

FUNDAMENTALS OF AI-AUGMENTED ENGINEERING: A CONVERSATIONAL APPROACH

Taner Onsay, Ph.D . & Sebnem Onsay
Authors
August 2026 · Revision 1.0

© 2026 AI First Engineering

Problem



Model



Compute



Analyze



Validate



Improve

*The engineering workflow you will use throughout this book
by Taner & Sebnem Onsay,*

CHAPTER - 2

PART 3 · ≈ 50 MIN

The Engineering Search — Vehicle Stopping Distance

A vehicle is approaching a hazard, and only so much road is available. What is the maximum speed at which it can still stop in time?

Problem

Model

Compute

Analyze

Validate

Improve



WHAT YOU BRING FROM PART 2

You arrive carrying four tools

IF / ELSE *A decision, written down*

IF `d_stop <= d_avail`

The hoist: accept or reject. The condition is a yes/no question — and the engineering lives in it.

FOR *Repeat, over items you can count*

FOR `each load in [250,300,275,250]`

Four crates, one check each. You knew the pass count before it started.

WHILE *Repeat, until a condition changes*

WHILE `torque is below spec ...`

The bolt from Part 2: you could not count the wrench turns in advance — the condition decided. Checked before every pass.

THE STOP *A loop must know when to stop*

`temp != 60.5` → **never False**

The heater that never stopped. A robust stop is an engineering specification.

Today's problem uses every one of these — and you will discover for yourself which loop it needs. Nothing new arrives until the search is already in your hands; if any card feels foggy, say so now.

Problem



Model



Compute



Analyze



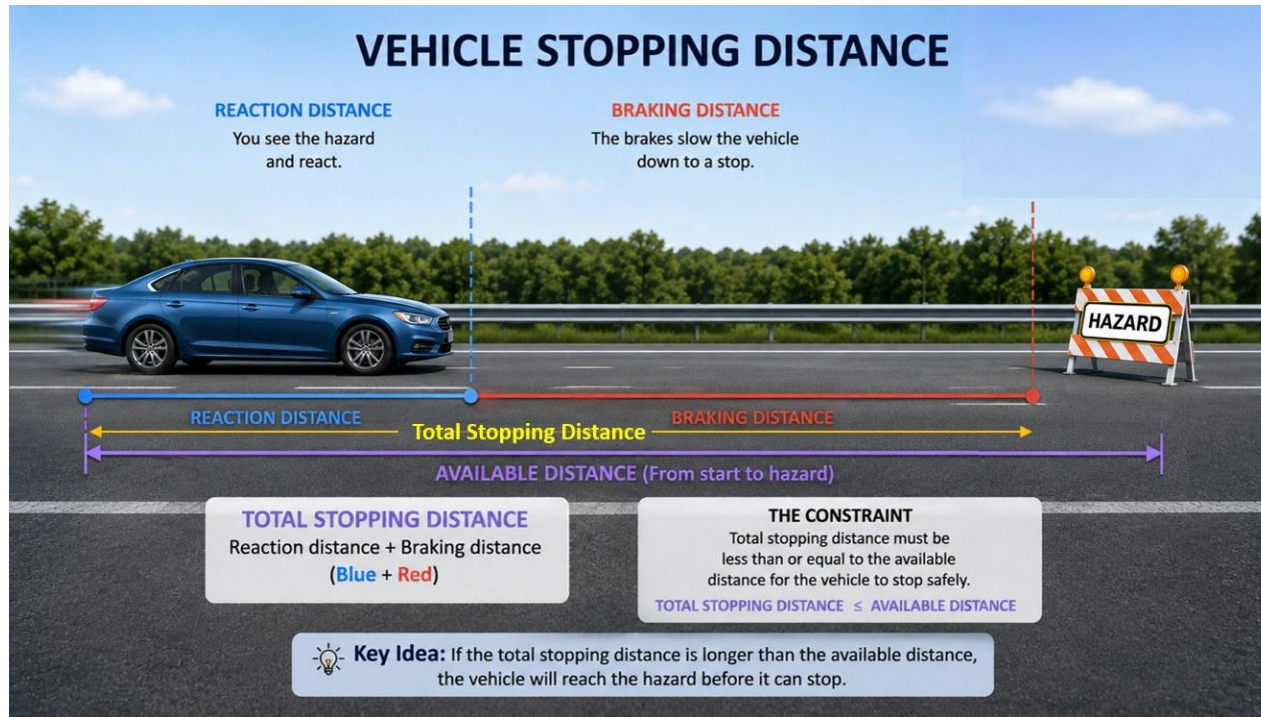
Validate



Improve



Two distances, one constraint



Reaction distance + braking distance must fit inside the available distance.



