Improving Robotics with Safety-Critical Virtualization

Group #4

Faculty Advisor: Dr. Azim
Capstone Coordinator: Dr. Mahmoud

Anthea Ariyajeyam	100556294
Alex Motyka	100582888
Kalev Gonvick	100582575
Erin Rutkowski-jones	100587683

What's our project?

Main goal: improve robotics which implement both safety-critical and non-safety-critical functionalities

Why is this useful?

- Embedded systems are increasing in functionality
  - Greater range of responsibilities (ie. both safety-critical and non-safety-critical)
- Incorporating safety-critical and non-safety-critical functions in the same environment creates the risk of interference

Stakeholders & Their Requirements

Stakeholders: Developers & Public (users of the systems)

Requirement	Details
Performance	• Able to support multiple functionalities without decreased performance • Enhance or have equal performance compared to existing operating systems
Safety & Reliability	• Provide safety-net protocols that allow the system to be safely shut down in the event of system failures • Perform both safety-critical and non-safety critical tasks without causing any interference

Objectives

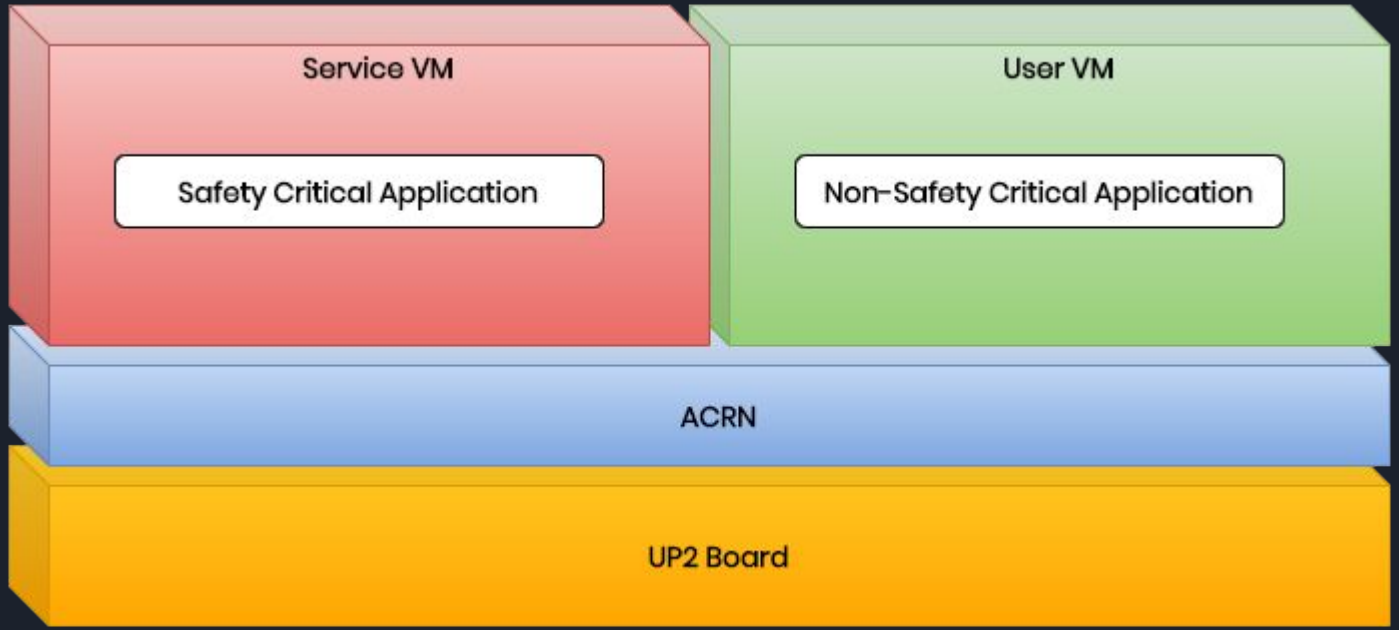
Original Objectives:

- Improve Performance & Reliability
- Provide Easier alternative for programming robots
- Demonstrate improvements with an autonomous wheelchair test case + compare

