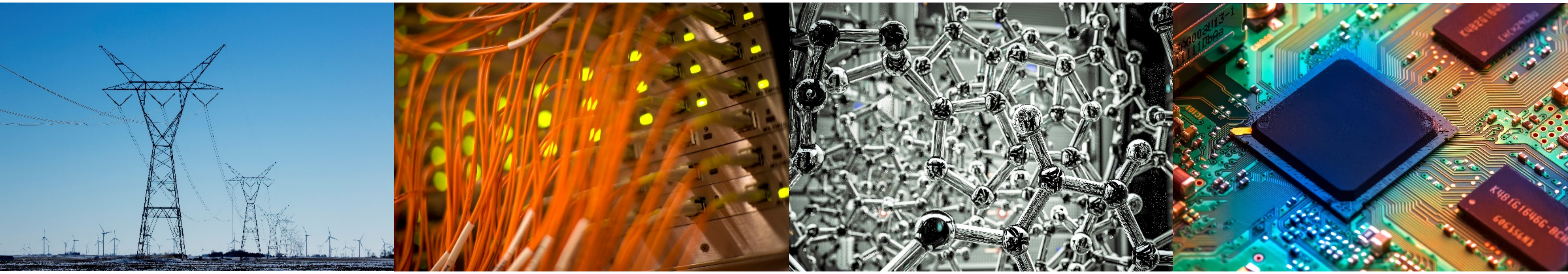
Fault-Injection on a Haptic Rendering Algorithm in the Raven Surgical Robot

Keywhan Chung, Xiao Li, **Zbigniew T. Kalbarczyk**, Ravishankar K. Iyer, and Thenkurussi Kesavadas
University of Illinois at Urbana-Champaign



