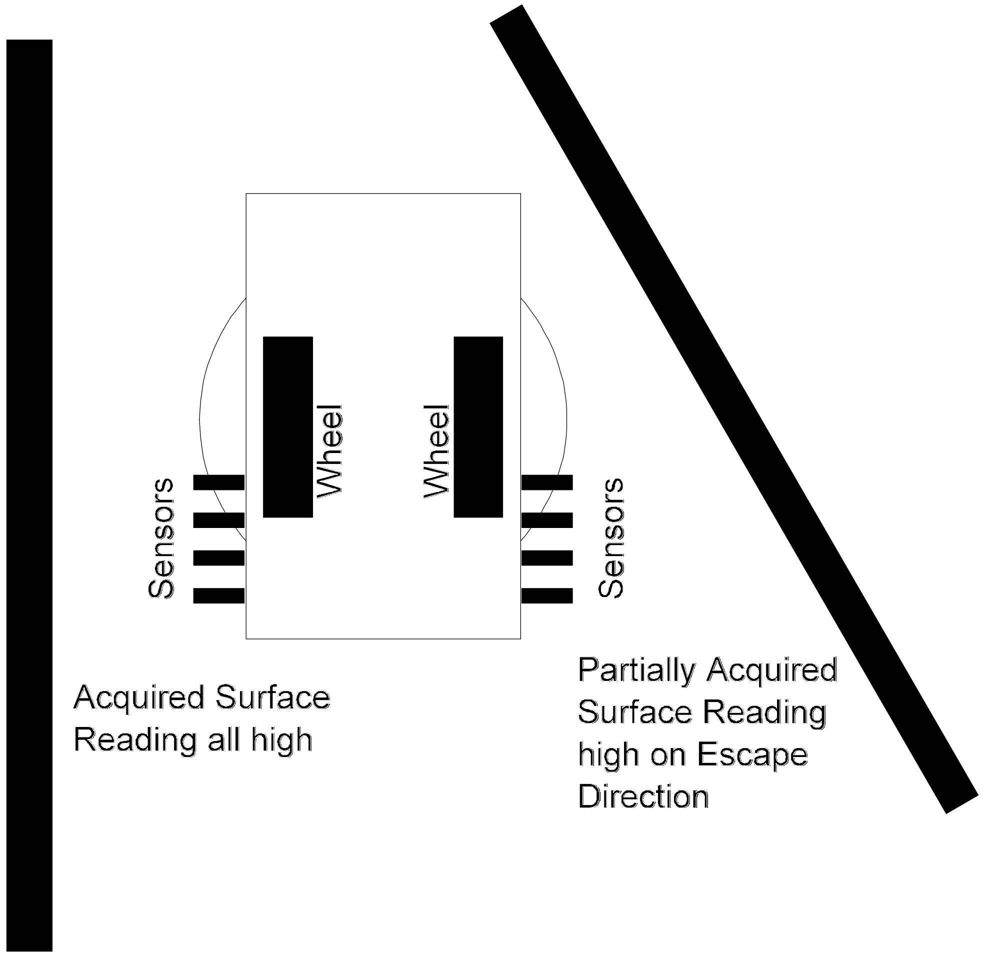
The immediate applications for this agent involve corner detection. The panning ability of the sensor head should make wall following very simple. However, proper alignment after making an angled turn is critical to reacquiring a surface. This can be done by analog sensors, but could also be done far more simply using digital proximity sensors. A proposed diagram follows.

