A Multi-Strategy Architecture for On-Line Learning of Robotic Behaviours using Qualitative Reasoning

Timothy Wiley TIMOTHYW@CSE.UNSW.EDU.AU
Claude Sammut CLAUDE@CSE.UNSW.EDU.AU
Bernhard Hengst BERNHARDH@CSE.UNSW.EDU.AU
School of Computer Science and Engineering, University of New South Wales,
Sydney, NSW 2052, Australia

Ivan Bratko BRATKO@FRI.UNI-LJ.SI
Faculty of Computer and Information Science, University of Ljubljana,
Večna pot 113, Ljubljana, Slovenia

Abstract

A Multi-Strategy Architecture improves the efficiency of on-line learning of robotic behaviours by taking inspiration from approaches humans use for learning complex behaviours. The hybrid approach first learns the qualitative dynamics of a robotic system from which a symbolic planner constructs an approximate solution to a control problem by qualitatively reasoning over the discovered dynamics. The parameters of the approximate solution are refined by numerical optimization, into a policy for a reactive controller. The hybrid approach is demonstrated on a multi-tracked robot intended for urban search and rescue.

1. Introduction

A cognitive architecture for developing complex robotic behaviours faces numerous challenges. To solve complex robotic tasks, such as locomotion, it is desirable to learn the appropriate low-level control (or actuator) actions, rather than use hand-crafted approaches. This robotic learning task is difficult when it is conducted on large continuous domains, using robotic systems that have noisy sensors, inaccurate actuators and are difficult to correctly model. We also pose an additional research challenge of on-line learning, that is, learning as the robot operates. In this paper, we detail a Multi-Strategy architecture for the on-line learning of control actions (Figure 1), that is inspired by approaches that humans typically take to learning complex control behaviours.

Control tasks are typically solved by some form of reinforcement learning, To learn a control policy, the system performs a succession of trials, which initially fail frequently. As more experience is gained, the control policy is refined to improve its success rate. In its early formulations (Michie & Chambers, 1968; Watkins, 1989; Sutton & Barto, 1998), reinforcement learning worked well as long as the number of state variables and actions was small. For larger domains and more complex tasks, such as visual AUV navigation (El-Fakdi & Carreras, 2013), or teaching a dog-like robot to jump (Theodorou, Buchli, & Schaal, 2010), hand-crafted computer simulations

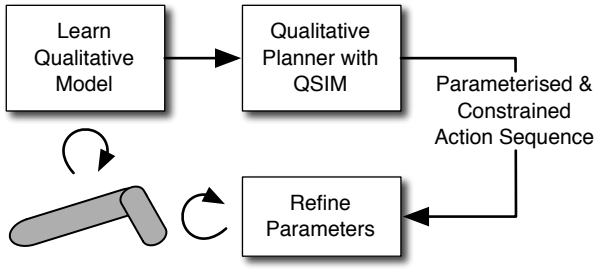


Figure 1: Multi-Strategy Architecture for learning robotic behaviours, using Qualitative Reasoning.

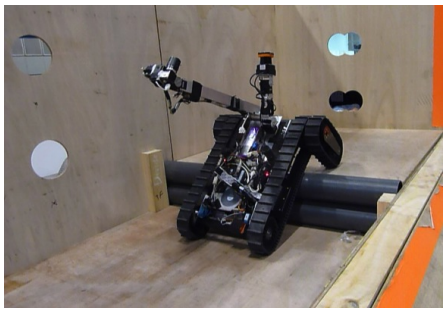


Figure 2: iRobot Negotiator platform for Urban Search and Rescue. The robot is shown climbing a step.

of a robot allow a large number of trials to be run, which are improved by a second stage of on-line learning. Alternatively, Behavioural Cloning learns control policies by first observing the actions of an expert human (Michie, Bain, & Hayes-Michie, 1990), and has been applied to tasks such as aircraft flight (Šuc, Bratko, & Sammut, 2004), helicopter flight (Ng et al., 2006), and a controlling a non-linear container crane (Šuc & Bratko, 1999).

