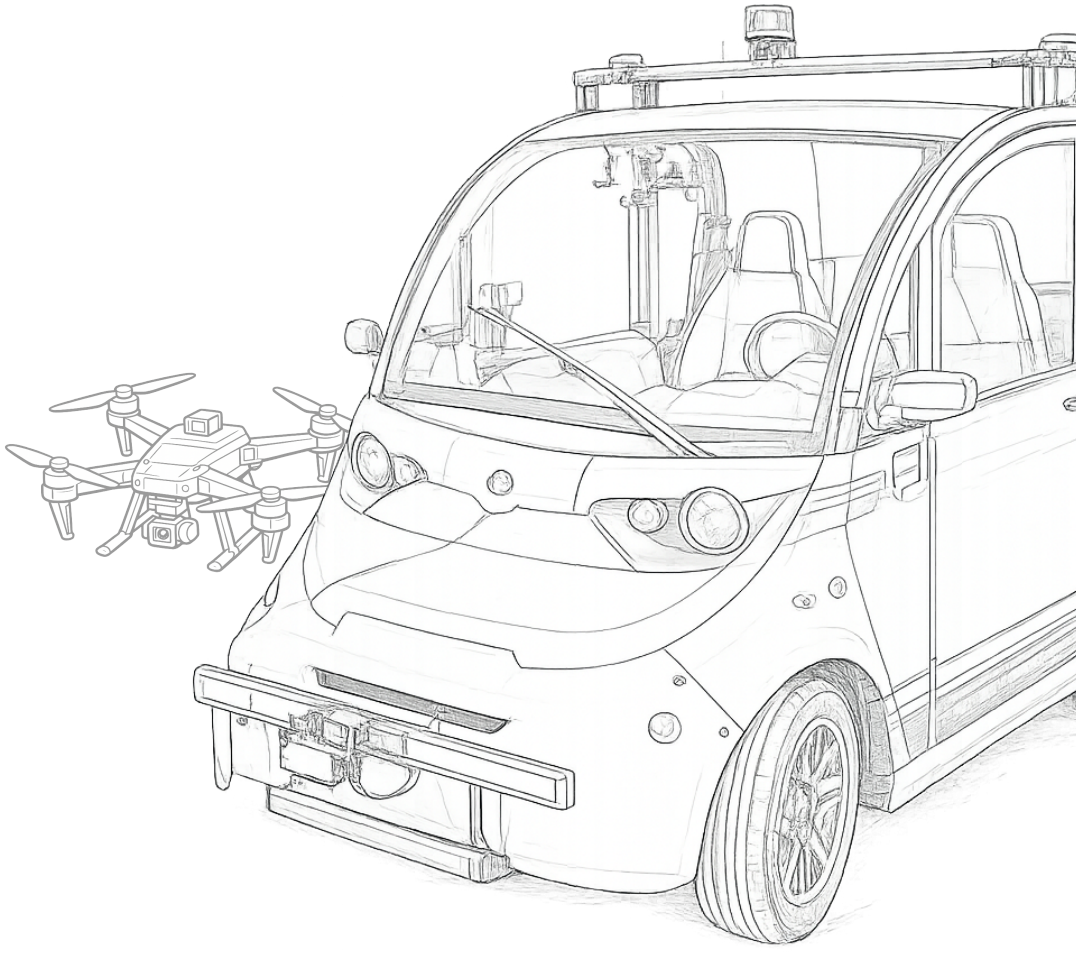
Course Reader Principles of Safe Autonomy (ECE484)



Sayan Mitra and others

Course Reader Principles of Safe Autonomy (ECE484)

Contents

Preface

Course Reader for Principles of Safe Autonomy prepared 2022-26. Eternally grateful to Pranav Sriram, Akshunna Vaishnav, Edward Guo, Arjun Ray, Chenhui Zhang, John Pohovey for editing and carefully reading the notes.

Sayan Mitra, Urbana
January 2026

1 Background

In this chapter, we review basic concepts in linear algebra, logic, probability. These concepts are used throughout the rest of the notes.

1.1 Coordinate frames and rotations

We often describe the position and orientation of a vehicle using multiple coordinate frames. The center of gravity frame is convenient for writing the equations of motion, the camera frame is convenient for vision-based sensing, on the other hand the world frame is convenient for navigation.

This section introduces the basic notation for frames, rotation matrices, and how to transform coordinates between frames in 2D and 3D.

Let W denote a world frame with origin O_W and orthonormal axes $(\hat{e}_x^W, \hat{e}_y^W, \hat{e}_z^W)$. Let B denote a body frame attached to a vehicle with origin O_B and axes $(\hat{e}_x^B, \hat{e}_y^B, \hat{e}_z^B)$. For a point p, its coordinates in the two frames are written as $p^W = [x^W, y^W, z^W]^\top$ and $p^B = [x^B, y^B, z^B]^\top$.

In order to convert coordinates from body frame B to world frame W, we need to know the position of the body frame origin O_B expressed in world coordinates. Let $p_{O_B}^W \in \mathbb{R}^3$ be the coordinates of O_B in the world frame; this vector gives the displacement from O_W to O_B expressed in W. We also need to know the orientation of the body frame relative to the world frame. For this purpose, we define the rotation matrix $R_{WB} \in \mathbb{R}^{3\times 3}$ by taking the body axes, moving them to the world origin (translation does not change direction), expressing them in world coordinates, and stacking the resulting vectors as columns:

$$R_{WB} = \begin{bmatrix} \hat{e}_x^B & \hat{e}_y^B & \hat{e}_z^B \end{bmatrix}_W, \quad (R_{WB})_{ij} = \hat{e}_i^W \cdot \hat{e}_j^B. \tag{1.1}$$

Exercise 1.1. (a) Using the definition of rotation matrices given above construct the 2D-rotation matrix R_{BA} from frame A to frame B, where frame A is rotated by angle θ from frame B. (b) For a point $p^A = [3, -5]^\top$ in frame A and $\theta = \frac{\pi}{3}$, compute the coordinates $p^B = R_{BA}p^A$. (c) Compute $R_{BA}^\top R_{BA}$, what do you observe? (d) Could this calculation

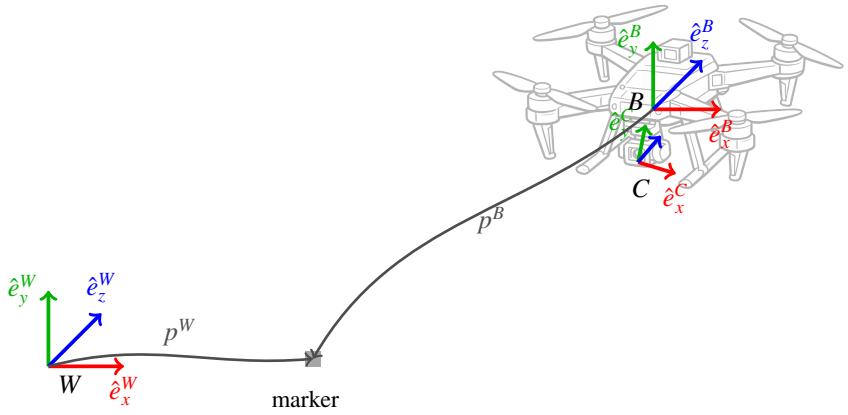


Figure 1.1
A point p expressed in world and body frames.

go wrong on a computer? Why? (e) Compute the inverse $R_{AB} = R_{BA}^{-1}$ and check that $R_{AB} = R_{BA}^{\top}$. (f) Compute p^A from p^B using one of the rotation matrices.

