

# REQUIREMENTS AND RISKS

## II: PLANNING FOR MISTAKES

Eunsuk Kang

# **LEARNING GOALS:**

- Evaluate the risks of mistakes from ML components using the fault tree analysis (FTA)
- Design strategies for mitigating the risks of failures due to AI mistakes

# RISK ANALYSIS

# **WHAT IS RISK ANALYSIS?**

# **WHAT IS RISK ANALYSIS?**

- What can possibly go wrong in my system, and what are potential impacts on system requirements?

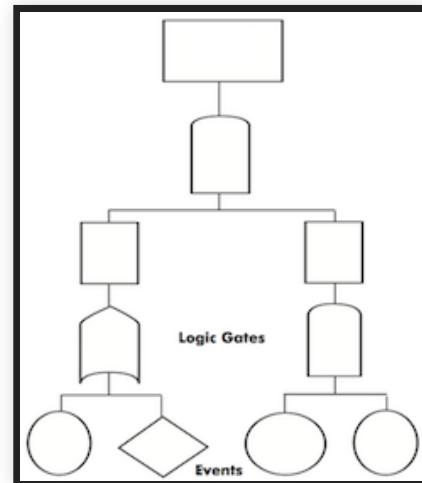
# WHAT IS RISK ANALYSIS?

- What can possibly go wrong in my system, and what are potential impacts on system requirements?
- Risk = Likelihood \* Impact

# WHAT IS RISK ANALYSIS?

- What can possibly go wrong in my system, and what are potential impacts on system requirements?
- Risk = Likelihood \* Impact
- A number of methods:
  - Failure mode & effects analysis (FMEA)
  - Hazard analysis
  - Why-because analysis
  - Fault tree analysis (FTA) <= Today's focus!
  - ...

# FAULT TREE ANALYSIS (FTA)



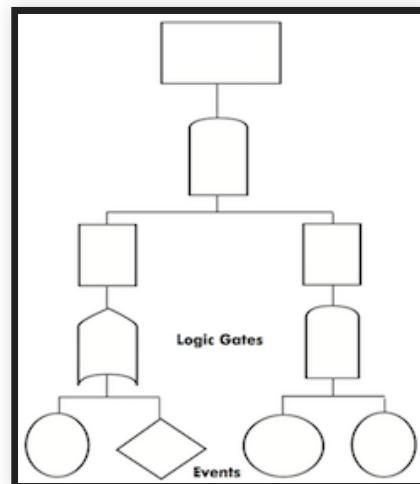
# FAULT TREE ANALYSIS (FTA)

- Fault tree: A top-down diagram that displays the relationships between a system failure (i.e., requirement violation) and its potential causes.
  - Identify sequences of events that result in a failure
  - Prioritize the contributors leading to the failure
  - Inform decisions about how to (re-)design the system
  - Investigate an accident & identify the root cause



# FAULT TREE ANALYSIS (FTA)

- Fault tree: A top-down diagram that displays the relationships between a system failure (i.e., requirement violation) and its potential causes.
  - Identify sequences of events that result in a failure
  - Prioritize the contributors leading to the failure
  - Inform decisions about how to (re-)design the system
  - Investigate an accident & identify the root cause
- Often used for safety & reliability, but can also be used for other types of requirements (e.g., poor performance, security attacks...)



# FAULT TREE ANALYSIS & ML

- ML is increasingly used in safety-critical domains such as automotive, aeronautics, industrial control systems, etc.,

# FAULT TREE ANALYSIS & ML

- ML is increasingly used in safety-critical domains such as automotive, aeronautics, industrial control systems, etc.,
- ML models are just one part of the system

# FAULT TREE ANALYSIS & ML

- ML is increasingly used in safety-critical domains such as automotive, aeronautics, industrial control systems, etc.,
- ML models are just one part of the system
- ML models will EVENTUALLY make mistakes
  - Output wrong predictions/values
  - Fail to adapt to the changing environment
  - Confuse users, etc.,

# FAULT TREE ANALYSIS & ML

- ML is increasingly used in safety-critical domains such as automotive, aeronautics, industrial control systems, etc.,
- ML models are just one part of the system
- ML models will EVENTUALLY make mistakes
  - Output wrong predictions/values
  - Fail to adapt to the changing environment
  - Confuse users, etc.,
- How do mistakes made by ML contribute to system failures? How do we ensure their mistakes do not result in a catastrophic outcome?

# FAULT TREES: BASIC BUILDING BLOCKS



Figure from *Fault Tree Analysis and Reliability Block Diagram* (2016), Jaroslav Menčík.

# FAULT TREES: BASIC BUILDING BLOCKS



- Event: An occurrence of a fault or an undesirable action
  - (Intermediate) Event: Explained in terms of other events
  - Basic Event: No further development or breakdown; leafs of the tree

Figure from *Fault Tree Analysis and Reliability Block Diagram* (2016), Jaroslav Menčík.

