

# Homeostatic motion planning with innate physics knowledge

**Lafratta, G.**<sup>1</sup>

**Porr, B.**<sup>1</sup>

**Chandler, C.**<sup>2</sup>

**Miller, A.**<sup>2</sup>

<sup>1</sup>School of Engineering, University of Glasgow

<sup>2</sup>School of Computing Science, University of Glasgow.

**Keywords:** Planning, homeostasis, closed loop, autonomous agents

## Abstract

Living organisms interact with their surroundings in a closed-loop fashion, where sensory inputs dictate the initiation and termination of behaviours. Even simple animals are able to develop and execute complex plans, which has not yet been replicated in

robotics using pure closed-loop input control. We propose a solution to this problem by defining a set of discrete and temporary closed-loop controllers, called “tasks”, each representing a closed-loop behaviour. We further introduce a supervisory module which has an innate understanding of physics and causality, through which it can simulate the execution of task sequences over time and store the results in a model of the environment. On the basis of this model, plans can be made by chaining temporary closed-loop controllers. The proposed framework was implemented for a real robot and tested in two scenarios as proof of concept.

## **1 Introduction**

Living organisms interact with their surroundings through sensory inputs in a closed-loop fashion (Maturana and Varela, 1980; Porr and Wörgötter, 2005). To recreate closed-loop navigation in a robot it is sufficient to directly connect a robot’s sensors to its motor effectors, and the specific excitatory or inhibitory connections determine the control strategy, as detailed in Braitenberg (1986). These sensor-effector connections represent a single closed-loop controller, which produces stereotyped behaviour in response to an immediate stimulus. For this reason, closed-loop control by itself is not suitable to formulate complex navigation plans.

A wide variety of techniques have been developed over the years to achieve safe spatial navigation in complex scenarios. These can be learning-based, reactive, or a combination of the two. Reinforcement Learning (RL) is possibly the most popular learning-based paradigm. In RL, an agent learns to associate actions with specific inputs

or "states" through repeated interactions with the environment, aiming to maximise the cumulative reward (Sun et al., 2021). Rewards represent feedback for the action taken, making for a closed-loop output control paradigm. RL has been very successful in recent years, when supplemented with techniques such as deep learning (Mnih et al., 2015; Silver et al., 2016; OpenAI, 2023). RL models are inherently slow to train for two main reasons. Firstly, rewards are sparse. Secondly, in output control, feedback about the action can only be obtained *after* the action has been completed; the action will therefore be corrected in the next training episode. Contrast this with input control, where feedback on the action is continuously provided by sensors and motor outputs can be corrected immediately. The outputs of a naive RL model are therefore typically associated with high error and/or low reward (cfr. benchmarks in Thodoroff et al. 2022). For example, Mila and Pal (2019) developed a Real-Time Actor-Critic paradigm which, despite being able to achieve faster convergence than the classical Soft Actor-Critic one, still required hundreds of thousands of training steps, which could take hours or days to complete (Dalton and Frosio, 2020). As a consequence, the training of RL models typically occurs offline. To this day, fully online RL remains unfeasible, with the consequence that an automated agent is not able to acquire knowledge tailored to its own environment in real-time. Instead, large models must be trained in order for a system to be able to generalise to a variety of unseen scenarios. This is not only computationally expensive, but also compromises the transparency of the model learned, especially if very large and/or trained on cloud platforms (Vasudevan et al., 2021).

Reactive algorithms, on the other hand, simply generate trajectories for a robot to follow based on the position of obstacles or targets in the environment without requiring

any training. Trajectories are usually calculated based on nodes in a graph, randomly generated (see Karur et al., 2021 for an in-depth review), or tailored to sensory inputs using force fields (e.g. Vector Field Histogram (Borenstein and Koren, 1991) or Artificial Potential Fields (Khatib, 1985)). From a control theory perspective, trajectories can be viewed as a series of desired inputs for a single controller, representing the desired locations of a robot in a global frame of reference. Loop closure in this case consists of providing feedback on how closely the robot follows its trajectory. Trajectory following is a form of input control; however, feedback input is usually provided by an external observer (e.g. a camera placed on the ceiling), as a robot is oblivious to its own objective trajectory. Contrast this with a Braitenberg vehicle, which only needs information about the location of a disturbance (e.g. obstacle or target) relative to itself to successfully perform a control action.

Like in RL, motor plans for robots should consist of sequences of actions, taken in response to inputs. However, in even simple animals like the common fruit fly, complex plans in unknown environments can be formulated completely on-the-fly (Honkanen et al., 2019). This challenges the notion that learning through trial and error is necessary at all for planning. In Spelke and Kinzler (2007) the concept of “core knowledge” is introduced. This is intended as a seemingly innate understanding of basic properties of one’s environment such as physics and causality, and is backed by studies on newborn animals. An agent equipped with core knowledge can form reasonable predictions over the outcome of an action without needing to perform it, which benefits the efficiency of the decision making process. A contribution in the direction of providing a system with innate knowledge has been attempted in few-shot RL. In this paradigm, the goal

is for a system to be able to choose and adapt policies to unseen scenarios with just a few training examples. This is achieved by using meta-learning (Luo et al., 2021; Bing et al., 2023) or transfer-learning (Sun et al., 2023) algorithms, or both (Shu et al., 2023) which, in brief, use previously trained RL models as a foundation for a novel model to be trained. While able to outperform pure RL in terms of training time, both the preexisting and the novel models are trained offline, suggesting that the response time of few-shot learning may still be unsuitable for real-time training.

Our research addresses the problems of how to achieve multi-step-ahead planning using closed-loop control, and of how to implement core knowledge in an agent as a tool to aid decision-making. To tackle the first problem, we define various control loops, which we will call “tasks”, each representing a distinct control strategy. For example, we may have a loop which only counteracts obstacles by turning left, or one that only turns right, etc. Control loops are contingent to specific sensory inputs that make them necessary (Porr and Wörgötter, 2005). They are therefore temporary and can be chained sequentially in a variety of combinations. As for the second problem, we introduce core knowledge (Spelke and Kinzler, 2007) as a simulation which runs in parallel to and mirrors the real world. This allows us to simulate sequences of tasks representing potential plans. Our framework, therefore, is characterised by the concurrent operation of temporary closed-loop controllers both physically and in simulation. This setup necessitates an overarching supervising module (Porr and Wörgötter, 2005) which oversees the task creation process. This module, which we will call “Configurator” (cfr. LeCun (2022)) directs the simulation, interprets its results to formulate a plan, and executes plans by creating and terminating tasks accordingly. Crucially, both task execution and

planning are formulated as homeostatic control problems: in the former, motor outputs are generated in order to counteract sensory stimuli which pose a disturbance to a desired state, and in the latter, a plan is chosen which maximises the time spent in that state.

The paper is organised as follows. In Section 2.2 we describe in detail the components of the proposed framework; in Section 2.3 we discuss the features of our specific implementation; in Section 3 we present the results of the experimental validation of the framework; in Section 4 we discuss the results in light of the state of the art.

## 2 Methods

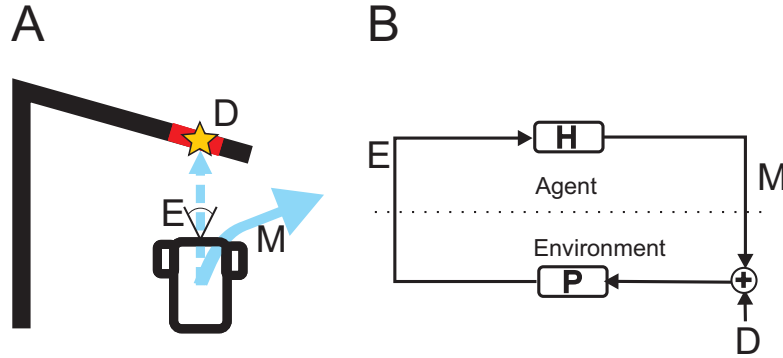


Figure 1: A) closed-loop obstacle avoidance. B) a closed-loop controller. D: disturbance (i.e. collision point), E: sensor error signal between robot and disturbance, M: motor output, P: environmental transfer function, H: transfer function of the agent/robot.