Corner Detection using CDS Cells



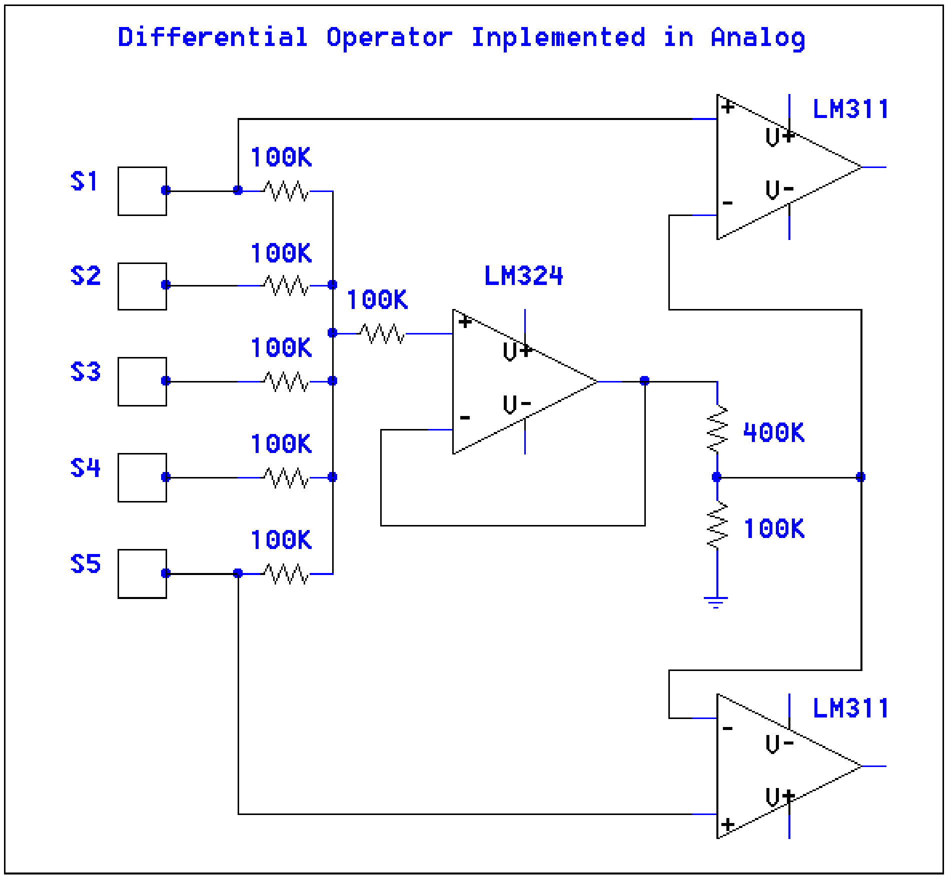
Because the agent's wheels are located on the central axis, proper alignment can be done using an under-turn-over-compensate procedure. The agent would first turn somewhat less than 90 degrees and then adjust inward until the surface is reacquired. With proper placement, it may only be necessary to perfect the turning angle as the sensors could determine the proper extension beyond the surface.

A further application could be the corner exit solution problem. When an agent enters a corner with the exit vector greater than 90 degrees from the entrance vector, instability and "jamming" can occur. To avoid this, the following procedure is suggested:


Wheel
Wheel
Sensors
Sensors
Acquired Surface Reading all high
Partially Acquired Surface Reading high on Escape Direction

A partially acquired surface on the opposing side of an acquired surface would indicate that the above condition is satisfied. The agent could then back up, turn 90 degrees to the side, and attempt to acquire the new surface at an angle of 90 degrees less than the previous angle. Again, this is feasible only when the agent can turn very tightly.

After two weeks of testing, it was concluded that the analog equivalent shown below is a superior method since minimal hardware control is required. Because the analog comparators are essentially op-amps in open loop configuration, this circuit can be realized using only a single 14-pin IC making it ideal for small boards.



Algorithms and Behaviors

In order to accommodate the contouring behavior sought, a novel control architecture was designed. Specifically, two experimental methods were implemented to assist the anemic HC11 in handle real-time contouring. Imposed linearity corrects for nonlinear sensors which can vary supply current. Calculations are greatly simplified of the displacement to analog reading conversion can be linearized. Overall control uses a system of partial solutions to eliminate instability.