ILLINOIS
Electrical & Computer Engineering
COLLEGE OF ENGINEERING

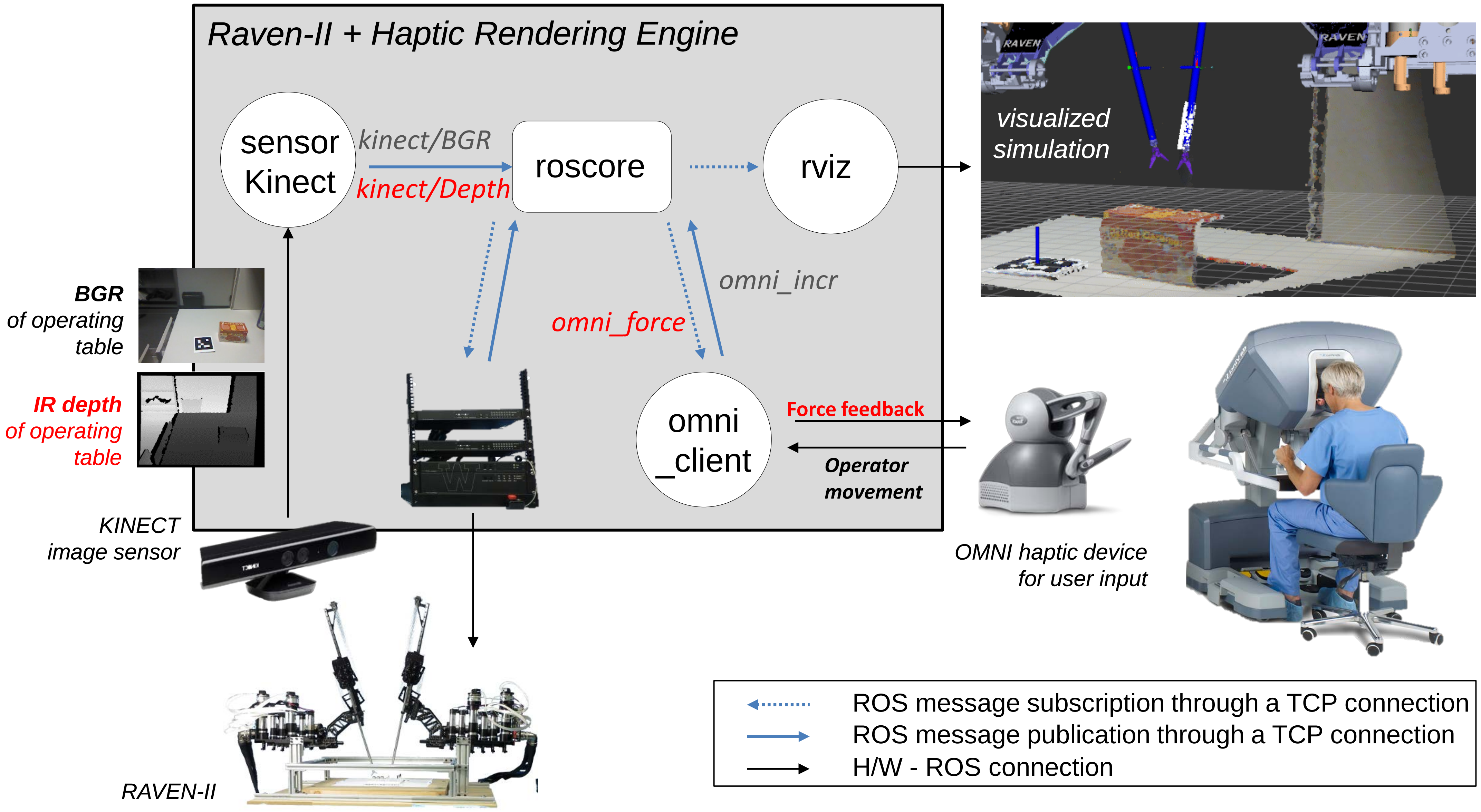
Introduction and Motivation

- Surgical robot widely adopted in medicine
- **Raven-II**: an open-architecture surgical robot
  - Built for research purposes
  - Based on open standards (Linux, ROS)
  - Hence, easy to add/upgrade/swap components and advance relevant technologies
- How about **reliability** and **security**?
  - What problems can the robot and its modules face?
  - How robust against them?
  - Any potential security threats?

