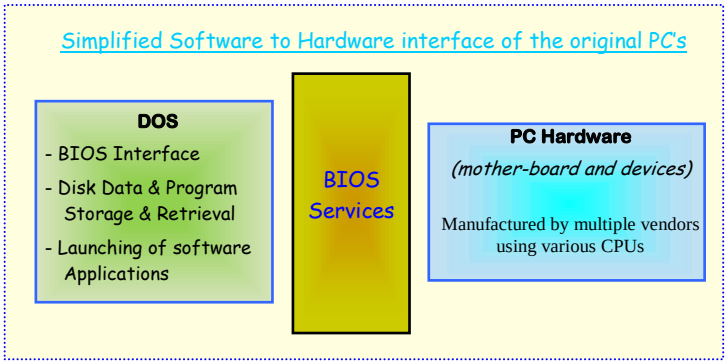
The following is a brief discussion of design concepts that under-lay the robot I have built and demonstrated in the top entry video. A fundamental criterion of my plan was that:

One Robot would be comprised of many drones, working in a defined area, to accomplish a goal.

In essence, the sum of all the host software functions and all the capabilities of the drones are **one** robot. No different than the metaphor: "You have hands to manipulate the world around you and you have feet to move you about". They are not separate entities; they are a part of all you are.

To that end, the first software-wrapper, written in Visual Basic (seen the 'Figure 8' challenge video) is an interface to a fleet of drones. The interface is built on a set of standards that removes idiosyncrasies found among drones constructed with different hardware. This has been accomplished by using the original design template for the personal computer of the late 1970's.

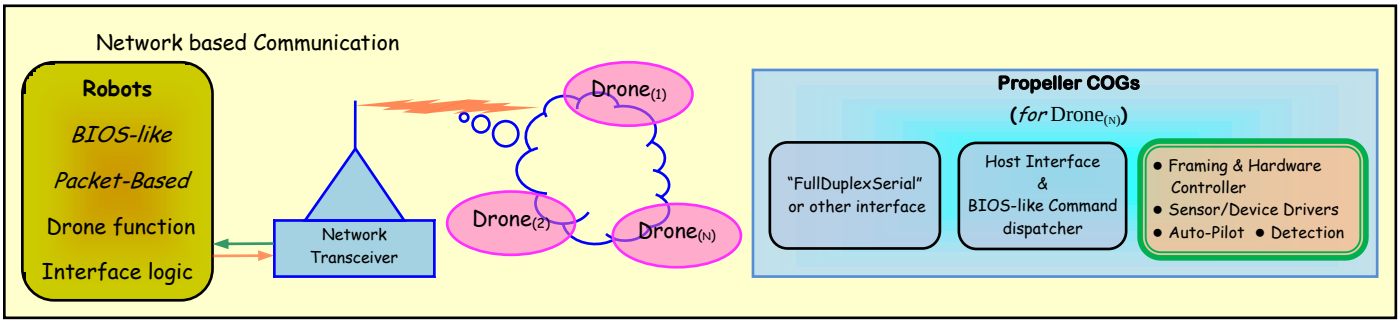


Putting aside your emotional feelings of who and what Microsoft is now, this Software-Hardware interface paradigm won the early PC design wars and was used for many years before better became established.

In the early years of DOS, we older code developers viewed DOS as having 3 layers. The lowest layer, 'BIOS' was an interface that theoretically guarantied, that our developed applications would run on any 'DOS' compliant hardware. The second layer gave developers large amounts of persistent off-line storage. The third layer has allowed human creativity to soar. Look what is being done today with software. It all started from simple ideas, brought to reality as someone's garage hobby many decades ago.

In my "Home-Based Robotic Development' environment, I have taken the original concept and modernized it a bit. It is important not to think of the above concept as a software platform but, think of it as a template to build a robot upon.

The first bit added was a "Packet-Based" BIOS-like set of calls across a network that interfaces a physical drone and the robot. In the above diagram, the following box would partially sit over the green 'DOS' box,



completely cross the gold colored 'BIOS Services' box and replace the blue 'PC Hardware' box. This concept

change now allows the creation of a robot where all its tools in the real world (insert your own drone ideas here) are managed and controlled by your robot application that runs on the digital device of your choosing.

The reason chosen for using "Packet-Based" BIOS-like calls is that transmitted packets always receive acknowledgment packets. If a packet transmitted to a drone is saved along with its acknowledgment, then we have captured a 'real world' drone response to a stimulus (a BIOS-like command).

In my robot, the "Get A2D Temp" BIOS-like command packet will be acknowledged with the sensor reading and relevant information such as: Heading, Battery Voltage, XBee signal strength.

This "Packet and Response" is the basis of my second fundamental robot design criteria (see below).

The next bit I modernized is the concept of data storage and retrieval (part of the green 'DOS' box). In my robot, all collected data is managed as a simple precept of the real world. Information within that precept is available to guide and direct all active drones in the field.

If a drone is asked to perform an ultra-sonic scan, the results are analyzed; objects found are added to the precept, empty scan regions are added to the precept. In essence, the precept is an on-going/developing map defining the area where all online drones can safely work in.

For simplicity, I define a "Session" to be a collection of all BIOS-like calls (and their responses), made to any of the on-line physical drones or bots over a period of time. Usually a session will accomplish a goal. Now, if we save all packets (and the responses) of a session, we have a firm record of the steps taken by the robot in order to reach its session goal.

The second fundamental design criteria under pinning my robot is "The Saved Session". A saved session has two purposes. If reloaded when an active fleet of drones exists, and the "Re-Run", the drone will perform and respond at their top speed. We are re-playing a script to achieve a goal. This is how my drone attempting the "Figure 8 Challenge" were controlled.

The second purpose of a saved session is that it can be loaded by the robot when there are NO active drones. If the session or script is run, the robot experiences real-world feed back even though there is no real world activity. An example of this is seen at the end of my video when my robot thinks its drones are really writing cursive script letters.

As a software developer, this design feature has shortened development time of numerous features considerably.

I hope this description of my robot's operation sheds some light on its basic operation as seen in the above video. The second software layer of my robot (I hope to introduce this fall), works exclusively building and managing the session's precept. Dead reckoning navigation is replaced by Localization. Directed recursive scanning fixes the location of objects and generates command streams that in the end will have caused a drone to map an entire area. All space is mapped via a 'Delaunay Triangulation' variant. A modified A* pathfinding algorithm is used to generate the routes of drones. Plus many more wanted robotic features.

Malcolm