A Formally Verified Checker of the Safe Distance Traffic Rules for Autonomous Vehicles

Albert Rizaldi, Fabian Immler

June 7, 2023

Abstract

The Vienna Convention on Road Traffic defines the safe distance traffic rules informally. This could make autonomous vehicle liable for safe-distance-related accidents because there is no clear definition of how large a safe distance is. We provide a formally proven prescriptive definition of a safe distance, and checkers which can decide whether an autonomous vehicle is obeying the safe distance rule. Not only does our work apply to the domain of law, but it also serves as a specification for autonomous vehicle manufacturers and for online verification of path planners. This formalization accompanies our paper "A Formally Verified Checker of the Safe Distance Traffic Rules for Autonomous Vehicles". [1]

Contents

1 Safe Distance

theory *Safe-Distance*
imports
HOL−Analysis.Multivariate-Analysis
HOL−Decision-Procs.Approximation
Sturm-Sequences.Sturm
begin

This theory is about formalising the safe distance rule. The safe distance rule is obtained from Vienna Convention which basically states the following thing.

"The car at all times must maintain a safe distance towards the vehicle in front of it, such that whenever the vehicle in front and the ego vehicle apply maximum deceleration, there will not be a collision."

To formalise this safe distance rule we have to define first what is a safe distance. To define this safe distance, we have to model the physics of the movement of the vehicle. The following model is sufficient.

$s = s_0 + v_0 * t + 1 / 2 * a_0 * t^2$

Assumptions in this model are :

- Both vehicles are assumed to be point mass. The exact location of the ego vehicle is the front-most occupancy of the ego vehicle. Similarly for the other vehicle, its exact location is the rearmost occupancy of the other vehicle.
- Both cars can never drive backward.

lemmas [*simp del*] = *div-mult-self1 div-mult-self2 div-mult-self3 div-mult-self4*

1.1 Quadratic Equations

lemma *discriminant*: $a * x^2 + b * x + c = (0::real) \Longrightarrow 0 \leq b^2 - 4 * a * c$
⟨*proof*⟩

lemma *quadratic-eq-factoring*:

assumes D : $D = b^2 - 4 * a * c$
assumes *nn*: $0 \leq D$
assumes *x1*: $x_1 = (-b + sqrt\ D)\ /\ (2 * a)$
assumes *x2*: $x_2 = (-b - sqrt\ D)\ /\ (2 * a)$
assumes a : $a \neq 0$
shows $a * x^2 + b * x + c = a * (x - x_1) * (x - x_2)$
⟨*proof*⟩

lemma *quadratic-eq-zeroes-iff*:
assumes D : $D = b^2 - 4 * a * c$
assumes *x1*: $x_1 = (-b + sqrt\ D)\ /\ (2 * a)$
assumes *x2*: $x_2 = (-b - sqrt\ D)\ /\ (2 * a)$
assumes a : $a \neq 0$
shows $a * x^2 + b * x + c = 0 \longleftrightarrow (D \geq 0 \wedge (x = x_1 \vee x = x_2))$ (**is** *?z* $\longleftrightarrow$ -)
⟨*proof*⟩

1.2 Convexity Condition

lemma *p-convex*:
fixes $a\ b\ c\ x\ y\ z$ :: *real*
assumes *p-def*: $p = (\lambda x.\ a * x^2 + b * x + c)$
assumes *less* : $x < y\ y < z$
and *ge* : $p\ x > p\ y\ p\ y \leq p\ z$
shows $a > 0$
⟨*proof*⟩

definition *root-in* :: *real* ⇒ *real* ⇒ (*real* ⇒ *real*) ⇒ *bool* **where**
root-in m M f = $(\exists\, x{\in}\{m\ ..\ M\}.\ f\ x = 0)$

definition *quadroot-in* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
quadroot-in m M a b c = *root-in m M* $(\lambda x.\ a * x\hat{}2 + b * x + c)$

lemma *card-iff-exists*: $0 <$ *card X* $\longleftrightarrow$ *finite X* $\wedge\ (\exists\, x.\ x \in X)$
⟨*proof*⟩

lemma *quadroot-in-sturm*[*code*]:
quadroot-in m M a b c $\longleftrightarrow (a = 0 \wedge b = 0 \wedge c = 0 \wedge m \leq M) \vee$
$(m \leq M \wedge$ *poly* [:*c*, *b*, *a*:] $m = 0) \vee$
count-roots-between [:*c*, *b*, *a*:] *m M* > 0
⟨*proof*⟩

lemma *check-quadroot-linear*:
fixes $a\ b\ c$ :: *real*
assumes $a = 0$
shows ¬ *quadroot-in m M a b c* $\longleftrightarrow$
$((b = 0 \wedge c = 0 \wedge M < m) \vee (b = 0 \wedge c \neq 0) \vee$
$(b \neq 0 \wedge (let\ x = -\ c\ /\ b\ in\ m > x \vee x > M)))$
⟨*proof*⟩

lemma *check-quadroot-nonlinear*:
assumes $a \neq 0$
shows *quadroot-in m M a b c* =
$(let\ D = b\hat{}2 - 4 * a * c\ in\ D \geq 0 \wedge$
$((let\ x = (-b + sqrt\ D)/(2*a)\ in\ m \leq x \wedge x \leq M) \vee$
$(let\ x = (-b - sqrt\ D)/(2*a)\ in\ m \leq x \wedge x \leq M)))$
⟨*proof*⟩

lemma *ncheck-quadroot*:
shows $\neg$*quadroot-in m M a b c* $\longleftrightarrow$
$(a = 0 \longrightarrow \neg$*quadroot-in m M a b c*$) \wedge$
$(a = 0 \vee \neg$*quadroot-in m M a b c*$)$
⟨*proof*⟩

1.3 Movement

locale *movement* =
fixes *a v s0* :: *real*
begin

Function to compute the distance using the equation
$s(t) = s_0 + v_0 * t + 1 / 2 * a_0 * t^2$
Input parameters:

- s_0: initial distance
- v_0: initial velocity (positive means forward direction and the converse is true)
- a: acceleration (positive for increasing and negative for decreasing)
- t: time

For the time $t < 0$, we assume the output of the function is s_0. Otherwise, the output is calculated according to the equation above.

1.3.1 Continuous Dynamics

definition *p* :: *real* ⇒ *real* **where**
$p\ t = s0 + v * t + 1/2 * a * t^2$

lemma *p-all-zeroes*:
assumes *D*: $D = v^2 - 2 * a * s0$
shows $p\ t = 0 \longleftrightarrow ((a \neq 0 \wedge 0 \leq D \wedge ((t = (-\ v + sqrt\ D)\ /\ a) \vee t = (-\ v - sqrt\ D)\ /\ a)) \vee$
$(a = 0 \wedge v = 0 \wedge s0 = 0) \vee (a = 0 \wedge v \neq 0 \wedge t = (-\ s0\ /\ v)))$
⟨*proof*⟩

lemma *p-zero*[*simp*]: *p 0 = s0*
⟨*proof*⟩

lemma *p-continuous*[*continuous-intros*]: *continuous-on T p*
⟨*proof*⟩

lemma *isCont-p*[*continuous-intros*]: *isCont p x*
⟨*proof*⟩

definition *p'* :: *real* ⇒ *real* **where**
*p' t = v + a * t*

lemma *p'-zero*: *p' 0 = v*
⟨*proof*⟩

lemma *p-has-vector-derivative*[*derivative-intros*]: (*p has-vector-derivative p' t*) (*at t within s*)
⟨*proof*⟩

lemma *p-has-real-derivative*[*derivative-intros*]: (*p has-real-derivative p' t*) (*at t within s*)
⟨*proof*⟩

definition *p''* :: *real* ⇒ *real* **where**
p'' t = a

lemma *p'-has-vector-derivative*[*derivative-intros*]: (*p' has-vector-derivative p'' t*) (*at t within s*)
⟨*proof*⟩

lemma *p'-has-real-derivative*[*derivative-intros*]: (*p' has-real-derivative p'' t*) (*at t within s*)
⟨*proof*⟩

definition *t-stop* :: *real* **where**
t-stop = − v / a

lemma *p'-stop-zero*: *p' t-stop* = (*if a = 0 then v else 0*) ⟨*proof*⟩

lemma *p'-pos-iff*: *p' x > 0* ⟷ (*if a > 0 then x > −v / a else if a < 0 then x < −v / a else v > 0*)
⟨*proof*⟩

lemma *le-t-stop-iff*: *a ≠ 0* ⟹ *x ≤ t-stop* ⟷ (*if a < 0 then p' x ≥ 0 else p' x ≤ 0*)
⟨*proof*⟩