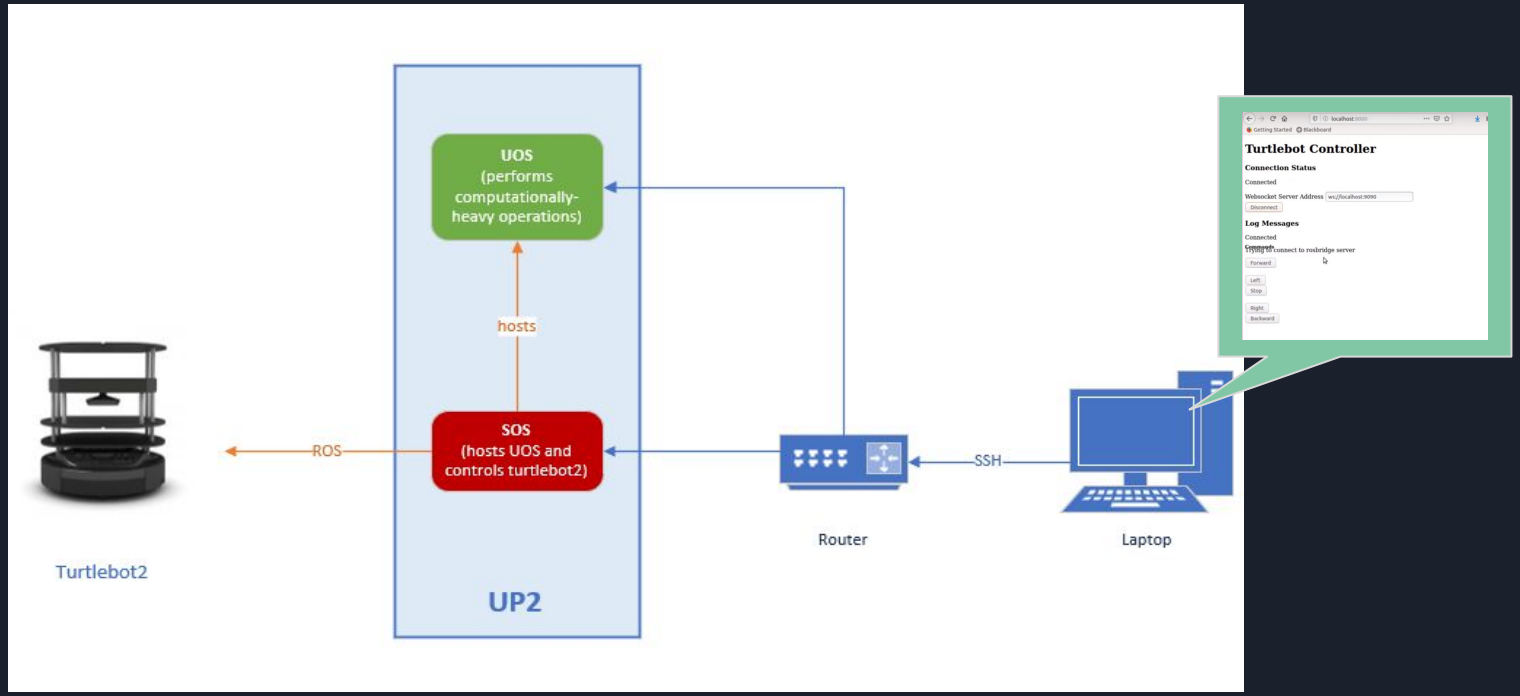
Modified Objectives:

- Improve Reliability and Safety
- Improve Performance
- Provide Easier alternative for programming robots
- Demonstrate improvements with a robotic test case + compare