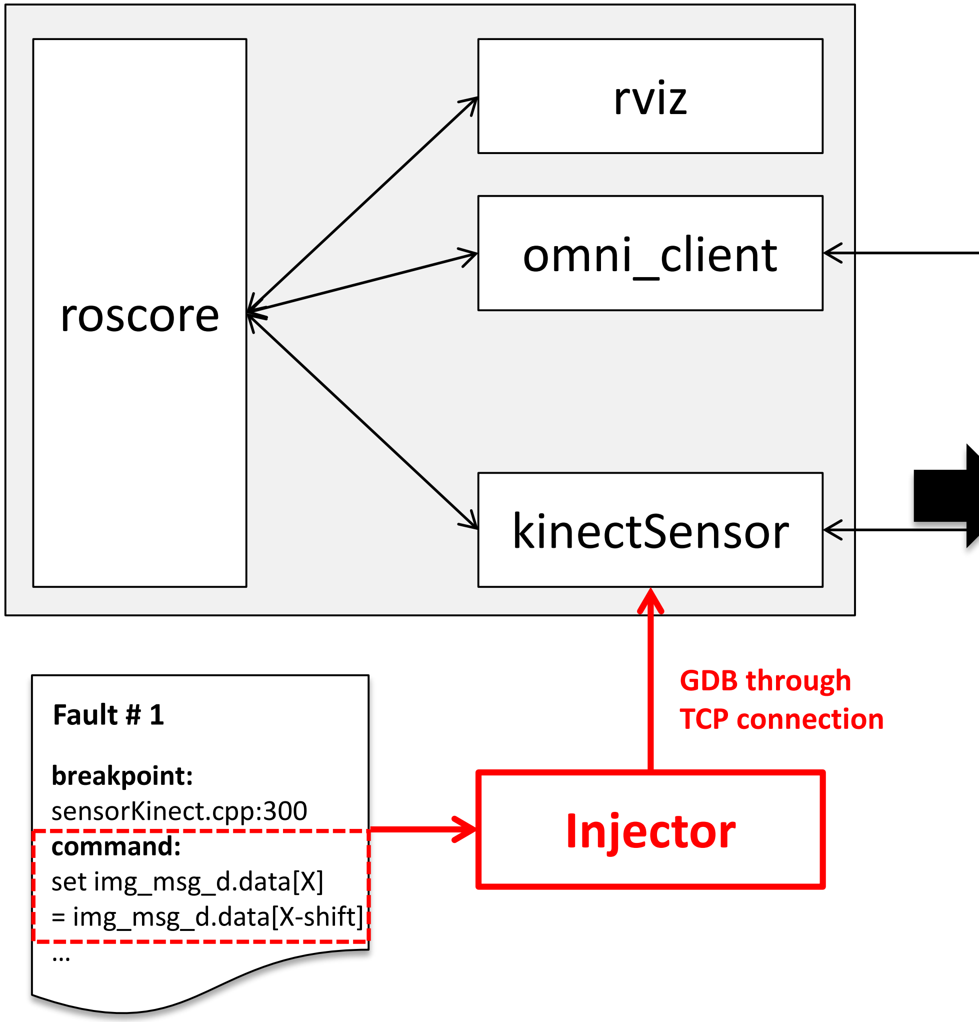
Evaluation through fault injection

- Sample application: Haptic Rendering Module designed for Raven-II
- Algorithm heavily rely on data from the image sensor
- Inject faults into the message from “image sensor node” to “control algorithm”

Environment Setup of Raven with the Haptic Rendering Engine



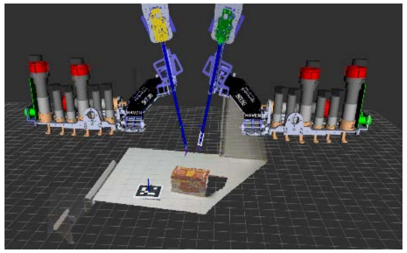
Fault Injection Setup

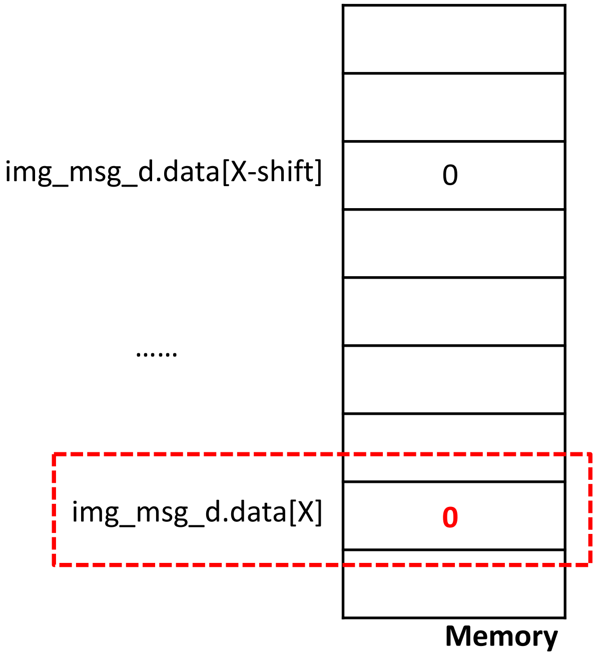


Fault # 1

breakpoint:
sensorKinect.cpp:300

command:
set img_msg_d.data[X]
= img_msg_d.data[X-shift]
...





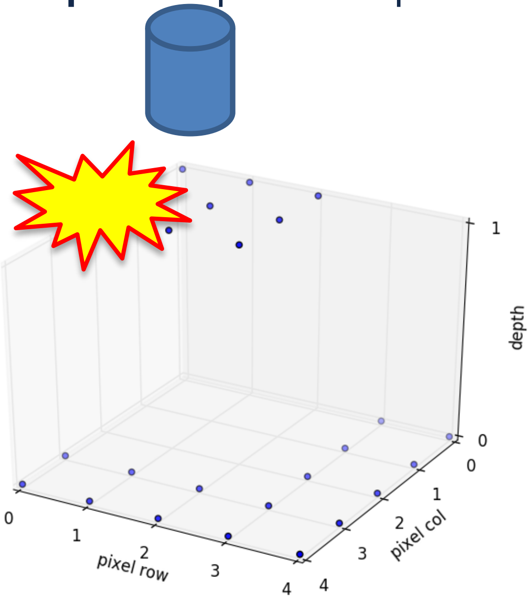
sensorKinect.cpp

```
...
while (ros::ok()){
   cap.grap(); // capture a frame from the image sensor
   cap.retrieve(bgrImage, CV_CAP_OPENNI_BGR_IMAGE);
   cap.retrieve(bgrImage, CV_CAP_OPENNI_DEPTH_MAP);
   ...
   // convert the image to a ROS message
   img_bridge_d.toImageMsg(img_msg_d);   breakpoint
   ...
   // publish the ROS message to roscore
   pub2.publish(img_msg_d);
   ...
}
...
```