## 2.1 Overview of proposed approach

In Figure 1A we present an informal definition of a task. A task begins when a disturbance (in this case obstacle  $D$ ) enters the robot’s sensor range. The sensors produce a signal  $E$  which is relayed directly to the motor effectors which produce reflex motor output  $M$ . A task executes until the disturbance ceases to produce a signal, at which point the task simply terminates. When no disturbance is present, the system falls back on a “default” task which characterises the resting state. Importantly, unlike in classical closed-loop control, where the control problem is tackled by a single controller at all times, each task is discarded upon termination and may be replaced by any other task.

Figure 2 provides an overview of how tasks and core knowledge can be combined to generate plans in the form of task sequences. Tasks are denoted with letter  $T$ , core knowledge is depicted as a thought balloon (Figure 2A-B), and the Configurator is represented in Figure 2C. The “default” behaviour is in this case arbitrarily defined as driving straight.

In the beginning, the Configurator is executing an initial task  $T_0$ , depicted in Figure 2D-E, where a robot by default drives straight ahead (this task type is denoted by  $T_S$ ) with no disturbance to counteract ( $D_0 = \emptyset$ ). Figure 2A represents the use by the Configurator of core knowledge to simulate tasks ahead of the present: the simulation results represent the scenarios likely to arise from the execution of the sequence. These scenarios, comprised of tasks and the disturbances encountered as a result of executing them, can be collected in a searchable structure, as depicted in Figure 2B. Nodes in this structure are denoted as  $q$ , and they each contain a task and any disturbance (denoted as a star) encountered in its simulation .

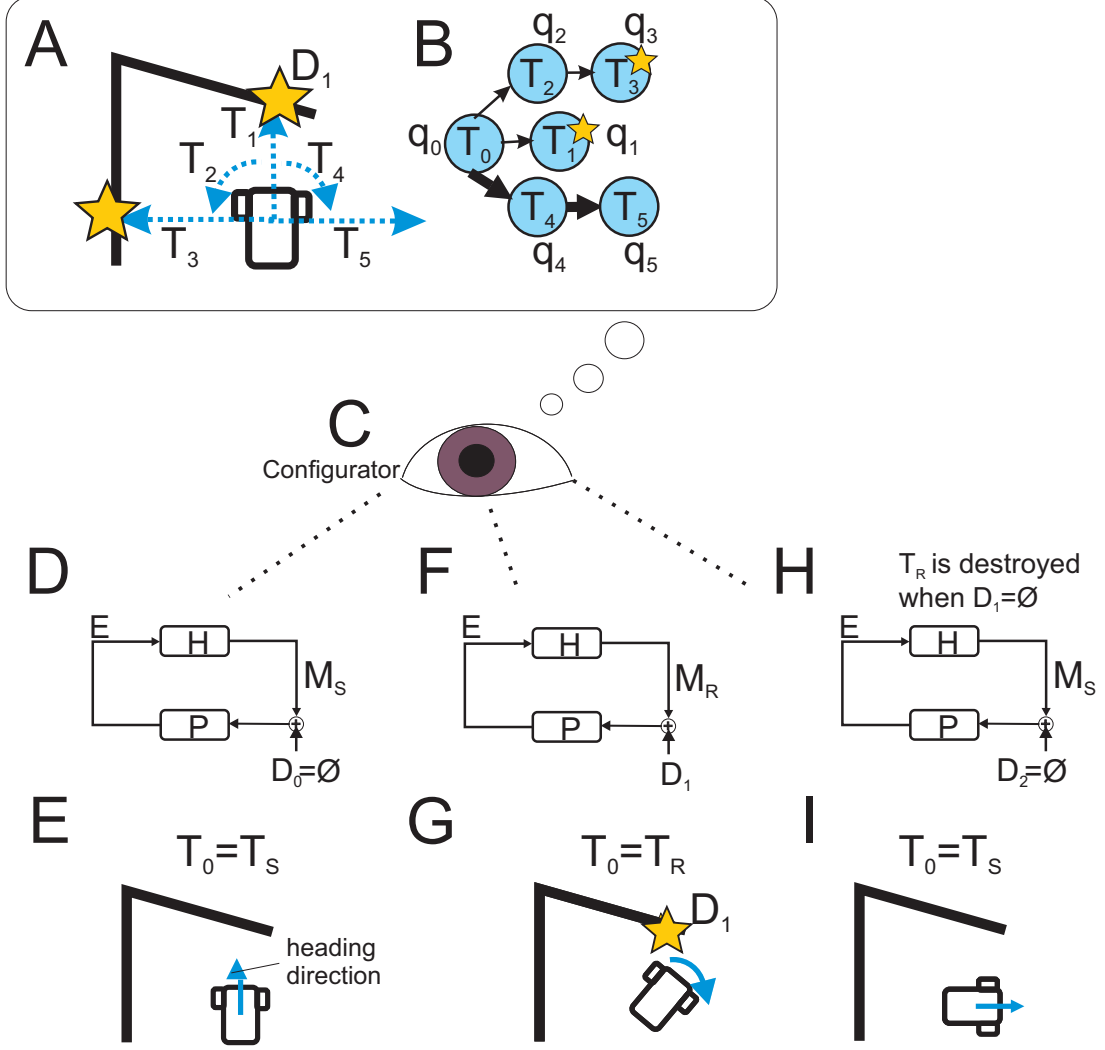


Figure 2: Example of the multi-step-ahead planning procedure carried out by the Configurator. Disturbances are indicated with stars.

The initial task  $T_0$  is represented in Figure 2B in node  $q_0$ , and from it stem three task sequences. First, default task  $T_1$  is created: through its simulation (Figure 2A), the Configurator finds that the robot will collide with obstacle  $D_1$  (depicted as a star) represented by the wall in front. Task  $T_1$  is in node  $q_1$  in Figure 2B. The second control strategy is a left turn (denoted with type  $T_L$ ) represented by task  $T_2$ , which executes successfully in simulation. Task  $T_2$  is followed by default task  $T_3$ , which ends in a collision. This sequence can be observed in Figure 2B as nodes  $q_2, q_3$ . Lastly, the



Configurator explores a third strategy of following  $T_0$  with task  $T_4$ , a right turn (denoted with type  $T_R$ ).  $T_4$  executes without disturbances and is followed by default task  $T_5$ . These tasks are nodes  $q_4, q_5$  in Figure 2B.

The process of finding a plan involves a search over the structure in Figure 2B. In our case, this search produces the disturbance-free sequence  $T_4, T_5$  (indicated by thick arrows in Figure 2B), which will be queued for execution. The execution is shown in Figures 2F-G: initial task  $T_0$  has been overwritten as a task of type  $T_R$ , constructed to counteract the obstacle  $D_1$ . As predicted, no disturbances are found to arise during its execution (the simulation is omitted): as in Figures 2H-I, when  $T_0$  ends, it is again overwritten as another default task  $T_S$ , initialised with a void disturbance  $D_2 = \emptyset$ .

## 2.2 Theoretical foundations

### 2.2.1 Tasks as closed-loop controllers

Formally, we define tasks as closed-loop controllers, as depicted in Figure 1B. Tasks are contingent to a disturbance  $D$  which enters the Environment transfer function  $P$ , and generates an error signal  $E$ . This signal is used by the Agent (i.e. the robot) transfer function  $H$  to generate a motor output  $M$  aimed at counteracting the disturbance (e.g. avoiding the obstacle in Figure 1A). Once the disturbance is counteracted, the error signal becomes zero: at this point, the control motor output  $M$  is no longer needed and the execution of the control loop terminates.