# FAULT TREES: BASIC BUILDING BLOCKS



- Event: An occurrence of a fault or an undesirable action
  - (Intermediate) Event: Explained in terms of other events
  - Basic Event: No further development or breakdown; leafs of the tree
- Gate: Logical relationship between an event & its immediate subevents
  - AND: All of the sub-events must take place
  - OR: Any one of the sub-events may result in the parent event

Figure from *Fault Tree Analysis and Reliability Block Diagram* (2016), Jaroslav Menčík.

# FAULT TREE EXAMPLE



Figure from *Fault Tree Analysis and Reliability Block Diagram* (2016), Jaroslav Menčík.

# FAULT TREE EXAMPLE



- Every tree begins with a TOP event (typically a violation of a requirement)

Figure from *Fault Tree Analysis and Reliability Block Diagram* (2016), Jaroslav Menčík.

# FAULT TREE EXAMPLE



- Every tree begins with a TOP event (typically a violation of a requirement)
- Every branch of the tree must terminate with a basic event

Figure from *Fault Tree Analysis and Reliability Block Diagram* (2016), Jaroslav Menčík.

# ANALYSIS

- What can we do with fault trees?
  - Qualitative analysis: Determine potential root causes of a failure through *minimal cut set analysis*
  - Quantitative analysis: Compute the probability of a failure

# MINIMAL CUT SET ANALYSIS



- Cut set: A set of basic events whose simultaneous occurrence is sufficient to guarantee that the TOP event occurs.
- *Minimal* cut set: A cut set from which a smaller cut set can't be obtained by removing a basic event.
- Q. What are minimal cut sets in the above tree?

# FAILURE PROBABILITY ANALYSIS

# FAILURE PROBABILITY ANALYSIS

- To compute the probability of the top event:
  - Assign probabilities to basic events (based on domain knowledge)
  - Apply probability theory to compute probabilities of intermediate events through AND & OR gates
  - (Alternatively, as sum of prob. of minimal cut sets)

# FAILURE PROBABILITY ANALYSIS

- To compute the probability of the top event:
  - Assign probabilities to basic events (based on domain knowledge)
  - Apply probability theory to compute probabilities of intermediate events through AND & OR gates
  - (Alternatively, as sum of prob. of minimal cut sets)
- In this class, we won't ask you to do this.
  - Why is this especially challenging for software?

# FTA PROCESS

# FTA PROCESS

## 1. Specify the system structure

- Environment entities & machine components
- Assumptions (ENV) & specifications (SPEC)

# FTA PROCESS

1. Specify the system structure
  - Environment entities & machine components
  - Assumptions (ENV) & specifications (SPEC)
2. Identify the top event as a requirement violation (REQ)

# FTA PROCESS

1. Specify the system structure
  - Environment entities & machine components
  - Assumptions (ENV) & specifications (SPEC)
2. Identify the top event as a requirement violation (REQ)
3. Construct the fault tree
  - Derive intermediate events from a violation of ENV or SPEC
  - Decompose the intermediate events further down based on the knowledge of the domain or components

# FTA PROCESS

## 1. Specify the system structure

- Environment entities & machine components
- Assumptions (ENV) & specifications (SPEC)

## 2. Identify the top event as a requirement violation (REQ)

## 3. Construct the fault tree

- Derive intermediate events from a violation of ENV or SPEC
- Decompose the intermediate events further down based on the knowledge of the domain or components

## 4. Analyze the tree

- Identify all possible minimal cut sets

# FTA PROCESS

## 1. Specify the system structure

- Environment entities & machine components
- Assumptions (ENV) & specifications (SPEC)

## 2. Identify the top event as a requirement violation (REQ)

## 3. Construct the fault tree

- Derive intermediate events from a violation of ENV or SPEC
- Decompose the intermediate events further down based on the knowledge of the domain or components

## 4. Analyze the tree

- Identify all possible minimal cut sets

## 5. Consider design modifications

- Eliminate certain cutsets, or
- Increase the size of min cutsets

# FTA PROCESS

## 1. Specify the system structure

- Environment entities & machine components
- Assumptions (ENV) & specifications (SPEC)

## 2. Identify the top event as a requirement violation (REQ)

## 3. Construct the fault tree

- Derive intermediate events from a violation of ENV or SPEC
- Decompose the intermediate events further down based on the knowledge of the domain or components

## 4. Analyze the tree

- Identify all possible minimal cut sets

## 5. Consider design modifications

- Eliminate certain cutsets, or
- Increase the size of min cutsets

## 6. Repeat

# EXAMPLE: BACK TO LANE ASSIST



- REQ: The vehicle must be prevented from veering off the lane.
- SPEC: Lane detector accurately identifies lane markings in the input image; the controller generates correct steering commands
- ENV: Sensors are providing accurate information about the lane; driver responses when given warning; steering wheel is functional