While you react

The vehicle keeps moving at full speed for your reaction time — before the brakes even touch.



While you brake

Deceleration brings the speed to zero — over a distance that grows rapidly with speed.



Reaction time is a real, personal, measurable quantity.

In the accompanying laboratory you will measure your own — a falling ruler plus the Chapter 1 free-fall model.



The constraint turns physics into an engineering decision.

Total stopping distance must fit the available distance. That single constraint decides the speed.

Problem

Model

Compute

Analyze

Validate

Improve



The stopping-distance model

REACTION DISTANCE — constant speed

$$d_{reaction} = v \cdot t_r$$

BRAKING DISTANCE — constant deceleration a

$$d_{braking} = v^2 / (2a)$$

$$d_{stop} = v \cdot t_r + v^2 / (2a)$$

total stopping distance, SI units throughout: v in m/s · t_r in s · a in m/s² · d in m



Speed hits twice

Reaction distance grows linearly with v — braking distance grows with v^2 . Doubling speed roughly quadruples the braking segment.



The driver matters

t_r represents the driver's response time. Distraction, fatigue, task complexity, and other human factors can change it.



The road matters

a is the braking capability. Dry pavement might allow ~ 7 m/s²; a wet road far less. One parameter, big consequences.

Problem

Model

Compute

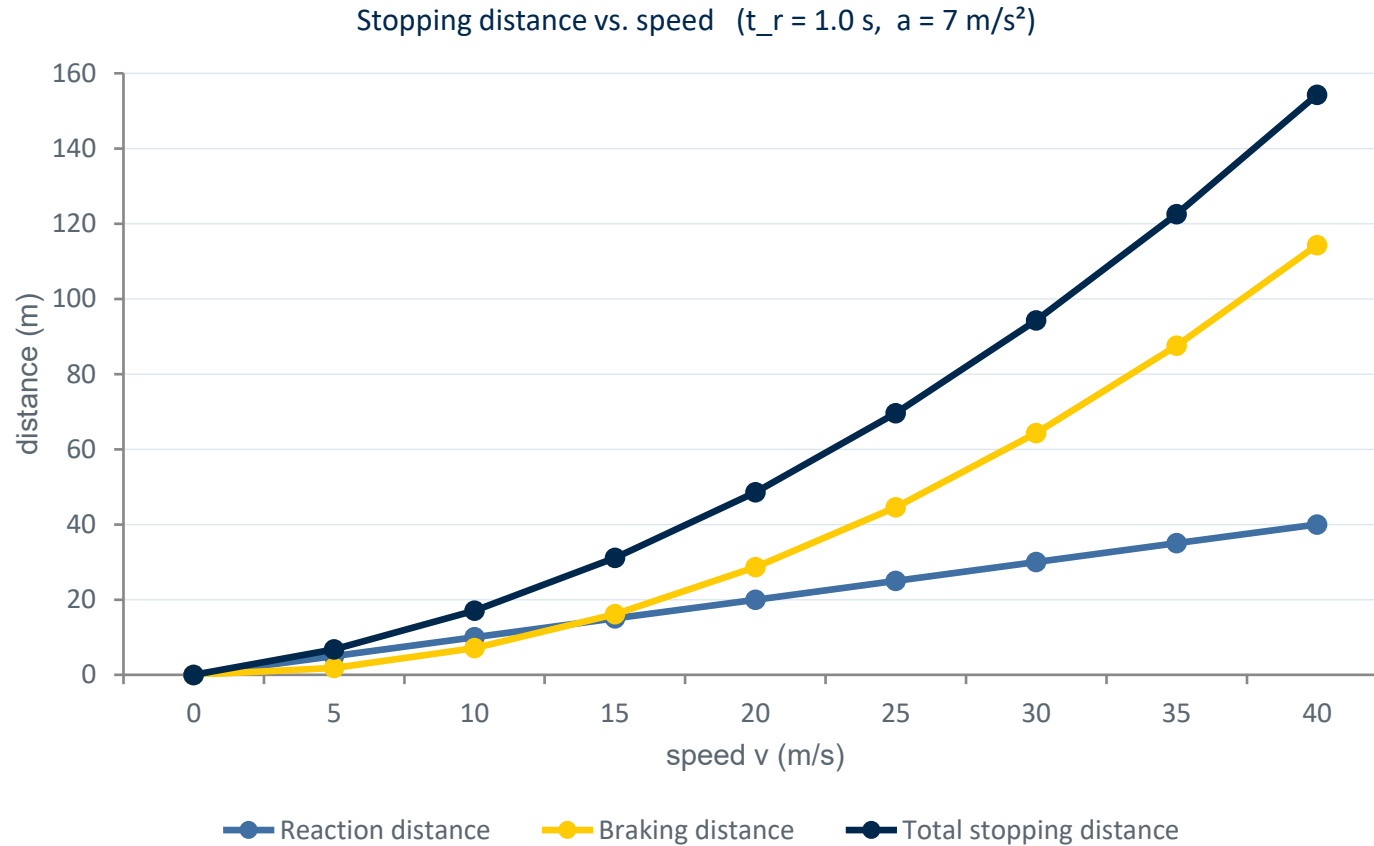
Analyze

Validate

Improve



Braking distance takes over as speed rises



READ THE CURVES

Reaction distance: a straight line — $v \cdot t_r$.

Braking distance: a curve — $v^2/2a$ — quietly dominant at speed.

The two are equal at $v = 2 \cdot a \cdot t_r = 14$ m/s. By 40 m/s braking is nearly 3× the reaction distance.

Drawn for $t_r = 1.0$ s, $a = 7$ m/s². Your own t_r will redraw this picture in the laboratory.



Turn the model around

Given the available stopping distance d_{avail} , the driver's reaction time t_r , and braking capability a :

What is the maximum speed v_{max} at which the vehicle can still stop in time?

FORWARD — easy

Given $v \rightarrow$ compute d_{stop} .

Plug in, evaluate, done. One pass through the formula.

BACKWARD — interesting