We define three discrete types of tasks representing the behaviours of driving straight ahead ( $T_S$ ), turning left ( $T_L$ ), and turning right ( $T_R$ ), and we call the corresponding outputs  $M_S, M_L, M_R$ , respectively. Of these, we arbitrarily set tasks  $T_S$  as the “default”

(see Section 1) behaviour, so that in a disturbance-free scenario the robot just drives straight ahead. Further, we allow our robot to exhibit obstacle avoidance and target pursuit. We therefore divide disturbances between obstacles and targets by assigning them, respectively, a label through the function  $type : D \rightarrow \{obstacle, target\}$ .

### 2.2.2 Hybrid control

The simulated task-chaining process has a discrete domain i.e. the task which is being simulated, and a continuous one, i.e. the evolution of sensory inputs over time during task simulation. For this reason, the rules underlying task simulation can be summarised using a nondeterministic hybrid automaton (Henzinger, 2000; Raskin, 2005; Lafferriere et al., 1999).

A hybrid automaton is a tuple  $\mathcal{HA} = (\mathbf{T}, C, F, K, I, J, R)$ , where

- $\mathbf{T}$  is a finite set of control modes, or the discrete states of the system.
- $C$  is a set of variables representing the continuous state of the system. We use  $\dot{C}$  to represent the derivatives of  $C$ , or the evolution of the continuous variables over time, and “primed” variables  $C'$  to denote the valuation of  $C$  with which the next control mode is initialised.
- $K \subseteq \mathbf{T} \times \mathbf{T}$  represents the edges, or permitted transitions between control modes
- $F, I$  are functions assigning each a predicate to each control mode.

$F : \mathbf{T} \rightarrow C \cup \dot{C}$  is the flow function.  $F(T_x)$  defines the possible continuous evolutions  $\dot{C}$  of the system in control mode  $T_x$ .

$I : \mathbf{T} \rightarrow C$  is the invariant function.  $I(T_x)$  represents the possible valuations for variables  $C$  in control mode  $T_x$ .

- $J, R$  are functions which assign a predicate to each edge.

$J : K \rightarrow C$  is the jump function.  $J(k)$  is a guard which determines when the discrete control mode change represented by edge  $k \in K$  is allowed based on the valuation of the continuous component  $C$ .

$R : K \rightarrow C \cup C'$  is the reset function.  $R(k)$  assigns to edge  $k$  possible updates for the variables when a discrete change occurs. This is represented by an assignment to primed variables  $C'$ .

In our implementation (discussed in more depth in Section 2.3), the discrete “control modes” are tasks and the continuous variables are sensory inputs. The flow functions are the velocities corresponding to task-specific motor outputs, the invariant predicates define inputs to which each task is contingent, the jump guards set conditions which must be met by the continuous inputs for a transition along a certain edge, and resets assign values to the continuous variables  $C'$  which will determine the initial inputs at the start of a certain task.

The unfolding over time of a hybrid automaton can be represented as a transition system  $(Q, \hookrightarrow)$ , where  $Q = \mathbf{T} \times C$  is a set of states, and  $\hookrightarrow \subseteq Q \times Q$  is a transition relation corresponding to edges connecting states. This transition system represents a model of the environment constructed through simulated interaction. We denote

$$Pre(q) = \{q' \in Q \mid (q' \hookrightarrow q)\} \quad (1)$$

$$Post(q) = \{q' \in Q \mid (q \hookrightarrow q')\} \quad (2)$$

Further, we define a path  $\rho = q_0 q_1 \dots q_n$  where  $q_i \in Post(q_{i-1}) \forall i \geq 1$ , where  $q_0$  is the root state in the transition system. Paths represent sequences that may potentially constitute a plan.

### 2.2.3 The Configurator as a supervisor

We define the Configurator as a type of supervisor (Ramadge and Wonham, 1984):

$$\text{Configurator} = (\mathcal{HA}, \mathcal{E}, \mathcal{P}, goal) \quad (3)$$

where  $\mathcal{HA}$  is a hybrid automaton as described in Section 2.2.2,  $goal = \{D_{goal}\}$  is the system’s overarching goal, or the disturbance which the *plan* aims to counteract,  $\mathcal{E} : \mathcal{HA} \rightarrow Q \times \hookrightarrow$  is an “explorer” function which generates transition system  $\mathcal{G} = (Q, \hookrightarrow)$  representing possible runs of  $\mathcal{HA}$ , and  $\mathcal{P} : Q \times \hookrightarrow \times goal \rightarrow Q$  is a “planner” function which extracts a goal-directed path from a transition system.

## 2.3 Implementation

### 2.3.1 Robot

We implement the outlined framework on an indoor robot based on the Alphabot<sup>1</sup> (see Figure 3A). The robot is equipped with a Raspberry Pi Model 3b+ (Broadcom BCM2837B0 processor with a 1.4 GHz clock speed and 1 GB RAM), an A1 Slamtec 2D LIDAR sensor and two 360° continuous rotation servos which move the wheels.

---

<sup>1</sup><https://www.waveshare.com/alphabot-robot.htm>

The LIDAR callback runs at 5 Hz and the motors are updated at 10 Hz. We define the motor update as  $step_{motor}$ .

### 2.3.2 Core knowledge as a physics simulation

We represent core knowledge as simulation in the physics engine Box2D<sup>2</sup> where the coordinates represents the robot’s local frame of reference. All data pertaining to position and its derivatives are expressed as 2D transforms of the form  $\{u, \theta\}$ , where  $u = (x, y)$  represents the position in Cartesian coordinates in meters and  $\theta$  is the orientation in radians of the object. Obstacles in Box2D are objects of dimensions 0.001m x 0.001m located at coordinates extracted from LiDAR input. The robot was modelled as a simple rectangle of dimensions 0.22m  $\times$  0.18m on the local x and y axis, respectively, with the center of mass shifted forward on the local x axis by 0.05m (see Figure 3A).

The physics simulation may progress for an arbitrarily long (simulated) time, and the world and its content are updated with each arbitrarily small time step  $step_{Box2D}$ . Simulated sensors can be added to objects which report collisions between objects. Task simulation is interrupted as soon as a collision is reported, and the first point of contact between fixtures is returned as the disturbance.

We limit the range of the sensor data used for the simulation to  $r = 1\text{m}$  from the origin. This represents a planning horizon, and the simulation stops when its limit is reached. The simulation is advanced using a kinematic model of the robot with a time step ( $step_{Box2D} = \frac{1}{60}\text{s}$ ), 3 position iterations, 8 velocity iterations so that a simulation carries on for a variable number of steps, indicated as  $n_{Box2D}$ .

---

<sup>2</sup><https://box2d.org/>

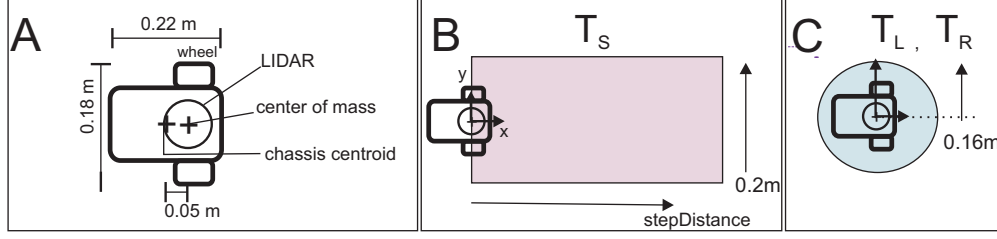


Figure 3: A: Schematic representation of the robot with dimensions and center of mass annotated. The shaded areas in B and C represent, the boundaries for the inclusion of objects in the simulation of tasks  $T_S$  (B),  $T_L$  and  $T_R$  (C).

