Software Process Control for Robots

Computer Measurement Laboratory
info@cmlab.biz

Modern robots are ubiquitous. They hide in quite a number of different locations from car manufacturing plants to the battlefield. They all have one common characteristic. They are running out of control

A robot is really two machines in one package. There is a physical component that is visible, tangible and readily mistaken for the robot itself. The second machine is the abstract machine that is the software. The software is really running the show. Most people do no see the software or even think about its existence and that is the real problem. The software is the robot. The mechanical and electrical bits are simply tools for the robot to manipulate to achieve its design ends.

The robotic machines that are of greatest concern are those that have mobility. These are the robots. We are beginning to deploy robots in a multitude of different roles. Most disturbing is the notion that some of these robots are beginning to assume a role in modern warfare. Further, they are now beginning to be empowered to function autonomously. All of these robots in their myriad of different roles have one thing in common. They are running out of control. The software systems that are at the heart of these machines quite literally have no conscience. What is very important is the simple fact that our machines are growing more versatile and capable by the day. Unfortunately, the software machines that are animating these systems are essentially mindless. There is quite literally nobody minding the store.

Recently, a Northrop Grumman MQ-8 Fire Scout UAV took off over Washington, D.C. When communication was lost with this vehicle it was supposed to circle until it reestablished communication. Instead it went on a joy ride over the city. While this particular vehicle was unarmed, other similar UAVs carry 70mm Hydra rocket pods or Hellfire missiles. No only do these other UAVs have the clearly demonstrated capability of joy riding they might well also have the capability of using their weapons in a very unpredictable fashion. They can do this because the software controlling these systems is running out of control.

As an abstract machine the control software for an autonomous robot is astonishingly complex. From a design standpoint, the complexity of its components far exceeds that of a typical oil refinery or a nuclear power plant. From an engineering standpoint, it would never occur to an engineer designing an oil refinery not to create a control function to monitor and control the continuous process throughout the plant. To do this the engineer would embed sensors at critical points in the refining process so that temperature and pressure at these critical points could be monitored in real time. Further these sensors are then connected to a control unit and a feedback loop is built into the system to modify temperature, pressure and flow rates to make sure that the processes within the refinery are kept within a steady state.

These sensors can also serve in another mode. They can serve to set off alarms when mechanical failures occur in the refining process. It would be at once inconceivable and unthinkable to build

a refinery without a control system in place from the very beginning of the design process.

The abstract machines that are the software systems are most clearly not built with the same engineering discipline that went into the mechanical systems that they animate. Unfortunately the software abstract machines are hand crafted by software craftsmen. These systems are, for the most part, neither carefully specified nor designed. They are simply built using the same technology that was in use before the industrial revolution.

It is not unusual to put an engineer in a software development position and have him or her develop a complete case of amnesia about his/her training in engineering. Before an engineer would build a widget, he or she would first design the part. This design would have real numbers on it for the dimensions of the widget. It would also list the properties of the materials out of which the widget would be built. In the rare cases where software designs exist, we never find numbers on these designs. Imagine, if you will, a set of blueprints for a new building project with no numbers on the blueprints anywhere. That is the state of the art for software design.

One of the key components that should be engineered into any complex software system is some type of an adaptive control mechanism. This software process control system would monitor the operation of the software to insure that it stayed within a certified range of operation. At the point that the software began to stray from its certified range of operation, the software process control system would then move to restore the monitored software to a steady state condition.

In our control theoretic approach to software process monitoring, we are interested in measuring and monitoring the activity of the software system. We have designed a security coprocessor that monitors the running software system and measure its activity while it is running. The central issue here is that once we understand the notion that a program exhibits activity that can be measured, we can begin to assert control on the program execution. We can certify certain activities as nominal and reject activities that are outside of this range. We can do this in real time because we are monitoring the activity of the system in real time.

In our approach to software process monitoring, we are interested in the activity of the software system. We have designed a monitor coprocessor that monitors the running software system and measures its activity while it is running. In this context the data may be seen as stimuli for the program that exhibits some activity in response to each datum. Data in the normal range of user activity will induce normal activity on the software. Data outside of this range will induce different or unusual activity on the system that may be readily observed. In this approach the focus shifts from trying to model and understand the data space to which a program may be subjected to the activity of the program in response to the data input.

The key notion, here, is that there are measurements that may be taken from the executing software system. These measurements are analyzed in real time by the control system to determine whether the software has strayed from its nominal operating characteristics.

If we are going to cede control to robots for roles in manufacturing to warfare, then we had better be sure that we can guarantee that their actions will not be harmful. Imagine a tracked robot with

a machine gun mounted on a turret on its top. Given a fair amount of autonomy, this vehicle is just as deadly to friend as foe. That is so because it is running out of control. The circumstances that dictate that it should fire its weapon at a foe are very different than those surrounding its discharge at a friendly force. The robot has no conscience (i.e. software process control system).

We are gradually losing (ceding) control of most systems that are part of our lives. We play less and less of a role in the operation of our vehicles. Owners, for example, of hybrid vehicles are never really sure when the engine is running. Pilots in modern aircraft increasingly play an advisory role to the avionics package. With the advent of UAVs it is increasingly clear that a pilot is now longer necessary. Very soon we will lose control of our own cars. We will be able to determine destination, environment and entertainment functions but little else.

The bottom line, here, is that our robots must operate under the aegis of a totally separate abstract software machine that is the software process control system. This is, de facto, the conscience of the robot. Only when we have been successful in implementing a machine conscience (vs. machine intelligence) do we stand a reasonable chance of control robots of the future.

The CML CARMA technology is, at its core, a software process control technology. It is admirably suited to the robotic software process control role. We have developed this technology for use in computer security contexts under the aegis of funding from the Office of the Secretary of Defense. The technology can be readily adapted for use in a much broader software process control environment.