Theme Report 3: Act

COMEPNG 2DX3: Microprocessor Systems Project

Wednesday, April 1st, 2025

Joey McIntyre (400520473)

Theme

In an intelligent system, the act theme represents a systems ability to execute decisions and affect its environment through its outputs or movements. It is the final stage in our understanding of intelligent systems in the 2DX3 course, and it is the final step in the Observe-Reason-Act loop. After observing inputs and using reasoning to interpret them, the system must act upon its decisions to complete the desired task. The act theme is crucial because in the context of intelligent systems, a system is anything that can perceive its environment through sensors and act upon that environment [1]. This means that without the ability to act, an intelligent systems observations and reasoning would be worthless as it has no useful effect on the real world. Acting is a broad theme as it could include moving a motor, turning on an LED, sending any for of communication, or really any operation that changes the state of the system or its surroundings. Embedded systems typically interact directly with the real world by processing sensor data and turning the results into action via actuators [2]. Thus, the act theme is crucial for enabling autonomous behaviour because It completes the observe-reason-act loop by allowing the system to respond to its conditions and carry out an appropriate response.

The relevance of the act theme is demonstrated in pretty much every intelligent system that responds to their inputs. For example, a smart thermostat activates a furnace or air conditioning when the temperature is observed to deviate from desired ranges. Similarly, self-driving vehicles, after detecting an obstacle, will hit the brakes or steer to avoid it. This demonstrated how important the act theme is, as simply observing this obstacle is useless if the autonomous vehicle cannot act to avoid said obstacle. In out labs and the final project, the act theme was critical to having our microcontroller preform real-world actions. While the observe theme (labs 1-3) focused on gathering data and the reason theme (labs 4-6) dealt with making decisions based on that data, the act theme (labs 7-8 and the final project) focuses on the systems responses. In labs 7 and 8, this involved using event-driven programming to trigger outputs in real time. The ability to act ensured out embedded system could not only sense and think, but also directly influence its environment, which is a fundamental requirement for any intelligent or autonomous system.

Background

The act theme is essential to smart systems because it is the part of the process where abstract decisions become concrete operations. No matter how well a system observes and reasons, its goals can only be accomplished by executing actions. In embedded microprocessor systems, acting often involves driving hardware. This includes controlling motors, toggling indicator LEDs, sending data to other devices, etc. This capability transforms a passive system into an interactive one that can control its environment. A good comparison of a system without the act stage is a brain without muscles; it can observe and think, but it can’t make anything happen. For an intelligent system to be useful in the real world, it must be able to appropriately respond to events or conditions in a timely manner. This is especially important in real-time applications. A good example of this is in systems that require urgency to ensure safety. A fire alarm must act by triggering sirens and sprinklers immediately after detecting smoke. Any delay, or worse, failure

to act renders the ability to detect smoke useless. The act theme ensures that when certain conditions are met (observed inputs and reasoning outcomes), the system will execute the required response to accomplish its overall goal.

One key aspect when it comes to implementing the act theme in microcontrollers is the use of event-driven programming and interrupts. In earlier labs, many actions were controlled by polling loops that continuously checked the status of sensors. As discussed in lecture, the limitation of polling is that it can waste CPU time as it is constantly polling for changes. This is why we utilize interrupts, which allow the processor to act the moment a specific event occurs, which improves responsiveness and efficiency. For example, instead of continuously checking a timer to toggle an output, a periodic interrupt can be set to trigger the output change at exact time intervals. This not only frees up the CPU to preform other important tasks or use less power but also gives much more precise timing. In lab 7, we were introduced to using hardware interrupts to respond to external inputs or time events. By doing so, out system could handle multiple events and act on them immediately without the need for supervision by the main program. This is very important in complex systems (such as out final project), where multimer inputs and outputs need to be managed seamlessly at the same time. Event-driven action is seen in many real-life scenarios such as engines that use interrupt signals from sensors to adjust fuel injection or wearable devices like Apple or Garmin watches, which wake from sleep and record data only when motion is detected to conserve energy. In summary the act theme is about control and responsiveness. It is what allows intelligent systems to translate decisions into real-world outcomes in a reliable and efficient manner.

Theme Exemplars

To demonstrate the act theme in intelligent systems, I will present two examples from our coursework that showcases how systems carry out actions in response to events or inputs. These examples come from lab 7 (Interrupts and Event Programming) and Project Deliverable 2. Both these examples illustrate the how decisions are turned to action. In the first example, the microcontroller preforms a simple timed output action using interrupts; in the second example, it controls a complex 3D spatial mapping system that interacts with the physical world. For each example, I will describe the method and implementation details and then discuss how I confirmed the systems behaviour.