The speed of the Box2D simulation is sensitive to the number of simulated objects: to reduce computational demands, we perform some simple filtering on the LiDAR coordinates as depicted in Figure 3B-C. For simulating tasks of type  $T_S$ , we only represent points located within a distance  $stepDistance$  from the robot’s center of mass, where  $stepDistance$  denotes a finite distance range in meters for that task type. For left and right turns  $T_L$  and  $T_R$  in Figure 3C we only retain points located within a radius of 0.16m from the robot’s center of mass. The filtered data thus obtained is further filtered using one of the following techniques. The first technique, “worldbuilding1”, consists of approximating each coordinate to two decimal places and eliminating redundant value. For the second, “worldbuilding2”, only every other point in the set constructed using “worldbuilding1” is retained. Simulation results can be used to determine motor commands for task execution in the real world. We calculate each motor command as a duration expressed as a number of motor update intervals

$$n_{motor} = n_{Box2D} \cdot step_{motor} \cdot step_{Box2D} \quad (4)$$

where  $n_{Box2D}$  is the number of steps of size  $step_{Box2D}$  taken to bring a task to termina-

tion in simulation and  $n_{Box2D} \cdot step_{Box2D}$  calculates the simulated time which the task has taken.

### 2.3.3 Hybrid automaton

The hybrid automaton modelling the simulated task-switching process is depicted in Figure 4. The discrete control modes corresponding to tasks  $\mathbf{T} = \{T_S, T_R, T_L\}$ , are

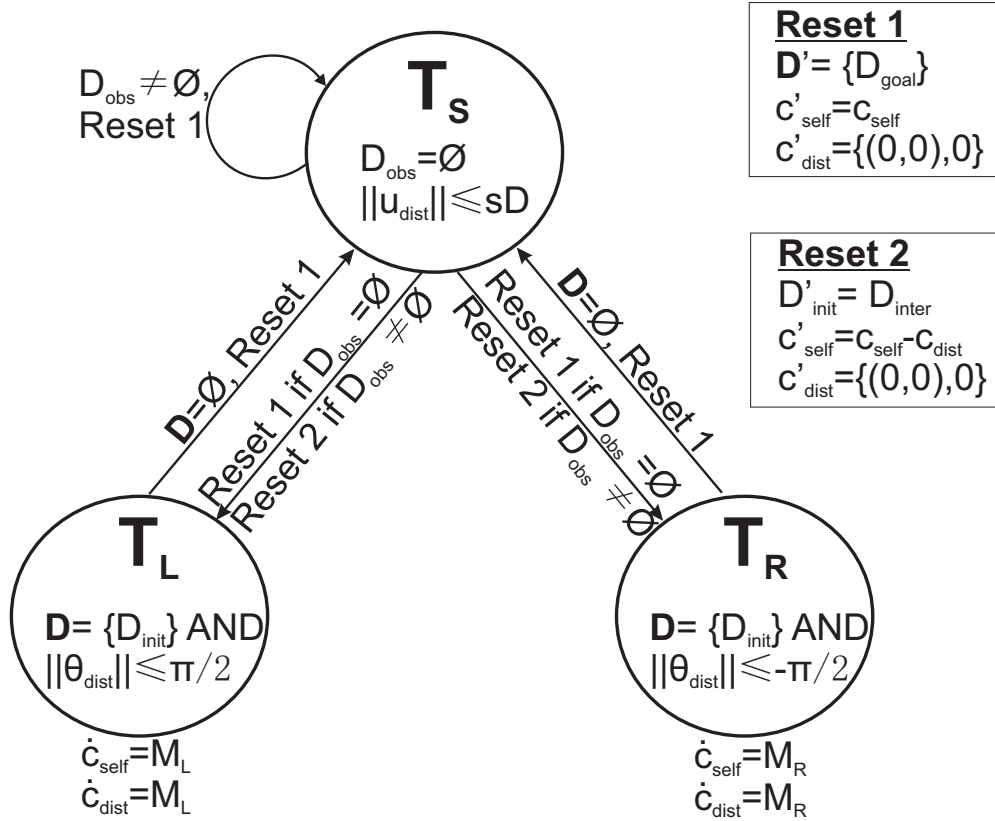


Figure 4:  $\mathcal{HA}$ : nondeterministic hybrid automaton modelling the closed-loop behaviour of the robot. We define  $u_{dist}, \theta_{dist} \in c_{dist}$ , and  $sD = stepDistance$ .

inscribed in circles connected by arrows representing the edges  $K$ . Continuous variables  $C = \{c_{self}, c_{dist}, \mathbf{D}\}$  correspond to sensor inputs in the form of 2D transforms. These inputs are: the position of the robot  $c_{self}$ , the change in the robot's position since the start of the task  $c_{dist}$ , and disturbance set  $\mathbf{D} = \{\mathbf{D}_{init}, \mathbf{D}_{inter}\}$ , comprised of the

disturbance to which the task is contingent  $D_{init}$ , and the disturbance interrupting task execution  $D_{inter}$ . Flow  $F$  predicates are annotated outside of each circle. Invariant  $I$  predicates are annotated inside each circle. Jump  $J$  and reset  $R$  predicates are located by the corresponding edges.

Task  $T_S$  models the behaviour of the system when it is driving straight, while tasks  $T_L$  and  $T_R$  model the behaviours of turning left and right, respectively. As indicated by the respective flow predicates, the position of the robot  $c_{self}$  and the position change since the start of the task  $c_{dist}$  evolve at a rate denoted by task-specific velocities  $M_S = \{(0.098\text{m/s}, 0\text{m/s}), 0\text{rad/s}\}$  for driving stright,  $M_L = \{(0\text{m/s}, 0\text{m/s}), 1.04\text{rad/s}\}$  for left turns, and  $M_R = \{(0\text{m/s}, 0\text{m/s}), -1.04\text{rad/s}\}$  for right turns, all in local coordinates. These values were calculated applying simple robot kinematics to empirical velocity measurements.

As indicated by the corresponding invariant predicate, task  $T_S$  executes as long as no obstacles (defined as  $D_{obs} = \{D \in \mathbf{D} \mid \text{type}(D) = \text{obstacle}\}$ ) are present and the robot has not yet covered a fixed distance  $stepDistance$  (abbreviated to  $sD$  in Figure). As indicated by the respective invariant predicates, tasks  $T_L$  and  $T_R$  keep executing as long as the only disturbance in the continuous variables is the one on which the task is contingent (predicate  $\mathbf{D} = \{D_{init}\}$ ), and the angle described by the turn has not yet reached  $\frac{\pi}{2}$  rad (for task  $T_L$ ), and  $-\frac{\pi}{2}$  rad for task  $T_R$ . Note that these invariant predicates still allow for a turn to execute in the case a task may be initialised with an empty disturbance.

Transitions along edge  $(T_S, T_S)$  are permitted if the terminated task has not encountered obstacles, as indicated by jump predicate  $D_{obs} \neq \emptyset$ . Transitions along edges



$(T_L, T_S)$  and  $(T_R, T_S)$ , are permitted if the left or right turns are terminated with no disturbances, as indicated by jump predicates  $\mathbf{D} = \emptyset$ . Transitions along edges  $(T_S, T_L)$  and  $(T_S, T_R)$  are always permitted, as indicated by the lack of jump guards.

The boxes on the right-hand side of Figure 4 contain the two reset predicates, “Reset1” and “Reset2”, which assign to primed variables  $C'$  depending on whether the most recently simulated task is terminated without disturbances, or if it encounters an obstacle, respectively. For all reset predicates, every time the system switches control mode, the distance covered in the previous task is reset to a zero transform  $c'_{dist} = \{(0\text{m}, 0\text{m}), 0\text{rad}\}$ . The “Reset1” predicate states that the next task will start where the previous one ended ( $c'_{self} = c_{self}$ ) and will be aimed at reaching the goal ( $\mathbf{D}' = \{D_{goal}\}$ ). The “Reset2” predicate states that the next task will be primed to counteract the obstacle ( $D'_{init} = D_{inter}$ ) which interrupted the previous task, and it will start where the previous task began ( $c'_{self} = c_{self} - c_{dist}$ ). “Reset1” always applies to edges  $(T_S, T_S)$ ,  $(T_L, T_S)$ ,  $(T_R, T_S)$ , and to edges  $(T_S, T_L)$  and  $(T_S, T_R)$  if the continuous state  $C$  does not contain obstacles ( $D_{obs} = \emptyset$ ). If obstacles are present ( $D_{obs} \neq \emptyset$ ) in these last two edges, “Reset2” applies.

