Odor Tracking Group
Critical Design Review

CPSC 483 – Spring 2003
Gutierrez-Osuna

Jason Hamor
Greg Albee
Ninh Dang
Simon Saugier

Table Of Contents

IX. I. Executive Summary

We have made significant progress since our proposal stage. We have made developments in every area. After receiving criticism on our initial design, we each chose a topic to focus on and began individual research. The topics we chose were: finding a dispersion model, interfacing external code with LabVIEW, researching and creating an odor-tracking algorithm, and figuring out how the existing dilution system works and how to model it.

We decided to use the Gaussian model for our dispersion model. This model simulates the dispersion of a gas through an atmosphere after enough time has elapsed for the gas to have sufficiently diffused. Although this model is not time-variant, it provides a sufficient model of an odor plume for our purposes.

We have figured out how to interface C/C++ with LabVIEW, and thus we will code our project in C/C++. We were relieved to discover that the version of LabVIEW that is currently available to us is able to create external code modules in C/C++.

We have researched five different types of odor-tracking algorithms, and have decided to develop a hybrid model for the time being, that is adapted to use the best parts of each algorithm, and also to meet our constraints (no wind sensor, etc.) We may modify the model in the future, based on its performance in testing.

We have investigated the existing odor-dilution system, and we understand how it works. We have also obtained an equation that can be used to estimate the response of the system to a chemical concentration.

Given these advances, we are poised to move forward on our project and begin coding and integration.

II. Dispersion Model

To test our odor-tracking algorithm, we will need an odor/gas dispersion model. The model will have to be capable of representing the dispersion of odor from one or multiple odor sources. More specifically, we will need to use a model that will be able to provide us with a certain chemical concentration at any three-dimensional Cartesian coordinate (X, Y, Z). Given these criteria, we essentially have three models to choose from - the Eulerian model, the Lagrangian model, and the Gaussian model.

The major advantage of the Eulerian and Lagrangian models over the Gaussian model is that the first two models consider the change of the odor plume with respect to time. The concentration at a certain point in space is dependent not only on the specific location, but also the amount of time that has elapsed since the odor source began emitting. While this representation of time variant dispersion is very realistic, the consequence of this realism is that these two models are incredibly complex. For this reason, we ended up choosing the Gaussian model. However, it is worth considering all three models in this paper to properly explain and defend our assumptions, considerations, and concerns.

The Eulerian model is based on the concept of tracking a certain "puff" of gas as it travels through the air (Fig. 1). This model is based on collision theory; the puff of gas

collides with the atmosphere around it, resulting in a different shape, size, and diffusion. The equation is given below (Eq. 1).

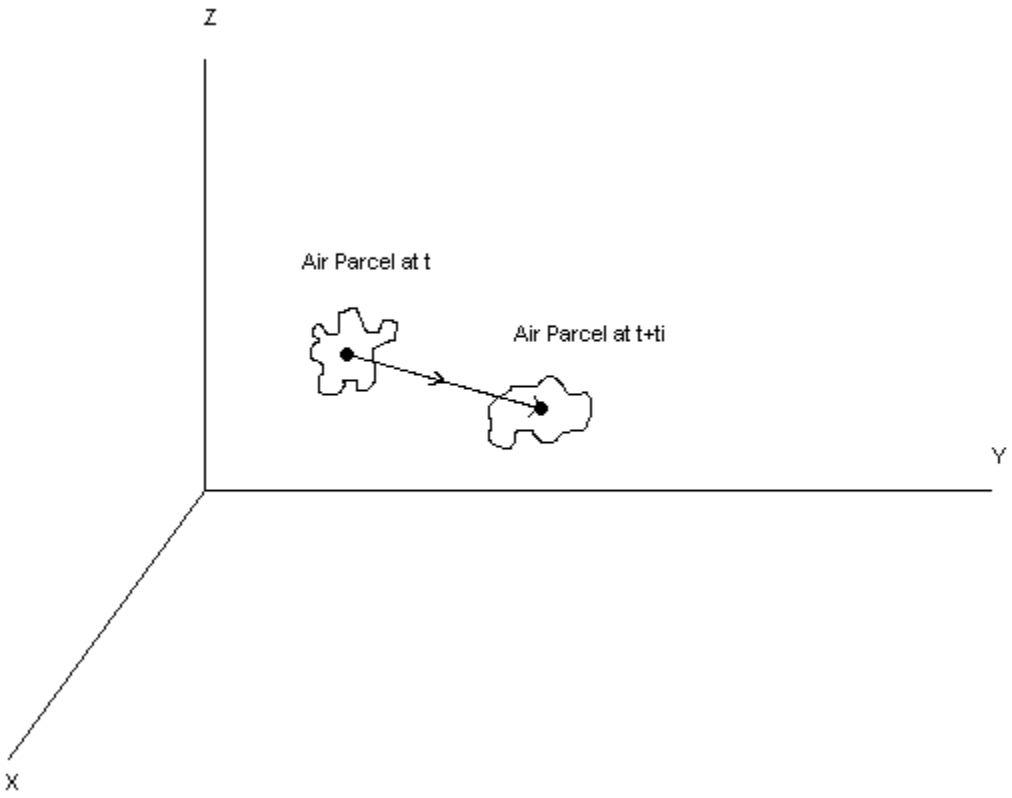


Fig. 1

$$\bar{c}(x,z) = \frac{2Qqh_0^a}{h^{a+1}u_0}\left(1+\frac{zR}{h}\right)^p \frac{J_{(1}(i)R^q J_{y1}(ti(z/h)^q}{J^2_{(1}(ti)} \bullet e^{-\frac{(ti)^2 q^2 K_{z0} x}{h h_0^r u_0}}$$

$$c(x,y,z) = \bar{c}(x,z)\frac{1}{\sqrt{2\pi}\sigma_y} e^{-\frac{y^2}{2\sigma_y^2}}$$