Solution. (a) In 2D, the axes of frame A expressed in frame B coordinates are $\hat{e}_x^A = [\cos\theta,\ \sin\theta]^\top$ and $\hat{e}_y^A = [-\sin\theta,\ \cos\theta]^\top$. Stacking them gives the rotation matrix

$$R_{BA} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

(b) With $\theta = \pi/3$, $c = \cos\theta = 1/2$ and $s = \sin\theta = \sqrt{3}/2$. Then

$$p^B = R_{BA}p^A = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}\begin{bmatrix} 3 \\ -5 \end{bmatrix} = \begin{bmatrix} 3c + 5s \\ 3s - 5c \end{bmatrix} = \begin{bmatrix} \frac{3+5\sqrt{3}}{2} \\ \frac{3\sqrt{3}-5}{2} \end{bmatrix}.$$

(c) $R_{BA}^\top R_{BA} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} c & -s \\ s & c \end{bmatrix} = \begin{bmatrix} c^2+s^2 & 0 \\ 0 & c^2+s^2 \end{bmatrix} = I$ since $c^2 + s^2 = 1$. This is an important property of rotation matrices called *orthonormality*. This implies that rotation matrices are invertible with inverse equal to the transpose, i.e., $R_{BA}^{-1} = R_{BA}^\top$. (d) In practice, due to floating-point round-off errors, the computed value of a rotation matrix may not be exactly orthonormal.

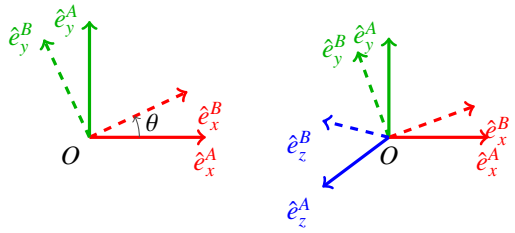


Figure 1.2
Two orthonormal frames at the same origin: world axes (solid) and body axes (dashed) rotated by an angle.

(e) The inverse is the transpose:

$$R_{AB} = R_{BA}^{-1} = R_{BA}^{\top} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}.$$

(f) Using $p^A = R_{AB}p^B$ with $c = 1/2$, $s = \sqrt{3}/2$ and $p^B = [\frac{3+5\sqrt{3}}{2},\ \frac{3\sqrt{3}-5}{2}]^{\top}$,

$$p^A = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} \frac{3+5\sqrt{3}}{2} \\ \frac{3\sqrt{3}-5}{2} \end{bmatrix} = \begin{bmatrix} \frac{3+5\sqrt{3}}{4} + \frac{9-5\sqrt{3}}{4} \\ -\frac{3\sqrt{3}+15}{4} + \frac{3\sqrt{3}-5}{4} \end{bmatrix} = \begin{bmatrix} 3 \\ -5 \end{bmatrix}.$$

This exercise illustrates several important facts about 2D rotation matrices.

(1) Each column of R_{BA} is one of the axes of frame A expressed in frame B coordinates. Since the columns of such a matrix are linearly independent vectors, rotation matrices are invertible.
(2) For coordinate frames that differ by a rotation only (same origin), the mapping between coordinates is simply $p^B = R_{BA}p^A$. Remember the notational convention: reading the suffixes of R_{BA} from right to left gives the mapping from frame A to frame B.
(3) Rotation matrices are orthonormal, i.e., $R_{BA}^{\top}R_{BA} = I$.
(4) Orthonormality and invertibility imply that the inverse is the transpose, i.e., $R_{BA}^{-1} = R_{BA}^{\top}$.
(5) The inverse mapping is $p^A = R_{AB}p^B = R_{BA}^{\top}p^B$.

All of these facts extend to 3D coordinate frames as well and the next exercise illustrates this.

Exercise 1.2. Consider two 3D coordinate frames W (world) and B (body). Let the body axes expressed in the world frame be

$$\hat{e}_x^B = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}, \quad \hat{e}_y^B = \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}, \quad \hat{e}_z^B = \begin{bmatrix} \frac{1}{2} \\ -\frac{1}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}.$$

. Let $p_{O_B}^W = [1,\ -2,\ 0]^\top$ be the displacement of the body frame from the world frame and consider a point with coordinates $p^B = [2,\ 1,\ -1]^\top$ in the body frame.

(a) Construct R_{WB} and check the orthonormality of the column vectors.

(b) Unlike the coordinate transformation for frames that are both at the same origin, here we need to account for the displacement of the body frame from the world frame. The transform from body to world coordinates is given by:

$$p^W = p_{O_B}^W + R_{WB} p^B.$$

Compute the world coordinates p^W of the point p^B using this formula.

(c) Write the inverse transform from world to body coordinates.

(d) Compute p^B back from p^W using $p^B = R_{WB}^\top (p^W - p_{O_B}^W)$ and verify it matches the original p^B.

Solution. (a) Stacking the axes gives

$$R_{WB} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}.$$

(b) First compute

$$R_{WB} p^B = 2\hat{e}_x^B + \hat{e}_y^B - \hat{e}_z^B = \begin{bmatrix} \sqrt{2} - 1 \\ \sqrt{2} + 1 \\ 0 \end{bmatrix},$$

so

$$p^W = p_{O_B}^W + R_{WB} p^B = \begin{bmatrix} \sqrt{2} \\ \sqrt{2} - 1 \\ 0 \end{bmatrix}.$$

(c) Let $v = p^W - p^W_{O_B} = [\sqrt{2} - 1,\ \sqrt{2} + 1,\ 0]^\top$. Then

$$R_{WB}^\top v = \begin{bmatrix} \hat{e}_x^B \cdot v \\ \hat{e}_y^B \cdot v \\ \hat{e}_z^B \cdot v \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix} = p^B.$$