lemma *p'-continuous*[*continuous-intros*]: *continuous-on T p'*
⟨*proof*⟩

lemma *isCont-p'*[*continuous-intros*]: *isCont p' x*
⟨*proof*⟩

definition *p-max* :: *real* **where**
p-max = *p t-stop*

lemmas *p-t-stop* = *p-max-def*[*symmetric*]

lemma *p-max-eq*: *p-max* = *s0* − v^2 / *a* / *2*
⟨*proof*⟩

1.3.2 Hybrid Dynamics

definition *s* :: *real* ⇒ *real* **where**
s t = (*if t* ≤ *0* *then s0*
else if t ≤ *t-stop then p t*
else *p-max*)

definition *q* :: *real* ⇒ *real* **where**
q t = *s0* + *v* ∗ *t*

definition *q'* :: *real* ⇒ *real* **where**
q' t = *v*

lemma *init-q*: *q 0* = *s0* ⟨*proof*⟩

lemma *q-continuous*[*continuous-intros*]: *continuous-on T q*
⟨*proof*⟩

lemma *isCont-q*[*continuous-intros*]: *isCont q x*
⟨*proof*⟩

lemma *q-has-vector-derivative*[*derivative-intros*]: (*q has-vector-derivative q' t*) (*at t within u*)
⟨*proof*⟩

lemma *q-has-real-derivative*[*derivative-intros*]: (*q has-real-derivative q' t*) (*at t within u*)
⟨*proof*⟩

lemma *s-cond-def*:
t ≤ *0* ⟹ *s t* = *s0*
0 ≤ *t* ⟹ *t* ≤ *t-stop* ⟹ *s t* = *p t*
⟨*proof*⟩

end

locale *braking-movement* = *movement* +

assumes *decel*: $a < 0$
assumes *nonneg-vel*: $v \geq 0$
begin

lemma *t-stop-nonneg*: $0 \leq$ *t-stop*
⟨*proof*⟩

lemma *t-stop-pos*:
assumes $v \neq 0$
shows $0 <$ *t-stop*
⟨*proof*⟩

lemma *t-stop-zero*:
assumes *t-stop* $= 0$
shows $v = 0$
⟨*proof*⟩

lemma *t-stop-zero-not-moving*: *t-stop* $= 0 \Longrightarrow q\ t = s0$
⟨*proof*⟩

abbreviation *s-stop* $\equiv s$ *t-stop*

lemma *s-t-stop*: *s-stop* $=$ *p-max*
⟨*proof*⟩

lemma *s0-le-s-stop*: *s0* $\leq$ *s-stop*
⟨*proof*⟩

lemma *p-mono*: $x \leq y \Longrightarrow y \leq$ *t-stop* $\Longrightarrow p\ x \leq p\ y$
⟨*proof*⟩

lemma *p-antimono*: $x \leq y \Longrightarrow$ *t-stop* $\leq x \Longrightarrow p\ y \leq p\ x$
⟨*proof*⟩

lemma *p-strict-mono*: $x < y \Longrightarrow y \leq$ *t-stop* $\Longrightarrow p\ x < p\ y$
⟨*proof*⟩

lemma *p-strict-antimono*: $x < y \Longrightarrow$ *t-stop* $\leq x \Longrightarrow p\ y < p\ x$
⟨*proof*⟩

lemma *p-max*: $p\ x \leq$ *p-max*
⟨*proof*⟩

lemma *continuous-on-s*[*continuous-intros*]: *continuous-on* $T\ s$
⟨*proof*⟩

lemma *isCont-s*[*continuous-intros*]: *isCont* $s\ x$
⟨*proof*⟩

definition $s' :: real \Rightarrow real$ **where**
$s'\ t = (if\ t \leq t\text{-}stop\ then\ p'\ t\ else\ 0)$

lemma *s-has-real-derivative*:
assumes $t \geq 0\ v\ /\ a \leq 0\ a \neq 0$
shows $(s\ has\text{-}real\text{-}derivative\ s'\ t)\ (at\ t\ within\ \{0..\})$
⟨*proof*⟩

lemma *s-has-vector-derivative*[*derivative-intros*]:
assumes $t \geq 0\ v\ /\ a \leq 0\ a \neq 0$
shows $(s\ has\text{-}vector\text{-}derivative\ s'\ t)\ (at\ t\ within\ \{0..\})$
⟨*proof*⟩

lemma *s-has-field-derivative*[*derivative-intros*]:
assumes $t \geq 0\ v\ /\ a \leq 0\ a \neq 0$
shows $(s\ has\text{-}field\text{-}derivative\ s'\ t)\ (at\ t\ within\ \{0..\})$
⟨*proof*⟩

lemma *s-has-real-derivative-at*:
assumes $0 < x\ 0 \leq v\ a < 0$
shows $(s\ has\text{-}real\text{-}derivative\ s'\ x)\ (at\ x)$
⟨*proof*⟩

lemma *s-delayed-has-field-derivative*[*derivative-intros*]:
assumes $\delta < t\ 0 \leq v\ a < 0$
shows $((\lambda x.\ s\ (x - \delta))\ has\text{-}field\text{-}derivative\ s'\ (t - \delta))\ (at\ t\ within\ \{\delta<..\})$
⟨*proof*⟩

lemma *s-delayed-has-vector-derivative*[*derivative-intros*]:
assumes $\delta < t\ 0 \leq v\ a < 0$
shows $((\lambda x.\ s\ (x - \delta))\ has\text{-}vector\text{-}derivative\ s'\ (t - \delta))\ (at\ t\ within\ \{\delta<..\})$
⟨*proof*⟩

lemma *s'-nonneg*: $0 \leq v \Longrightarrow a \leq 0 \Longrightarrow 0 \leq s'\ x$
⟨*proof*⟩

lemma *s'-pos*: $0 \leq x \Longrightarrow x < t\text{-}stop \Longrightarrow 0 \leq v \Longrightarrow a \leq 0 \Longrightarrow 0 < s'\ x$
⟨*proof*⟩

1.3.3 Monotonicity of Movement

lemma *s-mono*:
assumes $t \geq u\ u \geq 0$
shows $s\ t \geq s\ u$
⟨*proof*⟩

lemma *s-strict-mono*:
assumes $u < t\ t \leq t\text{-}stop\ u \geq 0$
shows $s\ u < s\ t$

⟨*proof*⟩

lemma *s-antimono*:
assumes $x \leq y$
assumes *t-stop* $\leq x$
shows $s\ y \leq s\ x$
⟨*proof*⟩

lemma *init-s* : $t \leq 0 \Longrightarrow s\ t = s0$
⟨*proof*⟩

lemma *q-min*: $0 \leq t \Longrightarrow s0 \leq q\ t$
⟨*proof*⟩

lemma *q-mono*: $x \leq y \Longrightarrow q\ x \leq q\ y$
⟨*proof*⟩

1.3.4 Maximum at Stopping Time

lemma *s-max*: $s\ x \leq$ *s-stop*
⟨*proof*⟩

lemma *s-eq-s-stop*: *NO-MATCH t-stop* $x \Longrightarrow x \geq$ *t-stop* $\Longrightarrow s\ x =$ *s-stop*
⟨*proof*⟩

end

1.4 Safe Distance

locale *safe-distance* =
fixes $a_e\ v_e\ s_e$ :: *real*
fixes $a_o\ v_o\ s_o$ :: *real*
assumes *nonneg-vel-ego* : $0 \leq v_e$
assumes *nonneg-vel-other* : $0 \leq v_o$
assumes *decelerate-ego* : $a_e < 0$
assumes *decelerate-other* : $a_o < 0$
assumes *in-front* : $s_e < s_o$
begin

lemmas *hyps* =
nonneg-vel-ego
nonneg-vel-other
decelerate-ego
decelerate-other
in-front

sublocale *ego*: *braking-movement* $a_e\ v_e\ s_e$ ⟨*proof*⟩
sublocale *other*: *braking-movement* $a_o\ v_o\ s_o$ ⟨*proof*⟩
sublocale *ego-other*: *movement* $a_o - a_e\ v_o - v_e\ s_o - s_e$ ⟨*proof*⟩