*Eq.*1

While the accuracy and intricacy of this model make it an appealing choice, the complexity of the corresponding equation puts it beyond the present scope of this project. Perhaps future researchers will feel this equation is worth investigating, but the amount of research and time that it would take us to investigate and understand this equation make it a non-ideal choice, given our time constraints and focus.

The Lagrangian model is similar to the Eulerian model. In fact, given the same initial conditions, both equations would produce the same results. The Lagrangian model, however, places the origin of the coordinate system at the center of the "puff" of gas being considered. As the puff moves, the origin moves with it, so in fact the world is represented as moving, while the puff holds still. The representative equation is given below (Eq. 2).

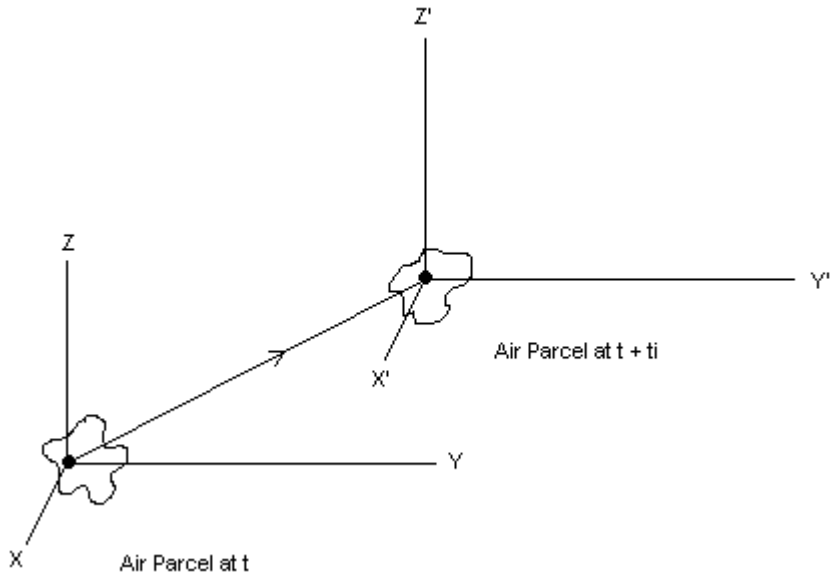


Fig. 2

$$c(r,t) = \frac{A(r)}{l^3}\sum_{i=1}^{n} m_i W(r_i - r, l)$$

$$A(r) = \frac{l^3}{\int_D W(r' - r, l)\,dr'}$$

$$W(r_i - r, l) = \frac{1}{(2\pi)^{3/2}} e^{-\frac{1}{2}\frac{|r_i - r|^2}{l^2}}$$

*Eq.*2

This model seems much more accessible and well defined that the Eulerian model. Indeed, the only unknown condition is "*l*", which is based on the gas being modeled. However, given that this equation calculates the concentration based upon the movement of every particle, we decided that this model was too intensive for our immediate purposes. Also, the research required to choose an appropriate "*l*" term for each gas we may need to simulate was beyond the scope of this project.

We ultimately decided to use the Gaussian model, due to its simplicity and ability to be coded in the allotted time. The major drawback of the Gaussian model, though, is that it is not time dependent. The Gaussian model provides the concentration of a gas at any point downwind, given that the gas has had adequate time to disperse into the room (Fig. 3,4) . The Gaussian model provides more of an average, and does not take into account eddies and swirls. It provides a very neat plume gradient, which is sufficient for our purposes. The equation is given below (Eq. 3).

