Simulator-Based Verification of Autonomous Vehicles: Agent-Based Test Generation

1:09pm, 7 October 2020

1 Introduction

- **Time** is implicit and discrete, with agents choosing actions at each timestep, and a time resolution constant specifying the simulation time between timesteps
- **State space** is continuous and feature-based techniques are probably more effective than discretisation
- **Actions** may be multi-body and/or multi-effector
- **Action space** is continuous and discretisation typically requires durative actions
- **Observability** is full, meaning that a sensor model is not required (although partial observability can be simulated)
- **Objectives** are goals states with transition costs/rewards, meaning in general that the horizon is indefinite, but a finite-horizon may also be imposed
- **Model** for transitions and objectives is unknown, but can be sampled through interaction with a simulator
- **Game** involving two or more players (agents), including a potentially adversarial design under verficiation (i.e. the ego agent)
- **Test generation** may be online (providing incomplete tests) or offline (providing complete tests)

Note: Ideally a test should specify the optimal effector-actions to execute in a given state with respect to some test objective. A complete test then would be one that specifies effector-actions to execute in every reachable state. Online testing cannot guarantee complete tests because effector-actions are only determined for states that are actually encountered in a given run, and this does not typically cover all reachable states. For example, if a simulation involves non-deterministic action-selection, non-deterministic action-effects, or exogenous events, then these sources of uncertainty mean that a single execution of online testing will only reveal one of many possible runs. It may be possible to impose restrictions on a simulation so as to avoid many of these sources of uncertainty, but the actions of the ego agent remain uncontrollable. For example, a test may be complete for a given ego policy, but may be incomplete when faced with a different ego policy (since that policy may cause the system to transition into a state that was not accounted for in the original test). It follows that a test is only complete if it accounts for every source of uncertainty and also works for every possible ego policy. This is a very strong requirement and is unlikely to be satisfiable in practice (e.g. if agents execute actions simultaneously, then the ego policy is a priori unknown). This seems to imply that the aim of our work is not *test generation* but rather *online testing with optional capturing of incomplete tests*: re-running such tests could be useful for repeatability, but test execution may fail, and there would be no guarantees that tests remain optimal with respect to their original test objectives.

2 Preliminaries

We rely on some standard mathematical notation: v_i is an element of vector $\boldsymbol{v} = (v_1, \ldots, v_n)$ with $\boldsymbol{v}_{-i} = (v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)$ the subvector of $\boldsymbol{v}$ excluding v_i, $|S|$ is the cardinality of set S, 2^S is the powerset of S, $\Delta(S)$ is the set of probability distributions over S, $\mathbb{R}$ is the set of real numbers with $\mathbb{R}^{\geq 0}$ the subset of non-negative real numbers, and $\mathbb{N}$ is the set of natural numbers. A function $f : X \to Y$ is a surjection if for each $y \in Y$ there exists some $x \in X$ such that $f(x) = y$. Given a function $f : X \to Y$, the inverse image of function f is a function $f^{-1}_{B,N} : Y \to 2^X$ defined for each $y \in Y$ as $f^{-1}_{B,N}(y) = \{x \in X \mid f(x) = y\}$.

2.1 Normal-Form Games

An (n-player) normal-form game is a tuple $(N, A, \boldsymbol{u})$ where $N = \{1, \ldots, n\}$ is a finite set of players, $A = A_1 \times \cdots \times A_n$ is a set of action profiles with A_i the (possibly infinite) set of actions available to player $i \in N$, and $\boldsymbol{u} = (u_1, \ldots, u_n)$ is a profile of utility functions with $u_i : A \to \mathbb{R}$ the utility function for i. A (mixed) strategy for player $i \in N$ is a probability distribution $\psi \in \Delta(A_i)$ with $\boldsymbol{\psi} \in \Delta(A_1) \times \cdots \times \Delta(A_n)$ a (mixed) strategy profile. A strategy ψ for player $i \in N$ is a pure strategy if $\psi(a) = 1$ for some $a \in A_i$. A strategy profile $\boldsymbol{\psi}$ is a pure strategy profile if ψ_i is a pure strategy for each $i \in N$. For convenience, we may denote a pure strategy ψ for player $i \in N$ directly by the action $a \in A_i$ such that $\psi(a) = 1$. Likewise, we may denote a pure strategy profile $\boldsymbol{\psi}$ directly by the action profile $\boldsymbol{a} \in A$ such that $\psi_i(a_i) = 1$ for each player $i \in N$. The expected utility of a strategy profile $\boldsymbol{\psi}$ is defined as:

$$u_i(\boldsymbol{\psi}) = \sum_{\boldsymbol{a} \in A} u_i(\boldsymbol{a}) \prod_{j \in N} \psi_j(a_j)$$