Transforming coordinates from frame B to frame A that differ by both rotation and translation requires knowing the both the position $p^A_{O_B}$ of the B's origin in A coordinates and the axes $\hat{e}_x^B$, $\hat{e}_y^B$, $\hat{e}_z^B$ expressed in A coordinates. From these, we construct the rotation matrix R_{AB} and use the transform

$$p^A = p^A_{O_B} + R_{AB}\, p^B.$$

This transformation can be interpreted as first rotating the point p^B into frame A using R_{AB}, then translating it by $p^A_{O_B}$ to account for the displacement of the origins. This is an affine transformation and can be written as linear transformation by using the trick of homogenenous coordinates, i.e., by augmenting both the p^A and p^B vectors and the transformation itself by 1:

$$\begin{bmatrix} p^A \\ 1 \end{bmatrix} = \begin{bmatrix} R_{AB} & p^A_{O_B} \\ 0^\top & 1 \end{bmatrix} \begin{bmatrix} p^B \\ 1 \end{bmatrix} = T_{AB} p^B.$$

The inverse coordinate transform from A to B is:

$$p^B = R_{AB}^\top (p^A - p^A_{O_B}).$$

Such transforms are widely used in robotics and computer vision. Multiple coordinate transformations can be chained together to express points from world frame to camera frame via the body frame, for example.

1.2 Exercises

Exercise 1.3. Let $R(\psi)$ be the 2D rotation matrix. Show that $R(\psi)^\top = R(-\psi)$ and that $R(\psi)^\top R(\psi) = I$.

Exercise 1.4. The body frame origin is at $p^W_{O_B} = [2,\ -1]^\top$ and the body is rotated by $\psi = \pi/6$. A point has body coordinates $p^B = [1,\ 0.5]^\top$. Compute p^W.

2 Safety

2.1 Data and models for safety

When we say that a system, such as a racing drone is safe, informally, we mean to convey that certain bad things, such as the drone crashing into a gate, almost never happens. In this chapter, we will make this notion of safety more precise and discuss mathematical methods for checking safety of autonomous systems. This discussion will also expose you to mathematical models of autonomous systems, different components in such models, and important modeling choices.

The common approach for checking any system or program, is through *testing*. It is common for 30-40% of development time for a product to be spent on Before developing a systems you must define what it means for the system to behave correctly and these are called the *requirements* of the system. Safety is a particular and important kind of requirement. A single *test* for a system $\mathcal{A}$ runs the system under a particular initial condition, and with a particular sequence of inputs, and observes whether the resulting run or *execution* passes or fails the requirements. For an autonomous car, testing or generating an execution means either test driving the car on the road or running the car's software in a simulator. Requirements here could be, for example, that "the ego car does not collide the the lead car" or "ego car does not leave the lane boundaries".

What we can and cannot learn from tests? If a test violates a requirement, that can give useful information. For example, it can pinpoint the particular input conditions (speed, road, lighting, traffic, etc.) that led to the collision. It can also help identify the bug in the code or the design of the software that needs to be fixed. Such a test or an execution of the system $\mathcal{A}$ is called *a counter-example* for the requirement.

While a finite set of tests can be used to show that a requirement is *not* satisfied, they cannot *prove* that such a requirement is satisfied for all executions of $\mathcal{A}$. This is because $\mathcal{A}$ can have infinitely many executions, even if it is a finite state system (why?).

"Testing can be used to show the presence of bugs, but never to show their absence!"

— Edsger W. Dijkstra

Assumtions or models can help bridge the gap. To learn or extrapolate about all—infinitely many—executions of $\mathcal{A}$ form a finite sampling of executions, we need to make some *assumptions* about $\mathcal{A}$. A collection of these mathematical assumptions defines a *model* $\hat{A}$ for $\mathcal{A}$.

Benefits of safety verification A model $\hat{\mathcal{A}}$ for $\mathcal{A}$ may be *proved* to satisfy the requirements. This proof does not "prove" anything for $\mathcal{A}$ (because of the "model to reality gap"), but it can serve as evidence and explanation for why $\mathcal{A}$ might satisfy the requirements. Such proofs can also be part of the certification package for $\mathcal{A}$. Indeed, safety standards for automotive and aerospace systems (e.g., DO178C) allow and encourage such models and their proofs as part of the certification process. Furthermore, the process of certification of invariably leads to more effective testing or usage of execution data, and identification of the assumptions under which $\mathcal{A}$ can be expected to be safe. This is possibly the most important benefit of modeling and safety analysis.

2.2 Automata, State machines

Definition 2.1. A *state machine* $\mathcal{A}$ is defined by (1) a set of states Q, (2) a set of *start states* $Q_0 \subseteq Q$, and (2) a set of transitions $\mathcal{D} \subseteq Q \times Q$.

Example 2.1 Consider a simple traffic light controller at an intersection. The traffic light can be in one of three states: Green (G), Yellow (Y), and Red (R). The initial state is G. The transitions are as follows: from G it can go to Y, from Y it can go to R, and from R it can go back to G. Thus, the automaton TrafficLight is defined by:

- $Q = \{G, Y, R\}$
- $Q_0 = \{G\}$
- $\mathcal{D} = \{(G, Y), (Y, R), (R, G)\}$

□

An *execution* for an automaton $\mathcal{A}$ is a particular run of that automaton. Mathematically, an execution is a (possibly infinite) sequence of states $q_0, q_1, \ldots$, such that $q_0 \in Q_0$ is a start state and $(q_i, q_{i+1}) \in \mathcal{D}$ is a valid transition of $\mathcal{A}$.

The TrafficLight automaton has a essentially a single execution: $G, Y, R, G, Y, R, \ldots$. Such an execution is called *infinite*. An execution can also be *finite*, for example, G, Y, R is a finite execution of length 3 for TrafficLight.

An automaton is said to be *finite* if its set of states Q is finite. Otherwise, it is *infinite*. An automaton is *deterministic* if for every state $q \in Q$ there is at most one state $q' \in Q$ such that $(q, q') \in \mathcal{D}$. Otherwise, it is *nondeterministic*. TrafficLight is a finite deterministic automaton.

Is testing a finite deterministic automaton trivial? Not always. Consider an automaton with $|Q| = 10^{50}$ states. Even if it is deterministic, testing its execution which may visit all its states is not necessarily feasible. Testing can be hard even for seemingly simple deterministic automata.

