Zeetah Software Architecture

Harjit Singh

Problem Statement

- During practice at APEC 2015, the mouse would start of a speed run at the edge between the start cell and the cell north of it
  - The “patch” was not too difficult, but…

    it made me think that I should “fix” the bug by making a change which **prevented** this class of bug

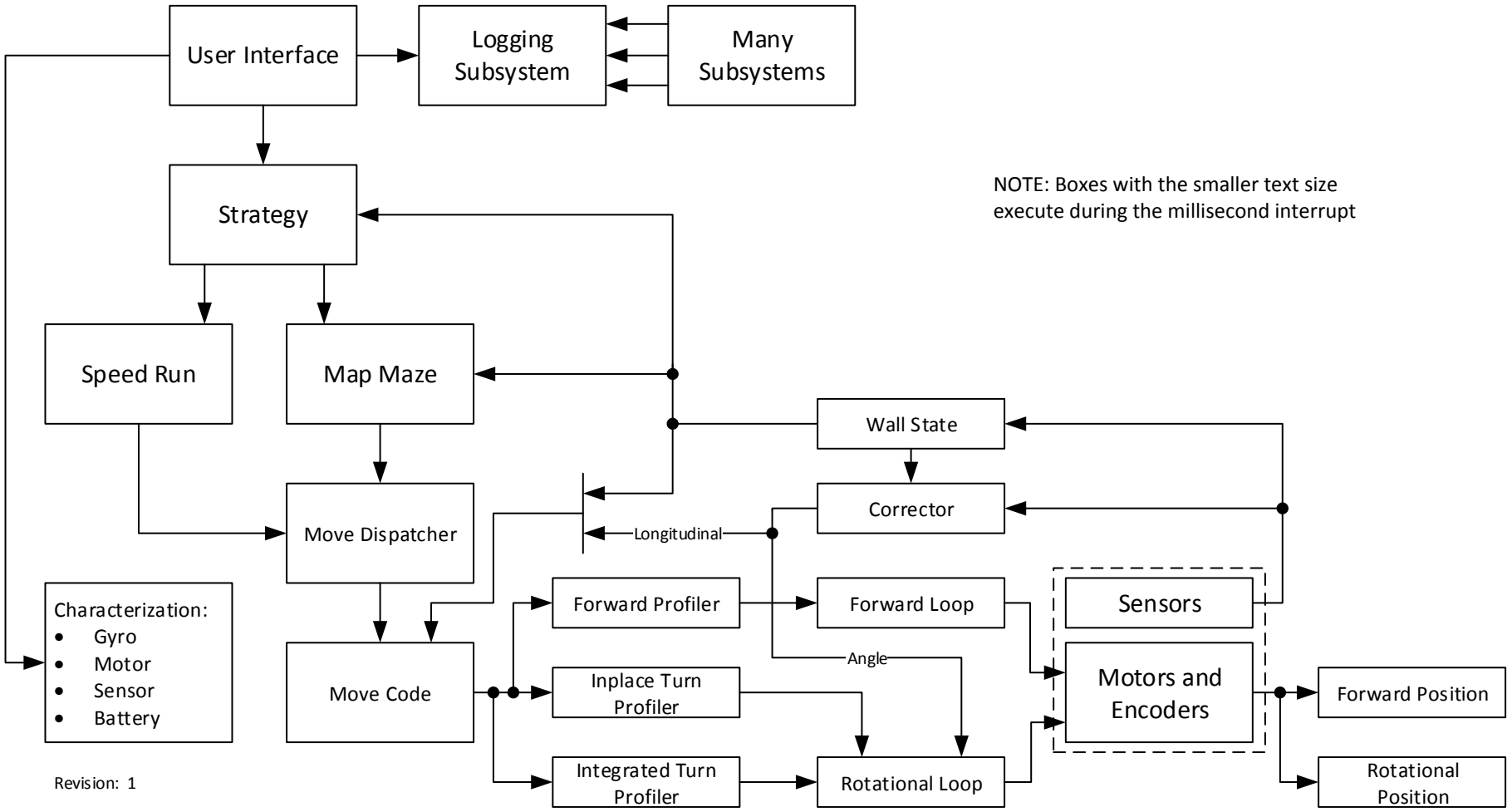
- 30% of the code the mouse is running is over **25 years old…**
  - … and portions have been patched many times since

Solution

- Write new bugs

- A couple of years ago I wrote a diagonal solver. This is the perfect opportunity to start using it
  - Diagonal solver needs:
    - A new coordinate system
    - Now have eight directions (N, NE, E, SE, S, SW, W, NW) instead of just four
    - These are hacked into the current system

“New” Architecture



User Interface

- On boot, the hardware is initialized into a benign state
- The millisecond interrupt is started
  - Need to do this to run the display, SPI and other code

- Select contest type – only on boot
- Whether we are going to connect to the host PC to transfer log data or clear log data
- Potentially change strategy
- Start run with progressive or over-riden run parameters

Strategy

- Based on contest type, this code will decide:
  - What to do when a run completes
    - Take mouse back to start or just stop
    - For example, at APEC
      - Where there is a maze time penalty, we could compute run time benefit of learning more of the maze VS doing a run with what is known
      - If the mouse has not been touched, auto start a speed run
  - Maintain:
    - Run counter, attempts, whether the mouse was touched or not
  - Based on input from the User Interface state
    - Setup run parameters
    - Calculate time for each move

Speed Run

- Assume all unknown walls in the maze exist, generate a path to destination
  - Destination can be Center or Start square
  - Path can use diagonal turns or not

- On completion of run, return to Strategy

Map Maze

- Called with learn mode:
  - “Learn to center”
  - “Learn to start”
  - “Learn completely”

- Return to Strategy on:
  - Completion of run

    Or
  - Determining that there are no unknown walls between the current location and the destination

Move Dispatcher

- Until the entire path has been executed, call the actual motion routines one after another

Move Code

- Based on the move type (forward, turn):
  - Setup the motion distance
  - Enable the appropriate profiler(s)
  - Sequence the state machine that:
    - Determines if walls are present or not and updates the Wall State
    - Updates the mouse location and orientation

Profilers

- Forward
  - Trapezoidal profiler
- Inplace Turn
  - Triangle profiler
  - When this is running forward profiler is off
- Integrated Turn
  - Cubic spiral turn profiler
  - This is run with the forward profiler

Loops

- Forward
  - Lead compensator with feed forward
- Rotational
  - Lead compensator with feed forward
- There is a very complete description of how to determine compensator and feed forward parameters on micromouseonline.com

Corrector

- Longitudinal
  - Looks at sensor data to detect falling edges
    - Location of falling edge along with mouse sideways offset is used to update mouse current position
    - I look for the three consecutive wall to three consecutive no wall readings to declare a falling edge
  - If there is a front wall, use the front sensors to compute the distance to the front wall and update the mouse current position
- Angle
  - Accumulating three sensor readings to compute angle of mouse
    - I only correct once per cell

Wall State

- Accumulate three sensor readings
- If the readings are above a certain threshold, there is a wall
- Update Wall State for the cell the mouse is in

Logging

- See MINOS 2014 presentation for details