### 2.3.4 The explorer $\mathcal{E}$

The explorer  $\mathcal{E}$  uses a best-first approach to construct transition system  $\mathcal{G}$  from automaton  $\mathcal{HA}$ . The priority of expansion of a state is defined by its cumulative past and expected cost (cfr. Hart et al. 1968), which we will define as function  $\phi : Q \rightarrow \mathbb{R}$ . Additionally, we use a heuristic function  $\sigma : Q \rightarrow \{0, 1\}$  to prevent the robot from turning more than 180 degrees on the spot and getting stuck rotating in place.

We calculate cost evaluation function  $\phi$  as:

$$\phi(q) = \gamma(q) + \chi(q) \quad (5)$$

where  $q$  are states in transition system  $\mathcal{G}$ ,  $\gamma(q)$  calculates the past cost associated with state  $q$  and  $\chi(q)$  is a heuristic function which estimates the cost to reach goal  $D_{goal}$  from state  $q$ .

We define the past cost function  $\gamma$  as:

$$\gamma(q) = \begin{cases} 0 & \text{if } D_{inter} = \emptyset \\ \frac{2r - \|u_{start} - u_{inter}\|}{2r} & \text{if } D_{inter} \neq \emptyset \end{cases} \quad (6)$$

where  $D_{inter}$  is a disturbance encountered in state  $q$ ,  $2r$  represents the maximum possible distance that the robot can be from a disturbance, and  $\|u_{start} - u_{inter}\|$  represents the Euclidean distance between task start position  $u_{start} = u_{self} - u_{dist}$  (see Section 2.2.2) and disturbance  $D_{inter}$ .

We define the cost heuristic function:

$$\chi(q) = \begin{cases} 0 & \text{if } goal = \emptyset \\ \frac{\|u_{self} - u_{goal}\|}{4r} & \text{otherwise} \end{cases} \quad (7)$$

where  $\|u_{self} - u_{goal}\|$  represents the distance between the robot's position at the end of the simulation of state  $q$  and target  $D_{goal}$ . We further define heuristic function  $\sigma$  as:

$$\sigma(q) : \|u_{self} - u_{pre}\| = 0 \wedge |\theta_{self} - \theta_{pre}| \geq \pi \quad (8)$$

where 2D transforms  $\{u_{self}, \theta_{self}\}$  and  $\{u_{pre}, \theta_{pre}\}$  represent the robot's position and orientation at the end of the simulation of state  $q$  and  $Pre(Pre(q))$ , respectively.

Algorithm 1 provides pseudocode for explorer  $\mathcal{E}$ . We start from a transition system  $\mathcal{G} = \{q_0\}$  and priority queue  $PQ = \{q_0\}$ , where  $q_0$  is a starting state representing the task being executed by the robot. The corresponding task is immediately terminated, in order for the system to explore the scenarios that would arise if the robot were to change its behaviour immediately, without waiting for the task at state  $q_0$  to terminate.

We use  $q_{exp}$  to denote the state with the lowest evaluation  $\phi(q_{exp})$  in the priority queue; its out-edges are explored with the highest priority. We further denote the state whose out-edges are actively being explored with  $q_{exp0}$  and the frontier state with  $q_{exp1}$ . Any permitted transitions from state  $q_{exp0}$  are explored in a depth-first fashion until frontier state  $q_{exp1}$  contains a task  $T_S$  (zero to two states ahead of state  $q_{exp}$ ). After the expansion of  $q_{exp}$  is complete,  $q_{exp}$  is reset to the node which has the smallest  $\phi$  value in the priority queue, and the process repeats until a path is found which leads to a goal or the planning horizon. Note how this implies that only tasks of the  $T_S$  type are evaluated and that  $q_{exp}$  is, therefore, always a task  $T_S$ .

### 2.3.5 Planner $\mathcal{P}$

The model generated by the explorer  $\mathcal{E}$  is used by planner  $\mathcal{P}$  defined in Section 2.2.3, which returns a path  $\rho_{plan}$  which brings the robot as close to the goal as possible. Formally we write:

$$\rho_{plan} = \{q_0, q_1, q_2, \dots, q_n \in \mathcal{G} \mid Post(q_n) = \emptyset \wedge q_n = \min_{q \in Q} \phi(q)\} \quad (9)$$

---

**Algorithm 1** Explorer  $\mathcal{E}$ 

---

```
inputs:  $\mathcal{HA}$ 
initialise  $PQ = \{q_0\}, \mathcal{G} = \{q_0\}, q_{exp} = q_0$ 
simulate the termination of  $q_0$ : set  $\mathbf{D}$  assumed empty,  $c_{self} = \{(0m, 0m), 0rad\}$ 
repeat
  remove  $q_{exp}$  from  $PQ$ 
  for all edges  $k = (q_{exp}, q_{exp1})$  for which jumps are permitted do
     $q_{exp0} = q_{exp}$ 
    repeat
      simulate  $T_{exp1} \in q_{exp1}$ 
      set  $q_{exp0} = q_{exp1}$ 
    until  $T_{exp1} = T_S$ 
    if  $\sigma(q_{exp1}) = 0$  then
      add  $T_{exp1}$  to  $PQ$ 
    end if
  end for
  set  $q_{exp} = \min_{q \in PQ} \phi(q)$ 
until no more jumps are permitted from  $q_{exp}$  or  $D_{goal}$  has been reached
return  $\mathcal{G}$ 
```

---

### 3 Results

We test the decision-making process by applying it to two real-world problems: one of avoiding of being trapped in a cul-de-sac and one of simultaneous collision avoidance and target pursuit. We contrast our new approach against a Braitenberg-style control strategy which uses a single closed-loop controller and therefore exhibits stereotyped behaviour.

Sequences of tasks and the specifications of the motor commands pertaining to each task are planned at the beginning of the experimental task and do not undergo correction throughout execution. The amount of time taken by the robot to formulate a plan is measured, as well as the total amount of objects simulated in Box2D and the number of states in the transition system, which are equivalent to the number of tasks simulated.

### 3.1 Experiment 1: cul-de-sac avoidance

Our first experiment tests our approach in a cul-de-sac avoidance scenario. This is a classical scenario where reactive behaviour is not sufficient to prevent getting stuck in a cul-de-sac. We set  $goal = \emptyset$  and  $stepDistance = 0.5m$ . The experiment was repeated six times with “worldbuilding1”, and twelve with “worldbuilding2”.

Figure 5 depicts a comparison of the behaviour planned using our new multiple-loop framework (Figure 5A), and the single-loop reactive behavioural case (Figure 5B), and the reasoning process employed using our framework (Figure 5C). As can be observed from the trace in Figure 5A, when the robot uses our multi-step-ahead planning strategy, it identifies the cul-de-sac immediately and, to prevent entering it, turns to its right which allows it to carry on undisturbed. By contrast, in the single-loop control experiment, the robot enters the cul-de-sac, as it is outside of the single loop’s range  $stepDistance$ . Moreover, as signified in yellow lightning, some collisions occur during execution.

Figure 5C offers a visual representation of the construction of transition system  $\mathcal{G}$  constructed in one of the experimental runs and the plan extracted from it. Initial state  $q_0$  is immediately terminated and from its end position (see robot schematic in Figure 5C), transition system building starts. The robot may safely proceed straight (state  $q_1$ ) or turn right and then advance (states  $q_4, q_5$ ). The sequence  $q_2, q_3$  cannot be safely executed since, in state  $q_3$ , the robot collides with the wall ahead of it. States  $q_1$  and  $q_5$  will have the same  $\phi$  value and rank higher in the priority queue than state  $q_3$ .