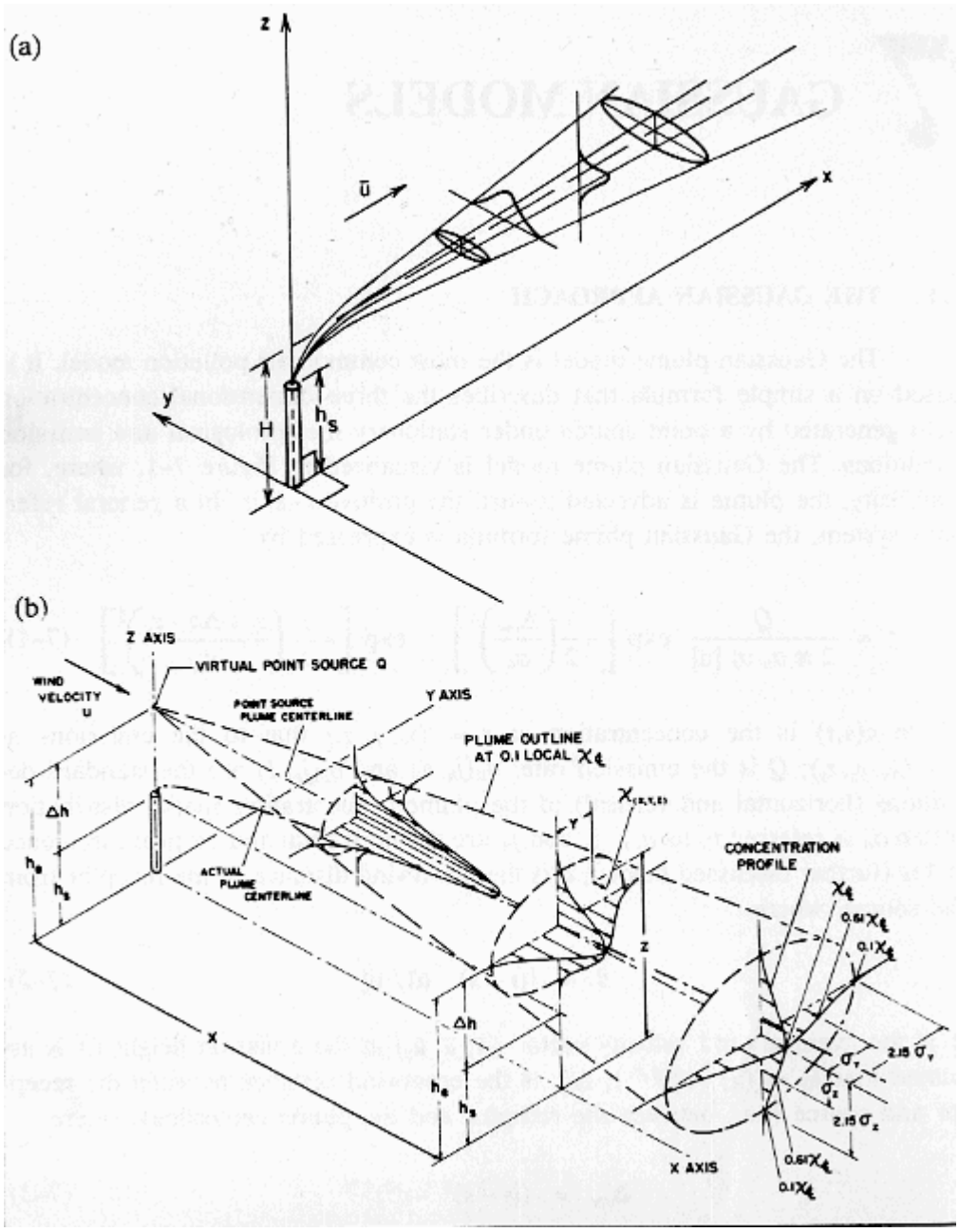

(a)
z
x
y
$\bar{u}$
H
h_s
(b)
Z AXIS
VIRTUAL POINT SOURCE Q
WIND VELOCITY u
POINT SOURCE PLUME CENTERLINE
Y AXIS
PLUME OUTLINE AT 0.1 LOCAL χ_{ℓ}
$\chi_{(x,y,z)}$
Y
CONCENTRATION PROFILE
χ_{ℓ}
$0.6\chi_{\ell}$
$0.1\chi_{\ell}$
Δh
h_e
h_s
ACTUAL PLUME CENTERLINE
Z
X
$2.15\sigma_y$
σ_y
σ_z
$2.15\sigma_z$
X AXIS

Fig. 3 [6]

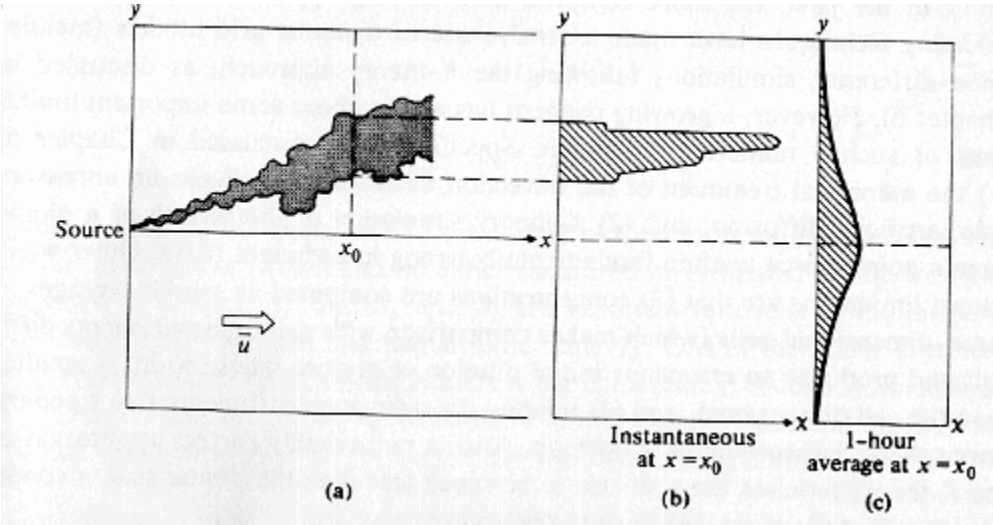


Fig. 4 [6]

$$c(x,y,z) = \frac{Q}{2\pi\sigma_y\sigma_z} e^{-\frac{1}{2}\left(\frac{y_r}{\sigma_y}\right)^2} e^{-\frac{1}{2}\left(\frac{h_e - z_r}{\sigma_z}\right)^2}$$