In game theory, a solution concept is a prediction of how a game will be played. Solution concepts are typically based on the idea that each player will try to maximise their expected utility under the assumption that other players will do the same. A feasible deviation from strategy profile $\boldsymbol{\psi}$ by player $i \in N$ is a strategy profile $\boldsymbol{\psi}' = (\boldsymbol{\psi}_{-i}, \psi')$ where ψ' is a strategy for i. A strategy profile $\boldsymbol{\psi}$ is a Nash equilibrium if no player $i \in N$ has a feasible deviation $\boldsymbol{\psi}'$ from $\boldsymbol{\psi}$ such that $u_i(\boldsymbol{\psi}') > u_i(\boldsymbol{\psi})$. Every normal-form game with a finite set of action profiles is guaranteed to have at least one (not necessarily pure) Nash equilibrium [3]. A Nash equilibrium is also guaranteed to exist for certain classes of normal-form game with an infinite set of action profiles [].

2.2 Markov Games

A (fully observable) Markov game is a tuple $(N, S, A, T, \boldsymbol{R})$ where $N = \{1, \ldots, n\}$ is a finite set of players, S is a (possibly infinite) set of states, $A = A_1 \times \cdots \times A_n$ is a set of action profiles with A_i the (possibly infinite) set of actions available to player $i \in N$, $T : S \times A \to \Delta(S)$ is a (stochastic) transition function, and $\boldsymbol{R} = (R_1, \ldots, R_n)$ is a profile of reward functions with $R_i : S \times A \times S \to \mathbb{R}$ the reward function for player $i \in N$. A Markov game is zero-sum if $\sum_{i \in N} R_i(s, \boldsymbol{a}, s') = 0$ for each $\boldsymbol{a} \in A$ and all $s, s' \in S$.

Markov games (also known as stochastic games [6]) are composed of a sequence of normal-form games, called stage games, where players seek to optimise their return for the whole game rather than for individual stage games. The current stage game is determined by the current state, and players transition between states by executing actions simultaneously. The game is said to be fully observable because players are assumed to observe the current (shared) state at each decision step. Let $T(s, \boldsymbol{a}, s')$ denote the probability of transitioning to state $s' \in S$ after executing action profile $\boldsymbol{a} \in A$ in state $s \in S$ according to probability distribution $T(s, \boldsymbol{a})$. The probability of transitioning to state $s' \in S$ after executing strategy profile $\boldsymbol{\psi}$ in state $s \in S$ is:

$$T(s, \boldsymbol{\psi}, s') = \prod_{\boldsymbol{a} \in A} T(s, \boldsymbol{a}, s') \prod_{j \in N} \psi_j(a_j)$$

The stage game for state $s \in S$ is a normal-form game $(N, A, \boldsymbol{u})$ where the utility function u_i for player $i \in N$ is defined for each action profile $\boldsymbol{a} \in A$ as the immediate expected reward:

$$u_i(\boldsymbol{a}) = \sum_{s' \in S} T(s, \boldsymbol{a}, s') R_i(s, \boldsymbol{a}, s')$$

Markov games subsume numerous other frameworks, including repeated games when $|S| = 1$, and Markov decision processes (MDPs) when $|N| = 1$.