The first example comes from Lab 7 – Interrupts and Event Programming. Milestone 9.1 does a good job of highlighting the act theme. In this task, we implemented a periodic time interrupt to make the system act at regular intervals without continuous polling. The goal was to flash an on-board LED with a 1 second period (1Hz frequency) and a 25% duty cycle, which simultaneously outputting this period signal to a GPIO pin. This demonstrates the act theme because the microcontroller is proactively generating an output behaviour (which is the blinking LED and a square wave) autonomously.

This milestone began with setting up a hardware time to trigger interrupts at a interval of 1 seconds. We used and adapted code from a prior studio exercise and configured the

microcontrollers time to produce an interrupt every 1000ms. In the interrupt service routing (ISR), the action taken was to turn on the LED and set a designated GPIO pin HIGH, then ensure it would turn LOW after 250ms (which achieves the 25% on-time). To implement the 250ms pulse width, we used a second timer or a simple delay loop inside the ISR (since the interval was relatively long, a brief busy-wait for 0.25s was acceptable in the context of this lab). A different approach could be to use two interrupts (one for LED on, one for LED off), but our simple implementation still met this labs requirement. We configured LED D1 as the output indicator and selected a square GPIO pin (on port N) to output the pulse waveform for measurement.

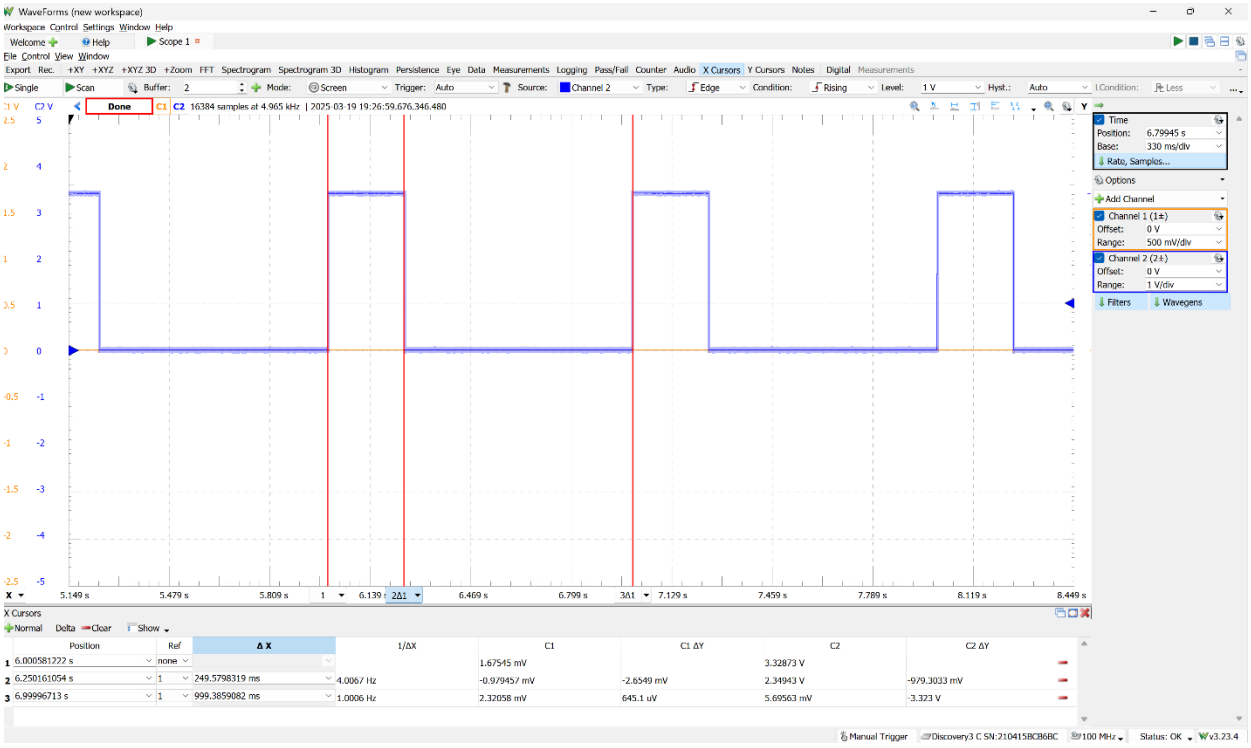

Figure 1: Oscilloscope Capture to 1Hz Interrupt-Driven LED pulse (showing 1s period and 25% duty cycle)