1.4.1 Collision

definition *collision* :: *real set* ⇒ *bool* **where**
collision time-set ≡ (∃ *t* ∈ *time-set*. *ego.s t* = *other.s t*)

abbreviation *no-collision* :: *real set* ⇒ *bool* **where**
no-collision time-set ≡ ¬ *collision time-set*

lemma *no-collision-initially* : *no-collision* {.. *0*}
⟨*proof*⟩

lemma *no-collisionI*:
(⋀*t*. *t* ∈ *S* ⟹ *ego.s t* ≠ *other.s t*) ⟹ *no-collision S*
⟨*proof*⟩

theorem *cond-1*: *ego.s-stop* < s_o ⟹ *no-collision* {*0*..}
⟨*proof*⟩

lemma *ego-other-strict-ivt*:
assumes *ego.s t* > *other.s t*
shows *collision* {*0* ..< *t*}
⟨*proof*⟩

lemma *collision-subset*: *collision s* ⟹ *s* ⊆ *t* ⟹ *collision t*
⟨*proof*⟩

lemma *ego-other-ivt*:
assumes *ego.s t* ≥ *other.s t*
shows *collision* {*0* .. *t*}
⟨*proof*⟩

theorem *cond-2*:
assumes *ego.s-stop* ≥ *other.s-stop*
shows *collision* {*0* ..}
⟨*proof*⟩

abbreviation *D2* :: *real* **where**
D2 ≡ (v_o − v_e)^2 − *2* ∗ (a_o − a_e) ∗ (s_o − s_e)

abbreviation $t_D{}'$ :: *real* **where**
$t_D{}'$ ≡ *sqrt* (*2* ∗ (*ego.s-stop* − *other.s-stop*) / a_o)

lemma *pos-via-half-dist*:
dist a b < *b* / *2* ⟹ *b* > *0* ⟹ *a* > *0*
⟨*proof*⟩

lemma *collision-within-p*:
assumes s_o ≤ *ego.s-stop ego.s-stop* < *other.s-stop*
shows *collision* {*0*..} ⟷ (∃ *t*≥*0*. *ego.p t* = *other.p t* ∧ *t* < *ego.t-stop* ∧ *t* < *other.t-stop*)

⟨*proof*⟩

lemma *collision-within-eq*:
assumes $s_o \leq$ *ego.s-stop ego.s-stop* $<$ *other.s-stop*
shows *collision* $\{0..\} \longleftrightarrow$ *collision* $\{0$ *..<* *min ego.t-stop other.t-stop*$\}$
⟨*proof*⟩

lemma *collision-excluded*: $(\bigwedge t.\ t \in T \Longrightarrow$ *ego.s* $t \neq$ *other.s* $t) \Longrightarrow$ *collision* $S \longleftrightarrow$ *collision* $(S - T)$
⟨*proof*⟩

lemma *collision-within-less*:
assumes $s_o \leq$ *ego.s-stop ego.s-stop* $<$ *other.s-stop*
shows *collision* $\{0..\} \longleftrightarrow$ *collision* $\{0$ *<..<* *min ego.t-stop other.t-stop*$\}$
⟨*proof*⟩

theorem *cond-3*:
assumes $s_o \leq$ *ego.s-stop ego.s-stop* $<$ *other.s-stop*
shows *collision* $\{0..\} \longleftrightarrow (a_o > a_e \wedge v_o < v_e \wedge 0 \leq D2 \wedge \mathit{sqrt}\ D2 > v_e - a_e\ /\ a_o * v_o)$
⟨*proof*⟩

1.4.2 Formalising Safe Distance

First definition for Safe Distance based on *cond-1*.

definition *absolute-safe-distance* :: *real* **where**
absolute-safe-distance $= -\ {v_e}^2\ /\ (2 * a_e)$

lemma *absolute-safe-distance*:
assumes $s_o - s_e >$ *absolute-safe-distance*
shows *no-collision* $\{0..\}$
⟨*proof*⟩

First Fallback for Safe Distance.

definition *fst-safe-distance* :: *real* **where**
fst-safe-distance $= {v_o}^2\ /\ (2 * a_o) - {v_e}^2\ /\ (2 * a_e)$

definition *distance-leq-d2* :: *real* **where**
distance-leq-d2 $= (a_e + a_o)\ /\ (2 * {a_o}^2) * {v_o}^2 - v_o * v_e\ /\ a_o$

lemma *snd-leq-fst-exp*: *distance-leq-d2* $\leq$ *fst-safe-distance*
⟨*proof*⟩

lemma *sqrt-D2-leq-stop-time-diff*:
assumes $a_e < a_o$
assumes $0 \leq v_e - a_e\ /\ a_o * v_o$
assumes $s_o - s_e \geq$ *distance-leq-d2*
shows *sqrt D2* $\leq v_e - a_e\ /\ a_o * v_o$
⟨*proof*⟩

lemma *cond2-imp-pos-vo*:
assumes $s_o \leq$ *ego.s-stop ego.s-stop* $<$ *other.s-stop*
shows $v_o \neq 0$
⟨*proof*⟩

lemma *cond2-imp-gt-fst-sd*:
assumes $s_o \leq$ *ego.s-stop ego.s-stop* $<$ *other.s-stop*
shows *fst-safe-distance* $< s_o - s_e$
⟨*proof*⟩

Second Fallback for Safe Distance.

definition *snd-safe-distance* :: *real* **where**
snd-safe-distance $= (v_o - v_e)^2$ / $(2 * (a_o - a_e))$

lemma *fst-leq-snd-safe-distance*:
assumes $a_e < a_o$
shows*fst-safe-distance* $\leq$ *snd-safe-distance*
⟨*proof*⟩

lemma *snd-safe-distance-iff-nonneg-D2*:
assumes $a_e < a_o$
shows $s_o - s_e \leq$ *snd-safe-distance* $\longleftrightarrow$ $0 \leq$ *D2*
⟨*proof*⟩

lemma *t-stop-diff-neg-means-leq-D2*:
assumes $s_o \leq$ *ego.s-stop ego.s-stop* $<$ *other.s-stop* $a_e < a_o$ $0 \leq$ *D2*
shows $v_e - a_e$ / $a_o * v_o < 0 \longleftrightarrow$ *sqrt D2* $> v_e - a_e$ / $a_o * v_o$
⟨*proof*⟩

theorem *cond-3′*:
assumes $s_o \leq$ *ego.s-stop ego.s-stop* $<$ *other.s-stop*
shows *collision* $\{0..\} \longleftrightarrow (a_o > a_e \wedge v_o < v_e \wedge s_o - s_e \leq$ *snd-safe-distance* $\wedge$
$v_e - a_e$ / $a_o * v_o < 0)$
⟨*proof*⟩

definition d :: *real* $\Rightarrow$ *real* **where**
$d\ t = ($
if $t \leq 0$ *then* $s_o - s_e$
else if $t \leq$ *ego.t-stop* $\wedge$ $t \leq$ *other.t-stop then ego-other.p* t
else if ego.t-stop $\leq t \wedge t \leq$ *other.t-stop then other.p* $t -$ *ego.s-stop*
else if other.t-stop $\leq t \wedge t \leq$ *ego.t-stop then other.s-stop* $-$ *ego.p* t
else other.s-stop $-$ *ego.s-stop*
)

lemma *d-diff*: $d\ t =$ *other.s* $t -$ *ego.s* t
⟨*proof*⟩

lemma *collision-d*: *collision* $S \longleftrightarrow (\exists\, t{\in}S.\ d\ t = 0)$

⟨*proof*⟩

lemma *collision-restrict*: *collision* {*0*..} ⟷ *collision* {*0*..*max ego.t-stop other.t-stop*}

⟨*proof*⟩

lemma *collision-union*: *collision* (*A* ∪ *B*) ⟷ *collision A* ∨ *collision B*
⟨*proof*⟩

lemma *symbolic-checker*:
collision {*0*..} ⟷
(*quadroot-in 0* (*min ego.t-stop other.t-stop*) (*1*/*2* ∗ (a_o − a_e)) (v_o − v_e) (s_o − s_e)) ∨
(*quadroot-in ego.t-stop other.t-stop* (*1*/*2* ∗ a_o) v_o (s_o − *ego.s-stop*)) ∨
(*quadroot-in other.t-stop ego.t-stop* (*1*/*2* ∗ a_e) v_e (s_e − *other.s-stop*))
(**is** - ⟷ *?q1* ∨ *?q2* ∨ *?q3*)
⟨*proof*⟩

end

1.5 Checker Design

definition *rel-dist-to-stop* :: *real* ⇒ *real* ⇒ *real* **where**
rel-dist-to-stop v a ≡ − v^2 / (*2* ∗ *a*)

context includes *floatarith-notation* **begin**
definition *rel-dist-to-stop-expr* :: *nat* ⇒ *nat* ⇒ *floatarith* **where**
rel-dist-to-stop-expr v a = *Mult* (*Minus* (*Power* (*Var v*) *2*)) (*Inverse* (*Mult* (*Num 2*) (*Var a*)))

definition *rel-dist-to-stop′* :: *nat* ⇒ *float interval option* ⇒ *float interval option* ⇒ *float interval option* **where**
rel-dist-to-stop′ p v a = *approx p* (*rel-dist-to-stop-expr 0 1*) [*v*, *a*]

lemma *rel-dist-to-stop′*: *interpret-floatarith* (*rel-dist-to-stop-expr 0 1*) [*v*, *a*] = *rel-dist-to-stop v a*
⟨*proof*⟩

definition *first-safe-dist* :: *real* ⇒ *real* ⇒ *real* **where**
first-safe-dist v_e a_e ≡ *rel-dist-to-stop* v_e a_e

definition *second-safe-dist* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
second-safe-dist v_e a_e v_o a_o ≡ *rel-dist-to-stop* v_e a_e − *rel-dist-to-stop* v_o a_o

definition *second-safe-dist-expr* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *floatarith* **where**
second-safe-dist-expr ve ae vo ao = *Add* (*rel-dist-to-stop-expr ve ae*) (*Minus* (*rel-dist-to-stop-expr vo ao*))

definition *second-safe-dist′* :: *nat* ⇒ *float interval option* ⇒ *float interval option*

