Fault Localization in Embedded Control System Software

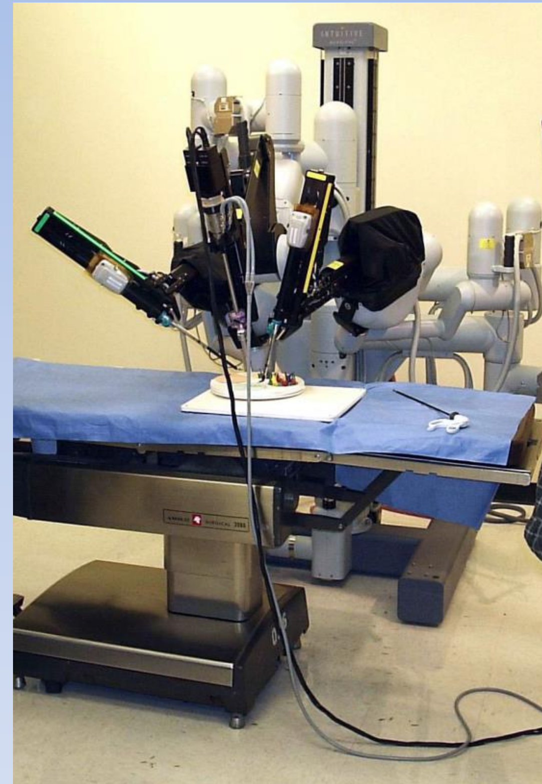
Kai Liang, Zhuofu Bai, Cenk Cavusoglu, Andy Podgurski,
Soumya Ray
Electrical Engineering and Computer Science
Case Western Reserve University, USA

Motivation

- Embedded control systems are ubiquitous
  - E.g. modern cars can have numerous systems controlling the automatic transmission, antilock brakes, airbags etc.
    - "The wheels are primarily there to keep the computers from dragging on the ground."—Paul Saffo
- Many such systems are used in safety critical situations
- When failures are observed, it is crucial to promptly locate and remove the faults that caused them

Robotic Surgery Systems

- Our domain of interest
- Cyber-physical systems that aid surgical procedures
- Benefits: less pain, shorter recovery time, minimize side effects
- Very complex control software being used in a highly uncertain, safety-critical environment