Solutions to Markov games are represented as functions, called policies, that map *histories* to strategies. An execution is a possibly infinite sequence of states and action profiles $(s_1, \boldsymbol{a}_1, s_2, \boldsymbol{a}_2, \dots)$. A history of length t is a finite execution $h_t = (s_1, a_1, \dots, a_{t-1}, s_t)$ ending in a state. Let H_t be the set of histories of length t with $D = \{1, 2, \dots, t_{\max}\}$ the set of decision-steps up to horizon $t_{\max} \in \mathbb{N} \cup \{\infty\}$ and $H = \{h \in H_t \mid t \in D\}$ the set of histories up to $t_{\max}$. A (stochastic or mixed) policy for player $i \in N$ is a function $\pi_i : H' \to \Delta(A_i)$ where $H' \subseteq H$. Let $\pi_i(h, a)$ denote the probability that player $i \in N$ should execute action $a \in A_i$ in history $h \in H'$ according to strategy $\pi_i(h)$. A policy π_i for player $i \in N$ is a complete policy if $H' = H$, otherwise π_i is a partial policy. A policy π_i for player $i \in N$ is a deterministic (or pure) policy if $\pi_i(h, a) = 1$ for each $h \in H'$ and some $a \in A_i$. For convenience, we may denote a deterministic policy π_i for player $i \in N$ directly by a policy $\pi'_i : H' \to A$ where $\pi'_i(h) = a$ such that $\pi_i(h, a) = 1$. A policy π_i for player $i \in N$ is Markovian if $\pi_i(h_t) = \pi_i(h_{t'})$ for all $h_t, h_{t'} \in H'$ such that $t = t'$ and $s_t = s_{t'}$. A Markovian policy for player $i \in N$ may be written as $\pi_i : X \to \Delta(A_i)$ where $X \subseteq S \times D$. A Markovian policy π_i for player $i \in N$ is a stationary policy if $\pi_i(s, t) = \pi_i(s, t')$ for all $(s, t), (s, t') \in X$, otherwise π_i is a non-stationary policy. A stationary Markovian policy for player $i \in N$ may be written as $\pi_i : S' \to \Delta(A_i)$ where $S' \subseteq S$. A tuple $\boldsymbol{\pi} = (\pi_1, \dots, \pi_n)$ is a policy profile over $H' \subseteq H$ if π_i is a policy over H' for each $i \in N$. The strategy profile for history $h \in H'$ according to policy profile $\boldsymbol{\pi}$ is $\boldsymbol{\pi}(h) = (\pi_1(h), \dots, \pi_n(h))$.

Solutions are only well-defined for certain classes of Markov game. Two common examples are finite-horizon Markov games and infinite-horizon (discounted reward) Markov games. A finite-horizon Markov game is a tuple $(\Gamma, t_{\max})$ where Γ is a Markov game and $t_{\max} \in \mathbb{N}$ is a horizon. An infinite-horizon Markov game is a tuple (Γ, γ) where Γ is a Markov game and $\gamma \in [0, 1)$ is a discount fbody. It is known that attention can be restricted to stationary Markovian policies in the case of infinite-horizon Markov

games, and to non-stationary Markovian policies in the case of finite-horizon Markov games. Unless stated otherwise, we will assume hereafter that all policies are either stationary or non-stationary Markovian policies. For an infinite-horizon Markov game (Γ, γ), the expected value of policy profile $\boldsymbol{\pi}$ for player $i \in N$ in state $s \in S$ is:

$$V_i(\boldsymbol{\pi}, s) = \sum_{s' \in S} T(s, \boldsymbol{\pi}(s), s') \left[R_i(s, \boldsymbol{\pi}(s), s') + \gamma V_i(\boldsymbol{\pi}, s') \right]$$

For a finite-horizon Markov game $(\Gamma, t_{\max})$, the expected value of policy profile $\boldsymbol{\pi}$ for player $i \in N$ in state $s \in S$, with $t \in D \cup \{0\}$ remaining decision-steps, is:

$$V_i(\boldsymbol{\pi}, s, t) = \begin{cases} \sum_{s' \in S} T(s, \boldsymbol{\pi}(s, t), s') \left[R_i(s, \boldsymbol{\pi}(s, t), s') + V_i(\boldsymbol{\pi}, s', t-1) \right] & \text{if } t > 0 \\ 0 & \text{if } t = 0 \end{cases}$$

A feasible deviation from policy profile $\boldsymbol{\pi}$ by player $i \in N$ is a policy profile $\boldsymbol{\pi}' = (\boldsymbol{\pi}_{-i}, \pi')$ where π' is a policy for i. A policy profile $\boldsymbol{\pi}$ is a Nash equilibrium if no player $i \in N$ has a feasible deviation $\boldsymbol{\pi}'$ from $\boldsymbol{\pi}$ such that $V_i(\boldsymbol{\pi}', h) \geq V_i(\boldsymbol{\pi}, h)$ for each $h \in H'$ and $V_i(\boldsymbol{\pi}', h') > V_i(\boldsymbol{\pi}, h')$ for some $h' \in H'$.