$\Rightarrow$ *float interval option* $\Rightarrow$ *float interval option* $\Rightarrow$ *float interval option* **where**
second-safe-dist' $p\ v_e\ a_e\ v_o\ a_o$ = *approx* p (*second-safe-dist-expr 0 1 2 3*) [v_e, a_e, v_o, a_o]

lemma *second-safe-dist'*:
interpret-floatarith (*second-safe-dist-expr 0 1 2 3*) [v, a, v', a'] = *second-safe-dist* $v\ a\ v'\ a'$
⟨*proof*⟩

definition *t-stop* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* **where**
t-stop $v\ a \equiv -\ v\ /\ a$

definition *t-stop-expr* :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ *floatarith* **where**
t-stop-expr $v\ a$ = *Minus* (*Mult* (*Var* v) (*Inverse* (*Var* a)))
end

definition *s-stop* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* **where**
s-stop $s\ v\ a \equiv s +$ *rel-dist-to-stop* $v\ a$

definition *discriminant* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* **where**
discriminant $s_e\ v_e\ a_e\ s_o\ v_o\ a_o \equiv (v_o - v_e)^2 - 2 * (a_o - a_e) * (s_o - s_e)$

definition *suff-cond-safe-dist2* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *bool* **where**
suff-cond-safe-dist2 $s_e\ v_e\ a_e\ s_o\ v_o\ a_o \equiv$
let D2 = *discriminant* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o$
in $\neg\ (a_e < a_o \wedge v_o < v_e \wedge 0 \leq D2 \wedge v_e - a_e\ /\ a_o * v_o <$ *sqrt D2*
)

lemma *less-sqrt-iff*: $y \geq 0 \Longrightarrow x <$ *sqrt* $y \longleftrightarrow (x \geq 0 \longrightarrow x^2 < y)$
⟨*proof*⟩

lemma *suff-cond-safe-dist2-code*[*code*]:
suff-cond-safe-dist2 $s_e\ v_e\ a_e\ s_o\ v_o\ a_o$ =
(*let D2* = *discriminant* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o$ *in*
$(a_e < a_o \longrightarrow v_o < v_e \longrightarrow 0 \leq D2 \longrightarrow (v_e - a_e\ /\ a_o * v_o \geq 0 \wedge (v_e - a_e\ /\ a_o * v_o)^2 \geq D2)))$
⟨*proof*⟩

There are two expressions for safe distance. The first safe distance *first-safe-dist* is always valid. Whenever the distance is bigger than *first-safe-dist*, it is guarantee to be collision free. The second one is *second-safe-dist*. If the sufficient condition *suff-cond-safe-dist2* is satisfied and the distance is bigger than *second-safe-dist*, it is guaranteed to be collision free.

definition *check-precond* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *bool* **where**
check-precond $s_e\ v_e\ a_e\ s_o\ v_o\ a_o \longleftrightarrow s_o > s_e \wedge 0 \leq v_e \wedge 0 \leq v_o \wedge a_e < 0 \wedge a_o < 0$

lemma *check-precond-safe-distance*:
check-precond s_e v_e a_e s_o v_o a_o = *safe-distance* a_e v_e s_e a_o v_o s_o
⟨*proof*⟩

1.5.1 Prescriptive Checker

definition *checker* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
checker s_e v_e a_e s_o v_o a_o ≡
let distance = $s_o - s_e$;
precond = *check-precond* s_e v_e a_e s_o v_o a_o;
safe-dist1 = *first-safe-dist* v_e a_e;
safe-dist2 = *second-safe-dist* v_e a_e v_o a_o;
cond2 = *suff-cond-safe-dist2* s_e v_e a_e s_o v_o a_o
in precond ∧ (*safe-dist1* < *distance* ∨ (*safe-dist2* < *distance* ∧ *distance* ≤ *safe-dist1* ∧ *cond2*))

lemma *aux-logic*:
assumes $a \Longrightarrow b$
assumes $b \Longrightarrow a \longleftrightarrow c$
shows $a \longleftrightarrow b \wedge c$
⟨*proof*⟩

theorem *soundness-correctness*:
checker s_e v_e a_e s_o v_o a_o ⟷ *check-precond* s_e v_e a_e s_o v_o a_o ∧ *safe-distance.no-collision* a_e v_e s_e a_o v_o s_o {0..}
⟨*proof*⟩

definition *checker2* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
checker2 s_e v_e a_e s_o v_o a_o ≡
let distance = $s_o - s_e$;
precond = *check-precond* s_e v_e a_e s_o v_o a_o;
safe-dist1 = *first-safe-dist* v_e a_e;
safe-dist2 = *second-safe-dist* v_e a_e v_o a_o;
safe-dist3 = − *rel-dist-to-stop* ($v_o - v_e$) ($a_o - a_e$)
in
if ¬ *precond then False*
else if distance > *safe-dist1 then True*
else if $a_o > a_e \wedge v_o < v_e \wedge v_e - a_e$ / $a_o * v_o < 0$ *then distance* > *safe-dist3*
else distance > *safe-dist2*

theorem *checker-eq-checker2*: *checker* s_e v_e a_e s_o v_o a_o ⟷ *checker2* s_e v_e a_e s_o v_o a_o
⟨*proof*⟩

definition *checker3* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
checker3 s_e v_e a_e s_o v_o a_o ≡
let distance = $s_o - s_e$;
precond = *check-precond* s_e v_e a_e s_o v_o a_o;
s-stop-e = s_e + *rel-dist-to-stop* v_e a_e;

$s\text{-}stop\text{-}o = s_o + rel\text{-}dist\text{-}to\text{-}stop\ v_o\ a_o$
in $precond \wedge (s\text{-}stop\text{-}e < s_o$
$\vee\ (s_o \leq s\text{-}stop\text{-}e \wedge s\text{-}stop\text{-}e < s\text{-}stop\text{-}o \wedge$
$(\neg(a_o > a_e \wedge v_o < v_e \wedge v_e - a_e\ /\ a_o * v_o < 0 \wedge distance * (a_o - a_e) \leq (v_o - v_e)^2\ /\ 2))))$

theorem *checker2-eq-checker3*:
$checker2\ s_e\ v_e\ a_e\ s_o\ v_o\ a_o \longleftrightarrow checker3\ s_e\ v_e\ a_e\ s_o\ v_o\ a_o$
⟨*proof*⟩

1.5.2 Approximate Checker

lemma *checker2-def'*: $checker2\ a\ b\ c\ d\ e\ f = ($
let $distance = d - a;$
$precond = check\text{-}precond\ a\ b\ c\ d\ e\ f;$
$safe\text{-}dist1 = first\text{-}safe\text{-}dist\ b\ c;$
$safe\text{-}dist2 = second\text{-}safe\text{-}dist\ b\ c\ e\ f;$
$C = c < f \wedge e < b \wedge b * f > c * e;$
$P1 = (e - b)^2 < 2 * distance * (f - c);$
$P2 = - b^2\ /\ c + e^2\ /\ f < 2 * distance$
in $precond \wedge (safe\text{-}dist1 < distance \vee$
$safe\text{-}dist1 \geq distance \wedge (C \wedge P1 \vee \neg C \wedge P2)))$
⟨*proof*⟩

lemma *power2-less-sqrt-iff*: $(x::real)^2 < y \longleftrightarrow (y \geq 0 \wedge abs\ x < sqrt\ y)$
⟨*proof*⟩

schematic-goal *checker-form*: $interpret\text{-}form\ ?x\ ?y \Longrightarrow checker\ s_e\ v_e\ a_e\ s_o\ v_o\ a_o$
⟨*proof*⟩

context includes *floatarith-notation* **begin**
definition $checker'\ p\ s_e\ v_e\ a_e\ s_o\ v_o\ a_o = approx\text{-}form\ p$
$(Conj\ (Conj\ (Less\ (Var\ (Suc\ (Suc\ 0)))\ (Var\ (Suc\ (Suc\ (Suc\ 0)))))$
$(Conj\ (LessEqual\ (Var\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))))))$
$(Var\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0)))))))$
$(Conj\ (LessEqual\ (Var\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))))))\ (Var\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))))))$
$(Conj\ (Less\ (Var\ 0)\ (Var\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))))))$
$(Less\ (Var\ (Suc\ 0))\ (Var\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))))))))))$
$(Disj\ (Less\ (Add\ (Var\ (Suc\ (Suc\ 0)))$
$(Mult\ (Minus\ (Power\ (Var\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))))\ 2))\ (Inverse\ (Mult\ (Var\ (Suc\ (Suc\ (Suc\ (Suc\ 0)))))\ (Var\ 0)))))$
$(Var\ (Suc\ (Suc\ (Suc\ 0)))))$
$(Conj\ (LessEqual\ (Var\ (Suc\ (Suc\ (Suc\ 0))))$
$(Add\ (Var\ (Suc\ (Suc\ 0)))$
$(Mult\ (Minus\ (Power\ (Var\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))))\ 2))\ (Inverse\ (Mult\ (Var\ (Suc\ (Suc\ (Suc\ (Suc\ 0)))))\ (Var\ 0))))))$