Unlike humans, these reinforcement learning approaches are generally incapable of making use of background knowledge about the domain to reduce the number of trials required to converge to a solution. For example, if a human is learning to drive a car that has a manual gear shift, the instructor does not tell the student, "Here is the steering wheel, the gear stick, and the pedals. Play with them until you figure out how to drive". Rather, the instructor will be quite explicit, providing a *qualitative* sequence of actions to perform. To change gears, the instructor might tell the student to simultaneously depress the clutch while releasing the accelerator, following by the gear change, and so on. However, the instructor cannot impart the *quantitative* "feel" of the pedals and gear stick, or control the student's muscles so that the hands and feet apply the right pressure and move at just the right speed. This can only be learned by trial-and-error. So despite the qualitative knowledge, the student will still make mistakes until the parameters of their control policy are tuned. However, with no prior knowledge, learning would take much longer since the student has no guidance about what actions to try, and in what order to try them. The qualitative constraints also give the learner what might be described as "common sense" in that it can reason about the actions it is performing. In the example above, the background knowledge was provided by a teacher but some or all of it could also be obtained from prior learning.

Applying a similar *hybrid* approach to learning robotic behaviours will speed up the trial-and-error learning of the low-level control policies. We describe a Multi-Strategy architecture (Figure 1), that extends earlier work in on hybrid learning systems (Ryan, 2002; Sammut & Yik, 2010). which incorporated a symbolic planner to reason about the background knowledge. However, the hybrid learning architecture must address challenges in building robotic systems.

The three-layer hierarchy of the ICARUS architecture (Bonasso et al., 1997; Gat, 1998) is a typical representation of a robotic system. The lowest *reactive* layer reads from a robot's sensors, and produces immediate controller actions sent to the robot's actuators. The intermediate *sequenc-*

ing layer configures the parameters of the reactive commands, and the upper *deliberative* layer is responsible for long-term planning. Thus a hybrid learning architecture must easily allow for representations that are suitable for both the low-level, numeric reactive commands, and high-level, deliberative symbolic planning

Our three-stage hybrid system (Figure 1), detailed in Section 2, first builds background knowledge of the robotic system at the reactive layer using a qualitative representation. Secondly a deliberative layer symbolic planner that incorporates Qualitative Reasoning (de Kleer & Brown, 1984; Forbus, 1984), produces a parametrised sequence of actions, which provide only an approximate solution to the control task. Finally, the parameters are quickly optimized at the reactive layer by trial-and-error learning. We demonstrate our architecture on a multi-tracked robot intended for Urban Search and Rescue (Figure 2).

1.1 Related Work

Qualitative Reasoning was used as the basis of planning actions by Hogge (1987) and Forbus (1989). Alternatively, Drabble (1993) and DeJong (1994) designed reactive monitoring systems with planning to predict and react to the next state of a system. These authors used similar versions of STRIPS-like actions to modify a qualitative model of the system, which in turn changed the qualitative behaviour of the system. Such representations of an action work well for systems like water tank with in-flow and out-flow valves, where the action of opening or closing a valve have discontinuous sudden affects. On a robotic system, actions, such as moving an actuator, are more subtle. An action changes the value of a variable of the system, but does not change the qualitative model of the system.

More recently, Troha and Bratko (2011) successfully performed planning using QSIM (Kuipers, 1986) within the robotics domain. The system first learns the qualitative behaviour of a robotic system and then plans a sequence of actions. However, their system was specialised to learning the effects of pushing objects with small two-wheel robots.

Mugan and Kuipers (2012) developed QLAP also for using qualitative reasoning in solving robotic tasks. QLAP learns a hierarchy of small quantitative controllers at the reactive ICARUS layer, that allow a set of variables to reach specified qualitative values. However, we prefer to learn a model that allows for symbolic planning of actions, as this model is reusable for planning different control tasks, and the low-level controllers we learn are optimised for their specific tasks.

1.2 Application Domain: Terrain Traversal in Urban Search and Rescue

The task of traversing rough terrain typically found in the field of Urban Search and Rescue, such as steps, staircases and loose rubble, is a complex control task. The iRobot manufactured Negotiator (Figure 2), typical of those used in the field, contains a main set of tracks to drive the robot and sub-tracks, or flippers, that can re-configure the geometry of the robot to climb over obstacles. The planner must choose the best sequence of actions to overcome the terrain obstacles.