After coding the periodic interrupt and ISR, we validated the results of this milestone in two ways. First, we observed the on-board LED to confirm it was blinking for 0.25 seconds and off for 0.75 seconds in a continuous cycle. This was a simple visual confirmation that we implemented the milestone correctly. Secondly, for a more accurate verification, we used the AD3's built in oscilloscope to measure the output waveform on the GPIO pin. Figure 1 shows the square wave output voltage measured with the AD3, where the pulse width (when the voltage is HIGH) is one quarter of the total period. The oscilloscope trace showed a clear 1Hz frequency and the expected duty cycle, which confirmed the interrupt-driven routine was functioning with accurate timing. We also compared this to a lab we did earlier in the semester where a similar LED blink was achieved using a polling method – and the interrupt-based approach achieved the same behaviour with simply less processor power being used during the off periods. This milestone really exemplified the act theme by demonstrating the microcontroller actively acting on a timed schedule and blinking an LED as a form of periodic output without any direct input

from us. This shows how an embedded system can act in a timely and autonomous manner using interrupts.

The second example came from the second and final project deliverable. The goal of this project was to design and build a fully integrated system that observes the environment around it and acts accordingly to preform a complex task. This project combined and built on what we learned from each lab and challenged us to develop an embedded 3D spatial mapping system that uses a time-of-flight distance sensor and a stepper motor. A crucial requirement of the system is taking in real world inputs like distance measurements and user commands and produce the appropriate actions like rotating the motor, starting a distance scan, and communicating the results to my PC and visualizing it using the open3D software. The act theme is strongly demonstrated here as the microcontroller controls the hardware setup and communicates with all the external devices in real time.

The main function of this system was to scan its surrounding environment with a 360-degree sweep and record 32 distance measurements, capturing distance data using the VL53L1X ToF sensor mounted on our stepper motor. Pressing the onboard PJ1 button triggered an interrupt that started the motor rotation and sensor data collection. The microcontroller controlled the stepper motor via the port H GPIO pins and triggered the sensor readings through I2C communication at incremental steps. A finite state machine managed the states and coordinated the rotating and the sensor data acquisition.

Timing was a very important part of this project. The motor speed ensured sufficient time for accurate sensor readings. After the scan, polar coordinates were converted to cartesian coordinates and stored in memory. The FSM then activated UART transmission of the collected data to my PC for visualisation. On-board LEDs also provided real time status indications for measurements being taken, data being transmitted, and for the motor moving.

