



Strong Software Reliability for Autonomous Space Robotics

Engineering The Future

Marie Farrell

March 28, 2023

Background: Me

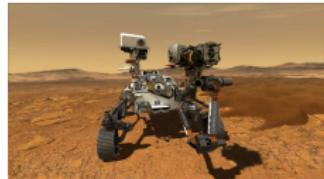
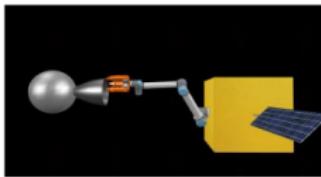
- PhD at Maynooth University: 2013–2017
- Postdoc at Universities of Liverpool and Manchester: 2018–2020 (FAIR-SPACE Hub)
- Postdoc at Maynooth University: 2020–2022 (VALU3S Project)
- Royal Academy of Engineering Research Fellow at The University of Manchester: 2022–2027



Engineering The Future: Moon Village



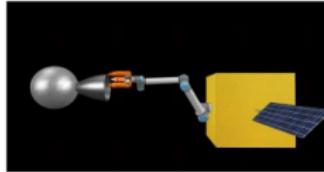
Strong Software Reliability for Autonomous Space Robotics



Problem:

- Space exploration necessitates the use of **autonomous** robotic systems.
- Current **verification** approaches are **not sufficient** to accurately specify the required autonomy.
- **Autonomy** introduces a unique set of concerns related to **verification** and **assurance**.

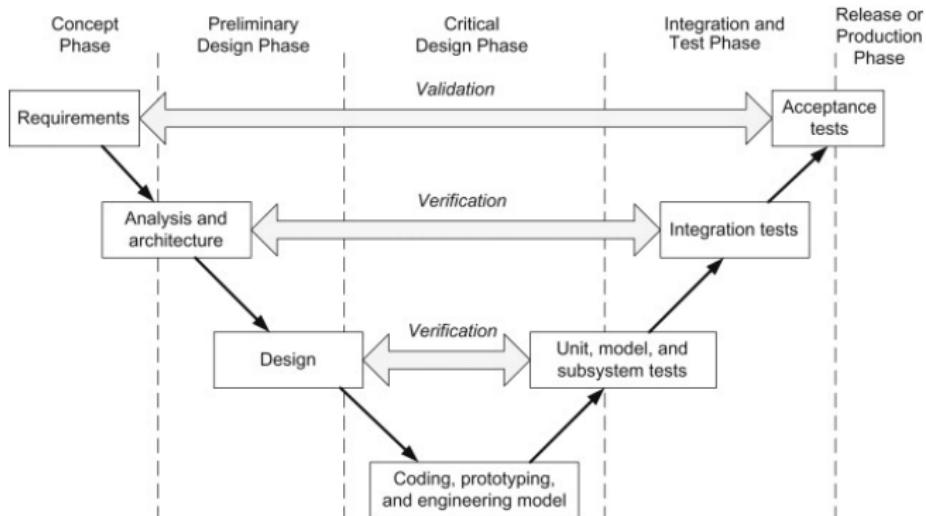
Strong Software Reliability for Autonomous Space Robotics



Aim:

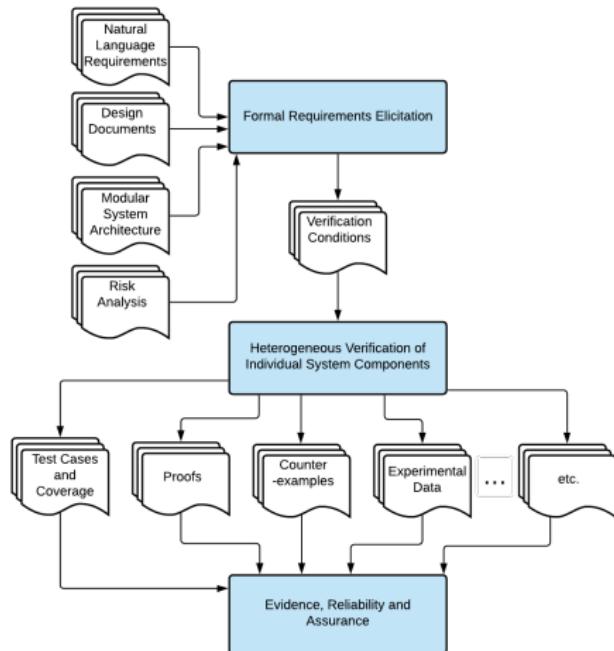
- Devise new ways of **describing, analysing** and **assuring** the correct autonomous behaviour of robotic space systems.