Given $d_{\text{avail}} \rightarrow$ find the largest safe v .

The unknown is inside the formula. How do we get it out?

Before any algebra and before any code: how would YOU find v_{max} by hand?

Problem



Model



Compute



Analyze



Validate



Improve



PREDICT



**50 m of road. Reaction time 1.0 s.
Braking capability 7 m/s^2 .
Roughly what is the maximum safe speed?**

Estimate it — no computing. Write a number down.

Hint: the curve you just saw already brackets the answer. Keep your number — the laboratory asks for it again with your own reaction time.

Problem ›

Model ›

Compute ›

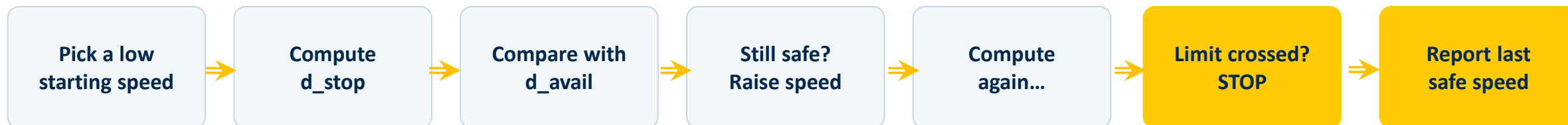
Analyze ›

Validate ›

Improve



How you would do it by hand

**CALCULATE****COMPARE****DECIDE****REPEAT****STOP**

You just invented a search.



Sweep a variable, test a condition each time, stop when the condition flips. You will meet far more powerful versions of this idea later in the course — here you own the simple one.

And notice which loop it wants.



You don't know in advance how many speeds you'll try — you know the condition that ends the process. That is a WHILE, straight from the definition in Part 2.

Problem



Model

**Compute**

Analyze



Validate



Improve



Excel: watch yourself repeat the procedure

[OPEN EXCEL DEMO](#)

Speed (m/s)	Reaction (m)	Braking (m)	Total (m)	Safe? (≤ 50 m)
10	10	7.1	17.1	SAFE
15	15	16.1	31.1	SAFE
20	20	28.6	48.6	SAFE
25	25	44.6	69.6	UNSAFE
30	30	64.3	94.3	UNSAFE

$t_r = 1.0 \text{ s}$ · $a = 7 \text{ m/s}^2$ · $d_{\text{avail}} = 50 \text{ m}$ — one formula typed once, filled down

WHY EXCEL FIRST?

- Every input $\rightarrow$ formula $\rightarrow$ result is visible in its cell.
- Each new row is you re-running the same procedure at a new speed.
- The answer is trapped between two rows: v_{max} lies between 20 and 25 m/s.
- **The repetition is now something you can SEE — and feel.**

Why are we manually repeating essentially the same calculation?

Problem



Model

**Compute**

Analyze



Validate



Improve



Your search has the WHILE shape from Part 2

PART 2 · THE THREE PARTS OF A WHILE

1 · **A condition**, checked before every pass — including the first.

2 · **A body**, that does the work of one pass.