# BREAKOUT: FTA FOR LANE ASSIST



- Draw a fault tree for the lane assist system with the top event as “Vehicle fails to stay within the lane”
- Draw on paper, scan & upload into Slack #lecture
- Or use the Google Slide template provided; make your own copy and paste the link into Slack

# EXAMPLE: FTA FOR LANE ASSIST



# FTA: CAVEATS

- In general, building a **complete** tree is impossible
  - There are probably some faulty events that you missed
  - "Unknown unknowns"
- Domain knowledge is crucial for improving coverage
  - Talk to domain experts; augment your tree as you learn more
- FTA is still very valuable for risk reduction!
  - Forces you to think about & explicitly document possible failure scenarios
  - A good starting basis for designing mitigations

# **STRATEGIES FOR HANDLING FAULTS IN ML- BASED SYSTEMS**

# ELEMENTS OF FAULT-TOLERANT DESIGN

# ELEMENTS OF FAULT-TOLERANT DESIGN

- Assume that:
  - Software/ML components will make mistakes at some point
  - Environment evolves, violating some of its assumptions

# ELEMENTS OF FAULT-TOLERANT DESIGN

- Assume that:
  - Software/ML components will make mistakes at some point
  - Environment evolves, violating some of its assumptions
- Goal: Minimize the impact of mistakes/violations on the overall system

# ELEMENTS OF FAULT-TOLERANT DESIGN

- Assume that:
  - Software/ML components will make mistakes at some point
  - Environment evolves, violating some of its assumptions
- Goal: Minimize the impact of mistakes/violations on the overall system
- Detection
  - Monitoring
  - Redundancy

# ELEMENTS OF FAULT-TOLERANT DESIGN

- Assume that:
  - Software/ML components will make mistakes at some point
  - Environment evolves, violating some of its assumptions
- Goal: Minimize the impact of mistakes/violations on the overall system
- Detection
  - Monitoring
  - Redundancy
- Response
  - Graceful degradation (fail-safe)
  - Redundancy (fail over)
  - Human in the loop
  - Undoable actions

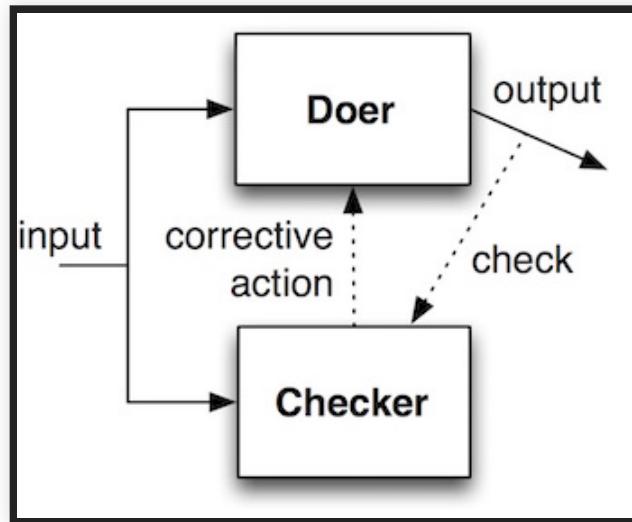
# ELEMENTS OF FAULT-TOLERANT DESIGN

- Assume that:
  - Software/ML components will make mistakes at some point
  - Environment evolves, violating some of its assumptions
- Goal: Minimize the impact of mistakes/violations on the overall system
- Detection
  - Monitoring
  - Redundancy
- Response
  - Graceful degradation (fail-safe)
  - Redundancy (fail over)
  - Human in the loop
  - Undoable actions
- Containment
  - Decoupling & isolation

# DETECTION: MONITORING

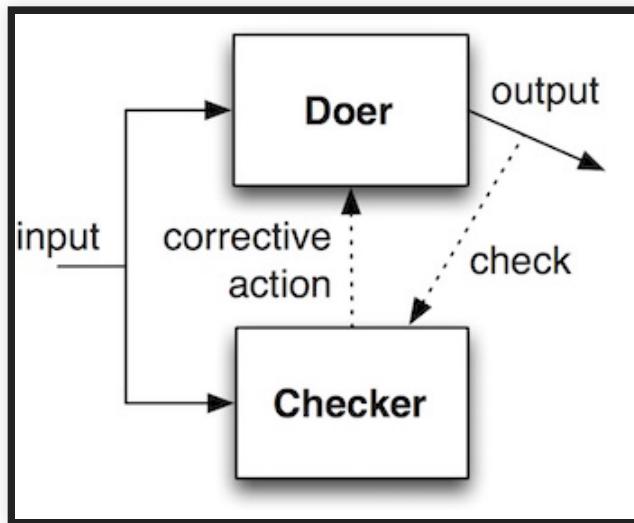


# DETECTION: MONITORING



- Goal: Detect when a component failure occurs

# DETECTION: MONITORING



- Goal: Detect when a component failure occurs
- Monitor: Periodically checks the output of a component for errors
  - Challenge: Need a way to recognize errors
  - e.g., corrupt sensor data, slow or missing response; low ML confidence