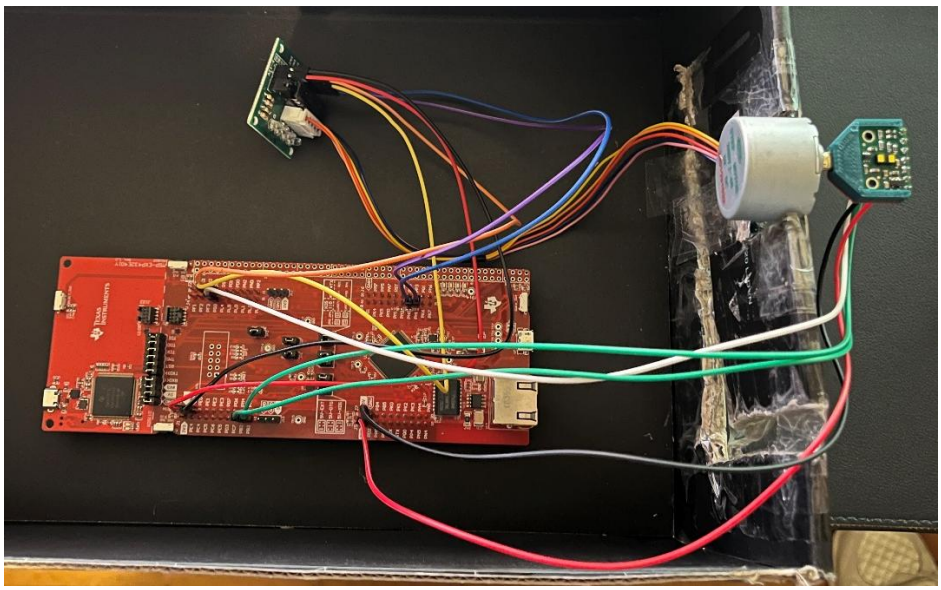

Figure 2: Hardware Implementation of the 3D Spatial Mapping System

Figure 2 shows the hardware components of the final project. When it came to validating the action of the system, this included visual verification of the motors full rotation with accurate and efficient data measurements being taken. In the final demonstration, the system autonomously scanned my assigned environment, transmitted data via UART, and visualized it using a python script and the Open3D software. The resulting 3D plot accurately matched the physical space, as can be seen in Figure 3 below. This confirmed that the system was acting as intended.

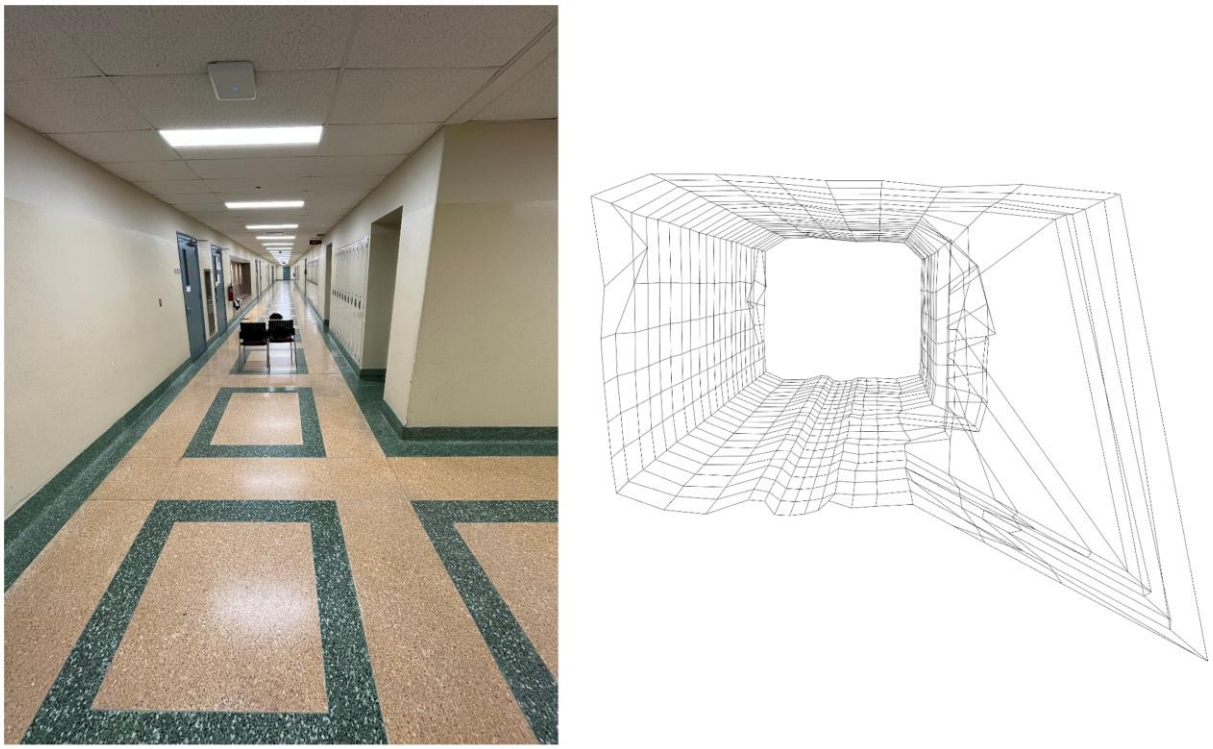

Figure 3: Side-by-Side Comparison of Scan Location and Scan Result

Debugging Exemplar

Debugging has been an essential component in every theme we have discussed so far, but it is particularly integral to the act theme. This is because a system that observes, reasons, and acts is bound to be somewhat complex with numerous components, so it can often be difficult to determine what's causing the system to not function correctly. A good example is during my project's integration phase, I came across an intermittent error where the VL53L1X Time-of-flight second occasionally failed to indicate that measurement data was ready, which caused unexpected scanning interruptions. Initially, the sensor would become stuck indefinitely waiting for data, which completely stopped the scanning process.