The cul-de-sac is discovered as all the sequences following state  $q_1$  (namely, state  $q_6$ , states  $q_7, q_8$  and states  $q_9, q_{10}$ ). State  $q_5$  will now therefore have highest priority,

and will be expanded. Three other state sequences are added to this node. Of these, driving right and straight (states  $q_{12}, q_{13}$ ) leads to a crash, while state  $q_{11}$  and sequence  $q_{14}, q_{15}$  are disturbance-free. In this case, state  $q_{11}$  reaches the limit of the planning horizon before state  $q_{15}$ , for which reason planning can end. The final plan, indicated by the bold arrow in Figure 5C, is  $\rho_{plan} = q_0 q_4 q_5 q_{11}$ , where  $q_0 = (T_S, \cdot)$ ,  $q_4 = (T_R, \cdot)$ ,  $q_5 = (T_S, \cdot)$ ,  $q_{11} = (T_S, \cdot)$ .

Figure 5D displays the descriptive statistics pertaining to simulation and planning speed. In runs where worldbuilding1 was used to create objects in the Box2D environment (in Figure, column labelled "cul-de-sac, wb1"), an average of 19.3 states (min: 16, max: 26) were simulated with 93.8 average total objects (SD=39.93) created over the course of the simulation. This took on average 0.295s (SD=0.069s). With worldbuilding2 (in Figure, column labelled "cul-de-sac, wb2"), an average of 20.16 states (min: 11, max: 21) were simulated with 40.75 average total objects (SD=20.67) in 0.166s on average (SD= 0.062s).

### 3.2 Experiment 2: avoidance and target behaviour

Figure 6 represents the results of an experiment where the robot had to reach a target  $goal = \{D_{goal} = \{(1.0m, 0m), 0rad\}\}$  while driving around an obstacle. We set  $stepDistance = 0.22m$  corresponding to the diameter of the robot, as in (Ulrich and Borenstein, 2000). The experiment was repeated ten times with "worldbuilding1", and eight with "worldbuilding2".

As shown in Figure 6B, in the reactive behaviour experiment the robot remains stuck behind the obstacle, while in our approach (Figure 6A) the agent is able to safely

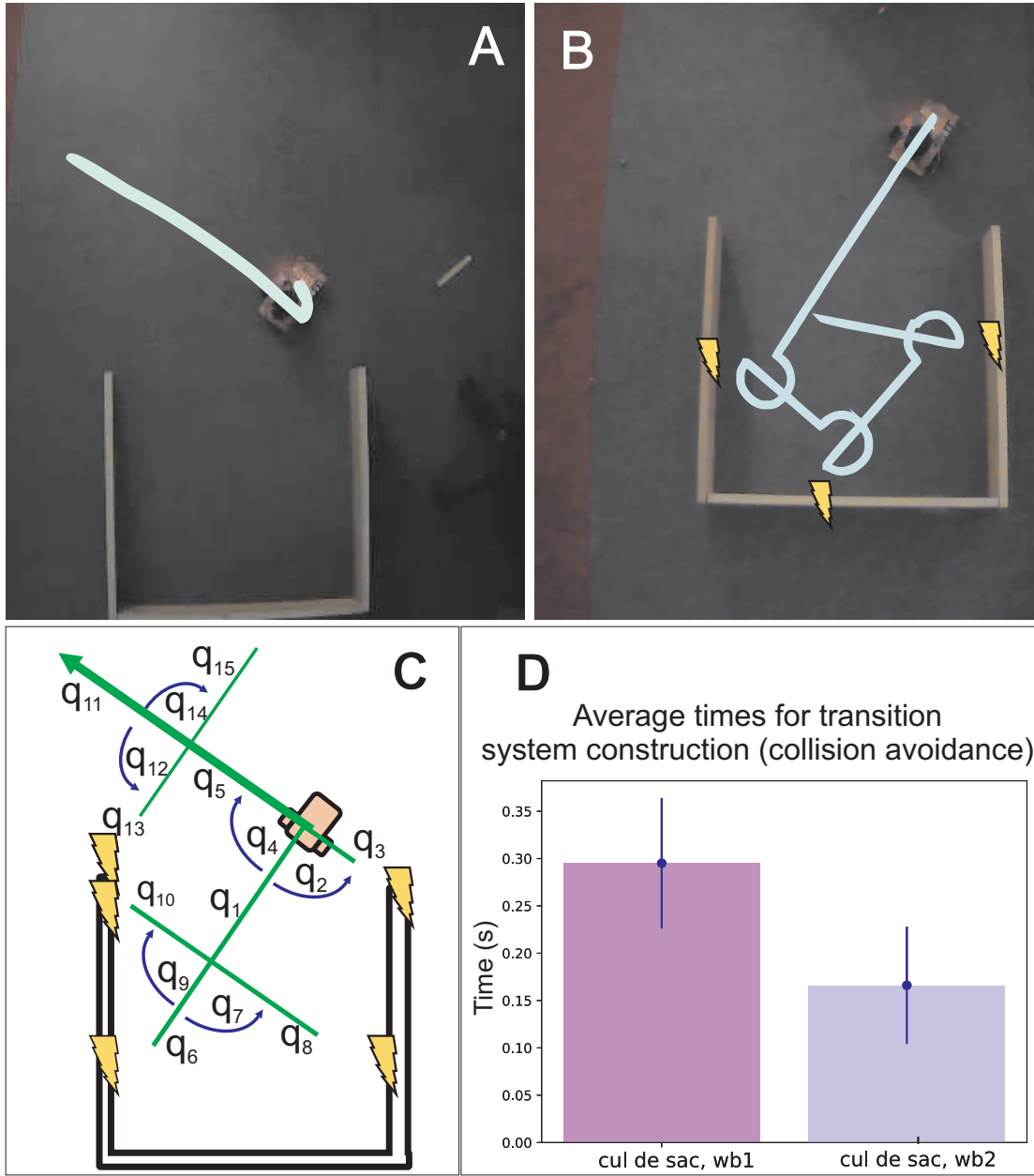


Figure 5: Behaviour exhibited by the robot in the obstacle-avoidance experiment. Panel A: with the proposed multiple-loop planning, panel B: with planning over a single control loop, panel C: reasoning in multiple-loop planning, D: descriptive statistics.

formulate a plan to reach the target.

Figure 6C depicts the transition system produced in one of the experimental runs. As shown, the robot can safely advance towards the target  $D_{goal}$  by entering states