3 · **A mover**, something in the body that pushes the condition toward false — or the loop never ends.

WHILE torque is below spec
 TIGHTEN a quarter turn

The tightening is the mover — every turn raises the torque. The heater with temp != 60.5 had no mover that worked — it never stopped.

THE SEARCH YOU JUST INVENTED

Condition: the NEXT speed still stops within the 50 m.

Body: raise the speed by one step.

Mover: stopping distance grows with speed — the condition **MUST** eventually fail. This loop cannot run forever.

You do not know how many speeds you will try.

You DO know the condition that ends the process.

That is the WHILE definition from Part 2, word for word.

All three parts present and healthy. Now we write the search down — first as pseudocode, the way every algorithm in this course gets written before any code.

Problem



Model



Compute



Analyze



Validate



Improve



You have already written the algorithm

WHAT THE SPREADSHEET DID

Inputs: t_r , a , d_{avail} — the constants at the top.

Process: each row computes $v \cdot t_r$ and $v^2/2a$.

Decision: the Safe? column asks $d_{\text{stop}} \leq d_{\text{avail}}$.

Repetition: every fill-down row is one more pass.

Stop: you stopped adding rows when SAFE turned UNSAFE.

Output: the last safe speed.

THE SAME PROCEDURE, WRITTEN DOWN

```
# PSEUDOCODE — max safe speed
GET  $t_r$ ,  $a$ ,  $d_{\text{avail}}$ 
SET  $v = 0$ ,  $\text{step} = 0.5$  # m/s
WHILE stopping distance at ( $v + \text{step}$ )
    is  $\leq d_{\text{avail}}$ :
     $v = v + \text{step}$ 
REPORT  $v$  as maximum safe speed
```

Anatomy check ✓ Inputs · process · decision · repetition · stopping criterion · output — all present. The spreadsheet WAS the algorithm, executed by hand. Now it is an engineering document — ready for any platform.

Problem



Model



Compute



Analyze



Validate



Improve



ONE JUMP YOU HAVE ALREADY MADE

Pseudocode → Python is only a change of dialect

PART 2 · THE HOIST, BOTH WAYS

```
# PSEUDOCODE
```

```
IF current + new <= limit
```

```
    accept the load
```

```
ELSE
```

```
    reject the load
```

You wrote this — and read its Python twin beside it.

THE SAME, IN PYTHON

```
if current + new_load <= SAFE_LIMIT:
```

```
    accept()
```

```
else:
```

```
    reject()
```

What changed: lowercase words, a colon, indentation — the assistant's dialect.

What did not: the condition, the two paths, the thinking.

Pseudocode → Python is a translation, not a new idea. You made this exact jump in Part 2. The next slide makes it again — same WHILE search, now in the dialect the machine executes. The thinking is already finished; only the spelling changes.

Problem



Model



Compute



Analyze



Validate



Improve



The WHILE search in Python

PYTHON — THE SEARCH, AUTOMATED

```
max_safe_speed.py

t_r, a, d_avail = 1.0, 7.0, 50.0  # s, m/s^2, m

v = 0.0      # start slow (m/s)
step = 0.5   # speed increment (m/s)

while stopping_distance(v + step, t_r, a) <= d_avail:
    v = v + step      # next speed is safe

print("Max safe speed:", v, "m/s")
```

READ IT LIKE AN ENGINEER

The condition IS the engineering: “the next speed still stops in time.”

The stop is guaranteed: d_{stop} grows with v , so the condition must eventually fail.

step = 0.5 sets the resolution of the answer. Hold that thought for validation.

stopping_distance(...) is not defined yet. That is Part 4's problem.

Predict: with these inputs, roughly what will it print? (Your Excel table already knows.)

Problem



Model



Compute



Analyze



Validate



Improve



THE ALGORITHM IS THE ENGINEERING IDEA. THE PLATFORM IS THE TOOL.

Same algorithm — any platform



Excel

NOW

Each fill-down row is one pass; the Safe? column is the decision. The repetition is visible — you did it by hand.



Python

NOW

while ... : automates the same CALCULATE → COMPARE → DECIDE → REPEAT → STOP and generalizes it to any t_r , a , d_{avail} .



VBA

LATER IN THE COURSE

A Do While ... Loop can automate the very same rows behind the spreadsheet cells. Same idea, inside Excel.



MATLAB

LATER IN THE COURSE

while ... end expresses the identical procedure. The algorithm remains; the environment changes.

You will not need every problem in every platform. Cross-platform repetition happens only when it teaches something — making an algorithm visible before automating it, or comparing two independent implementations for validation.

Problem



Model



Compute



Analyze



Validate



Improve