Software Development: V-Model



Integrating Multiple Heterogeneous Verification Techniques

- True autonomy requires a step change in **understanding** and **verification**.
- Issue of autonomous systems **assurance** remains unsolved.
- Adopt a sophisticated and **complementary** combination of robust V&V methods to support deployment in space.
- Provide formalisation of requirements for **Machine Learning** components.



Impact

This will ultimately lead to more **reliable**, more **usable** and more **effective** autonomous robots being deployed more **confidently** across a **range of sectors**.



But...

What Should I Verify?

Requirements Engineering



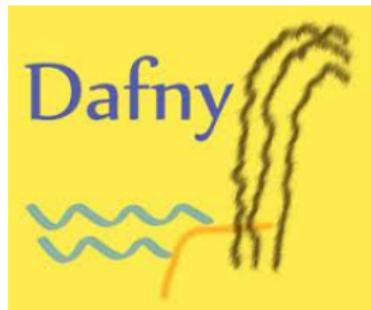
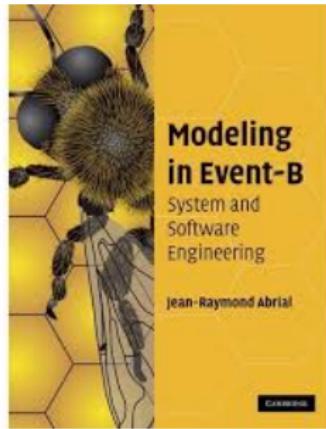
Illustration: The overall requirements engineering process

Moon Village



Requirement: Robots should always maintain a safe distance from astronauts.

Formal Methods



What Should I Verify?

Natural Language Requirements \neq Formal Properties

Formal Requirements Elicitation Tool (FRET)

The screenshot shows the FRET tool's main interface. On the left is a vertical toolbar with various icons. The central area features a large circular "Hierarchical Cluster" visualization containing several smaller green circles, with one labeled "FOILER". To the right of the cluster are five summary cards: "Total Projects 1", "Total Requirements 19", "Formalized Requirements 100.00 %", "System Components 7", and "Requirement Size 1528 bytes". Below these cards is a "Recent Activity" section listing ten FOL requirements (FOLRiver A001 through P003) along with their respective assumptions.