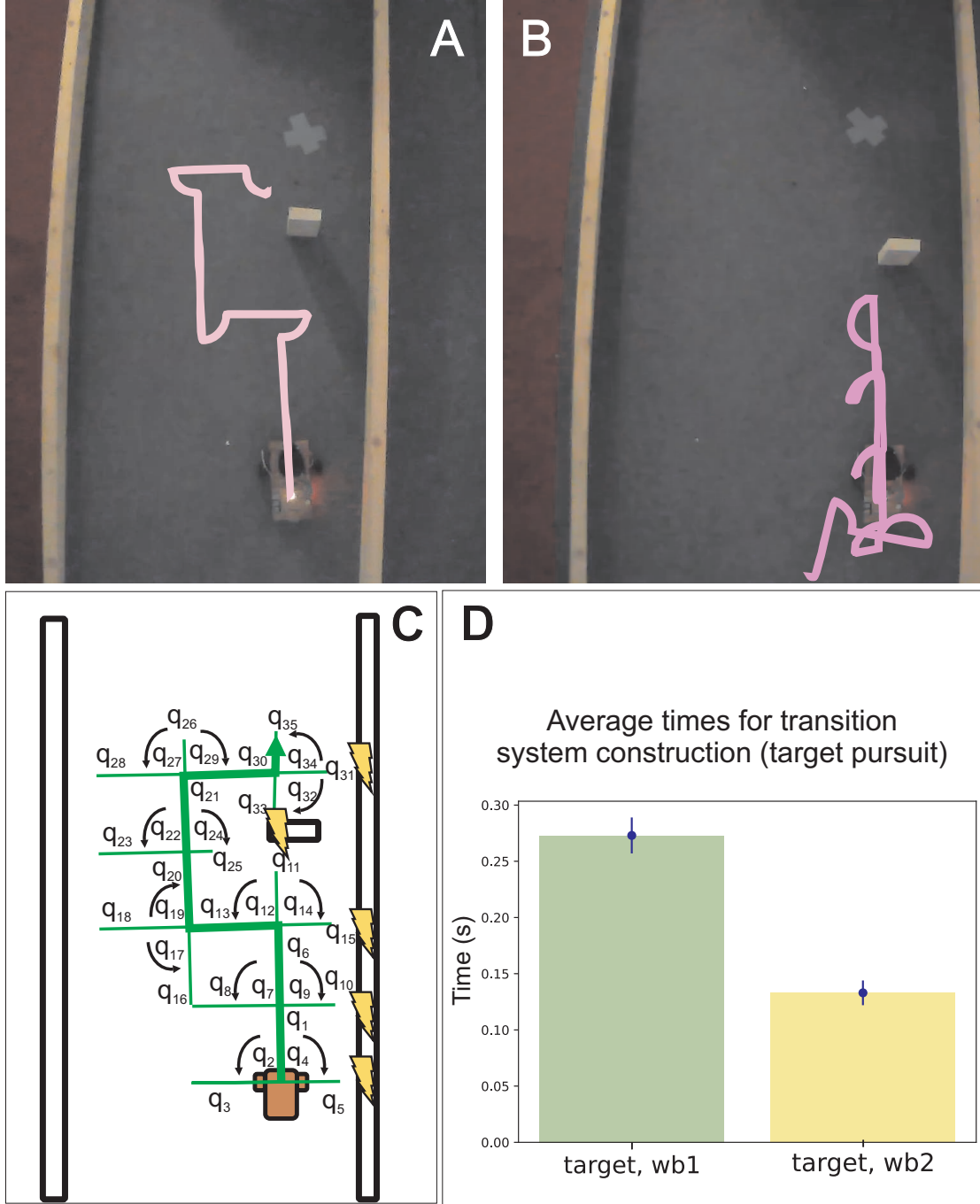


Figure 6: Initial frame and tracking of the behaviour exhibited by the robot in the target-seeking/overtaking experiment. Panel A: with the proposed multiple-loop planning, panel B: with one-loop-ahead disturbance detection, panel C: reasoning in multiple-loop planning, D: descriptive statistics. The target is represented in A-B by a grey cross.



$q_1, q_6$ . Further advancement through state  $q_{11}$  is prevented by an obstacle. The only collision-free option for the robot is to turn left (state  $q_{12}$ ) and proceed in a direction perpendicular to the target (state  $q_{13}$ ). The robot will then turn towards the disturbance by choosing to turn right (state  $q_{19}$ ) and then proceed straight ( $q_{20}$ ). The robot will then immediately attempt to return in line with the target by turning left and then proceeding straight (states  $q_{24}, q_{25}$ ), but these options are not collision-free. By transitioning to state  $q_{21}$ , the robot will drive straight towards the target; after this, it may safely turn right (state  $q_{29}$ ) and return in line with the target (state  $q_{30}$ ). The target is now very close: by turning left (state  $q_{34}$ ), the robot positions itself to face it and advances (state  $q_{35}$ ) until it reaches it, at which point transition system  $\mathcal{G}$  construction terminates. As indicated by the bold arrow in Figure 6C, the plan followed in Figure 6A is  $\rho_{plan} = q_1 q_6 q_{12} q_{13} q_{19} q_{20} q_{21} q_{29} q_{30} q_{34} q_{35}$ , where  $q_1 = (T_S, \cdot)$ ,  $q_6 = (T_R, \cdot)$ ,  $q_{12} = (T_L, \cdot)$ ,  $q_{13} = (T_S, \cdot)$ ,  $q_{19} = (T_R, \cdot)$ ,  $q_{20} = (T_S, \cdot)$ ,  $q_{21} = (T_S, \cdot)$ ,  $q_{29} = (T_R, \cdot)$ ,  $q_{30} = (T_S, \cdot)$ ,  $q_{34} = (T_L, \cdot)$ ,  $q_{35} = (T_S, \cdot)$ .

Descriptive statistics for the simulation and planning speed are shown in Figure 6D. Using worldbuilding1 (in Figure, column labelled "target, wb1"), the transition system was made up of an average of 37.8 states (min: 36, max: 39), and each simulation contained on average 114.4 objects (SD=21.08). The mean simulation time was 0.273s (SD= 0.016s). When worldbuilding2 (in Figure, column labelled "target, wb2") was employed, the mean number of states was similar at 38.25 with the same minimum and maximum values. The average number of objects was reduced to 64.38 (SD=11.70), and average simulation time dropped to 0.133s (SD=0.011s).

## 4 Discussion

In this paper we have introduced a novel decision-making process which exploits supervisory control of a hybrid automaton representing temporary simulated closed-loop controllers, called “tasks”, representing each a different control strategy. The simulation of the task-switching process occurs in a physics engine which constitutes the system’s core knowledge. The performance of the decision-making process was tested in a real robot in a collision avoidance problem and a target pursuit one. The robot was in both cases able to generate, in real-time, a model of tasks and their resulting disturbances over a 1m range, which was used to extract a plan. We contrasted this performance with a classical closed-loop system where planning can only occur one disturbance into the future and the system behaves deterministically. This control case produced less efficient behaviour in the best case scenario and failed by colliding with a wall or remaining stuck in the worst.

We construct a model of the environment as a hybrid transition system representing possible runs of a hybrid automaton. Problems concerned with the combination of discrete and hybrid control are referred to as Task And Motion Problems (TAMP). TAMPs are notoriously difficult to solve in real-time: in recent years neural networks have been used to train complex models aimed at combining planning over a hybrid transition system and learning control policies in a variety of scenarios (Kim et al., 2022; Driess et al., 2021). This obviously requires offline training and has potential for limited transparency and/or robustness depending on the depth of the network used. Additionally, even with previous training, planning times in Kim et al. (2022) fell into the order of hundreds of seconds; this is not suitable for real-world applications.

Online approaches to hybrid control for navigation have also been proposed. In Mavridis et al. (2019); Constantinou and Loizou (2018); Migimatsu and Bohg (2020), hybrid automata are constructed using various types of logic to check that they satisfy certain properties. Whether the control modes may be immediately available to the system (Migimatsu and Bohg, 2020) or may need to be computed (Mavridis et al., 2019), the computation of the control-mode-specific flow function can be an inefficient process. Computation statistics were not reported in Mavridis et al. (2019); Constantinou and Loizou (2018), however Migimatsu and Bohg (2020) report that the minimum time taken for convergence to an optimal continuous control was 0.85 seconds. A model-checking approach was proposed by Chandler et al. (2023) over a discrete-state system, which was able to plan over a sequence of five actions in around 0.01s. This suggests that the optimisation over continuous controls may be the bottleneck for TAMP problems.

While we do not focus on achieving smooth trajectories or more complex tasks such as grasping, we were able to achieve planning in real-time or near-real-time in a robot with very limited computational power. The closed-loop task simulation allowed us to easily obtain an estimation of the commands for the motor effectors. Starting from a reasonable estimation of a control output may relieve some of the computational load resulting from its optimisation, which should occur online and in a closed-loop fashion. Despite the increased planning time in our experiments compared to Chandler et al. (2023), the speed performance recorded in our experiments is still within human reaction time range (Wong et al., 2015) and within real-time requirements of the robot.