Imposing linearity on a multilevel system required that all levels have similar emission relationships. For infrared emitters, we assume and find experimentally, that a specific reading and displacement x can be related to any other reading of the same emission level by an equation of the form given in equ. 1.

$$A_x = A_{offset} - A_0 e^{-cx} \qquad \text{equ. 1}$$

Other levels will differ only in the initial offset constant and the decay constant c. The mathematics of converting to equal decay constants will not be discussed as this is not necessary for this application. The important factor is that some portion of this curve is almost linear. This section is found where the derivative approaches unity, or

$$\left|cA_0 e^{-cx}\right| = 1 \qquad \text{equ. 2}$$

The solution reduces to an x such that

$$x = \frac{\ln\left(\frac{1}{cA_0}\right)}{-c} = \frac{\ln(cA_0)}{c} \qquad \text{equ. 3}$$

For the R2 agent, there are eight seven active levels of emission, so seven such equations must be found and solved. Since on overall resolution of eight bits is required, each linear section required 256 divided by 7 or 36 values. The section thus consists of a central value with a deviation of 17 in each direction. The level offset can then be computed by realizing that the level must be shifted to the 36*(n-1) position along the overall resolution curve. For level 4, this would mean than a zero value would be assigned

36*3 as a base and the full deflection value of the central value +18 would be assigned (36*4)-1.

The result will be a piecewise linear system of seven emission levels created from independent non-linear curves. The control software need only determine the correct level to use and then apply the appropriate offset. This is done by starting at the highest level and detecting the overshoot. The exponential nature or the curves cause the higher side of the linear sections to deviate more and should thus be used. If too high a reading is returned, the emitter controllers drop the level by one and rescan. This is repeated until a satisfactory level is achieved.