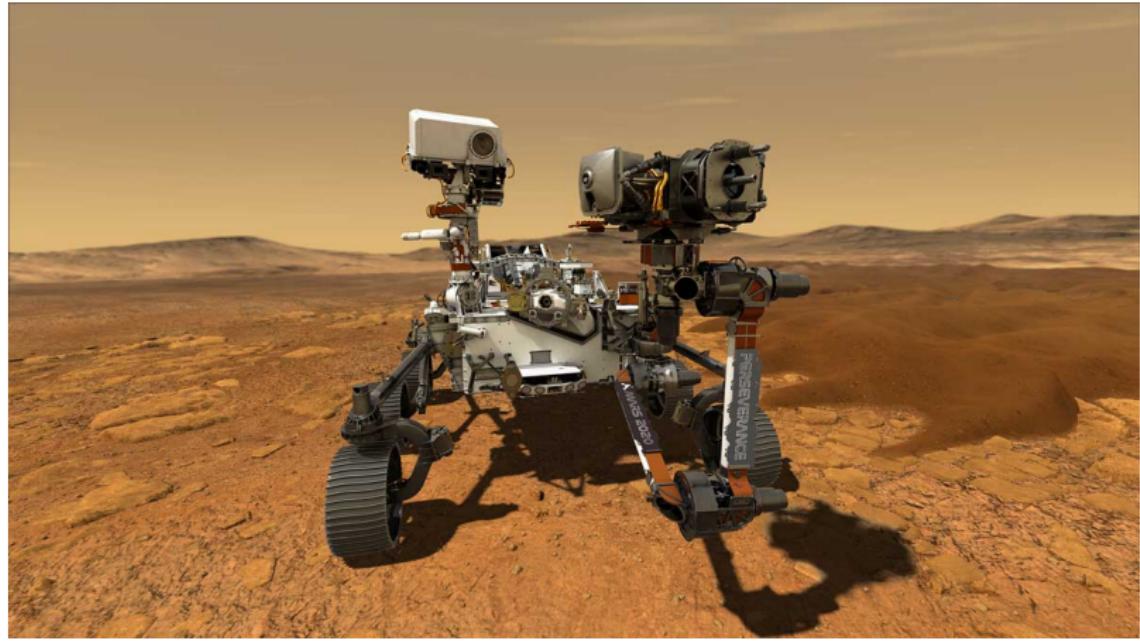
Requirement ID	Description	Assumptions
FOLRiver A001	when assumptions PlanAgent shall eventually satisfy $\alpha \Diamond \text{PlanAgent}$	
FOLRiver B002	when assumptions Interface shall eventually satisfy $(\text{Interface} \leftrightarrow \text{hasMoved}) \wedge (\text{position}_\text{river} = g) \wedge (\text{v} = g)$	
FOLRiver B003	when assumptions BatteryMonitor shall eventually satisfy $\Diamond = \Diamond \wedge b < 100$	
FOLRiver V002	when assumptions Vision shall eventually satisfy $\Diamond \text{lockable}(v)$	
FOLRiver G002	when assumptions goalAgent shall eventually satisfy $(g \rightarrow \text{chargePos} \leftrightarrow g \rightarrow \text{hotspot}(g))$	
FOLRiver G001	when assumptions goalAgent shall eventually satisfy $(\text{recharge} \leftrightarrow g \rightarrow \text{chargePos}) \wedge (g \rightarrow \text{chargePos} \leftrightarrow \text{recharge})$	
FOLRiver I001	when assumptions Interface shall eventually satisfy $\Diamond \text{moved} \leftrightarrow \text{plan} = \text{emptyPlan}$	
FOLRiver P003	when assumptions Planner shall eventually satisfy $\Diamond \text{Plan} \neq \Diamond \text{notPlanned}$	

Formal Requirements Elicitation Tool (FRET)



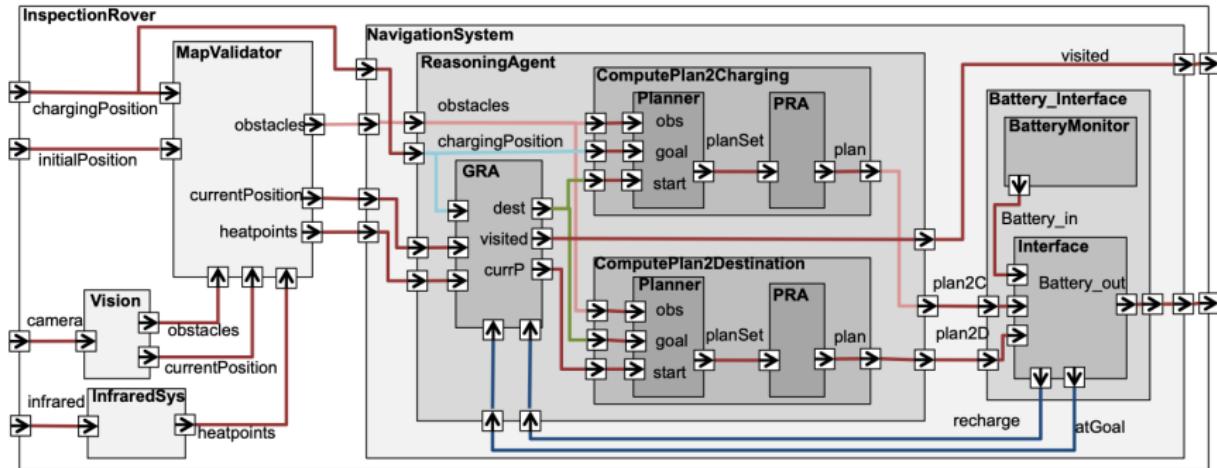
- supports the formalisation, understanding and analysis of requirements user-friendly interface
- intuitive diagrammatic explanations of requirement semantics
- users specify requirements in restricted natural language, called FRETISH, which embodies a temporal logic semantics