> **To do:** Confirm that this definition of a Nash equilibrium is correct with respect to histories (or states, or time-indexed states).

2.3 Stochastic Shortest Path Games

A (fully observable) stochastic shortest path (SSP) game is a tuple $(N, S, A, T, \boldsymbol{C}, G)$ where $N = \{1, \ldots, n\}$ is a finite set of players, S is a (possibly infinite) set of states, $A = A_1 \times \cdots \times A_n$ is a set of action profiles with A_i the (possibly infinite) set of actions available to player $i \in N$, $T : S \times A \to \Delta(S)$ is a (stochastic) transition function, $\boldsymbol{C} = (C_1, \ldots, C_n)$ is a profile of cost functions with $C_i : S \times A \times S \to \mathbb{R}^{\geq 0}$ the (non-negative) cost function for player $i \in N$, and $G = G_1 \cup \cdots \cup G_n$ is a non-empty set of goal states with $G_i \subseteq S$ the (possibly empty, possibly infinite) set of goal states for player $i \in N$. SSP games subsume numerous other frameworks, including finite- and infinite-horizon Markov games as well as SSP MDPs (which in turn subsume finite- and infinite-horizon MDPs as well as classical planning) [1, 4, 5, 2].

> **Note:** I have only found definitions of 2-player zero-sum SSP games in the literature, so this definition of an n-player general-sum SSP game is only an informed guess. The definition says that all players should seek to minimize their cost for the duration of the game, as is the case in Markov games, but that at least one player should seek to arrive in a goal state where the game will terminate. The definition does not explicitly capture the notion that an adversary would seek to block the goal-directed player from achieving their goal, but the fact that this notion appears in existing definitions may be due to those definitions focusing on zero-sum games.

3 Framework

Definition 1. *An agent-body-effector model is a tuple* $(N, B, E, f_{B,N}, f_{E,B})$ *where:*

- $N = \{1, \ldots, n\}$ *is a finite set of agents*
- B *is a finite set of bodies with* $f_{B,N} : B \to N$ *a surjection from bodies to agents*
- E *is a finite set of effectors with* $f_{E,B} : E \to B$ *a surjection from effectors to bodies and* $A(e)$ *the non-empty (possibly infinite) set of actions available to* $e \in E$

Corollary 1. $2 \leq |N| \leq |B| \leq |E|$.

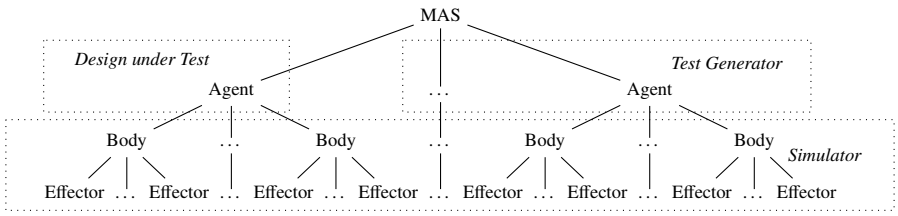


Figure 1: Agent-body-effector model

The set of bodies associated with agents $N' \subseteq N$ is defined as $B(N') = \{b \in f_{B,N}^{-1}(i) \mid i \in N'\}$. The set of effectors associated with bodies $B' \subseteq B$ is defined as $E(B') = \{e \in f_{E,B}^{-1}(b) \mid b \in B'\}$. Thus, the set of bodies B (resp. agents N) in an agent-body-effector model induces a partition of the set of effectors E, as illustrated in Figure 1. The set of (multi-)actions available to effectors $E' \subseteq E$ is defined as $A(E') = \times_{e \in E'} A(e)$. Thus, there exists a non-empty set of (multi-)actions available to each body (resp. agent).