*Eq.*3

The premise of this model, as can be seen in the illustrations, is to take the standard deviation of the concentration at a certain cross-section of the plume. This equation is fairly straightforward except for two cryptic terms; σ_Y and σ_Z. These two terms represent the standard deviation of the plume in the cross-wind (Y) and vertical (Z) directions. According to in Schnelle [5], "It should be noted that σ_Y and σ_Z are primarily functions of the stability and downwind distance X which must be experimentally determined." However, Rod Barratt [6] points us to the website www.industrialhygiene.com/calc/model, which provides us with a reliable estimation of these terms based on the relative stability of the atmosphere. Since we will be modeling an indoor environment, we can pick a stability rating and treat it as a constant.

The equations for σ_Y and σ_Z then become:

$$\sigma_Y = A \bullet \left(\frac{Xdist}{1000} \right)^{0.894}$$

$$\sigma_Z = C \bullet \left(\frac{Xdist}{1000} \right)^{D} + F$$

Where A, C, D, and F are all constants derived from the estimation provided by industialhygiene.com. Given these approximations, we are able to easily encode this equation in any programming language that can handle floating-point exponents.

III. Integrating External Code with LabVIEW

My task thus far has been to integrate external code with LabVIEW. This is a necessary step because many of our algorithms and processes would be too difficult, or impossible to implement entirely in LabVIEW. LabVIEW is of course an integral part of our system, because the existing odor-detection unit already has a LabVIEW interface that we will need to interact with the hardware.

LabVIEW already has an excellent resource for using external C/C++ code, which is called a Code Interface Node (CIN). A CIN is a block diagram module that allows you to pass variables of most common types into it, and output data that can continue to be used in the rest of the VI. A CIN can have any number of input and output terminals, and can have terminals declared as output-only if so desired. When the CIN is called, the only arguments passed to the CIN object code are pointer references to the input variables. When the code completes running, LabVIEW passes the output variable(s) to the next block on the VI diagram.

The steps to create a new CIN are as follows. First, on the block diagram, place a CIN block, found under the "Functions\Advanced" palette. Then place the block objects that you wish to input and output from the CIN on the front panel or block diagram. Right click on the CIN block and add parameters as necessary. A terminal can also be clicked on and made output only if desired. Wire the objects together on the block diagram. The completed block diagram should look like the Figure 1, which has two numbers entered into a CIN, which the CIN multiplies and outputs to a digital display. The entire CIN operation can be turned on and off via a Boolean switch if desired.

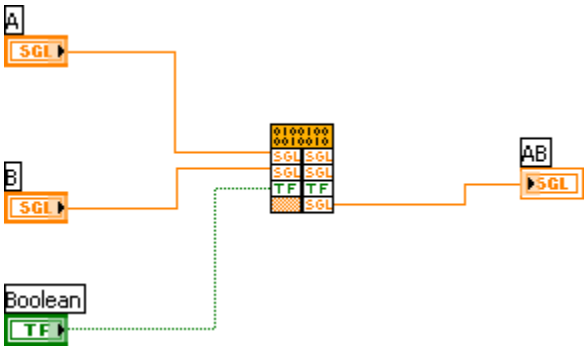


Figure 1: Example CIN module

Right click on the CIN and select "Create .c File", and put the file in a directory. LabVIEW supports many IDEs for code development, such as Metrowerks CodeWarrior, Microsoft Visual C++, and Gnu C for Unix environments. For my work, I have been using Microsoft Visual C++ 6.0. The following steps describe how to compile the .c file properly in this IDE. For other IDEs, consult the LabVIEW External Code documentation. In Visual Studio, create a new .dll project, and give it a name. To add the required files, go to Project\Add to Project\Files. Go to the Win32 subdirectory of the LabVIEW cintools directory and select the files "cin.obj, labview.lib, lvsb.lib, and