da Vinci surgical robot

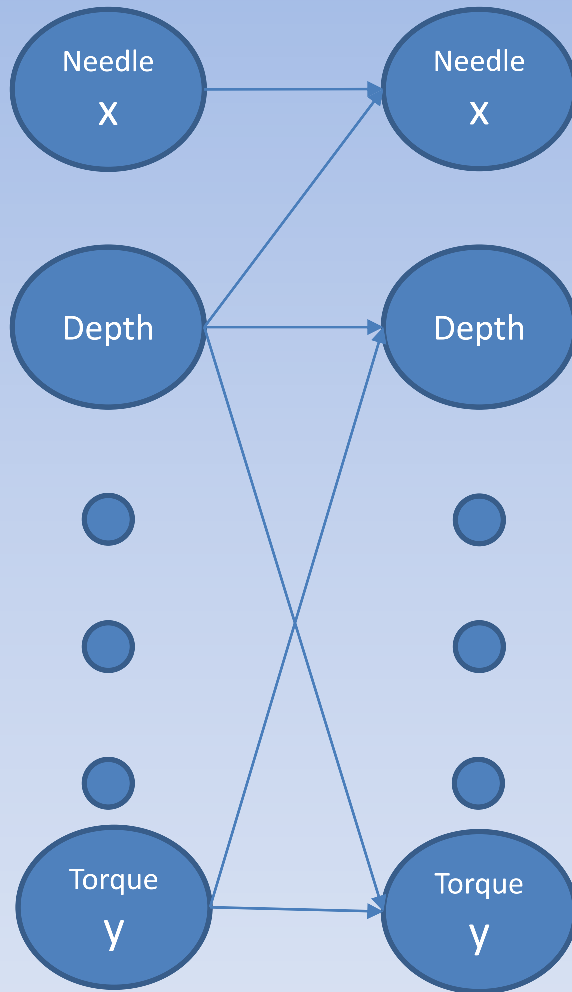
Our Approach

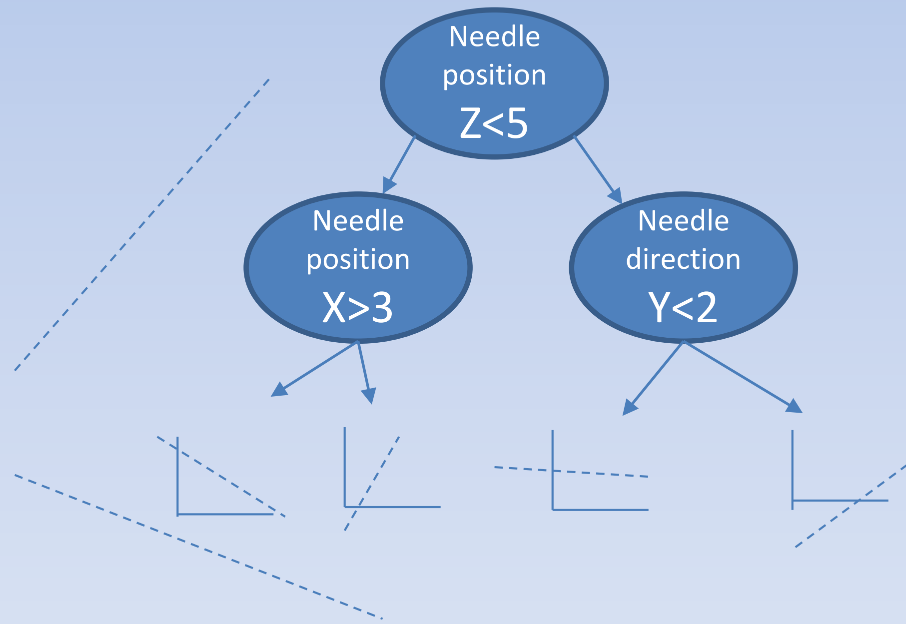
- Develop statistical models of normal behavior of such systems using simulators
  - Often built to test controllers without putting them on expensive hardware
- Identify variables responsible for "adverse and anomalous" (A&A) events when system operates
- Trace these variables through control code to determine faulty statements
  - Challenge: controllers are intricate mathematical code

Step 1: Build "Normal" Model

- Dynamic Bayesian Network (DBN)
- Conditional probability functions are regression trees with linear Gaussian models at leaves
  - General representation of nonlinear dynamics
- Structure and parameters learned from data (trajectories generated from simulator)
- Added feature selection to sparsify the model
- Prior work (IAAI13) shows these are able to adequately represent normal behavior and detect A&A events

DBN Example





$$\Pr(s_{t+1} \mid s_t) = \Pr(V_{t+1} \mid V_t) = \prod_{i=1}^{n} \Pr(V_{t+1}^{i} \mid V_t^{par(i)})$$

Step 2: Identify Variables Causing A&A Events

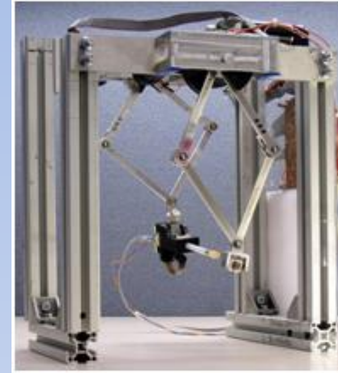
- An A&A event corresponds to a low-likelihood state according to our DBN
- A low-likelihood state must mean that some state variables have low likelihood
- We loop through each variable and check each likelihood against a range of normal likelihoods obtained from the training data
  - If outside this range, mark this variable as “bad”
  - In this step, the values of the variables are considered (unlike coverage-based fault localization)

Step 3: Identify Suspicious Statements

- In the control code, find the statements that define the "bad" variables
- Using the controller's PDG, rank statements so that the nearest common ancestor to all those statements has a high rank
  - (i.e. are most suspicious)
- Idea: If faults are rare (assumed), the nearest common ancestor could be a "common cause" for all the "bad" variables seen

Testbeds: Two RoS Systems

- Small Animal Biopsy Robot (SABiR)
  - Inject drugs/perform biopsies on live small animal targets with high accuracy
- Beating Heart Robot (BHR)
  - Needle tracking heart motion for robotic cardiovascular procedures
- Simulation implemented in MATLAB/Simulink for both the robots



Methodology

- We obtain 10 faulty controllers for each robot
  - 1 real fault, 9 mutation faults for SABiR
  - 10 mutation faults for BHR
- Baselines:
  - Two coverage based strategies (PFiC and Ochiai)
  - One value-based strategy (Elastic Predicates/ESP)
- All methods output ranked list of statements according to suspiciousness
  - We report the rank of the true faulty statement in this list

Fault Localization Results

SABiR	1 (Real)	2	3	4	5	6	7	8	9	10
FLECS	**39**	**3**	**3**	**2**	**28**	**27**	65	23	**10**	**3**
ESP	145	6	253	29	29	24	**29**	**21**	273	24
PFiC	166	166	162	162	162	163	166	162	163	166
Ochiai	163	163	163	163	162	163	166	162	163	166

BHR	1	2	3	4	5	6	7	8	9	10
FLECS	**1**	**18**	**2**	**2**	**18**	**18**	**1**	9	**2**	**10**
ESP	25	47	4	51	54	27	4	**3**	114	29
PFiC	84	84	84	84	84	84	84	84	84	84
Ochiai	84	84	84	84	84	84	84	84	84	84

Conclusion and Limitations

- Our approach is specific to controller code
  - Takes advantage of available simulators
  - Tracks variable values needed for localization
  - Uses the iterative calls to the code to help analysis
  - Does well on this kind of code relative to baselines
- Limitations
  - Assumptions and heuristics may not hold in all cases
  - Results are affected by granularity of instrumentation
  - Using the ranked list output likely does not reflect possible real usage scenarios