Example: Rover Navigation



Bourbouh, H., Farrell, M., Mavridou, A., Sljivo, I., Brat, G., Dennis, L. A., & Fisher, M. *Integrating Formal Verification and Assurance: An Inspection Rover Case Study*. NFM 2021.

Rover Architecture



R1: The rover shall not run out of battery.

R2: The rover shall not collide with an obstacle.

R3: The rover shall visit all reachable points of interest.

Rover Requirement in FRET

Update Requirement

Requirement ID	Parent Requirement ID	Project
R1.2	R1	▼

Rationale and Comments

Rationale

Charging station shall be selected as the next destination whenever the recharge flag is set to true

Comments

Requirement Description

A requirement follows the sentence structure displayed below, where fields are optional unless indicated with **. For information on a field format, click on its corresponding bubble.



if recharge **GRA** shall **immediately** satisfy goal=chargePosition

Status



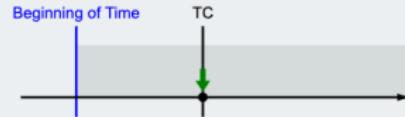
ASSISTANT

TEMPLATES

ENFORCED: in the interval defined by the entire execution.

TRIGGER: first point in the interval if (*recharge*) is true and any point in the interval where (*recharge*) becomes true (from false).
REQUIRES: for every trigger, if trigger holds then RES also holds at the same time point.

Beginning of Time



TC = (*recharge*), Response = (*goal* = *chargePosition*).

Diagram Semantics

Formalizations

Future Time LTL

Past Time LTL

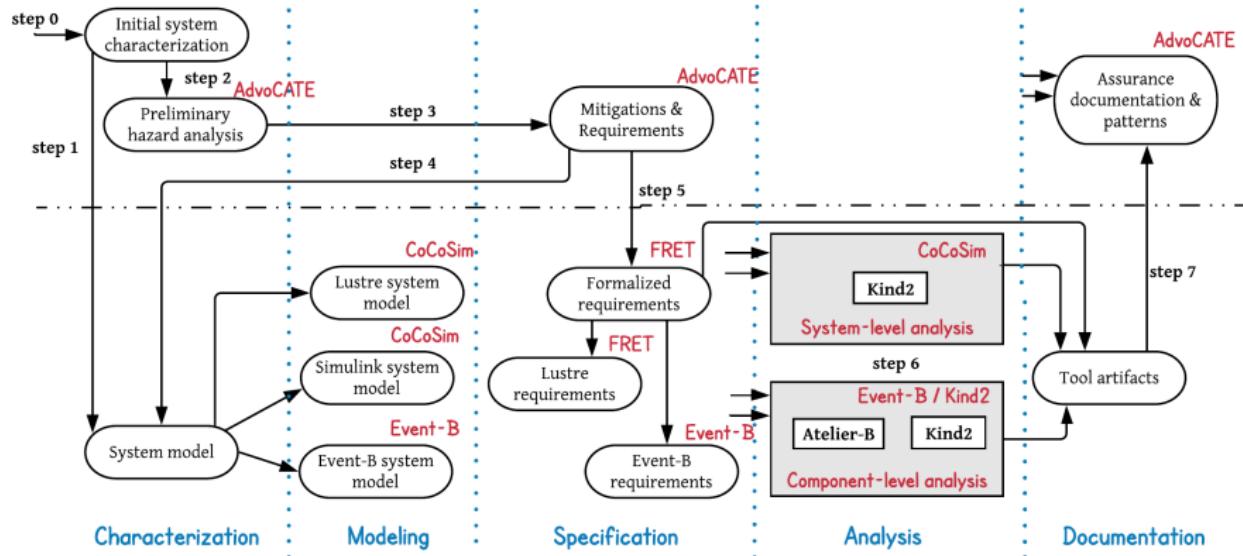
(H ((*recharge*) & ((Y (! *recharge*)) | F T))
-> (*goal* = *chargePosition*))

Target: *GRA* component.