# DETECTION: MONITORING



- Goal: Detect when a component failure occurs
- Monitor: Periodically checks the output of a component for errors
  - Challenge: Need a way to recognize errors
  - e.g., corrupt sensor data, slow or missing response; low ML confidence
- Doer-Checker pattern
  - Doer: Perform primary function; untrusted and potentially faulty
  - Checker: If doer output is faulty, perform a corrective action (e.g., default safe output, shutdown); should be trustworthy

# DOER-CHECKER EXAMPLE: AUTONOMOUS VEHICLE



# DOER-CHECKER EXAMPLE: AUTONOMOUS VEHICLE



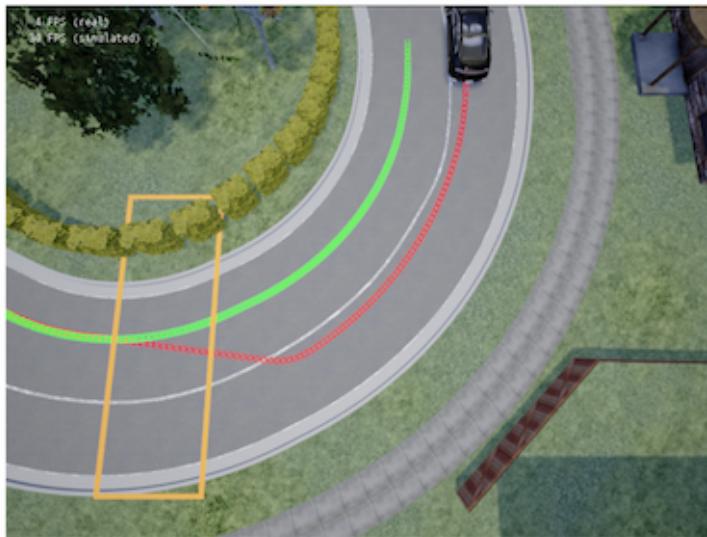
- ML-based controller (doer): Generate commands to steer the vehicle
  - Complex DNN; makes performance-optimal control decisions

# DOER-CHECKER EXAMPLE: AUTONOMOUS VEHICLE

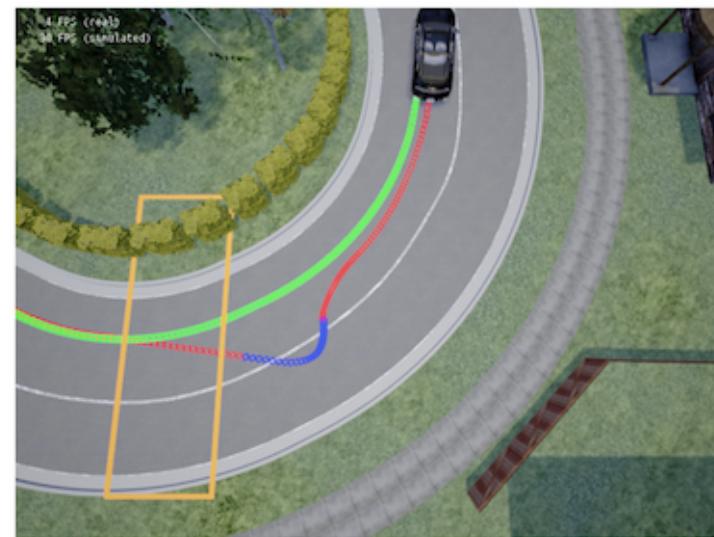


- ML-based controller (doer): Generate commands to steer the vehicle
  - Complex DNN; makes performance-optimal control decisions
- Safety controller (checker): Checks commands from ML controller; overrides it with a safe default command if the ML action is risky
  - Simpler, based on verifiable, transparent logic; conservative control

# DOER-CHECKER EXAMPLE: AUTONOMOUS VEHICLE



(a)



(b)