(*Conj* (*Less* (*Add* (*Var* (*Suc* (*Suc* 0)))
(*Mult* (*Minus* (*Power* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* 0)))))) 2)) (*Inverse* (*Mult* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* 0))))) (*Var* 0)))))
(*Add* (*Var* (*Suc* (*Suc* (*Suc* 0))))
(*Mult* (*Minus* (*Power* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* 0)))))))) 2))
(*Inverse* (*Mult* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* 0))))) (*Var* (*Suc* 0)))))))
(*Disj* (*LessEqual* (*Var* (*Suc* 0)) (*Var* 0))
(*Disj* (*LessEqual* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* 0)))))) (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* 0))))))))
(*Disj* (*LessEqual* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* 0)))))))
(*Add* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* 0))))))
(*Minus* (*Mult* (*Mult* (*Var* 0) (*Inverse* (*Var* (*Suc* 0)))) (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* 0))))))))))
(*Less* (*Mult* (*Power* (*Add* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* 0)))))))) (*Minus* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* 0))))))))) 2)
(*Inverse* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* 0)))))))
(*Mult* (*Add* (*Var* (*Suc* (*Suc* (*Suc* 0)))) (*Minus* (*Var* (*Suc* (*Suc* 0))))) (*Add* (*Var* (*Suc* 0)) (*Minus* (*Var* 0))))))))))))
([a_e, a_o, s_e, s_o, *Interval'* (*Float* 2 0) (*Float* 2 0), v_e, v_o, *Interval'* (*Float* 0 1) (*Float* 0 1)]) (*replicate* 8 0)

lemma *less-Suc-iff-disj*: $i < Suc\ x \longleftrightarrow i = x \vee i < x$
⟨*proof*⟩

lemma *checker'-soundness-correctness*:
assumes $a \in \{real\text{-}of\text{-}float\ al\ ..\ real\text{-}of\text{-}float\ au\}$
assumes $b \in \{real\text{-}of\text{-}float\ bl\ ..\ real\text{-}of\text{-}float\ bu\}$
assumes $c \in \{real\text{-}of\text{-}float\ cl\ ..\ real\text{-}of\text{-}float\ cu\}$
assumes $d \in \{real\text{-}of\text{-}float\ dl\ ..\ real\text{-}of\text{-}float\ du\}$
assumes $e \in \{real\text{-}of\text{-}float\ el\ ..\ real\text{-}of\text{-}float\ eu\}$
assumes $f \in \{real\text{-}of\text{-}float\ fl\ ..\ real\text{-}of\text{-}float\ fu\}$
assumes *chk*: *checker'* *p* (*Interval'* *al* *au*) (*Interval'* *bl* *bu*) (*Interval'* *cl* *cu*) (*Interval'* *dl* *du*) (*Interval'* *el* *eu*) (*Interval'* *fl* *fu*)
shows *checker* *a* *b* *c* *d* *e* *f*
⟨*proof*⟩

lemma *approximate-soundness-correctness*:
assumes $a \in \{real\text{-}of\text{-}float\ al\ ..\ real\text{-}of\text{-}float\ au\}$
assumes $b \in \{real\text{-}of\text{-}float\ bl\ ..\ real\text{-}of\text{-}float\ bu\}$
assumes $c \in \{real\text{-}of\text{-}float\ cl\ ..\ real\text{-}of\text{-}float\ cu\}$
assumes $d \in \{real\text{-}of\text{-}float\ dl\ ..\ real\text{-}of\text{-}float\ du\}$
assumes $e \in \{real\text{-}of\text{-}float\ el\ ..\ real\text{-}of\text{-}float\ eu\}$
assumes $f \in \{real\text{-}of\text{-}float\ fl\ ..\ real\text{-}of\text{-}float\ fu\}$
assumes *chk*: *checker'* *p* (*Interval'* *al* *au*) (*Interval'* *bl* *bu*) (*Interval'* *cl* *cu*) (*Interval'* *dl* *du*) (*Interval'* *el* *eu*) (*Interval'* *fl* *fu*)
shows *checker'-precond*: *check-precond* *a* *b* *c* *d* *e* *f*

and *checker'-no-collision*: *safe-distance.no-collision c b a f e d* $\{0..\}$
⟨*proof*⟩

1.5.3 Symbolic Checker

definition *symbolic-checker* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool*
where
symbolic-checker s_e v_e a_e s_o v_o a_o ≡
let e-stop = − v_e / a_e;
o-stop = − v_o / a_o
in check-precond s_e v_e a_e s_o v_o a_o ∧
(¬*quadroot-in 0* (*min e-stop o-stop*) (*1/2* ∗ (a_o − a_e)) (v_o − v_e) (s_o − s_e) ∧
¬*quadroot-in e-stop o-stop* (*1/2* ∗ a_o) v_o (s_o − *movement.p* a_e v_e s_e *e-stop*)
∧
¬*quadroot-in o-stop e-stop* (*1/2* ∗ a_e) v_e (s_e − *movement.p* a_o v_o s_o *o-stop*))

theorem *symbolic-soundness-correctness*:
symbolic-checker s_e v_e a_e s_o v_o a_o ⟷ *check-precond* s_e v_e a_e s_o v_o a_o ∧
safe-distance.no-collision a_e v_e s_e a_o v_o s_o $\{0..\}$
⟨*proof*⟩
end

end

2 Safe Distance with Reaction Time

theory *Safe-Distance-Reaction*
imports
Safe-Distance
begin

2.1 Normal Safe Distance

locale *safe-distance-normal* = *safe-distance* +
fixes δ :: *real*
assumes *pos-react*: *0* < δ
begin

sublocale *ego2*: *braking-movement* a_e v_e (*ego.q* δ) ⟨*proof*⟩

lemma *ego2-s-init*: *ego2.s 0* = *ego.q* δ ⟨*proof*⟩

definition τ :: *real* ⇒ *real* **where**
τ *t* = *t* − δ

definition τ′ :: *real* ⇒ *real* **where**
τ′ *t* = *1*

lemma τ*-continuous*[*continuous-intros*]: *continuous-on T* τ

⟨*proof*⟩

lemma *isCont-τ*[*continuous-intros*]: *isCont τ x*
⟨*proof*⟩

lemma *del-has-vector-derivative*[*derivative-intros*]: (*τ has-vector-derivative τ′ t*) (*at t within u*)
⟨*proof*⟩

lemma *del-has-real-derivative*[*derivative-intros*]: (*τ has-real-derivative τ′ t*) (*at t within u*)
⟨*proof*⟩

lemma *delay-image*: *τ ‘ {δ..} = {0..}*
⟨*proof*⟩

lemma *s-delayed-has-real-derivative*[*derivative-intros*]:
assumes *δ ≤ t*
shows ((*ego2.s ∘ τ*) *has-field-derivative ego2.s′* (*t − δ*) ∗ *τ′ t*) (*at t within {δ..}*)
⟨*proof*⟩

lemma *s-delayed-has-real-derivative′* [*derivative-intros*]:
assumes *δ ≤ t*
shows ((*ego2.s ∘ τ*) *has-field-derivative* (*ego2.s′ ∘ τ*) *t*) (*at t within {δ..}*)
⟨*proof*⟩

lemma *s-delayed-has-vector-derivative′* [*derivative-intros*]:
assumes *δ ≤ t*
shows ((*ego2.s ∘ τ*) *has-vector-derivative* (*ego2.s′ ∘ τ*) *t*) (*at t within {δ..}*)
⟨*proof*⟩

definition *u* :: *real ⇒ real* **where**
u t = (*if t ≤ 0 then* s_e
else if t ≤ δ then ego.q t
else (*ego2.s ∘ τ*) *t*)