lvsbmain.def". Then go back to Project\Add to Project\Files, and add the .c file that was created from LabVIEW. The next step is to edit the project settings. For the .c to compile to the format LabVIEW needs, some very specific options need to be set. Go to Project\Settings. In the Settings For field, select All Configurations. For the next steps, do not click the OK button until all project build options are set. On the C/C++ tab, set the category to Preprocessor. Add the absolute path to the cintools directory. The path must be in MS-DOS format, or be contained in quotes for Windows format directories (spaces, long filenames, etc.). Set the category to Code Generation. Change the Use Run-time Library field to Multithreaded DLL. Set the Struct Member Alignment to l Byte. On the Custom Build tab, set the Build Commands field to "<cintools path here>\win32\lvsbutil" "$(TargetName)" –d "$(WkspDir)\$(OutDir)". This command must be on one line in the Build Commands field. For the options preceded by "$", use the Directory and Files macro buttons at the bottom to fill them in, rather than typing them by hand. In the Output Files field, enter "$(OutDir)$(TargetName).lsb". At this point it is now safe to click the OK button for the project settings. Now enter the desired code into your .c file by opening the file from the Work Space Window. Save the project, and Build it. If the code compiles correctly, then select Build\Build <your file>.dll. Verify that the .lsb file was created and placed in the \Debug directory of your project. Now back on the block diagram in LabVIEW, right click the CIN and select Load Code Resource, and select the .lsb that was created from the compile. The VI should now be able to run and test it.

My initial attempts to get a CIN to work in LabVIEW were unsuccessful. The first documentation that I found for CIN's in LabVIEW has some imprecise wording in

them. When setting build options, the documentation does not specify that path directories must be in quotes or in MS-DOS format, so Visual Studio was unable to find and use the extra files LabVIEW requires. The first documentation that I found also did not say that when setting build options, to complete all changes before pressing the OK button. I had been doing each one at a time and pressing OK after each. After a search on the National Instruments website, I found more detailed build instructions in the developer section of the website. That web page contained the instructions as I outlined above, which worked properly for me.

The next tasks I will be responsible for are to implement more tasks in LabVIEW. For instance, we will need a way to display our odor plume graph on a LabVIEW front panel. This should involve representing an equation in LabVIEW or perhaps using external C code once again, and using Lab View's graphing capabilities to display the data in a meaningful fashion. I will also need to develop a way to integrate the existing LabVIEW module that interfaces with the e-nose hardware with our LabVIEW modules.

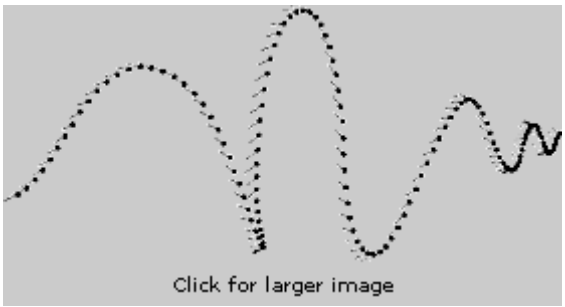
IV. Tracking Algorithm

One of the main parts of our project is to discover the "best" odor-tracking algorithm for robots. For the purposes of this project, our criteria for choosing a tracking algorithm are simplicity of implementation and accuracy of results. We have researched through many different resources and come up with our own tracking algorithm. The following are four existing tracking algorithms from different research groups. The first tracking algorithm is based on bacteria chemotaxis [1]. The second tracking algorithm is

based on the male silkworm moth to track for female moth sex-attractant pheromone [2]. The third tracking algorithm is based on gradient tracking [1]. And the fourth tracking algorithm is based on odor concentration and wind direction [3].

1- *The random walk of bacteria:* This algorithm is adapted from Holland and Mehuish [1]. Bacteria make a random movement depending on the concentration of the odor. At the start, bacteria randomly move forward a distance "d". They then stop and get the concentration at that location. They compare the current concentration with the last saved concentration. If the current concentration is greater than the last one, they will turn randomly +/-5 degree and continue move forward. If the current concentration is less than the last concentration, they will turn 180° and move forward (reversal process). They repeat the random walk process until they find the source.

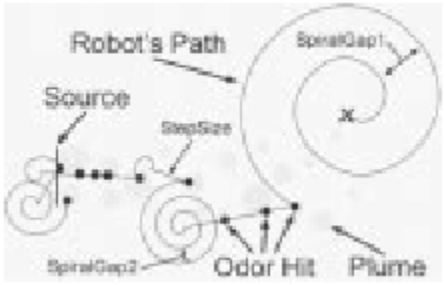
2- *Male silkworm moth track a female moth pheromone plume:* Female moths emit a sex attractant called Bombykol. When a male moth detects this substance, it starts the following movements to search for the female pheromone plume. First, it moves in the upwind direction. Second, it starts a sinusoidal zigzag movement across the horizontal of the odor plume. If it loses contact within a specific amount of time, it will loop back and start a new search. The male moth repeats the tracking strategy until it finds the female moth [2]. Please see the attached image.



Male moth behavior in odor plume

3- *Robot movement based on gradient:* "A possible way to find the sources of an odor plume is to estimate the local concentration gradient and move the robot in the direction of the gradient's increase" [1]. To experimentally test this method, a robot is supplied with two sensors on the left and the right sides. It compares the gradients of the two sensors to determine the direction of concentration's increase and makes a movement following that direction.

4- *The Spiral Surge Algorithm:* to start, the robot will move spirally (SpiralGap1) until it discovers an odor packet. Then, it will sample the wind direction and move upwind for a distance (StepSize). "If during the surge another odor packet is encountered, the robot resets the surge distance but does not resample the wind direction. After the surge distance has been reached, the robot begins a spiral casting behavior, looking for another plume hit. The casting spiral can be tighter than the plume finding spiral (SpiralGap2)" [3]. The robot will repeat the process until it finds the odor source. Please see the attached picture.