SIMULATE



Verifying Rover Navigation



Example: Aircraft Engine Controller

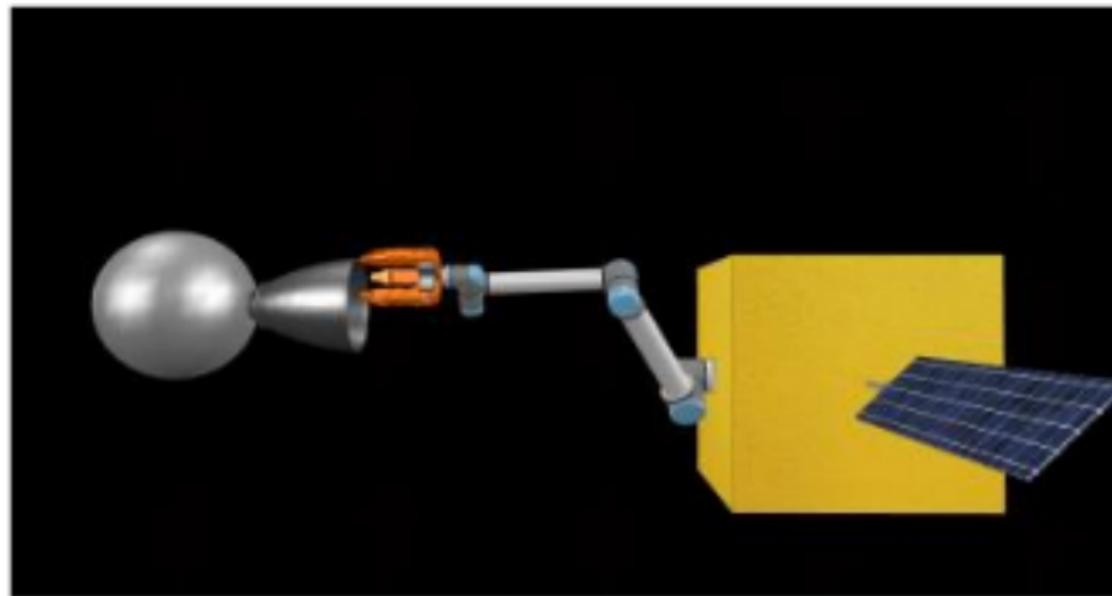
ID	FRETISH
UC5_R_1	if ((sensorfaults) & (trackingPilotCommands)) Controller shall satisfy (controlObjectives)
UC5_R_2	if ((sensorfaults) & (!trackingPilotCommands)) Controller shall satisfy (controlObjectives)
UC5_R_3	if ((sensorfaults) & (trackingPilotCommands)) Controller shall satisfy (operatingLimitObjectives)
UC5_R_4	if ((sensorfaults) & (!trackingPilotCommands)) Controller shall satisfy (operatingLimitObjectives)
UC5_R_5	if ((mechanicalFatigue) & (trackingPilotCommands)) Controller shall satisfy (controlObjectives)
UC5_R_6	if ((mechanicalFatigue) & (!trackingPilotCommands)) Controller shall satisfy (controlObjectives)
UC5_R_7	if ((mechanicalFatigue) & (trackingPilotCommands)) Controller shall satisfy (operatingLimitObjectives)
UC5_R_8	if ((mechanicalFatigue) & (!trackingPilotCommands)) Controller shall satisfy (operatingLimitObjectives)
UC5_R_9	if ((lowProbabilityHazardousEvents) & (trackingPilotCommands)) Controller shall satisfy (controlObjectives)
UC5_R_10	if ((lowProbabilityHazardousEvents) & (!trackingPilotCommands)) Controller shall satisfy (controlObjectives)
UC5_R_11	if ((lowProbabilityHazardousEvents) & (trackingPilotCommands)) Controller shall satisfy (operatingLimitObjectives)
UC5_R_12	if ((lowProbabilityHazardousEvents) & (!trackingPilotCommands)) Controller shall satisfy (operatingLimitObjectives)
UC5_R_13	if (trackingPilotCommands) Controller shall satisfy (changeMode(nominal)) (changeMode(surgeStallPrevention))
UC5_R_14	if (!trackingPilotCommands) Controller shall satisfy (changeMode(nominal)) (changeMode(surgeStallPrevention))

Farrell, M., Luckcuck, M., Sheridan, O., & Monahan, R. *FRETting about requirements: formalised requirements for an aircraft engine controller*. In REFSQ 2022.