Solving autonomous terrain traversal through learning is an active field of research (Gonzalez et al., 2010; Tseng et al., 2013; Vincent & Sun, 2012). Following these approaches, we apply our planner to the step climbing task. This task is more complex than it may appear and requires

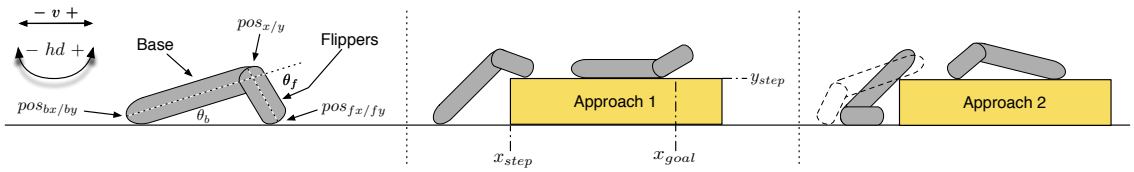


Figure 3: Representation of Negotiator and the step climbing task with the two broad approaches to climbing the step, driving forward (Approach 1) and reversing (Approach 2).

skills essential to traversing other terrain such as stairs and rubble. The Negotiator must configure itself to climb an obstacle that is higher than its main tracks using two broad approaches (Figure 3), sometimes requiring a lengthy sequence of actions. In Approach 1 the flippers are raised above the step before the robot drives forward. Approach 2 is significantly more complex. The robot reverses up to the step then, by supporting the its weight on the flippers, the base is raised and placed on the step. The flippers are reconfigured and the robot reverses onto the step. Taking Approach 1 is favourable to Approach 2 as the process of supporting the robot's weight on the flippers is very unstable. However, Approach 1 cannot be used if the step is too high. Thus, a learning system should prefer Approach 1 when possible, but also be able to determine when Approach 2 is the only viable way to climb the step.

2. Multi-Strategy Architecture for Learning Robotic Behaviours

Our three-stage hybrid architecture learning robotic behaviours (Figure 1), inspired by how humans learn to solve complex tasks, is based on the Multi-Strategy Architecture of Sammut and Yik (2010).

The first stage builds a qualitative representation for the background knowledge of the robotic system. A qualitative model of the dynamics of the robot and its interaction with the environment is learnt from numeric training data collected by on-line sampling. We extend Padé (Žabkar et al., 2011), which is an existing tool for learning qualitative models from numeric data, suitable for use with noisy domains such as robotics. Our extensions to Padé make the learnt qualitative model suitable for qualitative task-planning.

The second stage uses the background knowledge and our qualitative planner find a sequence of qualitative actions, forming the approximate solution that solves a given task. Our planner (Wiley, Sammut, & Bratko, 2013; Wiley, Sammut, & Bratko, 2014a) uses the Qualitative Simulation (QSIM) algorithm (Kuipers, 1986) to predict the outcome of performing a qualitative action. Each qualitative action is only an approximation of the precise actuator movements, and is constrained to a range of quantitative values. The quantitative regions are the parameters of the plan.

The third stage uses an on-line numeric trial-and-error learner to refine (or optimize) the precise quantitative values for each action of the plan. We formalise the trial-and-error approach of Sammut and Yik (2010), by phrasing the trial-and-error learner as a Semi-Markov Decision Problem (SMDP) using Options (Sutton, Precup, & Singh, 1999). At present, we focus on refining the parameters to a satisficing solution which allow the plan to be successfully executed. However, the SMDP representation allows for optimisation, such as finding the fastest plan.

In this paper, we present an overview of each stage of the Multi-Strategy architecture with sample results for solving the step-climbing task on the Negotiator (Figure 3). Section 3 gives a brief introduction to qualitative modelling (used throughout the stages), Section 4 details the learning of the qualitative model, Section 5 gives an overview of our qualitative planner, and Section 6 details the formalisation of the trial-and-error parameter refinement.

3. Qualitative Modelling

In a qualitative representation of a robot (and its interaction with the environment) the variables of the system are described relative to distinguished symbolic *landmarks* within the domain of each variable (Equation 1). The *qualitative value* of a variable, described in (2), is the variable's *magnitude* (either at a landmark or between two landmark) and a *direction of change* (the magnitude is increasing, steady or decreasing over time). The landmarks may optionally include positive and negative infinity. The symbol $\mathbb{Q}$ denotes *qualitative* representations, as distinguished from *quantitative* representations, denoted with $\mathbb{T}$. For example, the x-position of Negotiator (pos_x) may be qualitatively described as increasing between the robot's starting position and a feature of the environment, such as a step.