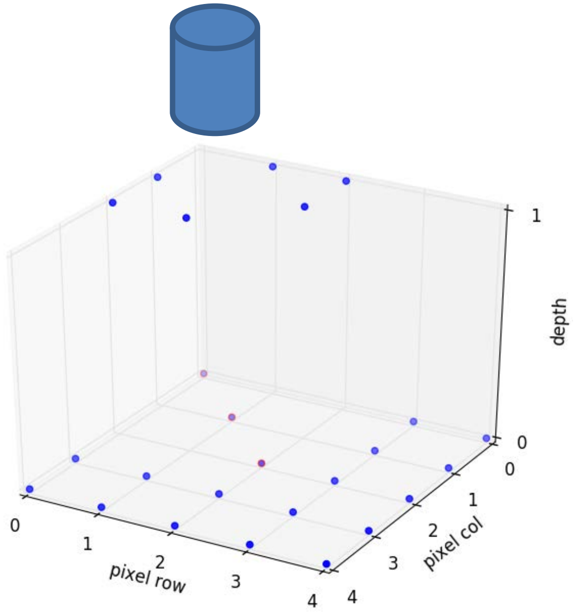
Fault I: Loss of Granularity in Depth Map

- **Reliability Issue**: Message can loose information during transition: e.g., hardware failure, network problem, etc.
- Leads to loss of granularity
- **Fault Model:**
  - **Neutralize the depth** of a portion of pixels chosen at random

1	1	1	0	0
1	1	1	0	0
1	1	1	0	0
0	0	0	0	0
0	0	0	0	0



Original Depth Map
(ground truth)



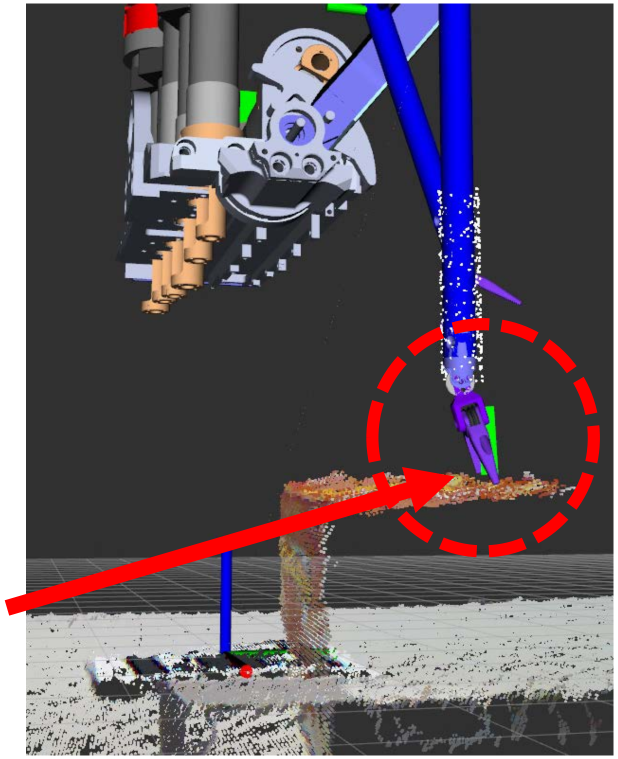
0	1	1	0	0
1	0	1	0	0
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

Corrupted Depth Map
(actual input for algorithm)