Example 2.2 [Collatz automaton] Consider the following automaton $\mathcal{A}$ over the state space $Q = \mathbb{N}$ (the set of natural numbers). The initial states are $Q_0 = \{q_0\}$ for some $q_0 \in \mathbb{N}$. The transitions $\mathcal{D}$ are defined as follows: for any state $n \in Q$,

$$(n, n/2) \in \mathcal{D} \text{ if } n \text{ is even, and } (n, 3n + 1) \in \mathcal{D} \text{ if } n \text{ is odd.}$$

For any given initial state q_0, this automaton has a unique execution. For example, if $q_0 = 6$ then the execution is

$$3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, \ldots$$

which eventually loops on 4, 2, 1. However, if $q_0 = 7$ then the execution is

$$7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, \ldots$$

which is much longer before it reaches the loop. Determining whether the execution for any given q_0 eventually reaches the loop 4, 2, 1 is an open problem in mathematics known as the Collatz conjecture. □

Explicitly enumerating the states and transitions of an automaton can become unwieldy very quickly. So, we will use variables and programs to describe state machines. Actually, programs *are* state machines where the valuations of the variables are the states.

Example 2.3 [Cruise control] Consider two cars moving on a straight road segment with initial positions $x_{20} > x_{10} > 0$, and velocities $v_{10} > v_{20} \geq 0$. Car 1 has a sensor with range $d_s > 0$ and it brakes with a deceleration $a_b > 0$. Under what assumptions on the model parameters can we show that there is no collision?

States. We assume that the cars have no lateral motion. So, the important state variables here are the positions and the velocities of the two cars $x_1, x_2, v_1, v_2 \in \mathbb{R}$ along the road, which we choose to be in the positive direction of the x-axis. Actually, v_2 never changes and we can make it into a constant parameter of the model instead of a state variable. So, the state space Q for this automaton $\mathcal{A}$ is $Q = \mathbb{R}^3$. For any vector $\mathbf{x} \in Q$, we can stack the three state components to be in some fixed order, say $\mathbf{x} = \langle x_1, x_2, v_1 \rangle$. Notice the state space is uncountably infinite.

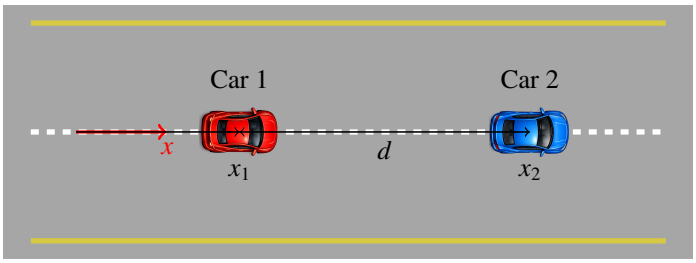


Figure 2.1
One-dimensional cruise control with Car 1 following Car 2.

Initial states. The initial states Q_0 of $\mathcal{A}$ is defined simply as

$$Q_0 := \{\mathbf{x} \in Q \mid \mathbf{x}.x_1 = x_{10}, \mathbf{x}.x_2 = x_{20}, \mathbf{x}.v_1 = v_{10}\}.$$

Here $|Q_0| = 1$, but in general it need not be finite or countable.

State transitions. The transitions describe how the state variables change over unit time. In this example, the positions are updated according to the velocities. The velocity of Car 2 remains constant. The velocity of Car 1 is updated based on whether Car 2 is within the sensor range d_s or not. If Car 2 is within the sensor range, then Car 1 brakes with deceleration a_b. Otherwise, it maintains its velocity. Thus, $\mathcal{D} \subseteq Q \times Q$ can be described by the following code snippet:

$$\begin{aligned} v_1' &= \begin{cases} \max(0, v_1 - a_b), & x_2 - x_1 < d_s, \\ v_1, & x_2 - x_1 \geq d_s, \end{cases} \\ x_2' &= x_2 + v_2, \\ x_1' &= x_1 + v_1'. \end{aligned}$$

What this means is that the set of transitions is

$$\begin{aligned} \mathcal{D} := \{(\mathbf{x}, \mathbf{x}') \in Q \times Q \mid \mathbf{x}'.x_2 = \mathbf{x}.x_2 + \mathbf{x}.v_2 \wedge \mathbf{x}'.x_1 = \mathbf{x}.x_1 + \mathbf{x}'.v_1 \\ (\mathbf{x}.x_2 - \mathbf{x}.x_1 < d_s) \Rightarrow (\mathbf{x}'.v_1 := \max(0, \mathbf{x}.v_1 - a_b)) \\ (\mathbf{x}.x_2 - \mathbf{x}.x_1 \geq d_s) \Rightarrow (\mathbf{x}'.v_1 := \mathbf{x}.v_1)\}. \end{aligned}$$

Now we have completely defined the automaton $\mathcal{A}$ for this example. By the way, although this model is very simple, one-dimensional scenarios are commonly used for safety assurance arguments for cars and even aircraft landing ISO (2011); Fabris (2012); Perry et al. (2013). □

The automaton in Example 2.3 is deterministic. Do you see why? For any state $\mathbf{x}$ the unique next state can be written as $\mathbf{x}' = f(\mathbf{x})$, were f the function in the code snippet shown above. This is not that realistic.

Exercise 2.1. Rewrite the code snippet to make the automaton in Example 2.3 nondeterministic. For example, you can rewrite the assignment of v_1 as to account for a range of deceleration values.

$$v_1 := \textbf{ choose } [\max(0, v_1 - a_b - \varepsilon), \max(0, v_1 - a_b)]$$

How can we accommodate a sensor that sometimes fails? Or one that works only in some range of relative velocities? A situation where Car 2 changes its velocity arbitrarily from a range? For each of these cases, write the new $\mathcal{D}$.

2.3 Safety and requirements

Colloquially, "safety" is often used as a catch-all for *requirements of an autonomous system*. More precisely however, a *requirement* for a system $\mathcal{A}$ is a property that all executions of $\mathcal{A}$ must meet or satisfy. A *safety requirement* is a particular type of requirement that states that "something bad never happens" during any execution of $\mathcal{A}$. In other words, a safety requirement forbids certain bad states from being reached at any point in time during any execution of $\mathcal{A}$. We already saw an example of a safety requirement that an "ego car should never collide with a lead car". An example of a bad state here is any state where the distance between the two cars is less than or equal to zero. An example of a requirement that is not a safety requirement is the state n eventually reaches the loop $4, 2, 1$ in the Collatz automaton in Example 2.1.

Example 2.4 Some examples of requirements written in English are:

- *A car should never come within 0.5m of another car.*
- *A car should never exceed the speed-limit.*
- *A car entering an intersection should exit it within 20 seconds.*
- *A car should drive at least 28 miles per gallon (mpg) over any interval in its operating life.*

The first three are related to driving safety. The last one is about performance and environmental impact. All are requirements of the system. □

Given an execution α we may refer to the i^{th} state in α as $\alpha[i] = q_i$. The first requirement above for our example with two cars can be written as:

$$\forall\ \alpha, \forall\ k, \alpha[k].x_2 - \alpha[k].x_1 > 0.5.$$

Notice the quantification over all executions α and over all states $\alpha[k]$ in α.