$$var^{\mathbb{Q}} := [l_0, l_1, \ldots, l_n] \tag{1}$$

$$var^{\mathbb{Q}} = l_i \ldots l_{i+1}/\texttt{dec} \qquad var^{\mathbb{Q}} = l_i/\texttt{std} \qquad var^{\mathbb{Q}} = l_i \ldots l_{i+1}/\texttt{inc} \tag{2}$$

Variables in the representation of the robot map to actuator settings (such as θ_f), sensor readings (such as θ_b), or calculated values (such as pos_x, pos_y, etc. calculated by position-tracking). Variables are divided into two categories: control and state variables. A *Control Variable*, ($cvar$) corresponds to an actuator, and when changed causes actions to be invoked on the robot, such as driving forward. A *State Variable*, ($svar$), is any other variable. On the Negotiator, the control variables are heading (hd), velocity (v), and the flipper angle (θ_f). The *Qualitative State Space*, $S^{\mathbb{Q}}$ (Equation 3), of the robot is defined by the qualitative domains of all state and control variables. Each *qualitative state*, $s_i^{\mathbb{Q}} \in S_Q$, assigns a qualitative value to each variable.

$$S^{\mathbb{Q}} := svar_1^{\mathbb{Q}} \times \ldots \times svar_m^{\mathbb{Q}} \times cvar_1^{\mathbb{Q}} \times \ldots \times cvar_n^{\mathbb{Q}} \tag{3}$$

A *Qualitative Model* combines the qualitative description of the robot, rules governing the legal operation of the robot, and features of the environment such as rubble, walls or steps. A qualitative model, $M^{\mathbb{Q}}$ (Equation 4), is specified by the qualitative state space of the robot, and a set of *Rules* (Equation 5) that define the legal qualitative states in $S^{\mathbb{Q}}$ which the robot can take. If a qualitative state matches the precondition of a rule, then the qualitative constraints of the rule are applied. Qualitative constraints are specified as *Qualitative Differential Equations (QDEs)*. Table 1 lists common QDE constraints. For example, on the Negotiator the relationship between the angle of the base (θ_b) and the angle of the flipper (θ_f) varies between $\texttt{M}^+$, $\texttt{M}^-$, or no relation.

$$M^{\mathbb{Q}} := \left\langle S^{\mathbb{Q}}, Rules \right\rangle \tag{4}$$

$$Rules := \{\{Preconditions\} \rightarrow Constraint\} \tag{5}$$

Table 1: Common types of QDE Constraints.

QDE	Description
$\texttt{M}^{+}(x, y)$	Monotonicity - the rate of change of x and y match.
$\texttt{M}^{-}(x, y)$	Inverse monotonicity - the rate of change of x and y are inverted.
$\texttt{sum}(x, y, z)$	$z = x + y$
$\texttt{mult}(x, y, z)$	$z = x \times y$
$\texttt{deriv}(x, y)$	y is the derivative of x with respect to time.
$\texttt{const}(x, k)$	$x = k/std$
qnull	The qualitative state is invalid.

4. Learning a Qualitative Model

The first stage of the Multi-Strategy Architecture learns a qualitative model of the robotic system from numeric training data. We extend the Padé tool (Žabkar et al., 2011) for inducing qualitative models with a variety of techniques necessary for learning qualitative models suitable for task planning in robotic domains. Padé learns a qualitative model of a target function with respect to the arguments of the function. In a robotic system, the qualitative dynamics of the *state* variables is a function of the *control* variables *over time* (Wiley, Sammut, & Bratko, 2013). Therefore, we learn a target function, $\mathbf{svars} = f(\mathbf{cvars})$, being the set of state variables, (**svars**), as a function of the set of control variables, (**cvars**). However, the labelling process of Padé can only estimate the qualitative Q-behaviour of a 'single'-valued target function. Therefore, we use a *pair-wise decomposition* to learn a set of target functions, as a set of partial derivatives representing the Q-behaviours between each pair of state and control variables (Equation 6).

$$F = \left\{ f_{i,j} = \frac{\delta svar_i}{\delta cvar_j} \; \forall \; svar_i \in \mathbf{svars}, cvar_j \in \mathbf{cvars} \right\} \tag{6}$$

Padé uses a three-step process for inducing a qualitative model from numeric training data. First, a numeric training data set of the robotic system is collected. Secondly, each learning example in the training data is labelled by the local qualitative behaviour (Q-behaviour), of the learning example. Finally, a general-purpose machine learning algorithm induces the qualitative model from the labelled data.