Final ACRN Design



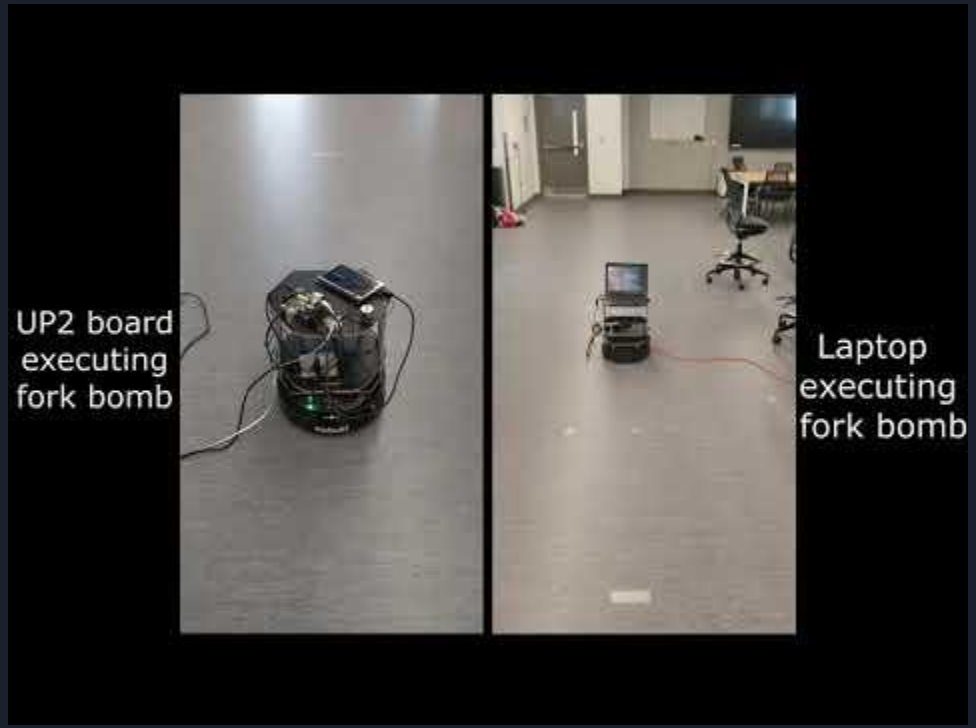
Physical Testing Configuration



Test Cases

Test Case Number	Test Case	Goal
1	Controlling Turtlebot Remotely via ROS Keyboard Teleop	Control the robot manually from a remote computer to move it six feet forward
2	Controlling Turtlebot Remotely via Web Application	Control the robot manually from a remote web application to move it
3	Completing a Basic Route Autonomously	Control the robot using a script that sequentially publishes multiple repetitive forward, left, and right commands in order to make the turtlebot navigate safely through an obstacle course

Demonstration



https://www.youtube.com/watch?v=Q18-37-W1OE&feature=emb_logo

Test Results: Hypervisor Solution

Test Case	Fork Bomb?	Average Time (seconds)	Additional Comments
Keyboard Teleop	Present	Passed (avg. 12.8s)	For the second case the UOS started to system hang.
	Not Present	Passed (avg. 12.83s)	
Webcontroller	Present	Passed (avg. 12.93s)	
	Not Present	Passed (avg. 12.97s)	
Autonomous	Present	Passed (avg. 12.37s)	
	Not Present	Passed (avg. 12.16s)	

Test Results: Traditional Setup

Test Case #	Fork Bomb?	Average Time (seconds)	Additional Comments
Keyboard Teleop	Present	⅔ Failed	Stalled at times or did not complete the task
	Not Present	Passed (avg. 13.18s)	
Webcontroller	Present	⅔ Failed	Stalled at times or did not complete the task
	Not Present	Passed (avg. 13.38s)	
Autonomous	Present	3/3 Failed	Either crashed or stalled
	Not Present	Passed	Collision occurred

Test Summary

- The hypervisor solution always completed its task without performance hits
- The hypervisor doesn’t restrict us when using ROS
- The laptop was unpredictable when a fork bomb was executed
  - Sometimes the bot would either stall, stop or spin
  - A crash was inevitable
- The hypervisor solution was actually performing faster when the UOS crashed

Main Challenges Faced

- No support for our hardware
  - Our board was APL UP2 (UEFI)
  - They only support APL UP2 (SBL)
  - Solved by going through source code instead of looking at their documentation
- Preparing for different scenarios
  - Mirroring projects made with ROS in a new setup for the different virtual environments
  - Solved by many iterations of test case designs
- Hardware limits on I/O
  - We are limited by a certain amount of USBs between the different operating systems
  - USBs are segmented across the board during runtime of both operating systems
  - Solved by showing test scenarios that require the least amount of user input

Area	V1.0@Q1'19	Q2'19	Q3'19	Q4'19	2020
HW	• APL NUC (UEFI) • KBL NUC (UEFI) • APL UP2 (SBL)	• APL NUC (UEFI) • KBL NUC (UEFI) • APL UP2 (SBL)	• APL NUC (UEFI) • KBL NUC (UEFI) • APL UP2 (SBL) • Denverton SoC	• APL NUC (UEFI) • KBL NUC (UEFI) • APL UP2 (SBL) • Denverton SoC	

Main Challenges Faced Cont.

- ROS modifications
  - Service VM used Ubuntu 18.04 with ROS melodic, and does not support Turtlebot 2
  - The User VM used Ubuntu 16.04 with ROS kinetic
  - Modifications needed to be made to ROS melodic in order to communicate between each other
  - Kobuki Yujin Robot (Turtlebot 2's main communication driver) had to be changed the most

Further Research

- More robust performance testing
  - Run a robotic system with multiple components
  - Would provide a more in depth test scenario for the hypervisor
  - Compare hypervisor against non-hypervisor solution
- Test with Zephyr
  - Used to run real-time robotics
- Test compatibility with other Common Linux Development Kits
  - R+, Arduino

Conclusion

- ACRN used to create a safety-prioritizing system
- ACRN modified to work with UP2 board ($400) instead of NUC ($800)
- Tested and confirmed to protect service OS from user OS crashes
- ROS successfully integrated → easy to deploy in existing systems

Questions

https://youtu.be/XOe2sJLMD1c

Resources

Images:

https://www.picpedia.org/highway-signs/r/research.html (further research slides)