lemma *init-u*: *t ≤ 0 ⟹ u t* = s_e ⟨*proof*⟩

lemma *u-delta*: *u δ* = *ego2.s 0*
⟨*proof*⟩

lemma *q-delta*: *ego.q δ* = *ego2.s 0* ⟨*proof*⟩

definition *u′* :: *real ⇒ real* **where**
u′ t = (*if t ≤ δ then ego.q′ t else ego2.s′* (*t − δ*))

lemma *u′-delta*: *u′ δ* = *ego2.s′ 0*
⟨*proof*⟩

lemma *q′-delta*: *ego.q′ δ = ego2.s′ 0* ⟨*proof*⟩

lemma *u-has-real-derivative*[*derivative-intros*]:
  assumes *nonneg-t*: *t ≥ 0*
  shows (*u has-real-derivative u′ t*) (*at t within* {*0*..})
⟨*proof*⟩

definition *t-stop* :: *real* **where**
  t-stop = ego2.t-stop + δ

lemma *t-stop-nonneg*: *0 ≤ t-stop*
  ⟨*proof*⟩

lemma *t-stop-pos*: *0 < t-stop*
  ⟨*proof*⟩

lemma *t-stop-zero*:
  assumes *t-stop ≤ x*
  assumes *x ≤ δ*
  shows $v_e = 0$
  ⟨*proof*⟩

lemma *u′-stop-zero*: *u′ t-stop = 0*
  ⟨*proof*⟩

definition *u-max* :: *real* **where**
  u-max = u (*ego2.t-stop + δ*)

lemma *u-max-eq*: *u-max = ego.q δ* − v_e^2 / a_e / *2*
⟨*proof*⟩

lemma *u-mono*:
  assumes *x ≤ y y ≤ t-stop*
  shows *u x ≤ u y*
⟨*proof*⟩

lemma *u-antimono*: *x ≤ y ⟹ t-stop ≤ x ⟹ u y ≤ u x*
⟨*proof*⟩

lemma *u-max*: *u x ≤ u-max*
  ⟨*proof*⟩

lemma *u-eq-u-stop*: *NO-MATCH t-stop x ⟹ x ≥ t-stop ⟹ u x = u-max*
⟨*proof*⟩

lemma *at-least-delta*:
  assumes *x ≤ δ*
  assumes *t-stop ≤ x*
  shows *ego.q x = ego2.s* (*x* − *δ*)

⟨*proof*⟩

lemma *continuous-on-u*[*continuous-intros*]: *continuous-on T u*
⟨*proof*⟩

lemma *isCont-u*[*continuous-intros*]: *isCont u x*
⟨*proof*⟩

definition *collision-react* :: *real set* ⇒ *bool* **where**
collision-react time-set ≡ (∃ *t*∈*time-set*. *u t* = *other.s t*)

abbreviation *no-collision-react* :: *real set* ⇒ *bool* **where**
no-collision-react time-set ≡ ¬ *collision-react time-set*

lemma *no-collision-reactI*:
assumes ⋀*t*. *t* ∈ *S* ⟹ *u t* ≠ *other.s t*
shows *no-collision-react S*
⟨*proof*⟩

lemma *no-collision-union*:
assumes *no-collision-react S*
assumes *no-collision-react T*
shows *no-collision-react* (*S* ∪ *T*)
⟨*proof*⟩

lemma *collision-trim-subset*:
assumes *collision-react S*
assumes *no-collision-react T*
assumes *T* ⊆ *S*
shows *collision-react* (*S* − *T*)
⟨*proof*⟩

theorem *cond-1r* : *u-max* < s_o ⟹ *no-collision-react* {*0*..}
⟨*proof*⟩

definition *safe-distance-1r* :: *real* **where**
safe-distance-1r = v_e ∗ δ − $v_e{}^2$ / a_e / *2*

lemma *sd-1r-eq*: (s_o − s_e > *safe-distance-1r*) = (*u-max* < s_o)
⟨*proof*⟩

lemma *sd-1r-correct*:
assumes s_o − s_e > *safe-distance-1r*
shows *no-collision-react* {*0*..}
⟨*proof*⟩

lemma *u-other-strict-ivt*:
assumes *u t* > *other.s t*
shows *collision-react* {*0*..<*t*}

⟨*proof*⟩

lemma *collision-react-subset*: *collision-react s* ⟹ *s* ⊆ *t* ⟹ *collision-react t*
⟨*proof*⟩

lemma *u-other-ivt*:
assumes *u t* ≥ *other.s t*
shows *collision-react* {*0* .. *t*}
⟨*proof*⟩

theorem *cond-2r*:
assumes *u-max* ≥ *other.s-stop*
shows *collision-react* {*0* ..}
⟨*proof*⟩

definition *ego-other2* :: *real* ⇒ *real* **where**
ego-other2 t = *other.s t* − *u t*

lemma *continuous-on-ego-other2*[*continuous-intros*]: *continuous-on T ego-other2*
⟨*proof*⟩

lemma *isCont-ego-other2*[*continuous-intros*]: *isCont ego-other2 x*
⟨*proof*⟩

definition *ego-other2'* :: *real* ⇒ *real* **where**
ego-other2' t = *other.s' t* − *u' t*

lemma *ego-other2-has-real-derivative*[*derivative-intros*]:
assumes *0* ≤ *t*
shows (*ego-other2 has-real-derivative ego-other2' t*) (*at t within* {*0*..})
⟨*proof*⟩

theorem *cond-3r-1*:
assumes *u δ* ≥ *other.s δ*
shows *collision-react* {*0* .. *δ*}
⟨*proof*⟩

definition *distance0* :: *real* **where**
distance0 = $v_e * \delta - v_o * \delta - a_o * \delta^2$ / *2*

definition *distance0-2* :: *real* **where**
distance0-2 = $v_e * \delta + 1$ / *2* * $v_o{}^2$ / a_o

theorem *cond-3r-1'*:
assumes $s_o - s_e \leq$ *distance0*
assumes *δ* ≤ *other.t-stop*
shows *collision-react* {*0* .. *δ*}
⟨*proof*⟩

theorem *distance0-2-eq*:
assumes $\delta > other.t\text{-}stop$
shows $(u\ \delta < other.s\ \delta) = (s_o - s_e > distance0\text{-}2)$
⟨*proof*⟩

theorem *cond-3r-1'-2*:
assumes $s_o - s_e \leq distance0\text{-}2$
assumes $\delta > other.t\text{-}stop$
shows *collision-react* $\{0\ ..\ \delta\}$
⟨*proof*⟩

definition *safe-distance-3r* :: *real* **where**
$safe\text{-}distance\text{-}3r = v_e * \delta - {v_e}^2\ /\ 2\ /\ a_e - v_o * \delta - 1/2 * a_o * \delta^2$

lemma *distance0-at-most-sd3r*:
$distance0 \leq safe\text{-}distance\text{-}3r$
⟨*proof*⟩

definition *safe-distance-4r* :: *real* **where**
$safe\text{-}distance\text{-}4r = (v_o + a_o * \delta - v_e)^2\ /\ 2\ /\ (a_o - a_e) - v_o * \delta - 1/2 * a_o * \delta^2 + v_e * \delta$

lemma *distance0-at-most-sd4r*:
assumes $a_o > a_e$
shows $distance0 \leq safe\text{-}distance\text{-}4r$
⟨*proof*⟩

definition *safe-distance-2r* :: *real* **where**
$safe\text{-}distance\text{-}2r = v_e * \delta - {v_e}^2\ /\ 2\ /\ a_e + {v_o}^2\ /\ 2\ /\ a_o$

lemma *vo-start-geq-ve*:
assumes $\delta \leq other.t\text{-}stop$
assumes $other.s'\ \delta \geq v_e$
shows $u\ \delta < other.s\ \delta$
⟨*proof*⟩

theorem *so-star-stop-leq-se-stop*:
assumes $\delta \leq other.t\text{-}stop$
assumes $other.s'\ \delta < v_e$
assumes $\neg\ (a_o > a_e \wedge other.s'\ \delta < v_e \wedge v_e - a_e\ /\ a_o * other.s'\ \delta < 0)$
shows $0 \leq - {v_e}^2\ /\ a_e\ /\ 2 + (v_o + a_o * \delta)^2\ /\ a_o\ /\ 2$
⟨*proof*⟩

theorem *distance0-at-most-distance2r*:
assumes $\delta \leq other.t\text{-}stop$
assumes $other.s'\ \delta < v_e$
assumes $\neg\ (a_o > a_e \wedge other.s'\ \delta < v_e \wedge v_e - a_e\ /\ a_o * other.s'\ \delta < 0)$
shows $distance0 \leq safe\text{-}distance\text{-}2r$
⟨*proof*⟩

theorem *dist0-sd2r-1*:
assumes $\delta \leq$ *other.t-stop*
assumes $\neg$ ($a_o > a_e \wedge$ *other.s'* $\delta < v_e \wedge v_e - a_e$ / $a_o *$ *other.s'* $\delta < 0$)
assumes $s_o - s_e >$ *safe-distance-2r*
shows $s_o - s_e >$ *distance0*
⟨*proof*⟩

theorem *sd2r-eq*:
assumes $\delta >$ *other.t-stop*
shows (*u-max* < *other.s* δ) = ($s_o - s_e >$ *safe-distance-2r*)
⟨*proof*⟩

theorem *dist0-sd2r-2*:
assumes $\delta > - v_o$ / a_o
assumes $s_o - s_e >$ *safe-distance-2r*
shows $s_o - s_e >$ *distance0-2*
⟨*proof*⟩
end