Definition 2. *Let* $(N, B, E, f_{B,N}, f_{E,B})$ *be an agent-body-effector model. A simulator-based testing game is a stochastic shortest path game* $(N, S, A, T, \boldsymbol{C}, G)$ *if:*

- $N = \{1, \ldots, n\}$ *such that* $n \geq 2$ *with* 1 *the ego agent and* $2, \ldots, n$ *the tester agents*
- $A_i = A(E(B(\{i\})))$ *is the set of actions available to agent* $i \in N$

To do: Formally define the notion of proper policies.

Lemma 1. *Let* $\mathcal{G}$ *be a simulator-based testing game. There exists a proper policy profile for* $\mathcal{G}$ *if and only if* $\mathcal{G}$ *does not exhibit dead-ends.*

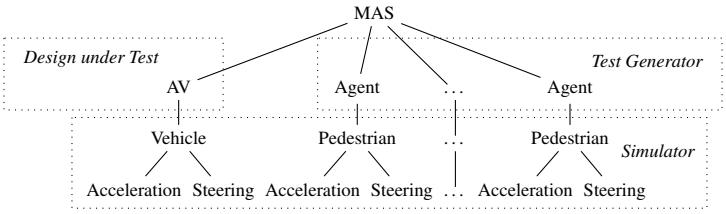
Proof Sketch. The proof may be analogous to the proofs for SSP MDPs [2], assuming it has not been proven already. □

Definition 3. *A time-limited test generation game is a tuple* $(\mathcal{G}, t_{\max})$ *where* $\mathcal{G}$ *is a test generation game and* $t_{\max} \in \mathbb{N}$ *is a (finite-)horizon.*

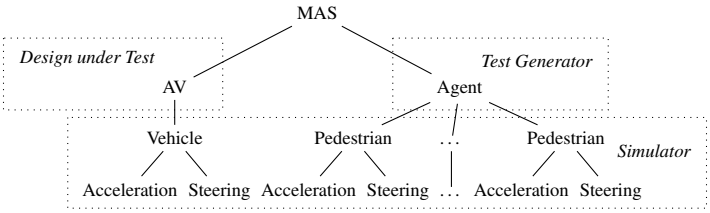
Proposition 1. *Let* $(\mathcal{G}, t_{\max})$ *be a time-limited test generation game. There exists a proper policy profile for* $(\mathcal{G}, t_{\max})$.

Proof Sketch. The set of states S can be translated into a new set of states $S' = S \times \{0, \ldots, t_{\max}\}$. The set of goal states G can be translated into a new set of goal states $G' = (G \times \{0, \ldots, t_{\max}\}) \cup (S \times \{0\})$. The translated game $(\mathcal{G}', t_{\max})$ does not exhibit dead-ends because for any state $s \in S'$ there always exists a state $s' \in S \times \{0\}$ such that s' is reachable from s. Thus, it follows from Lemma 1 that a proper policy profile for G is guaranteed to exist. □

4 Experiments



(a) single-body ego agent, n single-body tester agents



(b) single-body ego agent, 1 multi-body tester agent

Figure 2: Agent-based test generation in CAV-GYM:PEDESTRIANS

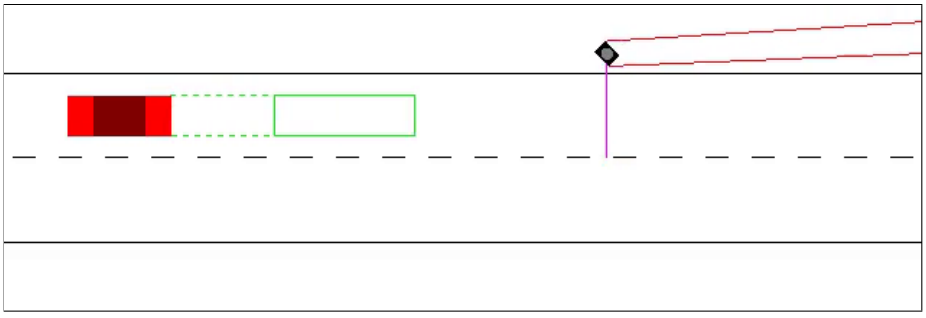


Figure 3: CAV-GYM:PEDESTRIANS

References

[1] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vols. I and II*. Athena Scientific, 1995.