# DOER-CHECKER EXAMPLE: AUTONOMOUS VEHICLE



- Yellow region: Slippery road, causes loss of traction; unexpected by ML



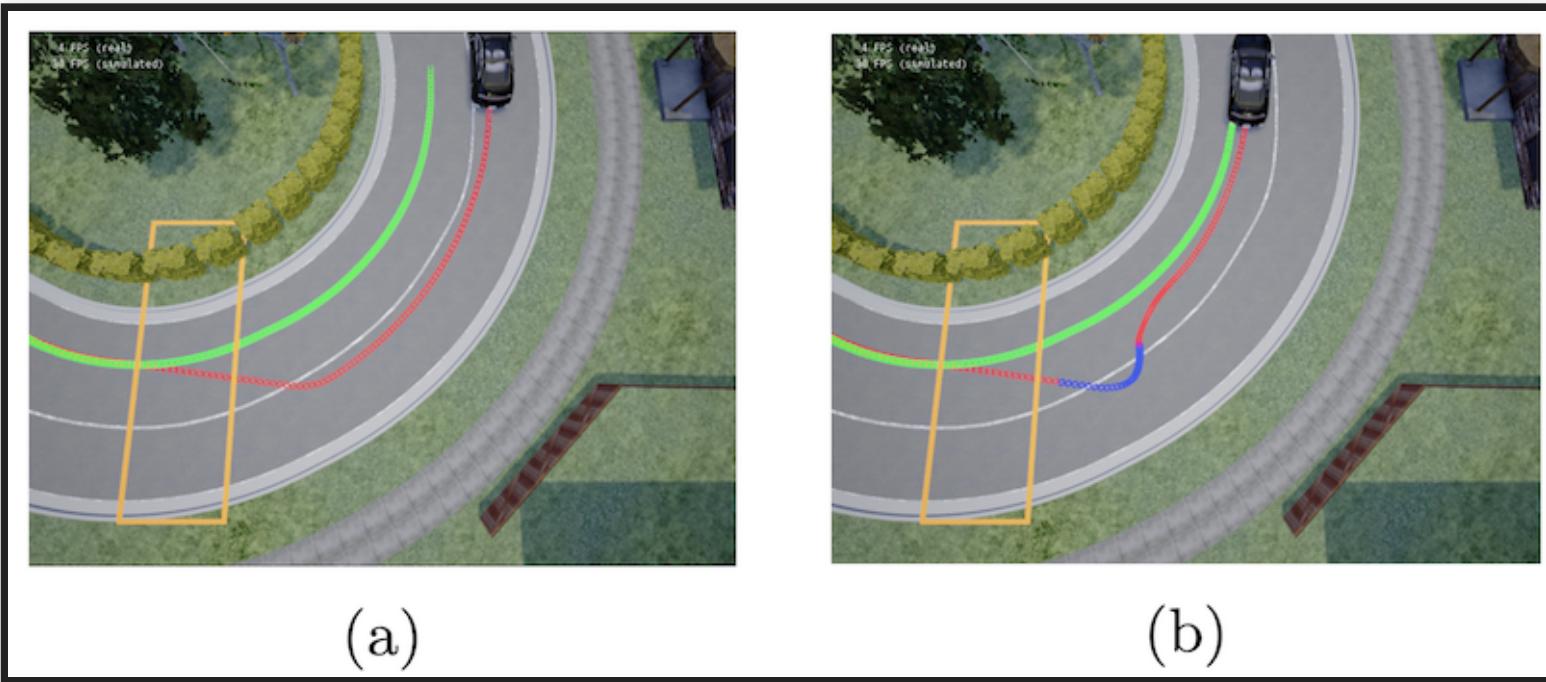
# DOER-CHECKER EXAMPLE: AUTONOMOUS VEHICLE



- Yellow region: Slippery road, causes loss of traction; unexpected by ML
- ML-based controller (doer): Model ignores traction loss; generates unsafe steering commands (a)



# DOER-CHECKER EXAMPLE: AUTONOMOUS VEHICLE



- Yellow region: Slippery road, causes loss of traction; unexpected by ML
  - ML-based controller (doer): Model ignores traction loss; generates unsafe steering commands (a)
  - Safety controller (checker): Overrides with safe steering commands (b)



# RESPONSE: GRACEFUL DEGRADATION (FAIL-SAFE)

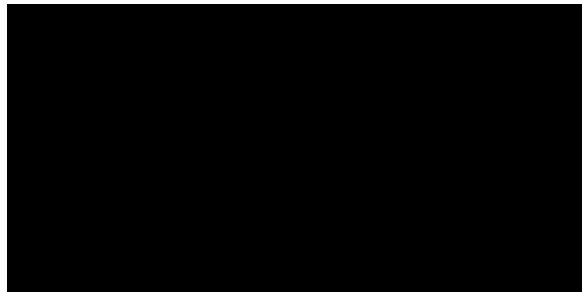


# RESPONSE: GRACEFUL DEGRADATION (FAIL-SAFE)



- Goal: When a component failure occurs, achieve system safety by reducing functionality and performance

# RESPONSE: GRACEFUL DEGRADATION (FAIL-SAFE)



- Goal: When a component failure occurs, achieve system safety by reducing functionality and performance
- Relies on a monitor to detect component failures