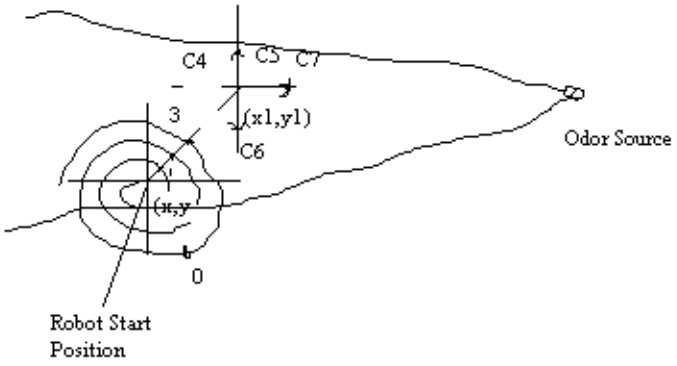


Spiral Surge Localization Algorithm

After looking at all the articles related to odor-tracking algorithms, we have come up with the following proposed tracking algorithm:

To start, the robot is in its origin location (x,y). It begins moving spirally and taking concentration samples at an interval time "t". The robot will move spirally three

layers and then stops. It will compare the concentrations at different points and different layers (C1, C2, C3). If C1<C2<C3, it will move a predefined distance ”d” and stop at a location (x1, y1) with a different concentration (C4). The robot then will move randomly in the direction of 90° different with direction of x1 a distance “d”. The stopping point now has a concentration of C5. If C5>C4, the robot will move to the direction of concentration’s increase. If C5<C4, it will move to the opposite direction a distance “2d” and stop at a point C6. If C6>C4, the robot will move to the direction of concentration’s increase. If C6<C4, it will once again change direction towards a point C7 and repeat the tracking process until it finds the source. At the location (x1,y1), if the robot cannot detect any odor after randomly rotating in different directions, it will assume it is lost and will repeat the casting behavior. Please see the attached figure.



To find out the location of (x1,y1), we will use the following formula:

$$X1 = r \cos\theta$$

$$Y1 = r \sin\theta$$

Where r and θ are calculated by the following formulas:

$$r = kt$$

$$\theta = ct$$

(t = time, k = tightness, and c = # of spirals)

We chose this algorithm because it relatively simple to implement, and we feel it will yield good results in practice.

There are some difficulties that we have encountered when we worked on tracking algorithm research. To discover tracking algorithms that rely on understanding how animals like moths, dogs, and lobsters track to their sources has taken some time for us to fully absorb and comprehend. Many research groups are still experimenting in this area of study. There are also not many papers about tracking algorithms published by the research community. Dr. Ricardo Gutierrez-Osuna will help us to discover more sources about tracking algorithms.

We will implement the proposed tracking algorithm in the next several weeks while we continue searching for other new tracking algorithms.

V. Existing Dilution System

Jason was first assigned to study the dilution system that would be used later in the project. The dilution system consists of three Model 1010 Precision Gas Diluters from Custom Sensor Solutions, a sensor array chamber with four TGS26xx series odor sensors from Figaro USA and one HIH series temperature/humidity sensor from Honeywell, a vacuum pump, a PC Board (PCB) with all the circuitry, two Data

Acquisition Cards (DAQs) from National Instruments, and a user interface program made in LabVIEW from National Instruments.

The three gas diluters are used for three different gases. There is one diluter hooked up for each gas and then the three are mixed together to form one odor mixture. Each diluter has a tube from the gas container to the input of the diluter. Each diluter is also controlled by the LabVIEW program via the remote control terminals on the back of the diluter. Based on the concentration entered in the LabVIEW program, the Data Acquisition Cards send a voltage in the range of 0 to 5 volts, which corresponds to a concentration range of 0 to 100 percent. The output of each diluter is attached to a three-to-one tube, which combines the three diluters' outputs into one output, and acts as a mixing chamber.

This odor mixture is then passed through the sensor array. The sensor array consists of four different odor sensors and one temperature/humidity sensor. Since the temperature and humidity level can affect how the sensors react, it is important to have a sensor that reads these levels in order to properly interpret the results. The four odor sensors are based on chemical reactions. The metal oxides in the sensors join with the oxygen in the air and create a resistive barrier layer around the sensor, which increases the sensors' electrical resistance. Odorous gases tend to be deoxidizing gases. As they come into contact with the sensors, they react with the sensors and remove the oxygen from the barrier layer that was formed. This in effect lowers the electrical resistance of the sensors. The sensors' resistance follows this general equation,

$$R_s = A[C]^{\square}$$

where R_s = sensor resistance, A = some constant, [C] = gas concentration, and α = slope of the R_s curve. This resistance is then inserted into a voltage divider circuit (Figure ?), and the resulting voltage is then sent to the LabVIEW program via the Data Acquisition Cards. With this circuit, as the sensor resistance decreases, the resulting sensor voltage increases. Therefore, generally as concentration of an odor increases, the sensor resistance decreases, and the sensor voltage increases.