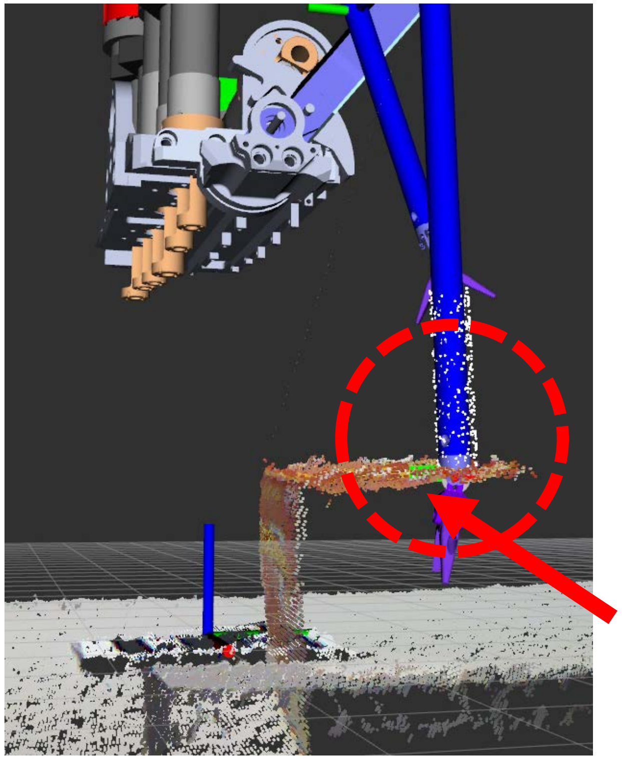
Fault I: Injection Result

Tip contact successfully rendered & blocked penetration

Rendered force feedback



without Fault Injected



with Fault Injected

Robot arm penetrated the object

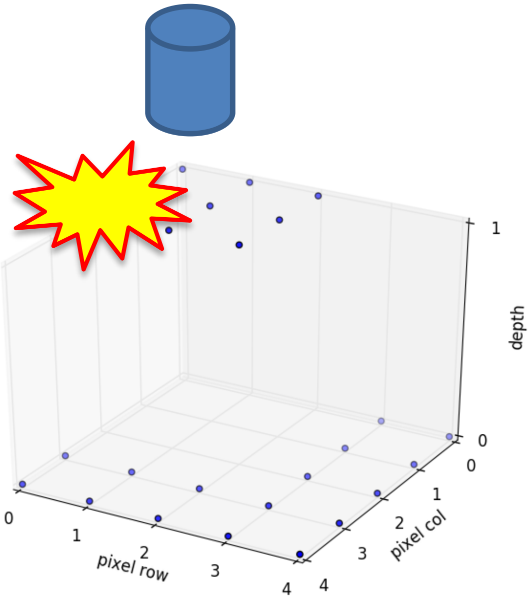
Rendered force feedback (horizontal not vertical)

- In reality: no blockage at surface
  - Damage underlying surface (e.g., patient tissue)
  - Robot suffers a heavy load without notice

Fault II: Shifted Depth Map

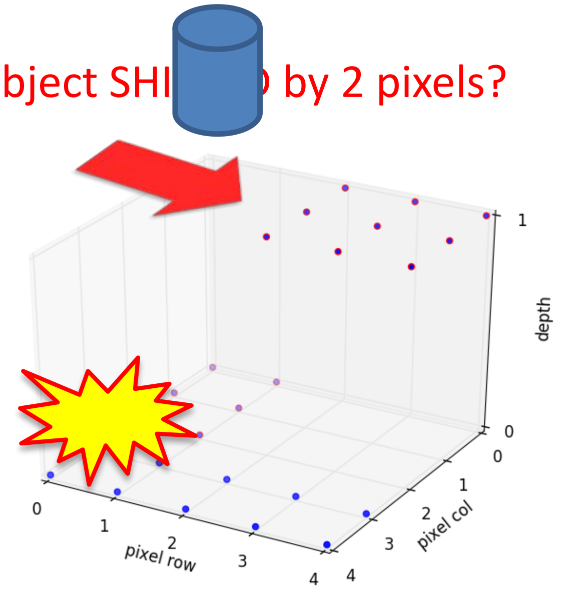
- **Security Threat**: Attacker can manipulate the message w/ malicious intent
- **Fault Model:**
  - **Shift the memory contents** as if the object has moved

1	1	1	0	0
1	1	1	0	0
1	1	1	0	0
0	0	0	0	0
0	0	0	0	0



Original Depth Map
(ground truth)

Object SHIFTED by 2 pixels?