# RESPONSE: GRACEFUL DEGRADATION (FAIL-SAFE)

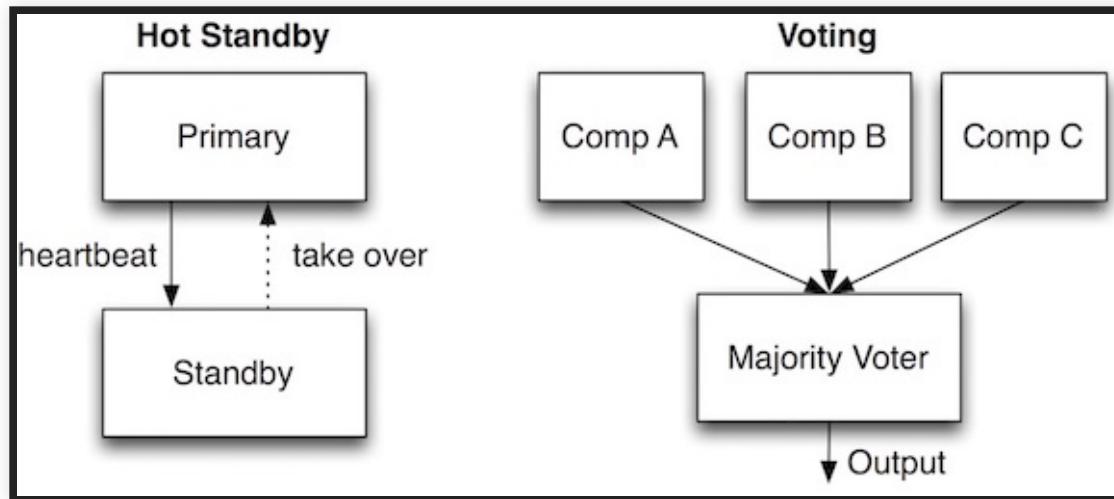


- Goal: When a component failure occurs, achieve system safety by reducing functionality and performance
- Relies on a monitor to detect component failures
- Example: Perception in autonomous vehicles
  - If Lidar fails, switch to a lower-quality detector & be more conservative about maintaining distance

# DETECTION & RESPONSE: REDUNDANCY

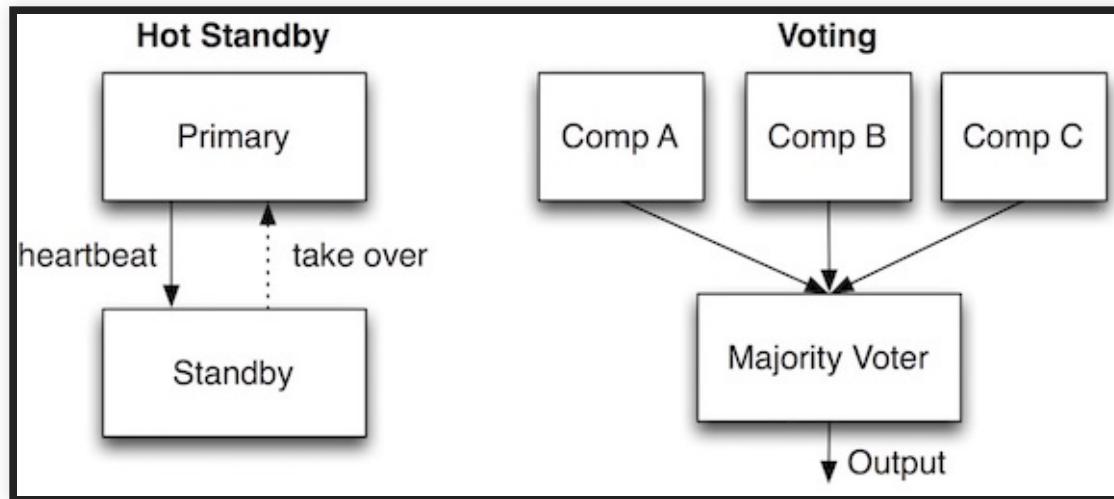


# DETECTION & RESPONSE: REDUNDANCY



- Detection: Compare output from redundant components

# DETECTION & RESPONSE: REDUNDANCY



- Detection: Compare output from redundant components
- Response: When a component fails, continue to provide the same functionality

# DETECTION & RESPONSE: REDUNDANCY



- Detection: Compare output from redundant components
- Response: When a component fails, continue to provide the same functionality
- Hot Standby: Standby watches & takes over when primary fails

# DETECTION & RESPONSE: REDUNDANCY



- Detection: Compare output from redundant components
- Response: When a component fails, continue to provide the same functionality
- Hot Standby: Standby watches & takes over when primary fails
- Voting: Select the majority decision