Example 2.5 [Unicycle lane keeping] We use the discrete-time unicycle model of an ego vehicle (E) following a lead vehicle (L) on a straight lane of width w. To define this model, first, we specify the state variables and then the state transitions.

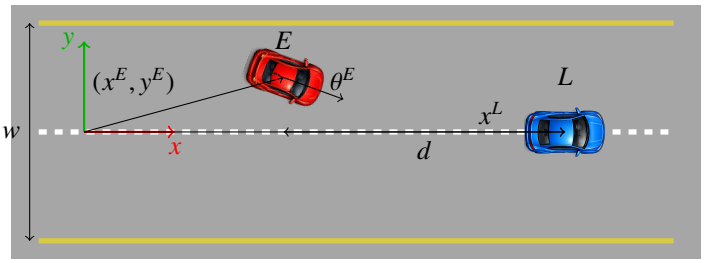


Figure 2.2
Lane-keeping setup with ego vehicle E and lead vehicle L, showing the state variables and parameters.

States. We assume that the lane is aligned with the world x-axis and the lead car L is ahead of the ego car E and moves on the lane centerline. For simplicity, we model only the longitudinal position of the lead car. For the ego car E, the state is defined by its position and heading measured counterclockwise from the lane centerline. Thus, the full state of the system is the vector $\mathbf{x} = [x^E, y^E, \theta^E, x^L]^\top$ and the state space Q of the system is $\mathbb{R}^4$.

Initial states. We set the initial state on the lane centerline with aligned heading, $y_0^E = 0$ and $\theta_0^E = 0$, and assume the lead car is ahead with $x_0^L - x_0^E > d_{\min}$. This defines the set of initial states $Q_0 \subseteq Q$

$$Q_0 = \{(x^E, y^E, \theta^E, x^L) \in Q \mid y^E = 0, \theta^E = 0, x^L - x^E > d_{\min}\}. \tag{2.1}$$

Concept break. A boolean formula involving the state variables, such as $y^E = 0 \wedge \theta^E = 0 \wedge x^L - x^E > d_{\min}$ as in the above definition of Q_0, defines a subset of the state space. A boolean formula, also called a **predicate** P evaluates true or false for each state $\mathbf{x} \in Q$, i.e., $P : Q \to \mathbb{B}$, and defines the subset $\{\mathbf{x} \in Q \mid P(\mathbf{x}) = \mathit{true}\}$.

Safety requirements. We want to ensure a minimum longitudinal gap $d_{\min}$ between the two cars is *always* maintained, and that the ego car E *always* stays within the $w/2$ lateral bounds of the centerline. These are requirements that must *never* be violated. Mathematically, these requirements can be expressed as safe or unsafe subsets of Q or as predicates over Q.

Lane safety	$\lvert y^E \rvert \le w/2$	$\mathcal{S}_{\text{inlane}} = \{(x^E, y^E, \theta^E, x^L) \mid \lvert y^E \rvert \le w/2\}.$
Gap	$x^L - x^E \ge d_{\min}$	$\mathcal{S}_{\text{gap}} = \{(x^E, y^E, \theta^E, x^L) \mid x^L - x^E \ge d_{\min}\}.$

State transitions. The state transitions define how the state variables change over one time step $\Delta t > 0$. We will describe the state transitions in three parts: (1) the ego car's controller decides the translational velocity v^E and the rotational velocity ω^E based on current distances as sensed by E, (2) the lead car updates its position based on its bounded velocity v_k^L, and (3) the ego car updates its position.

The ego car's controller uses several parameters: $v_{\max}^E$ (maximum speed), $\omega_{\max}^E$ (maximum yaw rate), and k_y (lateral feedback gain). The controller implements a bang-bang speed control to maintain the minimum gap $d_{\min}$, and a deviation-based heading controller that steers the car back toward the centerline when $y_k^E \neq 0$. We will learn more about designing such controllers in later lectures. Here, we simply state the control law. In our model, these equations represent how the software or the brains of the ego car decide on the controls at each time step based on sensed inputs such as d_k (gap) and y_k^E (lateral position).

$$d_k = x_k^L - x_k^E, \tag{2.2}$$

$$d_{\text{close}} = d_{\min} + v_{\max}^E \Delta t, \tag{2.3}$$

$$\begin{aligned} v_k^E &= \begin{cases} 0, & |y_k^E| \geq w/2, \\ 0, & d_k \leq d_{\text{close}}, \\ v_{\max}^E, & \text{otherwise}, \end{cases} \\ \theta_{k+1}^E &= -k_y\, y_k^E, \qquad |\theta_{k+1}^E| \leq \pi/2, \\ \omega_k^E &= (\theta_{k+1}^E - \theta_k^E)/\Delta t. \end{aligned} \tag{2.4}$$

The plant dynamics for both cars are given by the discrete-time unicycle equations:

$$\begin{aligned} x_{k+1}^L &= x_k^L + \Delta t\, v_k^L, \qquad v_k^L \in [0, v_{\max}^L]. \\ x_{k+1}^E &= x_k^E + \Delta t\, v_k^E \cos(\theta_k^E), \\ y_{k+1}^E &= y_k^E + \Delta t\, v_k^E \sin(\theta_k^E), \\ \theta_{k+1}^E &= \theta_k^E + \Delta t\, \omega_k^E. \end{aligned} \tag{2.5}$$

These equations define how the state $\mathbf{x}_k$ updates to $\mathbf{x}_{k+1}$ at each time step k. Notice that $\mathbf{x}_k$ does not uniquely determine $\mathbf{x}_{k+1}$ because v_k^L is not specified by the model; this reflects uncertainty about the lead car's behavior. The model is *nondeterministic* because multiple next states $\mathbf{x}_{k+1}$ are possible from a given $\mathbf{x}_k$. Combining all the equations we can write $\mathbf{x}_{k+1} = f(\mathbf{x}_k, v_k^L)$ where $v_k^L \in [0, v_{\max}^L]$ is an input to the system. In more detail, the values of the state variables from the pre-state $\mathbf{x}.x^E, \mathbf{x}.y^E, \mathbf{x}.\theta^E, \mathbf{x}.x^L$, are used to first compute the derived values $\mathbf{x}.d$ and then the controls $\mathbf{x}.v^E, \mathbf{x}.\omega^E$ are computed using (2.2)–(2.4) or chosen $v^L \in [0, v_{\max}^L]$. Finally, the post-state $\mathbf{x}'$ is computed using (2.5). We can also define the transition relation $\mathcal{D} \subseteq Q \times Q$ as $\mathcal{D} = \{(\mathbf{x}_k, \mathbf{x}_{k+1}) \mid \mathbf{x}_{k+1} = f(\mathbf{x}_k, v_k^L), v_k^L \in [0, v_{\max}^L]\}$. This defines the automaton $\mathcal{A} = \langle Q, Q_0, \mathcal{D} \rangle$ and the Post operator for this system:

$$\mathit{Post}(S) = \{\mathbf{x}' \in Q \mid \exists \mathbf{x} \in S, v^L \in [0, v_{\max}^L], \mathbf{x}' = f(\mathbf{x}, v^L)\}. \tag{2.6}$$

□

2.4 Reachable states, safety, and invariance

Perspective change. *To reason about all executions from a finite set of executions (or tests) we need to represent and manipulate* sets *of states and executions.*

To discuss sets of executions, first let us define one step transitions for automaton $\mathcal{A}$ over sets of states. For any set of states $S \subseteq Q$

$$Post(S) := \{\mathbf{x}' \in Q \mid \exists \mathbf{x} \in S, (\mathbf{x}, \mathbf{x}') \in \mathcal{D}\}.$$

Post function defines how a set of states S changes after every individual state in the set S performs 1-step transition.