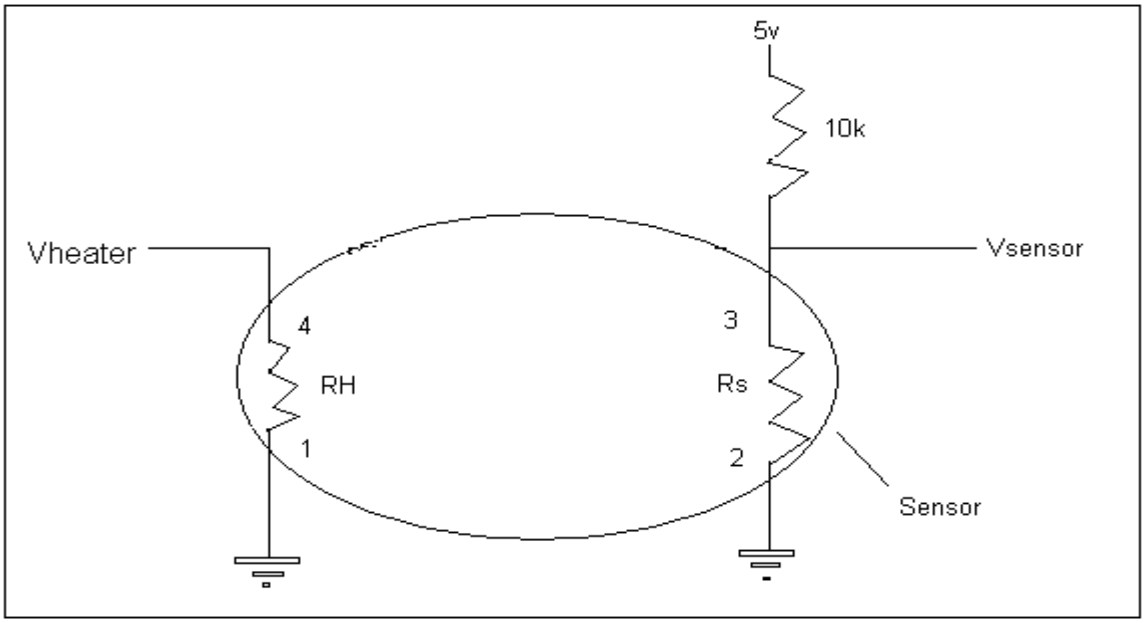


Figure ?. Sensor Circuit.

The pump involved is a vacuum pump and is used to pull the odor through the diluters and through the sensors. The pump is driven by a circuit that turns the pump on whenever the voltage is greater than 4 volts. Otherwise, it is turned off. The PC Board basically just holds all the circuitry in a small area to conserve space. The Data Acquisition Cards interface the circuitry with the LabVIEW program on the computer, and they also drive the diluters and the pump as well as collect the sensor responses.

The LabVIEW program acts as the interface between the user and the dilution system. The program allows the user to set parameters for the heater voltage and the

dilution profile. The heater voltage inputs consist of an upper limit, a lower limit, a period, and a pre-heat stage. The heater voltage upper limit is the maximum amplitude voltage, which should be less than or equal to about six volts. The lower limit is the minimum amplitude voltage, which should be greater than or equal to zero volts. The period is the length of time for one heater voltage cycle, which should be about 2-3 seconds. The pre-heat stage is the length of time to heat the sensors before using them, which should be about 15 minutes. The dilution profile consists of a table of concentrations for each diluter for each step and the step duration. For the table of concentrations, the columns correspond to each gas, and the rows correspond to each step or time you run the dilution system. The step duration refers to the length of time of one dilution step. This program also allows you to save the sensor responses to a file, load a dilution profile from a file, and manually turn the pump on and off.

The analytes used in this system are ammonia, isopropyl alcohol, and acetone. The group who designed this system normalized the gases to determine suitable relative concentrations for each analyte so that all three could be detected when all three were present. The normalized concentration for ammonia was 11.1 percent. The normalized concentration for isopropyl alcohol was 0.41 percent. The normalized concentration for acetone was 0.14 percent.

For our design, we determined what inputs and outputs of the dilution system were needed. We decided that the heater voltage limits, period, pre-heat stage, and the step duration would be set within the system, since they would be constant in our project environment. We will have the dispersion model send a dilution profile consisting of the concentrations of each odor at the robot's location. The robot simulator would take either

the sensor voltages or the actual sensor resistances for all five sensors in the array. It would then interpret the results and determine the concentrations of each odor and decide on where to move next based on its tracking algorithm. We considered incorporating a separate module that handled the sensor responses as the inputs and gave the individual odor concentrations as its outputs. We considered this option so that as the sensor response method became more complicated and changed, we can just work on this module instead of modifying the robot simulator each time. After talking with Dr. Gutierrez-Osuna, we determined that we would be dealing only with a single odor source, so the sensor response method could easily be handled within the robot simulator.