What do we mean?

'it forces you to think about the actual meaning behind the natural-language requirements'. - Collins Aerospace/UTRC Ireland

Example: Active Debris Removal



Autonomous Grasping for Active Debris Removal

ID	FRET Formalisation
R1	SV shall satisfy $(\text{grasp}(\text{TGT}, \text{BGP}) \ \& \ \text{closer}(\text{SV}, \text{TGT}))$
R1.1	Camera shall satisfy $\text{distance}(\text{Camera}, \text{TGT}) \geq 0.5$
R1.2	TGT shall satisfy if $\neg \text{contact}(\text{SVA}, \text{TGT})$ then $\text{motionless}(\text{TGT})$
R1.3	Camera shall satisfy $\text{valid}(\text{p})$
R1.3.1	Camera shall satisfy $\text{maxRes}(\text{p}) = 1280 * 720$
R1.3.2	Camera shall satisfy $\text{length}(\text{p}) > 0$
R1.4	Imagepreprocessing shall satisfy $\text{length}(\text{filteredimage}) \leq \text{length}(\text{p}) \ \& \ \text{length}(\text{filteredimage}) > 0$
R1.5	Findoptimalgrasp shall satisfy if $\text{exists}(\text{BGP})$ then $\text{return}(\text{BGP})$
R1.5.1	Findoptimalgrasp shall satisfy $\text{offset}(\text{BGP}, \text{TGT}) = 1 \ \& \ -20 \leq \text{fingersurfaceyaw} \ \& \ \text{fingersurfaceyaw} \leq 20$
R1.5.2	findoptimallgrasp shall satisfy $\text{length}(\text{grasps}) \geq 0$
R1.6	Findoptimalgrasp shall satisfy if $\neg (\text{exists}(\text{BGP}))$ then printerror
R1.7	Controller shall satisfy $\text{executeJointTrajectory}(\text{SVA}, \text{BGP})$
R1.8	SVA shall satisfy $\text{captured}(\text{TGT}) \Rightarrow \text{contactpoint}(\text{SVA}, \text{TGT}) = \text{BGP}$
R1.9	SV shall satisfy $\text{totalpullingdistance} \geq 0.3 \ \& \ \text{totalpullingdistance} \leq 0.5$
R2	SV shall always satisfy $\neg \text{collide}(\text{SV}, \text{TGT})$
R2.1	SV shall always satisfy $\neg (\text{position}(\text{SV}) = \text{position}(\text{TGT}))$
R2.2	SV shall always satisfy $\text{contactpoint}(\text{SVG}, \text{TGT}) = \text{BGP}$.
R2.2.1	SV shall satisfy if $\neg \text{grasped}$ then $\text{contactpoint}(\text{SV}, \text{TGT}) = \text{null}$
R2.2.2	SV shall satisfy if grasped then $\text{contactpoint}(\text{SVG}, \text{TGT}) = \text{BGP} + \text{errormargin}$
R2.3	SVG shall satisfy $\text{captured}(\text{TGT}) \Rightarrow \text{force} = 180$

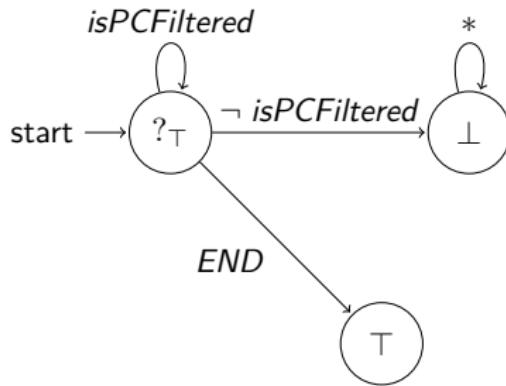
Farrell, M., Mavrakis, N., Ferrando, A., Dixon, C., & Gao, Y. *Formal modelling and runtime verification of autonomous grasping for active debris removal*. Frontiers in Robotics and AI. 2022.