Exercise 2.2. (a) Write the *Post* function for the automaton in Example 2.3 as a logical formula (as in the definition of $\mathcal{D}$).

(b) Write the *Post* function for the most general automaton you created in Exercise 2.1 as a logical formula (as in the definition of $\mathcal{D}$).

Exercise 2.3. Show that *Post*() is a monotonic function. That is, if $S_1 \subseteq S_2$ then $Post(S_1) \subseteq Post(S_2)$.

Thought experiment. For a deterministic automaton, given a single state $\mathbf{x}$, computing the next state generated by $\mathcal{D}$ is straight forward. Now that you have written a math formula for *Post*, think about how you could compute $Post(\{\mathbf{x}\})$ for any single state $\mathbf{x} \in Q$? How would you compute $Post(S)$ for a set $S \subseteq Q$? Are you assuming a particular shape for S and a particular representation? What if S is a complicated irregular set?

The above thought experiment should convince you that computing $Post(S)$ can become difficult when the S, $\mathcal{D}$ are complex. Computing the set of all executions of length k essentially involves computing $Post^k(Q_0)$ which is inductively defined as:

$$Post^k(S) = \begin{cases} S & k = 0 \\ Post(Post^{k-1}(S)) & k > 0 \end{cases} \tag{2.7}$$

Exercise 2.4. Prove that for any automaton $\mathcal{A} = \langle Q, Q_0, \mathcal{D} \rangle$ the set of states reached by any execution at the end of k transitions *equals* $Post^k(Q_0)$. *Hint. (1) To prove equality of sets $A = B$, you'd want to show $A \subseteq B$ and $B \subseteq A$. (2) Try induction on the length of the execution.*

Reachable states. A state $\mathbf{x} \in Q$ is said to be *reachable* if it is the last state of any execution. Equivalently, $Post^k(Q_0)$ is the set of states reachable after k step. Why is this important? Well, if you wanted to check that the system is safe with respect to an unsafe set $U \subseteq Q$ (Starting from an initial set Q_0) then all you would have to check is that all

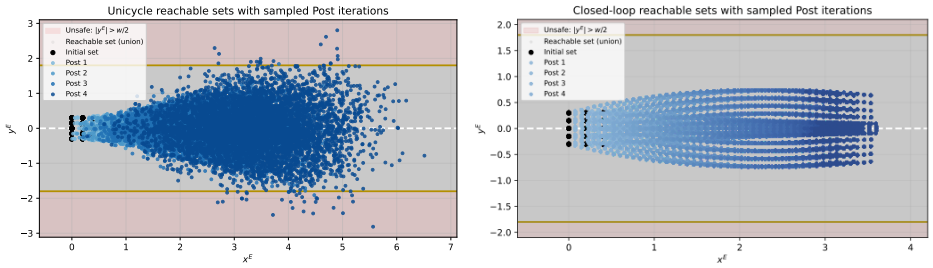


Figure 2.3
Approximation of the reachable sets for the open-loop unicycle (*left*) and the closed-loop (*right*). The red region is the unsafe set U. The reachable set (blue) intersects U indicating that the system is unsafe.

reachable are not in U. That is,

$$\forall\, k \geq 0, Post^k(Q_0) \cap U = \emptyset.$$

We have already seen that this can be a requirement for a single execution (Example 2.3). In general, computing the set of all reachable states which covers all executions can be intractable.

The transition relation $\mathcal{D}$ can be complicated because of nondeterminism and nonlinearity, and computing $Post^k(\cdot)$ for large k may not terminate. The trick we will see next can help bypassing the problem of computing $Post^k(\cdot)$.

Invariance. The idea of invariant property or invariance is common in physics and computer science. Any property or quantity related to a system that *remains unchanged* is an invariant. For example, the total energy of a lossless system in an invariant. Conserved quantities correspond to some invariant. The sum of the angles of a polygon is an invariant under scaling and linear transformations.

For an automaton $\mathcal{A} = \langle Q, Q_0, \mathcal{D} \rangle$ an invariant $I \subseteq Q$ is any set of states that contains all the reachable states, i.e., $\mathsf{Reach}_{\mathcal{A}} \subseteq I$. Invariants are useful because, if $I \cap U = \emptyset$ then we can infer $\mathsf{Reach}_{\mathcal{A}} \cap U = \emptyset$, which means the system is safe.

We now give a simple method for checking that a set of states $I \subseteq Q$ is an invariant.

Proposition 2.1. *For any set of states $I \subseteq Q$ such that (i) $Q_0 \subseteq I$ and (ii) $Post(I) \subseteq I$, then $\forall\, k$, $Post^k(Q_0) \subseteq I$. That is, I is an invariant of $\mathcal{A}$, $\mathsf{Reach}_{\mathcal{A}} \subseteq I$.*

Proof. We prove this by induction on k.

For $k = 0$ (base case) $Post^0(Q_0) = Q_0$ [by Definition of $Post^k(\cdot)$]. And, $Q_0 \subseteq I$ [by (i)]. Therefore, $Post^0(Q_0) \subseteq I$.

For $k > 0$ (inductive step), we assume $Post^k(Q_0) \subseteq I$. By applying $Post(\cdot)$ to both sides and using monotonicity of $Post(\cdot)$, we get $Post(Post^k(Q_0)) \subseteq Post(I)$. That is $Post^{k+1}(Q_0) \subseteq Post(I)$. Using (ii) it follows $Post^{k+1}(Q_0) \subseteq I$. □

Proposition 2.1 implies that if we can somehow find a set (an invariant) satisfying conditions (i) and (ii), then we do not have to compute $Post^k(\cdot)$ and in addition if $I \cap U = \emptyset$ then we can happily conclude that the system is safe.