4.1 Sampling the Robotic System

The first step of Padé generates the training data set, D shown in (7), by random sampling of the execution of the robot as it moves about the environment. For each learning example in D, $\mathbf{cvars}^{\mathbb{T}}$ and $\mathbf{svars}^{\mathbb{T}}$ are the sets of observed quantitative values of the control and state variables respectively. A variable for passage of time, $time^{\mathbb{T}}$, is included in the sampled data set as this improves the accuracy of Padé's labelling. Only one training data set is collected, which is used to induce a qualitative model for each pair-wise decomposition, $f_{i,j} \in F$.

$$D\left(time^{\mathbb{T}}, \mathbf{cvars}^{\mathbb{T}}, \mathbf{svars}^{\mathbb{T}}\right) \tag{7}$$

Table 2: Sampling bias of the Negotiator data in Figure 4a, for the θ_f-θ_b relationship.

Region	Percentage of samples
$\theta_f \leq 12$	82.8%
$12 < \theta_f \leq 136$	4.8%
$136 < \theta_f \leq 180, 0.5 < \theta_b$	5.3%
$136 < \theta_f \leq 180, \theta_b \leq 0.5$	7.1%

4.2 Labelling

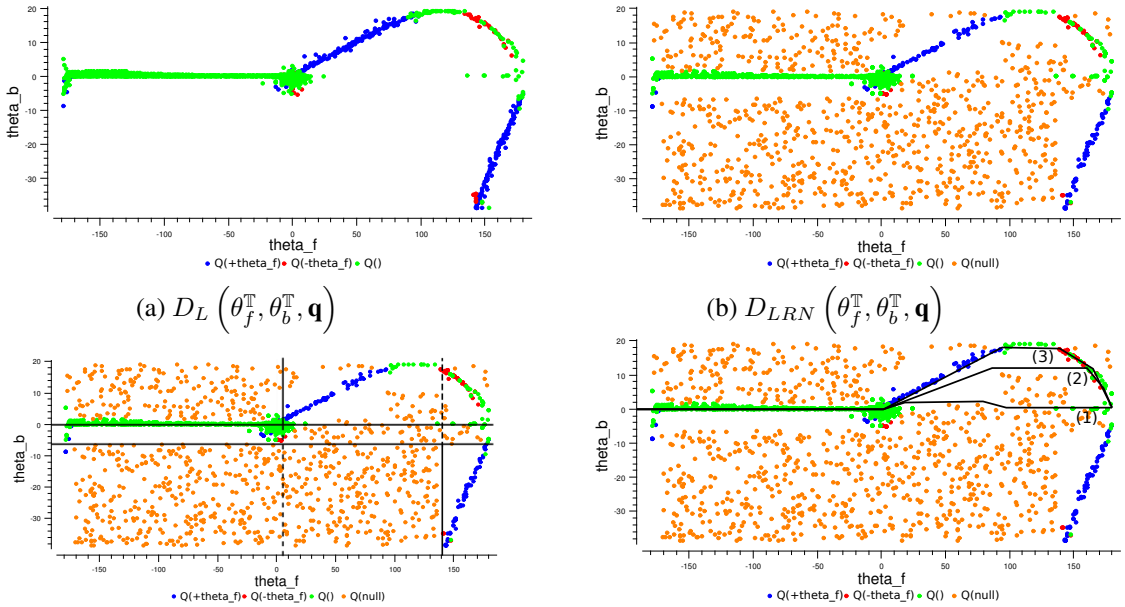
The second stage of Padé estimates the qualitative behaviour of each target function $f_{i,j} \in F$, that is the qualitative partial derivatives, using the sampled data set D. A labelled data set, D_L shown in (8), is produced for each $f_{i,j} \in F$ where, subscript L denotes a labelled data set, and $\mathbf{q}$ is the label for the local Q-behaviour of each learning example $(cvar_j, svar_i)$. A Q-behaviour is the local qualitative partial derivative. A label may be $svar_i = Q(+cvar_j)$ which states that the value of $svar_i$ is increasing with $cvar_j$, or $svar_i = Q(-cvar_j)$ which describes a decreasing relation with $cvar_j$, or $svar_i = Q()$ which states there is no relation between $svar_i$ and $cvar_j$. The local Q-behaviour of each learning example is estimated by ***Tube-Regression*** (Žabkar et al., 2011). Tube-regression estimates the Q-behaviour from a 'tube' of neighbouring learning examples centred around the example under consideration. Tube-regression also accounts for noise in D. We additionally enforce that when estimating a local Q-behaviour of one target function in F, tube-regression takes into account all state, control and time variables, as all of these variables may impact the Q-behaviour.