What Can We Verify?

Autonomous Grasping for Active Debris Removal

```
1 method imageprocessing(t: real , p: array<Point>, v: real , nb: int , rf: real)
2     returns (filteredimage: array<Point>)
3     requires 0 < p.Length; // R1.3.2
4     requires v > 0.0;
5     ensures filteredimage.Length ≤ p.Length; //R1.4
6     ensures filteredimage.Length > 0 ; // R1.4
7 {
8     filteredimage := removeDepth(p,t); //remove distant points from p.
9     filteredimage := downSample(filteredimage,v); / voxel representation of p.
10    filteredimage := filter(filteredimage,nb,rf); //removes noise and speckles.
11 }
```

R1.4:



THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG
PILE OF LINEAR ALGEBRA, THEN COLLECT
THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL
THEY START LOOKING RIGHT.



Requirements for Machine Learning

Req ID	Requirement
[RRAV-001]	The neural network shall output the cross track distance error (perpendicular distance from the rover to the centerline.) Error to truth must not exceed X.
[RRAV-002]	Neural network shall output cross track heading error (the angle between the rover heading and the centerline.) Error to truth must not exceed X.
[RRAV-003]	Upon receiving an image, the Neural Network shall output the distance and the angle within X seconds (latency).
[RRAV-004]	Neural network shall output a sensible distance: the value must be between 0 and half the width of the taxiway plus X (buffer X so that it can still report if it is off the taxiway).
[RRAV-005]	Neural network shall output a sensible angle: the value must be between -90 and 90 degrees.
[RRAV-006]	The neural network shall achieve a minimum of X% accuracy on training and Y% accuracy on testing.
[RRAV-007]	(Local robustness) The neural network shall be robust to small perturbations in the image (pixels).
[RRAV-008]	(Semantic variations) The neural network shall be robust to irrelevant variations in the scene.
[RRAV-009]	The neural network shall safely navigate intersections.
[RRAV-010]	The magnitude of the cross track distance error shall drop below X m within T seconds and remain there.
[RRAV-011]	The magnitude of the cross track heading error shall drop below X degrees within T seconds and remain there.

Requirements for Machine Learning

Req ID Requirement Pattern (source: NASA)

[IC-001]	The sw shall achieve an average PARAMETER value of X .
[IC-002]	The sw shall estimate PARAMETER to within $+ - X$ with a $Y\%$ confidence.
[IC-003]	The sw shall estimate the confidence of the PARAMETER estimate.
[IC-004]	The requirement shall be verified by measuring the average of the parameter over N repetitions.
[IC-005]	The sw shall estimate PARAMETER with an $X\%$ confidence interval of no more than $+ - Y$.
[IC-006]	The sw shall calculate the PARAMETER confidence interval at an $X\%$ confidence level.
[IC-007]	The sw shall calculate the PARAMETER as a probability distribution.
[IC-008]	The sw shall determine PARAMETER with a high level of confidence.
[IC-009]	The sw shall detect $X\%$ of occurrences of EVENT.
[IC-010]	The risk-ratio requirements shall be verified using a statistically significant set of SCENARIOS.
[IC-011]	The sw shall cause EVENT at a rate less than X times per Y DURATION.
[IC-012]	The sw shall detect CONDITION that implies EVENT is probable.
[IC-013]	The sw shall take action so that the risk ratio thresholds are satisfied.

Farrell, M., Mavridou, A. & Schumann, J. *Exploring Requirements for Software that Learns: A Research Preview*. REFSQ 2023.

Summary

- Two important questions:
 - ▶ What Should I Verify?
 - ▶ What Can I verify?
- Tools like FRET can help.
- Requirements engineering for autonomous systems is very difficult.
- Heterogeneous/corroborative verification is the way forward.

Questions?

marie.farrell@manchester.ac.uk