We will need to implement an electronic nose simulator before using the dilution system provided by a previous class. We will use this simulator to provide a sensor response based on the given concentration so that we can quickly and efficiently modify the robot simulator's algorithms for interpreting the sensor response and tracking the odor source. Once these algorithms are working, we will replace the electronic nose simulator with the actual electronic nose system and continue to test and refine our algorithms. For our electronic nose simulator we planned on using the general equation, mentioned earlier, and is given here:

$$R_s = A[C]^{\alpha}$$

The values for A and α vary according to the gases present. Therefore, since we are only using one odor source, we decided to use the following equation and values associated with methane (CH_4),

$$R = R_0\left(1 + K_{CH_4}[CH_4]\right)^{\alpha}$$

where R = sensor resistance, R_0 = 27.4 kΩ, $K_{CH4} = 4.37 \times 10^{-3}$ ppm^{-1}, $[CH_4]$ = concentration of methane gas, and $\alpha = 0.34$. We used this equation because it was similar to the general equation for sensor response. It also did not really matter which gas was used since the only purpose of the electronic nose simulator is to test the robot simulator's algorithms for interpreting the sensor response and tracking the odor source.

During the next stage of our project, we will begin coding the electronic nose simulator, since we now know what programming language we will be using and we have the equation for the simulator. As soon as the other two modules are completed, we can begin testing the algorithms. Once we feel that everything is working according to plan, we will attach the dilution system and continue to test and modify the algorithms to work with real sensor responses.

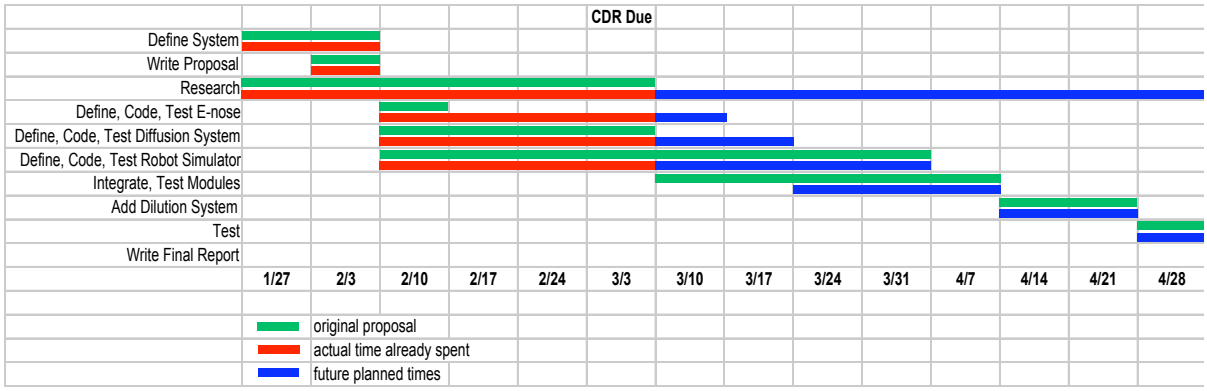
Once the coding for the electronic nose is completed and awaiting the completion of the other modules, we will begin running tests on the dilution system to begin the pattern recognition to use for interpreting the sensor responses later on. We will have to do more research on the methods of pattern recognition and figure out how we will do ours. From this pattern recognition, we will be able to determine the algorithm for the robot simulator to use in interpreting the sensor responses when attached to the dilution system.

We will also have to modify some of the LabVIEW interface for the dilution system so that our dilution module can automatically send the concentration of the odor at the robot's location. We will also need the sensor responses to be sent directly to the robot simulator so that it can track the odor to the odor source.

VI. Future Progress

The advances described here have allowed us to progress in our initial goal of creating three modules; a dispersion module to model the odor concentration in the air, a tracking module to implement our tracking algorithm, and a dilution system module to model the existing dilution system. The next step we will take is to begin coding. Having each done our research, we are going to come together again as a group and combine our results. The dispersion model will be easy to code in C or C++, and coding will begin immediately. This module will then be integrated with LabVIEW. We will also begin coding the tracking algorithm we have chosen. We must figure out how to accept the appropriate input from LabVIEW in this module, and figure out how to interpret it. Given the information expressed in this paper, this next stage should not be too difficult.

VII. Scheduling



So far, we are close to being on schedule according to our Gantt chart. One area that appears to be taking too long is research. However, this is misleading; we have done sufficient research to proceed with our development, but we have also discovered that we

will have to continue to do research for the lifetime of the project. Other than this, we are roughly on schedule, and we foresee no obvious or major difficulties to befall us in the weeks to follow.

VIII. References

[1] Marques, L. et al. *Olfaction-based Mobile Robot Navigation.* Thin Solid Films 418 (2002) 51-58

[2] Danny,G. et al. *Modeling Approaches to Understand Odor-guided Locomotion.* http://flightpath.neurobio.arixona.edu/Model/index.html

[3] Hayes, T Adam et al. *Distributed Odor Source Localization.* IEEE Sensors Journal, Vol.2, No.3, June 2002

[4] Barratt, Rod, Atmospheric Dispersion Modeling, An Introduction to Practical Applications, 2002

[5] Schnelle, Jr., Partha R. Dey, Karl B., Atmospheric Dispersion Modeling Compliance Guide, 1999

[6] Zannetti, P., Air Pollution Modeling, 1990