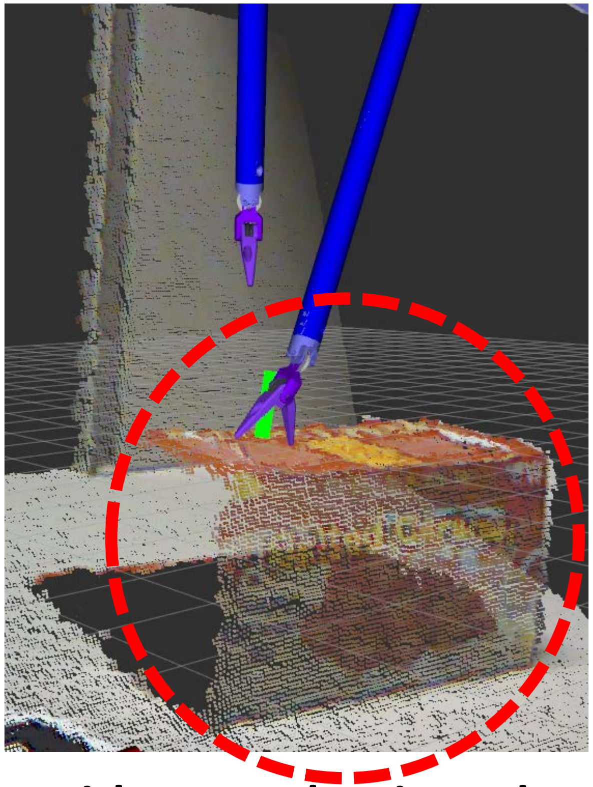


0	0	1	1	1
0	0	1	1	1
0	0	1	1	1
0	0	0	0	0
0	0	0	0	0

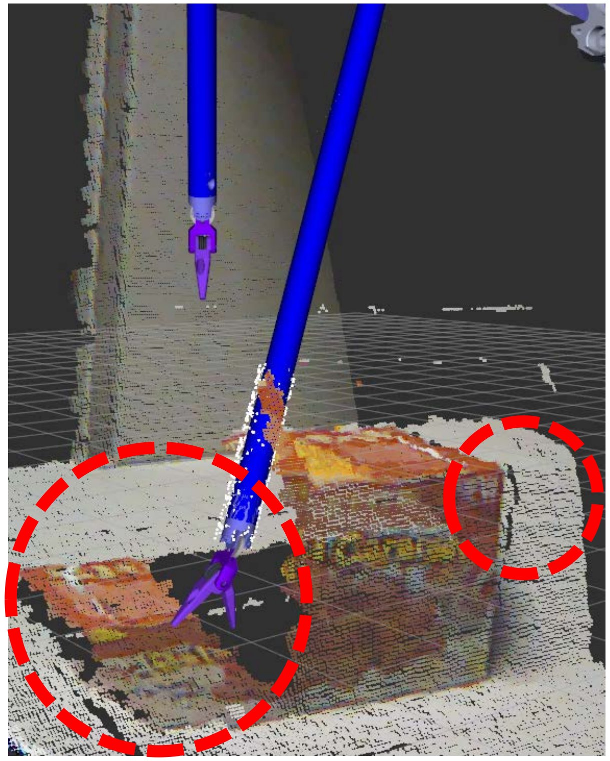
Shifted Depth Map
(actual input for algorithm)

Fault II: Injection Result

Object under operation rendered in 3D while operating table remaining flat



without Fault Injected



with Fault Injected

Volume added to table surface

Object under operation flattened (same depth level as table)

- If we also corrupted the BGR message,
  can obfuscate the operator to think that the object is in **a different location**

Conclusion

- Using fault injection,
  demonstrated possibility of neutralizing a haptic feedback engine:
  - Reliability Issue: hardware failure in image source or network issue
  - Security Threat: intentional manipulation of input data
- Need validation of input source and detection of corruption
- Future Work:
  - Advances in fault models
  - Additional source of faults (e.g., corrupt the “omni_force” message)
  - Vulnerabilities in applications and the ROS framework
  - Protection against known faults