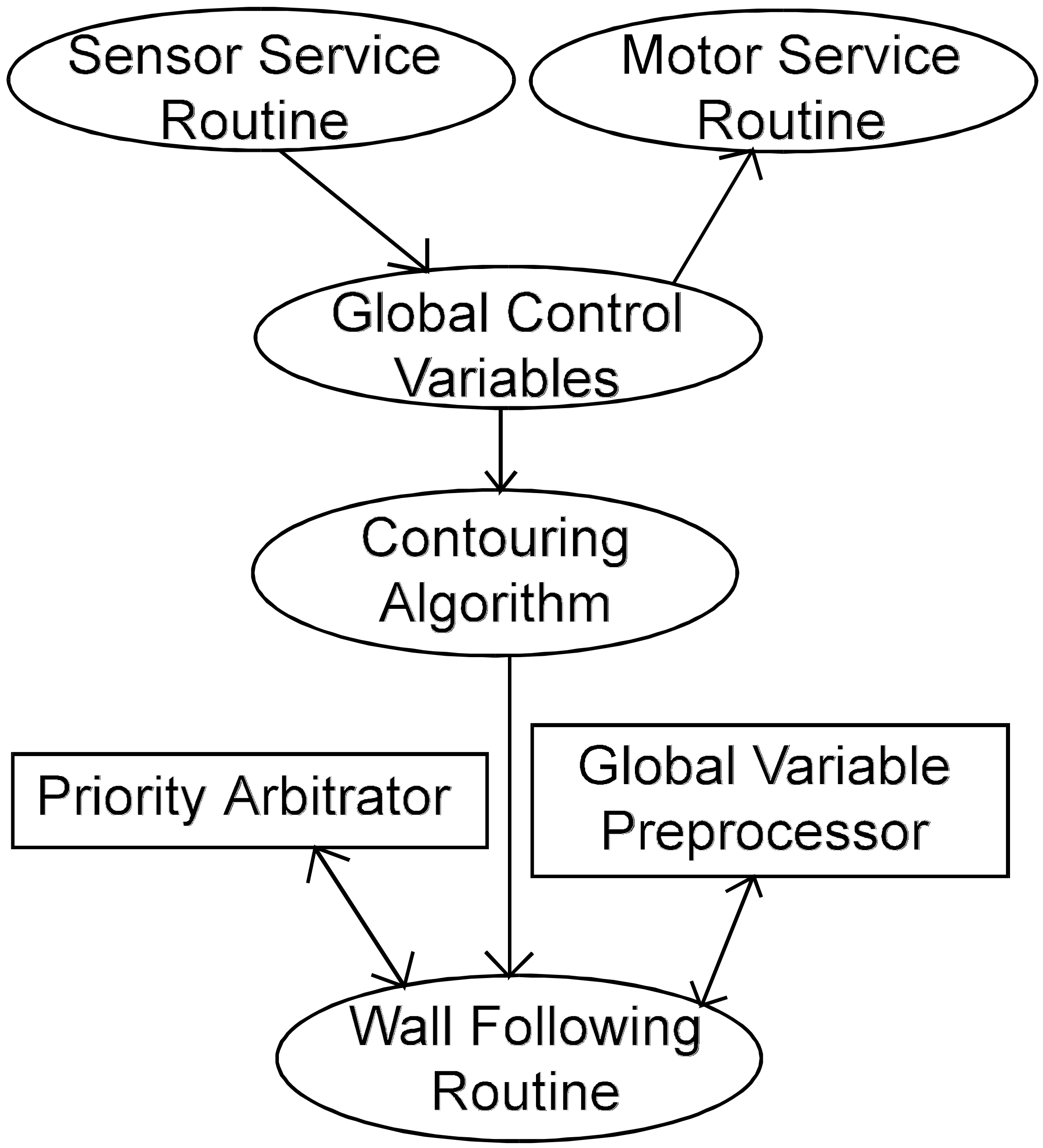
Partial solutions

The system of partial solutions was developed to add stability to behaviors prone to instability problems: wall following and contouring. Partial solutions is similar to a bottom-up program style but with important differences. The algorithm to control an overall function such as contouring is referred to as a goal. To achieve a solution to this goal, several partial solutions must be achieved. These include wall following, corner detection, convex turning, concave turning, barrier detection, and alignment.

A typical approach to contouring using two lateral and one forward sensor might be structured as follows. The agent detects an object and aligns to it. Forward linear motion is started and compensated with the distance is to great or to small. If an object is detected immediately in front of the agent, the agent turns in place until there is no object and then continues. One immediately notices several problems. First, what happens when the agent attempts to corner? The turning will certainly leave the agent too far away from the wall and the correction inward will engage. Once started, this behavior does not stop until the wall is found and the agent is an acceptable distance away. Unfortunately, the inward turn will cause the sensors detecting the wall to be brought tangent to the surface and the agent will attempt to run through the surface until a wall is reacquired.

The equivalent partial solutions approach is much different. Using the same sensor arrangement, the agent makes steady corrections to the wall via the partial solution of correcting inward when too far away and outward when to close. By using the differential measurement, angle to the surface is also recorded. The instability of the aforementioned situation is avoided by switching to another partial solution: angle correction. If the agent is to far away, it is imperative that the agent be running tangent to the surface for the distance measurement to be meaningful. This, the first partial solution is to align with the surface. The partial solution of adjusting to the distance can now be done. As soon as a small correction is made in distance, the angle partial solution takes over and again forces tangential alignment. Cornering is slightly more complex as the possibility of the agent loosing the surface does exist. Since two partial solutions are insufficient to corner sharply, a third partial solution is incorporated. As soon as the distance to the surface jumps between successive readings, a corner is confirmed and the distance correction partial solution is stopped. The cornering partial solution is to rotate towards the corner with a small sideways displacement. As long as the distance to surface is high, the second face of the surface has not been found and the partial solution is to rotate until a minimal distance is found. The head is then pointing directly at the corner of the object and the second partial solution of achieving a tangential alignment starts. As soon as the corner is list, the partial solution is again to reacquire the corner. When a satisfactory distance is achieved while the angle to surface is perpendicular, the partial solution of cornering has been achieved and the distance correcting partial solution for wall-following is employed again.