$$D_L\left(cvar_j^{\mathbb{T}}, svar_i^{\mathbb{T}}, \mathbf{q}\right) \text{ for } f_{i,j} \in F \tag{8}$$

The pair-wise labelled data produces models which are over-generalised. For example, consider learning a qualitative model for the behaviour of the Negotiator on flat ground. Figure 4a shows the pair-wise labelling $D_L(\theta_f^{\mathbb{T}}, \theta_b^{\mathbb{T}}, \mathbf{q})$, which describes the relationship between the angles of the base and flippers. The qualitative model induced from this labelling is shown in Figure 5a. We use a variety of techniques to improve the results of labelling.

Sampling from the robot causes the sampled data set D to contain severe sampling bias towards regions of the state space the robot frequently visited. Table 2 details the sampling bias of the data sampled in Figure 4a, broken down into four key regions of the θ_f-θ_b function. To avoid the sampling bias, D_L is re-sampled which first requires D_L to be discretised. The labelled data set is discretised, denoted as $Disc \circ D_L$, using Entropy Discretisation (Fayyad & Irani, 1993), producing a mapping between each learning example of to one bin of the discretisation. Some bins may be empty. Thus, the re-sampled data set D_{LR} (denoted by subscript R) is constructed by choosing r (Equation 9) random labelled learning examples with replacement from each non-empty bin of $Disc \circ D_L$. Re-sampling equalises the number of learning examples in each bin.

$$r = \frac{|D_L|}{\text{no. non-empty bins}(Disc \circ D_L)} \tag{9}$$



(a) $D_L\left(\theta_f^{\mathbb{T}}, \theta_b^{\mathbb{T}}, \mathbf{q}\right)$

(b) $D_{LRN}\left(\theta_f^{\mathbb{T}}, \theta_b^{\mathbb{T}}, \mathbf{q}\right)$

(c) Example partitioning from C4.5 of the state space from the induced model (solid lines), with QSIM's extended partitioning (dotted lines).

(d) QSIM simulations. Trajectory (1) is without any post-processing, (2) is after removing unsupported states, and (3) is after corresponding values are added.

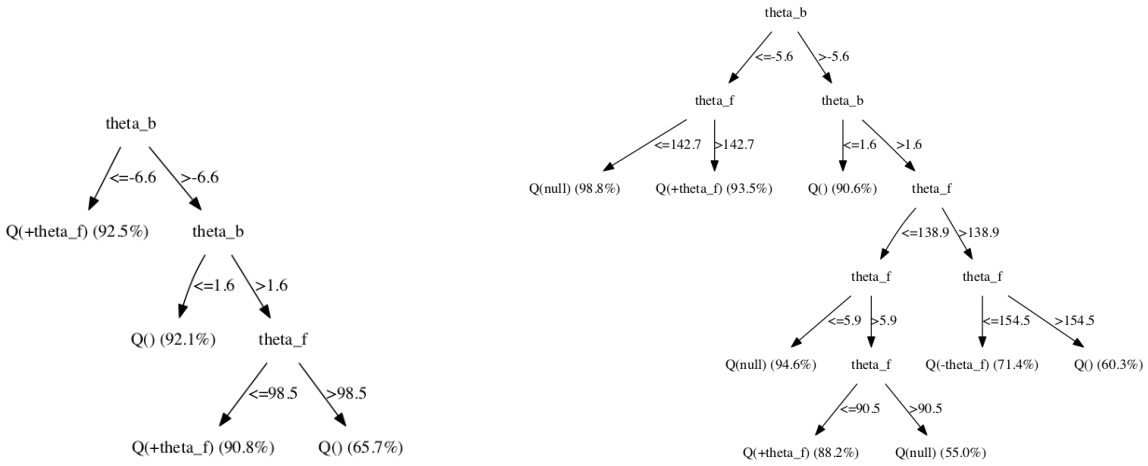
Figure 4: Labelled Data Sets for the θ_f-θ_b relationship on the Negotiator