In conclusion, our findings demonstrate that not only it is possible to formulate

optimal plans based on pure input control, but it is also achievable in real-time over the discrete and continuous domains, thanks to core knowledge. We believe that our framework has potential to scale well to TAMP problems, in at least two dimensions, and provide a valuable complement to the state-of-the-art.

## 5 Acknowledgements

This work was supported by a grant from the UKRI Engineering and Physical Sciences Research Council Doctoral Training Partnership award [EP/T517896/1-312561-05]; the UKRI Strategic Priorities Fund to the UKRI Research Node on Trustworthy Autonomous Systems Governance and Regulation [EP/V026607/1, 2020-2024]; and the UKRI Centre for Doctoral Training in Socially Intelligent Artificial Agents [EP/S02266X/1].

## References

- Zhenshan Bing, David Lerch, Kai Huang, and Alois Knoll. Meta-reinforcement learning in non-stationary and dynamic environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45:3476–3491, 3 2023. ISSN 19393539. doi: 10.1109/TPAMI.2022.3185549.
- Johann Borenstein and Yoram Koren. The vector field histogram—fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7:278–288, 1991. doi: 10.1109/70.88137.

V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, 1986.

Christopher Chandler, Bernd Porr, Alice Miller, and Giulia Lafratta. Model checking for closed-loop robot reactive planning. *Electronic Proceedings in Theoretical Computer Science*, 395:77–94, 11 2023. doi: 10.4204/EPTCS.395.6.

Christos C. Constantinou and Savvas G. Loizou. Automatic controller synthesis of motion-tasks with real-time objectives. *Proceedings of the IEEE Conference on Decision and Control*, 2018-December:403–408, 7 2018. doi: 10.1109/CDC.2018.8619825.

Steven Dalton and Iuri Frosio. Accelerating reinforcement learning through gpu atari emulation. *Advances in Neural Information Processing*, 33:19773–19782, 2020.

Danny Driess, Jung Su Ha, and Marc Toussaint. Learning to solve sequential physical reasoning problems from a scene image. *International Journal of Robotics Research*, 40:1435–1466, 12 2021. ISSN 17413176. doi: 0.1177/02783649211056967.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968. ISSN 21682887. doi: 10.1109/TSSC.1968.300136.

Thomas A. Henzinger. The theory of hybrid automata. *Verification of Digital and Hybrid Systems*, pages 265–292, 2000. ISSN 1043-6871. doi: 10.1007/978-3-642-59615-5\_13.

Anna Honkanen, Andrea Adden, Josiane Da Silva Freitas, and Stanley Heinze. The insect central complex and the neural basis of navigational strategies. *Journal of*

*Experimental Biology*, 222, 2 2019. ISSN 00220949. doi: 10.1242/JEB.188854/2882.

Karthik Karur, Nitin Sharma, Chinmay Dharmatti, and Joshua E. Siegel. A survey of path planning algorithms for mobile robots. *Vehicles 2021, Vol. 3, Pages 448-468*, 3: 448–468, 8 2021. ISSN 2624-8921. doi: 10.3390/VEHICLES3030027.

O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 500–505, 1985. ISSN 10504729. doi: 10.1109/ROBOT.1985.1087247.

Beomjoon Kim, Luke Shimanuki, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Representation, learning, and planning algorithms for geometric task and motion planning. *International Journal of Robotics Research*, 41:210–231, 2 2022. ISSN 17413176. doi: 10.1177/02783649211038280.

Gerardo Lafferriere, George J Pappas, and Sergio Yovine. A new class of decidable hybrid systems. *Hybrid Systems: Computation and Control: Second International Workshop*, pages 137–151, 1999.

Yann LeCun. A path towards autonomous machine intelligence. *OpenReview.net*, 6 2022.

Qian Luo, Maks Sorokin, and Sehoon Ha. A few shot adaptation of visual navigation skills to new observations using meta-learning. *Proceedings - IEEE International Conference on Robotics and Automation*, 2021-May:13231–13237, 2021. ISSN 10504729. doi: 10.1109/ICRA48506.2021.9561056.

Humberto R. Maturana and Francisco J. Varela. *Autopoiesis and Cognition*, volume 42.

Boston Studies in the Philosophy of Science New York, N.Y., 1980. ISBN 978-90-277-1016-1. doi: 10.1007/978-94-009-8947-4.

Christos N. Mavridis, Constantinos Vrohidis, John S. Baras, and Kostas J. Kyriakopoulos. Robot navigation under mitl constraints using time-dependent vector field based control. *Proceedings of the IEEE Conference on Decision and Control*, 2019-December:232–237, 12 2019. ISSN 25762370. doi: 10.1109/CDC40024.2019.9028890.

Toki Migimatsu and Jeannette Bohg. Object-centric task and motion planning in dynamic environments. *IEEE Robotics and Automation Letters*, 5:844–851, 4 2020. ISSN 23773766. doi: 10.1109/LRA.2020.2965875.

Simon Ramstedt Mila and Christopher Pal. Real-time reinforcement learning. 2019. doi: <https://doi.org/10.48550/arXiv.1911.04448>.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dhharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–542, 2 2015. ISSN 00280836. doi: 10.1038/NATURE14236.

OpenAI. Gpt-4 technical report, 2023.

- B Porr and F Wörgötter. Inside embodiment-what means embodiment to radical constructivists? *Kybernetes*, 34:105–117, 2005.
- P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *Analysis and Optimization of Systems*, pages 475–498, 10 1984. doi: 10.1007/BFB0006306.
- Jean-François Raskin. An introduction to hybrid automata. *Handbook of Networked and Embedded Control Systems*, pages 491–517, 2005. doi: 10.1007/0-8176-4404-0\_21.
- Yang Shu, Zhangjie Cao, Jinghan Gao, Jianmin Wang, Philip S. Yu, and Mingsheng Long. Omni-training: Bridging pre-training and meta-training for few-shot learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45:15275–15291, 12 2023. ISSN 19393539. doi: 10.1109/TPAMI.2023.3319517.
- David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–490, 1 2016. ISSN 00280836. doi: 10.1038/NATURE16961.
- Elizabeth S. Spelke and Katherine D. Kinzler. Core knowledge. *Developmental Science*, 10:89–96, 1 2007. ISSN 1363755X. doi: 10.1111/J.1467-7687.2007.00569.X.



- Huihui Sun, Weijie Zhang, Runxiang Yu, and Yujie Zhang. Motion planning for mobile robots - focusing on deep reinforcement learning: A systematic review. *IEEE Access*, 9:69061–69081, 2021. ISSN 21693536. doi: 10.1109/ACCESS.2021.3076530.
- Yuewen Sun, Kun Zhang, and Changyin Sun. Model-based transfer reinforcement learning based on graphical model representations. *IEEE Transactions on Neural Networks and Learning Systems*, 34:1035–1048, 2 2023. ISSN 21622388. doi: 10.1109/TNNLS.2021.3107375.
- Pierre Thodoroff, Wenyu Li, and Neil D Lawrence. Benchmarking real-time reinforcement learning. *Proceedings of Machine Learning Research*, 181:26–41, 2022.
- Iwan Ulrich and Johann Borenstein. Vfh\*: Local obstacle avoidance with look-ahead verification. pages 2505–2511, 2000.
- Rama K. Vasudevan, Maxim Ziatdinov, Lukas Vlcek, and Sergei V. Kalinin. Off-the-shelf deep learning is not enough, and requires parsimony, bayesianity, and causality. *npj Computational Materials* 2021 7:1, 7:1–6, 1 2021. ISSN 2057-3960. doi: 10.1038/s41524-020-00487-0.
- Aaron L. Wong, Adrian M. Haith, and John W. Krakauer. Motor planning. *Neuroscientist*, 21:385–398, 8 2015. ISSN 10894098. doi: 10.1177/1073858414541484.