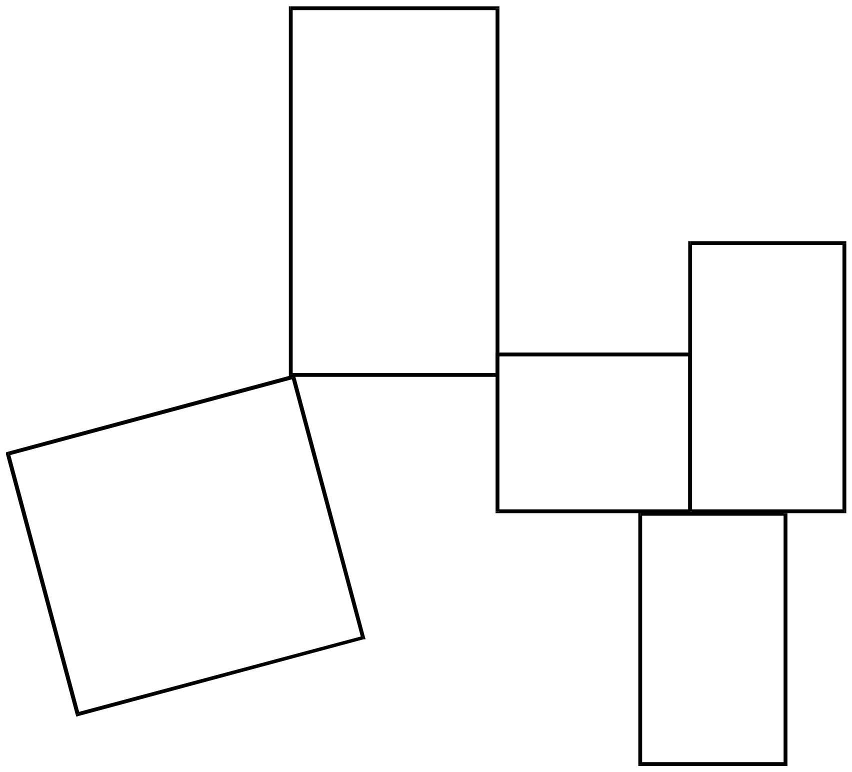
Because the partial solutions used to achieve a goal are controlled by current and past values of functions of global sensor value, the priority arbitration must be separate from the global variable processing. A wall following routine will communicate with the priority arbitrator and global variable preprocessor as a group of partial solutions to the contouring algorithm. Sensor and motor service routines are completely separate from the behaviors. The diagram on the following page illustrates this architecture.



Contouring is thus treated as a simple problem with one solution. Other more complex behaviors may require two solutions. The important concept is that all partial solutions can be arbitrarily used or ignored depending on previous results. In fact, one can almost picture a type of learning in which several partial solutions are attempted and the one that do not advance the agent towards the general solution are deleted or randomly altered. The applications for genetic algorithms are also considerable because the best partial solution can be represented as a systematic combination of several partial solutions.

Each agent has access to all partial solutions, so one might picture two agents which explored different regions of a structure and can transmit some code identifying the structure from another area of the structure and the sequence of partial solutions used.

Experimental Layout



To test the contouring algorithm, for environment shown was constructed. All boxes used were cardboard and had approximately equal reflectivities. The agent was released in an approach vector which would contact a side of the surface. Using the algorithms described, the agent then aligned itself with the surface, turned and rotated its head towards the surface, aligned itself again, and started contouring. The surface used tests all aspect of the algorithm as concave and convex corners are used throughout. The nonperpendicular edges test the partial solutions technique by proving that arbitration between distancing and angular alignment can contour any angled surface.

Conclusions

The demonstrations shown during the last week of class up to and including the required demonstration date verified the ability of the R2 agent to contour a surface of considerable complexity. After running for thirty minutes, it was determined that the algorithm is nearly fault-free. Manual corrections were necessary only about once per three circumnavigations. It is pertinent to point out that these manual corrections may not have been necessary. On certain occasions, the agent clipped a wall and was knocked off-course. The arbitration successfully corrected for this and it is possible that eventually, the partial solutions system would correct for even an extreme instability such as a frontal collision. The manual intervention was necessary to prevent damage to the agent since the prototype agent is not robust and might easily knock loose sensitive hardware.