To debug this issue, I used a timeout counter method directly in my code, as shown in Figure 4. In the Get_Distances() function, within the measurement loop, I incremented a test counter each time the sensor data was checked but found not ready.

```
for(int i = 0; i < 32; i++)
{
  int test = 0;

  // Wait until the ToF sensor's data is ready
  while (dataReady == 0)
  {
    status = VL53L1X_CheckForDataReady(dev, &dataReady);
    VL53L1_WaitMs(dev, 5);                                    // Wait 5ms between checks
    test++;

    // Timeout error message meaning sensor didnt become ready
    if(test == 100)
    {
      UART_printf("scan failed\r\n");
      i = (100);                                              // Force exit from loop
      dataReady = 1;
    }
  }
```

Figure 4: Debugging Timeout Mechanism

This debugging strategy allowed the system to detect any sensor malfunctions or unresponsive conditions by counting how many times it checked for new data without success. If the sensor failed to respond withing the 100 allowed attempts, the system logs a message via UART indicating to the user that the “scan failed” and safely exits the scanning loop rather than freezing indefinitely.

When observing UART output logs, instances when the “scan failed” were clearly indicated, which revealed sensor responsiveness issues. This allowed me to address the underlying initialization issue which I resolved by reinitializing the sensor after detecting a failure. Using this built-in timeout approach provided me with real-time error detection and seamless recovery from sensor errors. It really highlighted how important implementing explicit timeout-based debugging strategist directly in firmware is, and how its crucial for an embedded system to preform as intended. This was just one of many debugging methods I used in this project, with other including using the Keil debugger, using LEDs to indicate certain action (or inaction), and even using the AD3’s oscilloscope. All these methods combined allowed me to present a system that can observe, reason and act appropriately as per the specifications of this project.

Synthesis

The exemplars from Lab 7 and the final project collectively demonstrate the act theme by showing how embedded systems preform real-world actions based on its internal decision making. Lab 7 exemplifies the theme through precise and autonomous timing, using interrupts to produce periodic actions without constantly polling which uses valuable CPU power. The microcontroller effectively generated a repeating signal, demonstrating the efficiency and accuracy possible when using event-driven programming to manage outputs.

On a larger scale, the final project integrated multiple complex elements, including a stepper motor, ToF sensor, and UART communication, illustrating how complex systems act and make decisions. Here, the microcontroller coordinated motor movements and sensor data collection,

which highlighted intelligent systems ability to preform multiple actions simultaneously. Both theme examples emphasize the importance of timing, precision, and the seamless integration of various hardware components, reinforcing that effective action is central to embedded systems' functionality.

These exemplars show how out learning has progressed throughout the course. Initially, we learned basic event-driven techniques in Lab 7, then we integrated these concepts into a more advanced, fully autonomous spatial mapping system. Both exemplars rely on accurate observation and reasoning, illustrating the complete Observe-Reason-Act cycle, where action is the critical endpoint that makes intelligent systems relevant in the real world.

Reflection

Working through the act theme significantly deepened my understanding of how embedded systems interact meaningfully with their environments. Lab 7 demonstrated how interrupts could trigger a systems action in a more efficient manner, clearly improving upon the less efficient polling methods we initially used. Implementing periodic outputs gave me greater confidence in utilizing hardware timers and interrupt routines, which are fundamental skills in embedded system programming.

The final project provided an even deeper learning experience. Integrating sensors, motors, interrupts, and UART communications required countless hours of planning and troubleshooting, testing my skills in multitasking and real-time control. Dealing with issues like sensor responsiveness forced me to use a wide range of debugging methods, such as timeout counters, the Keil debugger, action LEDs and more, enhancing my ability to design fault-tolerant systems.

Overall, the experiences in the labs and the final project have solidified my understanding of the critical role action plays within intelligent systems, reinforcing the idea that the ability to act reliably and efficiently is essential for a system to be effective. Moving forward, I am much better prepared to design and debug systems capable of complex real-world interactions, combining theory and practical engineering skills I have gained throughout the 2DX3 course.

References

[1] S.-H. Park, “Reviewing ‘Artificial Intelligence Modern Approach’: Intro and Intelligent Agents,” *Medium*, Sep. 10, 2024. https://medium.com/@saehwanpark/a-deep-dive-into-artificial-intelligence-a-modern-approach-part-1-47e800369c69 (accessed Apr. 06, 2025).

[2] “How Do Embedded Systems Interact with Various Sensors,” *Pcbonline.com*, 2024. https://www.pcbonline.com/blog/embedded-system-interact-with-various-sensors.html