2.2 Safe Distance Delta

locale *safe-distance-no-collsion-delta* = *safe-distance-normal* +
assumes *no-collision-delta*: *u* $\delta <$ *other.s* δ
begin

sublocale *delayed-safe-distance*: *safe-distance* a_e v_e *ego.q* δ a_o *other.s'* δ *other.s* δ
⟨*proof*⟩

lemma *no-collision-react-initially-strict*:
assumes $s_o \leq$ *u-max*
assumes *u-max* < *other.s-stop*
shows *no-collision-react* {0 <..< δ}
⟨*proof*⟩

lemma *no-collision-react-initially*:
assumes $s_o \leq$ *u-max*
assumes *u-max* < *other.s-stop*
shows *no-collision-react* {0 .. δ}
⟨*proof*⟩

lemma *collision-after-delta*:
assumes $s_o \leq$ *u-max*
assumes *u-max* < *other.s-stop*
shows *collision-react* {0 ..} ⟷ *collision-react* {δ..}
⟨*proof*⟩

lemma *collision-react-strict*:
assumes $s_o \leq$ *u-max*

assumes *u-max* < *other.s-stop*
shows *collision-react* {δ ..} $\longleftrightarrow$ *collision-react* {δ <..}
⟨*proof*⟩

lemma *delayed-other-s-stop-eq*: *delayed-safe-distance.other.s-stop* = *other.s-stop*
⟨*proof*⟩

lemma *delayed-cond3'*:
assumes *other.s* δ ≤ *u-max*
assumes *u-max* < *other.s-stop*
shows *delayed-safe-distance.collision* {*0* ..} $\longleftrightarrow$
($a_o > a_e \wedge$ *other.s'* $\delta < v_e \wedge$ *other.s* $\delta -$ *ego.q* $\delta \leq$ *delayed-safe-distance.snd-safe-distance* $\wedge\ v_e - a_e$ / $a_o *$ *other.s'* $\delta < 0$)
⟨*proof*⟩

lemma *delayed-other-t-stop-eq*:
assumes $\delta \leq$ *other.t-stop*
shows *delayed-safe-distance.other.t-stop* + δ = *other.t-stop*
⟨*proof*⟩

lemma *delayed-other-s-eq*:
assumes *0* ≤ *t*
shows *delayed-safe-distance.other.s t* = *other.s* (*t* + δ)
⟨*proof*⟩

lemma *translate-collision-range*:
assumes $s_o \leq$ *u-max*
assumes *u-max* < *other.s-stop*
shows *delayed-safe-distance.collision* {*0* ..} $\longleftrightarrow$ *collision-react* {δ ..}
⟨*proof*⟩

theorem *cond-3r-2*:
assumes $s_o \leq$ *u-max*
assumes *u-max* < *other.s-stop*
assumes *other.s* $\delta \leq$ *u-max*
shows *collision-react* {*0* ..} $\longleftrightarrow$
($a_o > a_e \wedge$ *other.s'* $\delta < v_e \wedge$ *other.s* $\delta -$ *ego.q* $\delta \leq$ *delayed-safe-distance.snd-safe-distance* $\wedge\ v_e - a_e$ / $a_o *$ *other.s'* $\delta < 0$)
⟨*proof*⟩

lemma *sd-2r-correct-for-3r-2*:
assumes $s_o - s_e >$ *safe-distance-2r*
assumes *other.s* $\delta \leq$ *u-max*
assumes ¬ ($a_o > a_e \wedge$ *other.s'* $\delta < v_e \wedge v_e - a_e$ / $a_o *$ *other.s'* $\delta < 0$)
shows *no-collision-react* {*0*..}
⟨*proof*⟩

lemma *sd2-at-most-sd4*:
assumes $a_o > a_e$

shows *safe-distance-2r* $\leq$ *safe-distance-4r*
⟨*proof*⟩

lemma *sd-4r-correct*:
assumes $s_o - s_e >$ *safe-distance-4r*
assumes *other.s* $\delta \leq$ *u-max*
assumes $\delta \leq$ *other.t-stop*
assumes $a_o > a_e$
shows *no-collision-react* {*0*..}
⟨*proof*⟩

Irrelevant, this Safe Distance is unreachable in the checker.

definition *safe-distance-5r* :: *real* **where**
safe-distance-5r = ${v_e}^2$ / *2* / $(a_o - a_e)$ + ${v_o}^2$ / *2* / a_o + $v_e * \delta$

lemma *sd-5r-correct*:
assumes $s_o - s_e >$ *safe-distance-5r*
assumes *u-max* < *other.s-stop*
assumes *other.s* $\delta \leq$ *u-max*
assumes $\delta >$ *other.t-stop*
shows *no-collision-react* {*0*..}
⟨*proof*⟩

lemma *translate-no-collision-range*:
delayed-safe-distance.no-collision {*0* ..} $\longleftrightarrow$ *no-collision-react* {δ ..}
⟨*proof*⟩

lemma *delayed-cond1*:
assumes *other.s* $\delta >$ *u-max*
shows *delayed-safe-distance.no-collision* {*0* ..}
⟨*proof*⟩

theorem *cond-3r-3*:
assumes $s_o \leq$ *u-max*
assumes *u-max* < *other.s-stop*
assumes *other.s* $\delta >$ *u-max*
shows *no-collision-react* {*0* ..}
⟨*proof*⟩

lemma *sd-2r-correct-for-3r-3*:
assumes $s_o - s_e >$ *safe-distance-2r*
assumes *other.s* $\delta >$ *u-max*
shows *no-collision-react* {*0*..}
⟨*proof*⟩

lemma *sd-3r-correct*:
assumes $s_o - s_e >$ *safe-distance-3r*
assumes $\delta \leq$ *other.t-stop*
shows *no-collision-react* {*0* ..}

⟨*proof*⟩

lemma *sd-2-at-least-sd-3*:
assumes $\delta \leq$ *other.t-stop*
shows *safe-distance-3r* $\geq$ *safe-distance-2r*
⟨*proof*⟩
end

2.3 Checker Design

We define two checkers for different cases:

- one checker for the case that $\delta \leq$ *other.t-stop* (*other.t-stop* $= - v_o$ / a_o)
- a second checker for the case that $\delta >$ *other.t-stop*

definition *check-precond-r1* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
check-precond-r1 s_e v_e a_e s_o v_o a_o $\delta \longleftrightarrow s_o > s_e \wedge 0 \leq v_e \wedge 0 \leq v_o \wedge a_e < 0 \wedge a_o < 0 \wedge 0 < \delta \wedge \delta \leq - v_o$ / a_o

definition *safe-distance0* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
safe-distance0 v_e a_o v_o $\delta = v_e * \delta - v_o * \delta - a_o * \delta^2$ / *2*

definition *safe-distance-1r* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
safe-distance-1r a_e v_e $\delta = v_e * \delta - {v_e}^2$ / a_e / *2*

definition *safe-distance-2r* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
safe-distance-2r a_e v_e a_o v_o $\delta = v_e * \delta - {v_e}^2$ / *2* / $a_e + {v_o}^2$ / *2* / a_o

definition *safe-distance-4r* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
safe-distance-4r a_e v_e a_o v_o $\delta = (v_o + a_o * \delta - v_e)^2$ / *2* / $(a_o - a_e) - v_o * \delta$ − *1* / *2* $* a_o * \delta^2 + v_e * \delta$

definition *safe-distance-3r* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
safe-distance-3r a_e v_e a_o v_o $\delta = v_e * \delta - {v_e}^2$ / *2* / $a_e - v_o * \delta$ − *1* / *2* $* a_o * \delta^2$