As with a simulation, the algorithm structure developed works best when the agent is small compared to the surface covered. In this situation, there are many opportunities for correction after completing a turn. In the test environment, this was not the case. The agent sometimes had not fully compensated for a previous turn before another corner had to be handled. For the R2 agent, a minimal flat surface length of twice the agent length is preferable although constant arbitration will correct for faster changing surfaces. If speed is sacrificed, even tighter control is possible.

The author feels that the partial solutions architecture would greatly benefit from a better interfacing with a linearly enforced displacement sensor system. For instance, if the agent could realize that it is exactly two inches too far from a surface as opposed to just too far, then much quicker correction is possible. Also, with linearity, the differential reading could be translated into a crude angle measurement. Partial solutions also works best when many solutions are available. If more sensors could be added for the critical portions of contouring such as cornering, efficiency would certainly be higher. The success of the partial solutions architecture was not expected. The system actually rose from a simpler arbitration system in which different subroutines were used for a predetermined time and then made dormant. The active arbitration of these subroutines lead to the development of the partial solutions architecture.

One note of importance is that when debugging behaviors of any kind, the agent must be running in real time. It is often difficult to assess what the agent is or is not doing and what particular behaviors, if any, are working. The author strongly encourages the use of the memory mapped single byte display. While debugging, the display can flash the current state variables, the subroutines in action, or the number of times a certain routine has executed. This is vital when an arbitration is used. Since several solutions are being implemented at once, any failure cannot be directly attributed to one specific routine. However, by displaying some code unique to each routine, the programmer can see immediately what was executed when the algorithm became unstable. Were it not for the current requirements, the author would have included several such displays or an LCD.

References

[1] Joseph L. Jones and Anita M. Flynn. Mobile Robots: Inspiration to Implementation
K. Peters 1993

[2] Pattie Maes. Designing Autonomous Agents
MIT / Elsevier 1990

[3] Fred Martin. The 6.270 Robot Builder’s Guide
Epistomology and Learning Group, MIT 1992

[4] M68HC11 Reference Manual
Motorola 1991

[5] MC68HC11E9 Technical Data
Motorola 1991

[6] High-Speed CMOS Data
Motorola 1993

Appendix

Program Code

Global Variables

```
int     base_line=0x40;
int     wall_follow_wait=50;
int     flood_dif_stable=10;
int     side_hit=0x50;
int     ok_to_follow=1;
int     wall_follow_speed_high=100;
int     wall_follow_speed_med1=80;
int     wall_follow_speed_med2=80;
int     wall_follow_speed_low=30;
int     flood_bias=4;
int     sat=0x75;
int     wall=0x92;
int     notwall=0x8A;
int     encoder_bias=35;
int     msnum=5;
int     s0=0;
int     s1=0;
int     s2=0;
int     s3=0;
int     s4=0;
int     s5=0;
int     s6=0;
int     right_hard=165;
int     left_hard=170;
int     desired_right=0;
int     desired_left=0;
int     current_right_speed=0;
int     current_left_speed=0;
int     current_4000=0;
int     current_5000=255;
int     current_6000=255;
int     current_7000=0x40;
int     right_side=0;
int     front_hit=0x6e;
int     flood_dif=0;
int     col_dif=0;
int     ok_to_run=1;
int     crit=0x65;
```