[2] Mausam and Andrey Kolobov. *Planning with Markov decision processes: An AI perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[3] John Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

[4] Stephen D. Patek. *Stochastic and shortest path games: theory and algorithms*. PhD thesis, Massachusetts Institute of Technology, 1997.

[5] Stephen D. Patek and Dimitri P. Bertsekas. Stochastic shortest path games. *SIAM Journal on Control and Optimization*, 37(3):804–824, 1999.

[6] Lloyd S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953.

Appendix: Kinematics

A state is a tuple $s_i = (p_i, v_i, o_i)$ where $p_i \in \mathbb{R}^2$ is a position denoted $p_i = (p_i^x, p_i^y)$, $v_i \in [v_{\min}, v_{\max}]$ is a velocity with $v_{\min}, v_{\max} \in \mathbb{R}^{\geq 0}$ such that $v_{\min} < v_{\max}$, and $o_i \in (-\pi, \pi]$ is an orientation (in radians). Let $b \in \mathbb{R}^{\geq 0}$ be a wheelbase constant. The positions of front and rear wheels $f_i, r_i \in \mathbb{R}^2$ in state $s_i = (p_i, v_i, o_i)$ are:

$$f_i = \left(p_i + \frac{b}{2} \cdot (\cos o_i, \sin o_i) \right) \tag{1}$$

$$r_i = \left(p_i - \frac{b}{2} \cdot (\cos o_i, \sin o_i) \right) \tag{2}$$

A body has two effectors: throttle and steering. An action is a tuple $a_i = (t_i, e_i)$ where $t_i \in [t_{\min}, t_{\max}]$ is a throttle with $t_{\min}, t_{\max} \in \mathbb{R}$ such that $t_{\min} < t_{\max}$, and $e_i \in (e_{\min}, e_{\max}]$ is a steering angle (in radians) with $e_{\min}, e_{\max} \in (-\pi, \pi]$ such that $e_{\min} < e_{\max}$. Let $\lambda \in \mathbb{R}^{>0}$ be a time resolution constant. If action $a_i = (t_i, e_i)$ is executed in state $s_i = (p_i, v_i, o_i)$, then the successor state is $s_{i+1} = (p_{i+1}, v_{i+1}, o_{i+1})$ where:

$$f_{i+1} = f_i + (v_i \cdot \lambda \cdot (\cos o_i + e_i, \sin o_i + e_i)) \tag{3}$$

$$r_{i+1} = r_i + (v_i \cdot \lambda \cdot (\cos o_i, \sin o_i)) \tag{4}$$

$$p_{i+1} = \frac{f_{i+1} + r_{i+1}}{2} \tag{5}$$

$$v_{i+1} = \max\{v_{\min}, \min\{v_{\max}, v_i + (\lambda \cdot t_i)\}\} \tag{6}$$

$$o_{i+1} = \operatorname{atan2}\left(f_{i+1}^y - r_{i+1}^y, f_{i+1}^x - r_{i+1}^x\right) \tag{7}$$

The body kinematics from Equations 1–7 are illustrated in Figure 4.