# DETECTION & RESPONSE: REDUNDANCY



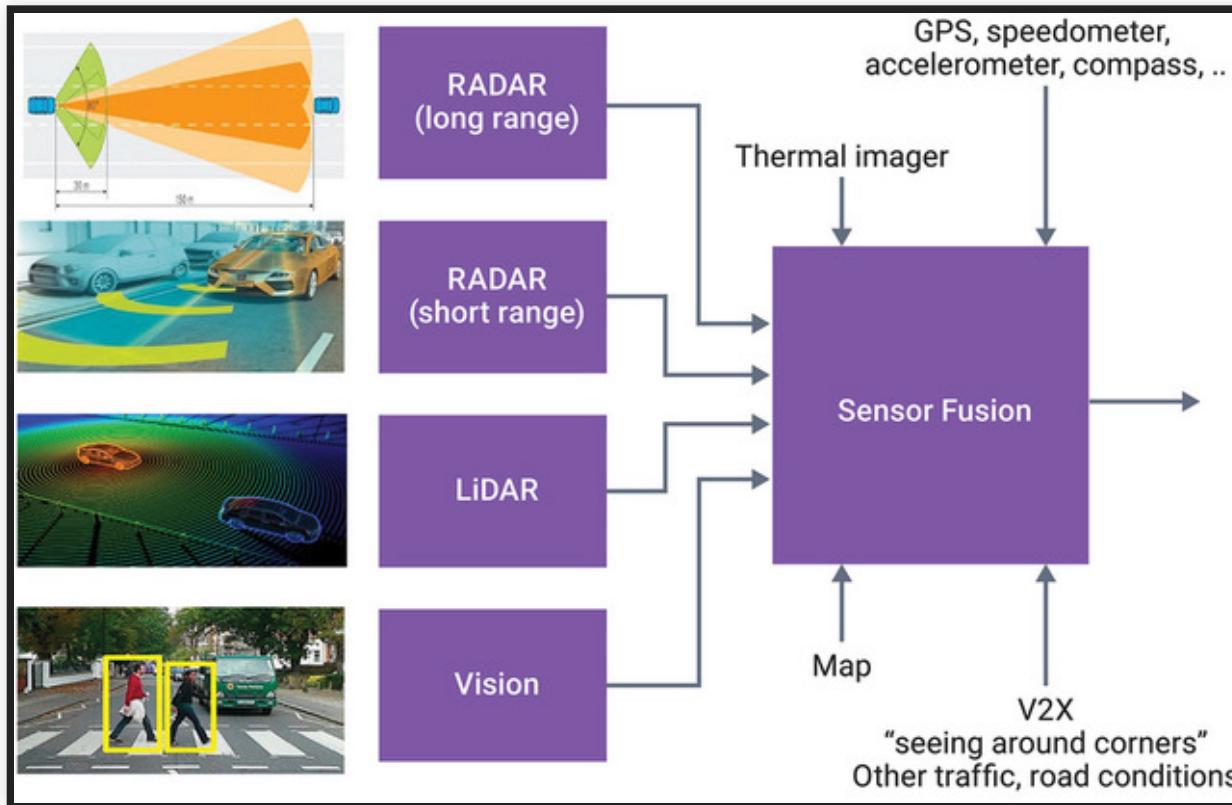
- Detection: Compare output from redundant components
- Response: When a component fails, continue to provide the same functionality
- Hot Standby: Standby watches & takes over when primary fails
- Voting: Select the majority decision
- Caution: Do components fail independently?
  - Reasonable assumption for hardware/mechanical failures
  - Q. What about ML components?

# REDUNDANCY EXAMPLE: ENSEMBLE LEARNING



- An example of redundancy by voting

# REDUNDANCY EXAMPLE: SENSOR FUSION



- Combine data from a wide range of sensors
- Provides partial information even when some sensor is faulty
- A critical part of modern self-driving vehicles

# RESPONSE: HUMAN IN THE LOOP

*Provide less forceful interaction, make suggestions, or ask for confirmation*

# RESPONSE: HUMAN IN THE LOOP

*Provide less forceful interaction, make suggestions, or ask for confirmation*

- AI and humans are good at predictions in different settings
  - AI better at statistics at scale and many factors
  - Humans understand context and data generation process; often better with thin data

# RESPONSE: HUMAN IN THE LOOP

*Provide less forceful interaction, make suggestions, or ask for confirmation*

- AI and humans are good at predictions in different settings
  - AI better at statistics at scale and many factors
  - Humans understand context and data generation process; often better with thin data
- AI for prediction, human for judgment?