Behaviors

```
void main()
{
        start();
        aaa();
}

void start()
{
        start_process(ms());
        start_process(ss());
}

void aaa()
{
        while (s0<crit) {}
        head_left();
        head_center();
        straight();
        head_left();
        slam_right();
        halt();
        wait(400);
        align_to_wall();
        desired_left=wall_follow_speed_low;
        desired_right=wall_follow_speed_low;
        while (1) {keep_on_wall();};
}

void keep_on_wall()
{
        follow_wall();
        align_to_wall();
}


void align_to_wall()
{
        if (flood_dif>0) {while (flood_dif>0)
                        { desired_left=10;
                          desired_right=-10;
                        }
                    }
        else          {while (flood_dif<0)
                        { desired_right=10;
                          desired_left=-10;
                        }
                    }
        halt();
        wait(200);
}
```

```
void follow_wall()
{
        while (ok_to_follow)
        {
              if (flood_dif<(-flood_dif_stable))
                { desired_right=wall_follow_speed_high;
                  desired_left=0;
                  wait(wall_follow_wait);
                  desired_right=wall_follow_speed_low;
                  desired_left=wall_follow_speed_low;
                  set_display(1);
                }

              else if (flood_dif>flood_dif_stable)
                { desired_left=wall_follow_speed_med1;
                  desired_right=0;
                  wait(wall_follow_wait);
                  desired_left=wall_follow_speed_low;
                  desired_right=wall_follow_speed_low;
                  set_display(2);
                }
              else if (s0<notwall)
                { desired_right=wall_follow_speed_med1;
                  wait(wall_follow_wait);
                  desired_right=wall_follow_speed_low;
                  set_display(3);
                }
              else if (s0>wall)
                { desired_left=wall_follow_speed_med1;
                  wait(wall_follow_wait);
                  desired_left=wall_follow_speed_low;
                  set_display(4);
                }
              else if (flood_dif<0)
                { desired_right=wall_follow_speed_med2;
                  wait(wall_follow_wait);
                  desired_right=wall_follow_speed_low;
                  set_display(5);
                }
              else if (flood_dif>0)
                { desired_left=wall_follow_speed_med2;
                  wait(wall_follow_wait);
                  desired_left=wall_follow_speed_low;
                  set_display(6);
                }
              else {set_display(7);};
        }
        slam_right();
        halt();
        wait(200);
        align_to_wall();
}
```

Diagnostic Algorithms

```
void check()
{       while (1)
        {
        set_display(s0);
        wait(1000);
        set_display(flood_dif);
        wait(1000);
        }
}

int check_left()
{
        select_encoder(0);
        wait(200);
        return (peek(0x1027)-encoder_bias);
}

int check_right()
{
        select_encoder(1);
        wait(200);
        return peek(0x1027);
}
```

Straight Line Motion Algorithm

```
void straight()
{
        int x;
        if (ok_to_run) {desired_right=70;
                        desired_left=70; }
        while (ok_to_run)
        {
        x=check_left()-check_right();
        set_display(x);
        if (x>0)
                {desired_right=50;}
        else if (x<0) {desired_right=90;}
        else desired_right=70;
        }
        halt();
        wait(200);
}
```

Service Routines

```
void ss()
{
        while (1)
        {
        floods(1,1);
        collimateds(1,1);
        laterals(1);
        set_flood_level(3,3);
        wait(5);
        if (s0>sat) {set_flood_level(1,1);
                     wait(5);
                     s0=analog(0)+20;
                     s1=analog(1)+20;
                     s2=analog(2)+20;
                    }
        else         {
                     wait(5);
                     s0=analog(0);
                     s1=analog(1);
                     s2=analog(2);
                    }
        flood_dif=(s1-s2+flood_bias);
        ok_to_run=((s0<crit)&(s1<crit)&(s2<crit));
        set_flood_level(0,0);
        set_collimated_level(7,7);
        wait(5);
        s3=analog(3);
        s4=analog(4);
        col_dif=(s3-s4);
        set_collimated_level(0,0);
        set_lateral_level(5);
        wait(5);
        s5=analog(5);
        s6=analog(6);
        set_lateral_level(0);
        ok_to_follow=((s0>base_line)&(s5<side_hit));
        }
}

void ms()
{       while (1)
        {
        {       if (desired_right>current_right_speed)
                        {motor(0,(current_right_speed+msnum));
                        current_right_speed=current_right_speed+msnum;}
                else
                {
                if (desired_right<current_right_speed)
                        {motor(0,(current_right_speed-msnum));
                        current_right_speed=current_right_speed-msnum;}
                else {};
                };
        };
        {       if (desired_left>current_left_speed)
                        {motor(1,(current_left_speed+msnum));
                        current_left_speed=current_left_speed+msnum;}
                        else
                {
                if (desired_left<current_left_speed)
                        {motor(1,(current_left_speed-msnum));
                        current_left_speed=current_left_speed-msnum;}
                else {};
                };
        };
        };
}

void wait(int m_seconds)
{
        long stop_time;
        stop_time=mseconds()+(long)m_seconds;
        while(stop_time>mseconds()) defer();
}

void halt()
{        desired_right=0;
        desired_left=0;
}
```