definition *checker-r1* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
checker-r1 s_e v_e a_e s_o v_o a_o $\delta \equiv$
let distance $= s_o - s_e$;
precond $=$ *check-precond-r1* s_e v_e a_e s_o v_o a_o δ;
vo-star $= v_o + a_o * \delta$;
t-stop-o-star $= -$ *vo-star* / a_o;
t-stop-e $= - v_e$ / a_e;
safe-dist0 $=$ *safe-distance-1r* a_e v_e δ;
safe-dist1 $=$ *safe-distance-2r* a_e v_e a_o v_o δ;

```
      safe-dist2    = safe-distance-4r a_e v_e a_o v_o δ;
      safe-dist3    = safe-distance-3r a_e v_e a_o v_o δ
  in
    if ¬ precond then False
    else if distance > safe-dist0 ∨ distance > safe-dist3 then True
     else if (a_o > a_e ∧ vo-star < v_e ∧ t-stop-e < t-stop-o-star) then distance >
safe-dist2
    else distance > safe-dist1
```

theorem *checker-r1-correctness*:
$(checker\text{-}r1\ s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \longleftrightarrow check\text{-}precond\text{-}r1\ s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \wedge$
safe-distance-normal.no-collision-react $a_e\ v_e\ s_e\ a_o\ v_o\ s_o\ \delta\ \{0..\})$
⟨*proof*⟩

definition *check-precond-r2* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
$check\text{-}precond\text{-}r2\ s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \longleftrightarrow s_o > s_e \wedge 0 \leq v_e \wedge 0 \leq v_o \wedge a_e < 0 \wedge a_o < 0 \wedge 0 < \delta \wedge \delta > -\ v_o\ /\ a_o$

definition *safe-distance0-2* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
$safe\text{-}distance0\text{-}2\ v_e\ a_o\ v_o\ \delta = v_e * \delta + 1\ /\ 2 * {v_o}^2\ /\ a_o$

definition *checker-r2* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**

```
  checker-r2 s_e v_e a_e s_o v_o a_o δ ≡
    let distance   = s_o − s_e;
    precond    = check-precond-r2 s_e v_e a_e s_o v_o a_o δ;
        safe-dist0 = safe-distance-1r a_e v_e δ;
        safe-dist1 = safe-distance-2r a_e v_e a_o v_o δ
    in
      if ¬ precond then False
      else if distance > safe-dist0 then True
      else distance > safe-dist1
```

theorem *checker-r2-correctness*:
$(checker\text{-}r2\ s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \longleftrightarrow check\text{-}precond\text{-}r2\ s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \wedge$
safe-distance-normal.no-collision-react $a_e\ v_e\ s_e\ a_o\ v_o\ s_o\ \delta\ \{0..\})$
⟨*proof*⟩

Combine the two checkers into one.

definition *check-precond-r* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
$check\text{-}precond\text{-}r\ s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \longleftrightarrow s_o > s_e \wedge 0 \leq v_e \wedge 0 \leq v_o \wedge a_e < 0 \wedge a_o < 0 \wedge 0 < \delta$

definition *checker-r* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**

```
  checker-r s_e v_e a_e s_o v_o a_o δ ≡
    let distance      = s_o − s_e;
```

precond = *check-precond-r* s_e v_e a_e s_o v_o a_o δ;
vo-star = $v_o + a_o * \delta$;
t-stop-o-star = $-$*vo-star* / a_o;
t-stop-e = $-v_e$ / a_e;
t-stop-o = $-v_o$ / a_o;
safe-dist0 = *safe-distance-1r* a_e v_e δ;
safe-dist1 = *safe-distance-2r* a_e v_e a_o v_o δ;
safe-dist2 = *safe-distance-4r* a_e v_e a_o v_o δ;
safe-dist3 = *safe-distance-3r* a_e v_e a_o v_o δ
in
if $\neg$ *precond then False*
else if distance > *safe-dist0 then True*
else if $\delta \leq$ *t-stop-o* $\wedge$ *distance* > *safe-dist3 then True*
else if $\delta \leq$ *t-stop-o* $\wedge$ ($a_o > a_e \wedge$ *vo-star* $< v_e \wedge$ *t-stop-e* < *t-stop-o-star*) *then distance* > *safe-dist2*
else distance > *safe-dist1*

theorem *checker-eq-1*:
checker-r s_e v_e a_e s_o v_o a_o $\delta \wedge \delta \leq -$ v_o / a_o $\longleftrightarrow$ *checker-r1* s_e v_e a_e s_o v_o a_o δ
⟨*proof*⟩

theorem *checker-eq-2*:
checker-r s_e v_e a_e s_o v_o a_o $\delta \wedge \delta >$ $-$ v_o / a_o $\longleftrightarrow$ *checker-r2* s_e v_e a_e s_o v_o a_o δ
⟨*proof*⟩

theorem *checker-r-correctness*:
(*checker-r* s_e v_e a_e s_o v_o a_o δ $\longleftrightarrow$ *check-precond-r* s_e v_e a_e s_o v_o a_o $\delta \wedge$ *safe-distance-normal.no-collision-react* a_e v_e s_e a_o v_o s_o δ {0..})
⟨*proof*⟩

end

3 Evaluation

theory *Evaluation*
imports
Safe-Distance
HOL−Library.Float
begin

3.1 Code Generation Setup for Numeric Values

definition *real-div-down* :: *nat* ⇒ *int* ⇒ *int* ⇒ *real* **where**
real-div-down p i j = *truncate-down* (*Suc p*) (*i* / *j*)

definition *real-div-up* :: *nat* ⇒ *int* ⇒ *int* ⇒ *real* **where**
real-div-up p i j = *truncate-up* (*Suc p*) (*i* / *j*)

context includes *float.lifting* **begin**
lift-definition *float-div-down* :: *nat* ⇒ *int* ⇒ *int* ⇒ *float* **is** *real-div-down*
⟨*proof*⟩

lift-definition *float-div-up* :: *nat* ⇒ *int* ⇒ *int* ⇒ *float* **is** *real-div-up*
⟨*proof*⟩
end

lemma *compute-float-div-up*[*code*]: *float-div-up p i j* = − *float-div-down p* (−*i*) *j*
including *float.lifting*
⟨*proof*⟩

lemma *compute-float-div-down*[*code*]:
float-div-down prec m1 m2 = *lapprox-rat* (*Suc prec*) *m1 m2*
including *float.lifting* ⟨*proof*⟩

definition *real2-of-real* :: *nat* ⇒ *real* ⇒ (*real* ∗ *real*) **where**
real2-of-real p x = (*truncate-down* (*Suc p*) *x*, *truncate-up* (*Suc p*) *x*)

context includes *float.lifting* **begin**
lift-definition *float2-of-real* :: *nat* ⇒ *real* ⇒ *float* × *float* **is** *real2-of-real*
⟨*proof*⟩
end

definition *float2-opt-of-real* :: *nat* ⇒ *real* ⇒ *float interval option* **where**
float2-opt-of-real prec x = *Interval′* (*fst* (*float2-of-real prec x*)) (*snd* (*float2-of-real prec x*))

hide-const (**open**) *Fraction-Field.Fract*
lemma *real-of-rat-Fract*[*simp*]: *real-of-rat* (*Fract a b*) = *a* / *b*
⟨*proof*⟩

lemma [*code*]: *float2-of-real p* (*Ratreal r*) =
(*let* (*a*, *b*) = *quotient-of r in*
(*float-div-down p a b*, *float-div-up p a b*))
including *float.lifting*
⟨*proof*⟩

fun *real-of-dec* :: *integer* × *integer* ⇒ *real* **where**
real-of-dec (*m*, *e*) =
real-of-int (*int-of-integer m*) ∗
(*if e* ≥ *0 then 10* ^ (*nat-of-integer e*) *else inverse* (*10* ^ (*nat* (−(*int-of-integer e*)))))

lemma *real-of-dec* (*m*, *e*) = *int-of-integer m* ∗ *10 powr* (*int-of-integer e*)
⟨*proof*⟩

3.2 Data Evaluation

definition *trans6* **where**
trans6 c chk se ve ae so vo ao =
chk (*c se*) (*c ve*) (*c ae*) (*c so*) (*c vo*) (*c ao*)

definition *checker-dec* **where**
checker-dec chk p u =
trans6 (*float2-opt-of-real* (*nat-of-integer u*) *o real-of-dec*) (*chk* (*nat-of-integer p*))

definition *checker-interval* = *checker-dec checker'*
definition *checker-symbolic* = *trans6 real-of-dec symbolic-checker*
definition *checker-rational* = *trans6 real-of-dec checker*
lemmas[*code*] = *movement.p-def*

⟨*ML*⟩

The precision of the input data is roughly 12 and yields similar performance as Sturm

⟨*ML*⟩
Number of data points:

- data01: 1121215
- data02: 1341135
- data03: 1452656

⟨*ML*⟩
Precision: 12, Uncertainty: 7 digits
⟨*ML*⟩

end

References

[1] A. Rizaldi, F. Immler, and M. Althoff. A formally verified checker of the safe distance traffic rules for autonomous vehicles. In S. Rayadurgam and O. Tkachuk, editors, *NASA Formal Methods*, pages 175–190, Cham, 2016. Springer International Publishing.