Exercise 2.5. Let us return to Example 2.3 with the candidate invariant requirement: $d := x_2 - x_1 > 0$. Is this really an invariant? Can we prove this using Proposition 2.1? What additional assumptions do we need?

Let us assume $v_{20} = 0$ for the rest of the discussion. It is easy to see that $d := x_2 - x_1 > 0$ is not an invariant. We have not said enough about the initial values of x_{10}, x_{20}, v_{10}. What if v_{10} is so large that in one step x_1 becomes $\geq x_2$.

Now, let us take a different approach to find an inductive invariant. If we can upper-bound the time that Car 1 spends *after* it detects Car 2, then we should be able to bound the total distance it travels while braking. To do this, we introduce a *timer* into our model.

```
if x2 − x1 < ds
  if v1 > ab
    v1 := v1 − ab
    timer := timer + 1
  else v1 := 0
else v1 := v1
x1 := x1 + v1
```

Exercise 2.6. Show that the following is an invariant using Proposition 2.1:

$$I_2 : timer + v_1/a_b \leq v_{10}/a_b.$$

Invariant I_2 is indeed an inductive invariant and it implies that $timer \leq v_0/a_b$. This implies that the total distance traveled by Car 1 after detection is at most v_{10}^2/a_b. Now

we see that if we assume that the sensing distance $d_s > v_{10}^2/a_b$, then the in all executions always $x_2 > x_1$. That is, in all reachable states there is no collision.

Identifying assumptions under which the system is guaranteed to work, is a key benefit of (absolute) safety analysis. These assumptions can be used to define what are called *operating design domains (ODD).*

2.5 Invariant properties of the unicycle lane-keeping system

Proposition 2.2. *$I_{gap} = \{\mathbf{x} \in Q \mid \mathbf{x}.x^L - \mathbf{x}.x^E \geq d_{\min}\}$ is an invariant.*

Proof. We use Proposition 2.1 to prove this. We need to check two conditions: (i) $Q_0 \subseteq I_{\text{gap}}$ because for any state in Q_0, $x_0^L - x_0^E > d_{\min}$ by (2.1).

(ii) Let $\mathbf{x} \in I_{\text{gap}}$ and let $\mathbf{x}' \in Post(\{\mathbf{x}\})$. We have to show that $\mathbf{x}' \in I_{\text{gap}}$. By the controller definition (2.2), there are two cases consider because here the ego car's speed v_k^E, and therefore, the longitudinal gap in the post state $\mathbf{x}'$ only depends on the longitudinal gap in the prestate $\mathbf{x}.d = x_k^L - x_k^E$:

1. Close $d_k \leq d_{\text{close}}$ and $v_k^E = 0$.
2. Not close $d_k > d_{\text{close}}$ and $v_k^E = v_{\max}^E$.

From the plant updates (2.5),

$$d_{k+1} = x_{k+1}^L - x_{k+1}^E = (x_k^L + \Delta t\, v_k^L) - (x_k^E + \Delta t\, v_k^E) = d_k + \Delta t(v_k^L - v_k^E). \tag{2.8}$$

Case 1 (Close): The gap in the post-state is $d_{k+1} = d_k + \Delta t\, v_k^L \geq d_k$ (since $v_k^L \geq 0$). Also, $d_k = x_k^L - x_k^E \geq d_{\min}$ (since $\mathbf{x} \in I_{\text{gap}}$).
Case 2 (Not close): The gap in the post-state is

$$\begin{aligned} d_{k+1} &= d_k + \Delta t(v_k^L - v_{\max}^E) \\ &\geq d_k - \Delta t\, v_{\max}^E && \text{(since } v_k^L \geq 0\text{)} \\ &\geq d_{\text{close}} - \Delta t\, v_{\max}^E && \text{(since not close } d_k = \mathbf{x}.d > d_{\text{close}}\text{)} \\ &= d_{\min} && \text{(since } d_{\text{close}} = d_{\min} + v_{\max}^E \Delta t\text{).} \end{aligned}$$

Thus $\mathbf{x}' \in I_{\text{gap}}$, so $Post(I_{\text{gap}}) \subseteq I_{\text{gap}}$. By Proposition 2.1, I_{gap} is an invariant. □

Proposition 2.3. *$I_{inlane} = \{\mathbf{x} \in Q \mid |\mathbf{x}.y^E| \leq w/2,\ y^E\theta^E \leq 0,\ |\theta^E| \leq \pi/2\}$ is an invariant of the discrete-time lane-keeping model.*

Proof. We use Proposition 2.1. Condition (i) holds because $y_0^E = 0$ and $\theta_0^E = 0$. For (ii), let $\mathbf{x} \in I_{\text{inlane}}$ and let $\mathbf{x}' \in Post(\{\mathbf{x}\})$. From the heading controller, $\theta_{k+1}^E = -k_y y_k^E$ so $y_k^E \theta_{k+1}^E \leq 0$ and $|\theta_{k+1}^E| \leq \pi/2$. From the plant updates (2.5),

$$y_{k+1}^E = y_k^E + \Delta t\, v_k^E \sin(\theta_k^E). \tag{2.9}$$

If $y_k^E > 0$, then $y_k^E \theta_k^E \leq 0$ implies $\theta_k^E \leq 0$, and since $|\theta_k^E| \leq \pi/2$ we have $\sin(\theta_k^E) \leq 0$. It follows that, $y_{k+1}^E \leq y_k^E \leq w/2$.

If $y_k^E < 0$, then $\theta_k^E \geq 0$, and since $|\theta_k^E| \leq \pi/2$ we have $\sin(\theta_k^E) \geq 0$. It follows that, $y_{k+1}^E \geq y_k^E \geq -w/2$.

If $y_k^E = 0$, then $y_{k+1}^E = 0$. Thus $|y_{k+1}^E| \leq w/2$, and the sign condition is preserved by the controller, so $\mathbf{x}' \in I_{\text{inlane}}$. Hence $Post(I_{\text{inlane}}) \subseteq I_{\text{inlane}}$, and I_{inlane} is an invariant. □

These propositions imply that for any reachable state $\mathbf{x}$ of the system, $\mathbf{x} \in I_{\text{gap}}$ and $\mathbf{x} \in I_{\text{inlane}}$. Thus, the the unicycle lane-keeping system maintains the minimum gap and stays within the lane bounds for all time.

Not all invariants are inductive.

For example, the predicate $|y^E| \leq w/2$ alone is not an inductive invariant because the sign condition $y^E \theta^E \leq 0$ is needed to ensure that the lateral position does not grow unbounded. Thus, while $|y^E| \leq w/2$ is an invariant (it contains all the reachable states), it is not an inductive invariant, i.e., it cannot be proved using the two conditions of inductive invariance in Proposition 2.1. When you have a candidate invariant, it is often useful to check if it is inductive. If not, you may need to strengthen it by adding additional conditions as we did above.