Turning Routines

```
void slam_right()
{
        desired_right=0;
        select_encoder(1);
        desired_left=80;
        while (peek(0x1027)<right_hard) {defer();};
        halt();
}

void slam_left()
{
        desired_left=0;
        select_encoder(0);
        desired_right=80;
        while (peek(0x1027)<left_hard) {defer();};
        halt();

}
```

Control Routines

```
void select_encoder(int var)
{       poke(0x1026,0x50);
        current_7000=(current_7000&0x7f)+(128*var);
        poke(0x7000,current_7000);
        poke(0x1027,0);
}

void head_motor(int var)
{
        current_7000=((current_7000&0x9f)|var);
        poke(0x7000,current_7000);
}

void head_left_main(int var)
{
        head_motor(0x20);
        while (!(peek(0x1000)&var)) {defer();};
        head_motor(0x40);
        right_side=0;
}

void head_right_main(int var)
{
        head_motor(0x00);
        while (!(peek(0x1000)&var)) {defer();};
        head_motor(0x40);
        right_side=1;
}

void head_right()
{       head_right_main(1);
}

void head_left()
{       head_left_main(4);
}

void head_center()
{
        if (right_side)
        {
                head_left_main(2);
                while (peek(0x1000)&0x02) {defer();};
                wait(500);
                head_right_main(2);
        }
        else
        {
                head_right_main(2);
        };
}
```

```
void set_flood_level(int right,int left)
{
	int x;
	x=(current_6000&0x1f)+((7-right)*32);
	current_6000=x;
	poke(0x6000,current_6000);
	x=(current_5000&0xe3)+((7-left)*4);
	current_5000=x;
	poke(0x5000,current_5000);
}

void set_collimated_level(int right,int left)
{
	int x;
	x=(current_6000&0xe3)+((7-right)*4);
	current_6000=x;
	poke(0x6000,current_6000);
	x=(current_5000&0x1f)+((7-left)*32);
	current_5000=x;
	poke(0x5000,current_5000);
}

void set_lateral_level(int level)
{
	int x;
	x=(current_5000&0xfc)+(((16-level)&0x0c)/4);
	current_5000=x;
	poke(0x5000,current_5000);
	x=(current_6000&0xfc)+((16-level)&0x03);
	current_6000=x;
	poke(0x6000,current_6000);
}

void set_leds(int value)
{
	int x;
	x=(current_4000&0xf8)+value;
	current_4000=x;
	poke(0x4000,current_4000);
}

void set_display(int value)
{
	poke(0x3000,value);
}

void collimateds(int right,int left)
{
	int x;
	x=(current_4000&0x3f)+(128*left)+(64*right);
	current_4000=x;
	poke(0x4000,current_4000);
}
```

```
void floods(int right,int left)
{
        int x;
        x=(current_4000&0xcf)+(16*left)+(32*right);
        current_4000=x;
        poke(0x4000,current_4000);
}

void laterals(int value)
{
        int x;
        x=(current_4000&0xf7)+(value*8);
        current_4000=x;
        poke(0x4000,current_4000);
}
```