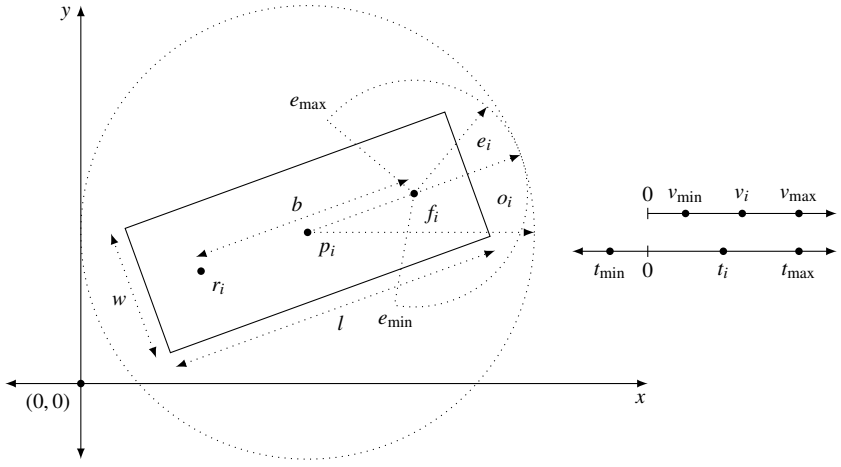


Figure 4: Body kinematics

Problem: Given state s_i and target orientation o_g such that o_g is reachable from s_i with a single action, find steering angle e_i such that executing action $a_i = (t_i, e_i)$ in s_i results in successor state $s_{i+1} = (p_{i+1}, v_{i+1}, o_{i+1})$ where $o_{i+1} = o_g$ for any throttle t_i.

Appendix: Algorithms

To do: Account for discretisation of continuous action spaces at the agent level, although strictly speaking only approximate Q-learning agents require finite action spaces.

Algorithm 1: Simulator

persistent: environment e, agents $N = \{1, \ldots, n\}$, terminator $\psi \subseteq S$
begin
  for each *episode* **do**
    $s \leftarrow$ reset e
    for each *timestep* **do**
      $\boldsymbol{a} \leftarrow (\textsc{ChooseAction}_1(s), \ldots, \textsc{ChooseAction}_n(s))$
      $\boldsymbol{r}, s' \leftarrow$ execute $\boldsymbol{a}$ in e
      for each *agent* $i \in N$ **do**
        $\textsc{ProcessFeedback}_i(s, a_i, r_i, s')$
      if $s' \in \psi$ **then break else** $s \leftarrow s'$

Algorithm 2: RandomAgent

persistent: exploration rate $\epsilon \in [0, 1]$
function ChooseAction(s)
  with probability ϵ **do**
    return random choice from $A(s)$
  return $\varnothing$

Algorithm 3: ProgrammedRandomAgent

persistent: programmed behaviour $\pi : S \to A$, terminator $\psi \subseteq S$, exploration rate $\epsilon \in [0, 1]$
function ChooseAction(s)
  if π *is active* **then**
    if $s \in \psi$ **then** set π as inactive **else return** $\pi(s)$
  with probability ϵ **do**
    set π as active
    return $\pi(s)$
  return $\varnothing$

Algorithm 4: ProgrammedReactiveAgent

persistent: programmed behaviour $\pi : S \to A$, trigger $\varphi \subseteq S$, terminator $\psi \subseteq S$

function ChooseAction(s)
  if π *is active* **then**
    if $s \in \psi$ **then** set π as inactive **else return** $\pi(s)$
  if $s \in \varphi$ **then**
    set π as active
    return $\pi(s)$
  return $\varnothing$

Algorithm 5: ProgrammedElectionAgent

persistent: programmed behaviour $\pi : S \to A$, terminator $\psi \subseteq S$, coordinator c

function ChooseAction(s)
  if π *is active* **then**
    if $s \in \psi$ **then** set π as inactive **else return** $\pi(s)$
  if *elected by* c **then**
    set π as active
    return $\pi(s)$
  return $\varnothing$

Algorithm 6: QLearningAgent

persistent: learning rate $\alpha \in [0, 1]$, discount fbody $\gamma \in [0, 1)$, exploration rate $\epsilon \in [0, 1]$,
feature $f_j : S \times A \to \mathbb{R}$ with weight $w_j \in \mathbb{R}$ for $j = 1, \ldots, m$

function ChooseAction(s)
  with probability ϵ **do**
    return random choice from $A(s)$
  return random choice from $\operatorname{argmax}_{a \in A(s)} \text{QValue}(s, a)$
procedure ProcessFeedback(s, a, r, s')
  $q \leftarrow \left(r + \gamma \cdot \max_{a' \in A(s')} \text{QValue}(s', a')\right) - \text{QValue}(s, a)$
  for each *feature* f_j **do**
    $w_j \leftarrow w_j + \alpha \cdot q \cdot f_j(s, a)$
function QValue(s, a)
  return $\sum_{j=1}^{m} f_j(s, a) \cdot w_j$