Self-Healing Autonomous Vehicles
Increasing System Resiliency with Automated Program Repair

Kevin Angstadt
Department of Computer Science
University of Virginia
angstadt@virginia.edu
26. February 2016
Robots Go Where We Cannot

- Limited, Delayed, or Impeded Communication
  - Spirit Rover Stopped Communication Due to Software Fault

- Redundant Hardware Protects Against Physical Damage
Let’s focus on a similar system that costs much less…
Quadcopters are not Immune

Drone delivering asparagus to Dutch restaurant crashes and bursts into flames
By Mary-Ann Russon
April 14, 2015 13:22 BST

Drone crashes into house wall in man's demonstration of software bug
By Aaron Sarkin
Jan 14, 2015, 8:20am CT | Last updated Jan 14, 2015, 11:34am CT
Cutting to the Chase

• Automated program repair can generate software patches to increase system resiliency in autonomous vehicles

• We apply these patches during operation using special hardware
Today’s Action-Packed Episode

• Overview of Quadcopters
  – Hardware System
  – Software System

• An Example Bug

• Automated Program Repair Crash Course

• Understanding Processors and Patching

• Demo of Software Resiliency
Quadcopter Communication

- **Manual Flight Commands**
- **Flight Plans + Position Information**

- **RC Controller**
- **Computer + Telemetry Radio**
Flying a Mission
How does a Quadcopter Fly?

Feedback Loop

INPUT

GPS Data
Accelerometer
Compass
Altimeter
RC Input
Motor Speed

OUTPUT

Motor Controllers

400Hz

What is the current position?
What is the current velocity?
What is the desired position?
What is the desired velocity?
What is the difference between these values?

Calculate Correction to Output

Quadcopter Moves
Quadcopters are Like Ogres

**flight_mode.pde**
- set_mode()
- update_flight_mode()

**control_stabilize.pde**
- stabilize_init()
- stabilize_run()

**AC_AttitudeControl.cpp**
- angle_of_rate_of_yaw()
- rate_controller_run()

**AP_MotorsMatrix.cpp**
- output()

**AP_HAL::RCOutput**
- write(ch, perido_us)

**update_flight_mode**: checks control_mode variable and calls flight_mode specific_run() function.

**stabilize_run**: Interprets pilot input and sets target roll, pitch and yaw angles (or rates)

**AttitudeControl::angle_of_rate_of_yaw**: calculates attitude error and converts them to high-level motor requests

**AP_MotorsMatrix::output**: converts high-level motor requests into individual motor outputs.

**AP_HAL::RCOutput::write**: sends PWM messages to ESCs

Source: dev.ardupilot.com
How do we fix this bug?
“What we would like ideally […] is the automatic detection and correction of bugs” –R. J. Abbott

AUTOMATED PROGRAM REPAIR
Defining the Problem

Given a program and evidence of a bug, fix that bug.

- Source code written by developers
- Binary that processor executes

Generate a list of changes to the program that, when applied, result in passing all tests or only normal runs

- Passing tests and one failing test
- Normal and anomalous runs
How do we do this automagically?

Given a program and evidence of a bug, fix that bug.

THINK EVOLUTION

Imagine dinosaurs rather than programs...
Dinosaurs vs. Program Repair

- Genetic code for next generation contained within current generation
- Code to correct the bug can be found elsewhere in the code
- Environmental factors result in survival of the fittest
- Test suites and indicative workloads result in survival of the fittest
- Many combinations of genetic material possible
- Search space for programs in enormous
The GenProg Plan

1. **Input**
2. **Evaluate Fitness**
3. **Discard**
4. **Accept**
5. **Output**

**Mutate**
Open Problems

• Fitness evaluation
  – How do we test quadcopter code (simulation?)

• Patch application
  – Easy: Land quadcopter, reflash, restart
  – Hard: What if the quadcopter cannot land? How do we apply a patch without reflashing and restarting the software?
Patching an Executing Binary

- Programming languages provide abstractions to help humans write software
- A compiler converts this high-level program into a binary that a processor can execute
- Since the processor executes the binary, we must apply the patch to this version of the program
What can a processor do?

- Execute a list of instructions
- *Instruction*: perform operation using one or two pieces of data
  - Add/Subtract/Divide/Multiply
  - Load a piece of data from memory
  - Store a piece of data to memory (sometimes)
  - Jump to a new location in the list of instructions
- Human-readable form is known as *assembly language*

```
.text
main:
   add $t0, $zero, 496
   add $t1, $zero, $zero
   add $t2, $zero, 1

loopbegin:
   beq $t0, $t2, loopend
   div $t0, $t2
   mfhi $t3
   one $t3, $zero, loopcont
   add $t1, $t1, $t2

loopcont:
   add $t2, $t2, 1
   bet $t0, $t1, summer
   add $t7, $zero, $zero
   j loopbegin

summer:
   add $t7, $zero, 1
   j loopbegin

loopend:
   jr $ra
```
Von Neumann Architecture

We can patch a binary by *inserting* into or *deleting* from the instruction list...

There are a number of steps in executing a CPU instruction:
- **Load Instruction** from memory
- **Load Data** from memory
- **Compute** (i.e. execute the instruction)
- **Store Result** back in memory

Memory

*We cannot necessarily change the list in memory.*

Processor

Load Instruction

Load Data

Compute

Store Result

Processor Memory (Registers)

Load Data

Store Result

Memory

...but we can’t necessarily change the list in memory

```
.text
main:
  add $t0, $zero, 496
  add $t1, $zero, $zero
  add $t2, $zero, 1
loopbegin:
  beq $t0, $t2, loopend
  div $t0, $t2
  mfhi $t3
  one $t3, $zero, loopcont
  add $t1, $t1, $t2
loopcont:
  add $t2, $t2, 1
  bgt $t0, $t1, summer
  add $t7, $zero, $zero
  j loopbegin
summer:
  add $t7, $zero, 1
  j loopbegin
loopend:
  jr $ra
```
Insertion-Only Patch

... add $t0, $t2, 496
add $t1, $t1, 3
add $t2, $t2, 1
sub $t2, $t2, 1
add $t3, $t1, $t2
...
Modifying the Instruction List

- Use a special piece of hardware to connect directly to the processor.
- Bypasses other system components.
- Used to control the processor and inspect data during testing.
- Can also execute an instruction not contained in the instruction list.

"The Brick"
Injecting Instructions

Processor

Memory

"The Brick"

Load Data

Inject Instruction

Compute

Processor Memory (Registers)

Store Result
Technical/Conceptual Challenges

• How do we convert a patch generated by GenProg to a list of instruction insertions?
• “The brick” is not always accurate
  – How much error can the program tolerate (acceptability envelope)?
Putting This All Together

• Quadcopter software utilizes a *feedback loop* and several layers of *abstraction* to sustain flight
• A programming error in one of these layers can cause *undesired behavior*
• *Automated program repair* borrows notions from evolution to search for patches to software
• Such a patch must be applied to the quadcopter software while still maintaining flight
• Special hardware can be used to inject instructions from the patch directly into the processor
THANK YOU!
Questions?