2.6 Linear algebra representation

We can write the discrete-time update in matrix form by collecting the full state as $\mathbf{x}_k = [x_k^E, y_k^E, \theta_k^E, x_k^L]^\mathcal{T}$ and the inputs as $u_k = [v_k^E, \omega_k^E, v_k^L]^\mathcal{T}$. Then

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \begin{bmatrix} \cos(\theta_k^E) & 0 & 0 \\ \sin(\theta_k^E) & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_k^E \\ \omega_k^E \\ v_k^L \end{bmatrix}. \tag{2.10}$$

2.7 Automata under Coordinate Transformations

In the above examples, we carefully chose the coordinate frames for the different automata models to make the math simple. For example, in Example 2.3, we chose a 1-D world aligned with the lane of travel of the cars. In Example 2.5, we aligned the x-axis of the world frame with the lane centerline. In more complex scenarios, we may have to deal with a world frame that is not aligned so conveniently. In this section, we briefly discuss how to transform automata models under coordinate transformations. Let us consider a general automaton $\mathcal{A} = \langle Q, Q_0, \mathcal{D} \rangle$ where $Q \subseteq \mathbb{R}^n$. Let $T : \mathbb{R}^n \to \mathbb{R}^n$ be a bijective

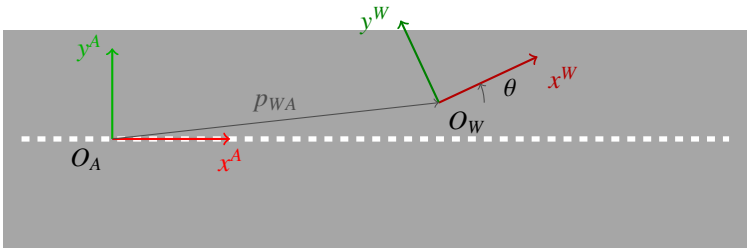


Figure 2.4
Lane-aligned frame A and a rotated, translated world frame W.

coordinate transformation function with inverse T^{-1}. We can define a new automaton $\mathcal{A}_T = \langle Q_T, Q_{0,T}, \mathcal{D}_T \rangle$ where $Q_T = \{T(\mathbf{x}) \mid \mathbf{x} \in Q\}$, $Q_{0,T} = \{T(\mathbf{x}) \mid \mathbf{x} \in Q_0\}$, and

$$\mathcal{D}_T = \{(\mathbf{x}_T, \mathbf{x}'_T) \in Q_T \times Q_T \mid (T^{-1}(\mathbf{x}_T), T^{-1}(\mathbf{x}'_T)) \in \mathcal{D}\}. \tag{2.11}$$

Equivalently, $(\mathbf{x}, \mathbf{x}') \in \mathcal{D}$ if and only if $(T(\mathbf{x}), T(\mathbf{x}')) \in \mathcal{D}_T$.

As a concrete example, consider the 1-D collision automaton $\mathcal{A}$ in Example 2.3 where the cars move along a lane-aligned frame A. We embed this model in 2D by assigning each car a constant lateral coordinate $y^A = 0$. Let $p_i^A = [x_i^A,\ y_i^A]^\top$ for $i \in \{1, 2\}$, and let $e_x^A = [1,\ 0]^\top$ denote the lane direction. Define a rigid transform from the lane-aligned frame A to a world frame W by

$$T_{WA}(p^A) = R_{WA} p^A + p_{WA}, \quad R_{WA} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{2.12}$$

The induced transformation on the state applies T_{WA} to each position and leaves the scalar speed v_1 unchanged.

In the lane-aligned frame, $y_1^A = y_2^A = 0$ and the transitions are

$$\begin{aligned} v_1' &= \begin{cases} \max(0, v_1 - a_b), & x_2^A - x_1^A < d_s, \\ v_1, & x_2^A - x_1^A \geq d_s, \end{cases} \\ p_1^{A'} &= p_1^A + v_1 e_x^A, \\ p_2^{A'} &= p_2^A + v_2 e_x^A. \end{aligned}$$

Applying T_{WA} to both the pre- and post-states gives the transformed transition relation $\mathcal{D}_T$ in W:

$$\begin{aligned} v_1' &= \begin{cases} \max(0, v_1 - a_b), & d < d_s, \\ v_1, & d \geq d_s, \end{cases} \\ p_1^{W'} &= p_1^W + v_1 R_{WA} e_x^A, \\ p_2^{W'} &= p_2^W + v_2 R_{WA} e_x^A, \end{aligned}$$

where the longitudinal gap can be expressed in transformed coordinates as

$$d = x_2^A - x_1^A = (e_x^A)^\top R_{WA}^\top (p_2^W - p_1^W) = \cos\theta(x_2^W - x_1^W) + \sin\theta(y_2^W - y_1^W). \tag{2.13}$$

Thus, the simple 1-D motion becomes motion along a fixed line in the rotated frame, and the braking condition depends on the projection of the separation vector onto the original lane direction.

Proposition 2.4. *For any bijection T, every transition of $\mathcal{A}$ maps to a unique transition of $\mathcal{A}_T$, and vice-versa. Consequently, if $\alpha = \mathbf{x}_0, \mathbf{x}_1, \ldots$ is an execution of $\mathcal{A}$, then $T(\alpha) = T(\mathbf{x}_0), T(\mathbf{x}_1), \ldots$ is an execution of $\mathcal{A}_T$. Moreover, for any $S \subseteq Q$,*

$$Post_T(T(S)) = T(Post(S)),$$

and a set I is an invariant for $\mathcal{A}$ if and only if $T(I)$ is an invariant for $\mathcal{A}_T$.

Exercise 2.7. Write the inverse transform T_{WA}^{-1} and verify that $p^A = T_{WA}^{-1}(p^W) = R_{WA}^\top(p^W - p_{WA})$.

Solution. Since $p^W = R_{WA}p^A + p_{WA}$, subtract p_{WA} and multiply by $R_{WA}^\top$ to get $p^A = R_{WA}^\top(p^W - p_{WA})$. Hence $T_{WA}^{-1}(p^W) = R_{WA}^\top(p^W - p_{WA})$.

References

Fabris, Simone. 2012. Method for hazard severity assessment for the case of undemanded deceleration, Technical report, TRW Automotive.

ISO. 2011. Road vehicles—functional safety, Technical Report ISO 26262, International Organization for Standardization (ISO).

Perry, Raleigh B., Michael M. Madden, Wilfredo Torres-Pomales, and Ricky W. Butler. 2013. *The simplified aircraft-based paired approach with the ALAS alerting algorithm*, Technical Report NASA/TM-2013-217804, NASA, Langley Research Center.

Index