# RESPONSE: HUMAN IN THE LOOP

*Provide less forceful interaction, make suggestions, or ask for confirmation*

- AI and humans are good at predictions in different settings
  - AI better at statistics at scale and many factors
  - Humans understand context and data generation process; often better with thin data
- AI for prediction, human for judgment?
- But be aware of:
  - Notification fatigue, complacency, just following predictions; see *Tesla autopilot*
  - Compliance/liability protection only?

# RESPONSE: HUMAN IN THE LOOP

*Provide less forceful interaction, make suggestions, or ask for confirmation*

- AI and humans are good at predictions in different settings
  - AI better at statistics at scale and many factors
  - Humans understand context and data generation process; often better with thin data
- AI for prediction, human for judgment?
- But be aware of:
  - Notification fatigue, complacency, just following predictions; see *Tesla autopilot*
  - Compliance/liability protection only?
- Deciding when and how to interact

# RESPONSE: HUMAN IN THE LOOP

*Provide less forceful interaction, make suggestions, or ask for confirmation*

- AI and humans are good at predictions in different settings
  - AI better at statistics at scale and many factors
  - Humans understand context and data generation process; often better with thin data
- AI for prediction, human for judgment?
- But be aware of:
  - Notification fatigue, complacency, just following predictions; see *Tesla autopilot*
  - Compliance/liability protection only?
- Deciding when and how to interact
- Lots of UI design and HCI problems

# RESPONSE: HUMAN IN THE LOOP

*Provide less forceful interaction, make suggestions, or ask for confirmation*

- AI and humans are good at predictions in different settings
  - AI better at statistics at scale and many factors
  - Humans understand context and data generation process; often better with thin data
- AI for prediction, human for judgment?
- But be aware of:
  - Notification fatigue, complacency, just following predictions; see *Tesla autopilot*
  - Compliance/liability protection only?
- Deciding when and how to interact
- Lots of UI design and HCI problems
- Q. Examples?

Speaker notes

Cancer prediction, sentencing + recidivism, Tesla autopilot, military "kill" decisions, powerpoint design suggestions

# RESPONSE: UNDOABLE ACTIONS

*Design the system to reduce the consequences of wrong predictions, allowing humans to override or undo*

Examples?

Speaker notes

Smart home devices, credit card applications, Powerpoint design suggestions

# EXAMPLE: LANE ASSIST



Possible mitigation strategies? Discuss with your neighbors

# EXAMPLE: FTA FOR LANE ASSIST



# MODIFIED FTA FOR LANE ASSIST



- Fault mitigation strategy: An additional sensor (infrared) for redundancy
  - Both sensors need to fail instead of just one
  - Reflected in the FTA as an additional basic event in the minimal cutset

# CONTAINMENT: DECOUPLING & ISOLATION

- **Design principle:** Faults in a low-critical (LC) components should not impact high-critical (HC) components

# POOR DECOUPLING: USS YORKTOWN (1997)



- Invalid data entered into DB; divide-by-zero crashes entire network
- Required rebooting the whole system; ship dead in water for 3 hours
- Lesson: Handle expected component faults; prevent propagation

# POOR DECOUPLING: AUTOMOTIVE SECURITY



- Main components connected through a common CAN bus
  - Broadcast; no access control (anyone can read/write)
- Can control brake/engine by playing a malicious MP3

*Experimental Security Analysis of a Modern Automobile, Koscher et al., (2010)*

# CONTAINMENT: DECOUPLING & ISOLATION

- **Design principle:** Faults in a low-critical (LC) components should not impact high-critical (HC) components

# CONTAINMENT: DECOUPLING & ISOLATION

- **Design principle:** Faults in a low-critical (LC) components should not impact high-critical (HC) components
- Apply the principle of least privilege
  - LC components should be allowed to access min. necessary functions

# CONTAINMENT: DECOUPLING & ISOLATION

- **Design principle:** Faults in a low-critical (LC) components should not impact high-critical (HC) components
- Apply the principle of least privilege
  - LC components should be allowed to access min. necessary functions
- Limit interactions across criticality boundaries
  - Deploy LC & HC components on different networks
  - Add monitors/checks at interfaces

# CONTAINMENT: DECOUPLING & ISOLATION

- **Design principle:** Faults in a low-critical (LC) components should not impact high-critical (HC) components
- Apply the principle of least privilege
  - LC components should be allowed to access min. necessary functions
- Limit interactions across criticality boundaries
  - Deploy LC & HC components on different networks
  - Add monitors/checks at interfaces
- Is an ML component in my system performing an LC or HC task?
  - If HC, can we "demote" it into LC?
  - Alternatively, if possible, replace/augment HC ML components with non-ML ones
  - Q. Examples?

# SUMMARY

- Accept that a failure is inevitable
  - ML components will eventually make mistakes
  - Environment may evolve over time, violating its assumptions
- Use risk analysis to identify and mitigate potential problems
- Design strategies for detecting and mitigating the risks from mistakes by ML