The training data set D contains 'positive'-only learning examples. Figure 4a shows large regions of the state space without any learning examples. These empty regions, due to the physical limitations of the Negotiator or obstacles in the environment, are the key reason for the induced models being over-generalised. To correct for 'positive'-only learning examples, we seed the resampled labelled data set with additional learning examples labelled by `qnull`, to produce a null-seeded data set D_{LRN} (denoted by subscript N). Every empty bin of $Disc \circ D_L$, is seeded with n (Equation 10) random learning examples. The quantitative values for variables of the sample are randomly chosen within the bounds of the bin. To avoid introducing sampling bias towards the seeded learning examples, the value of n is dependent on the size of the bin, and a $ratio$ of the number of learning examples in D_L. Figure 4b shows $D_{LRN}(\theta_f^{\mathbb{T}}, \theta_b^{\mathbb{T}}, \mathbf{q})$, seeded with a ratio of 75%.

$$n = ratio \times |D_L| \times \frac{\text{area}(\text{empty-bin} \in Disc \circ D_L)}{\text{area}(D_L)} \tag{10}$$

4.3 Inducing the Qualitative Model

For each re-sampled, seeded, labelled data set D_{LRN}, a qualitative model is induced. As suggested by Žabkar et al. (2011), we use C4.5 (Quinlan, 1993) to induce a pair-wise qualitative model in the form of a decision tree, $M_P^{\mathbb{Q}}$, for each pair-wise target function. Tube-regression deals with some noise of the robotic system, however, learning examples may be incorrectly classified. Therefore,



(a) $M_P^{\mathbb{Q}} \circ D_{PL}\left(\theta_f, \theta_b\right)$ (b) $M_P^{\mathbb{Q}} \circ D_{PLRN}\left(\theta_f, \theta_b\right)$

Figure 5: Induced Qualitative Models on data sets from Figure 4. The majority class percentage for each leaf is provided.

C4.5 is configured to stop splitting nodes if a percentage of the majority class drops below 90%, or the number of instances in a node is less than 5% of the total learning examples.

Each tree-based induced qualitative model is converted into qualitative rules (Equation 5). Further, each Q-behaviour between a pair of state and control variables is converted to a monotonic QDE (Table 1). For example, the Q-behaviour $svar_i = Q(+cvar_j)$ equates to $\mathtt{M}^{+}\left(svar_i, cvar_j\right)$. To construct the complete qualitative model of the robot, $M^{\mathbb{Q}}$ (Equation 4), the set of qualitative states and the list of qualitative rules must be defined. The qualitative rules of $M^{\mathbb{Q}}$, described in (11), are the combination of all rules from all induced qualitative models for every pair-wise target function. The qualitative states, $S^{\mathbb{Q}}$ of $M^{\mathbb{Q}}$, are ultimately defined by the ordered landmarks of each qualitative variable. Thus, for $S^{\mathbb{Q}}$ we choose the landmarks for each qualitative variable as the collection of values used in each split of the induced qualitative trees.

$$\text{Rules of } M^{\mathbb{Q}} = \bigcup_{\forall f_{i,j} \in F} \left\{ M_P^{\mathbb{Q}} \circ D_{LRN}\left(cvar_j^{\mathbb{T}}, svar_i^{\mathbb{T}}, \mathbf{q}\right) \right\} \tag{11}$$

Using the splits of the induced decision trees to define the qualitative states of $S^{\mathbb{Q}}$ creates a discrepancy with how the decision trees from C4.5 divide the state space. Figure 4c highlights this issue. The decision tree of C4.5 recursively breaks down sub-regions of the state space. However, in $S^{\mathbb{Q}}$ the bounds of each sub-tree are extended. This partitioning of $S^{\mathbb{Q}}$ creates regions which contain zero 'supporting' examples, which causes QSIM, used in the planner, to produce incorrect trajectories. This problem is resolved by assuming that any state of $S^{\mathbb{Q}}$ without at least one supporting learning example is an invalid region. A `qnull` rule is added for each unsupported state. The extended model $M_U^{\mathbb{Q}}$, is denoted by subscript U. Figure 4d shows how detecting unsupported qualitative states alters the trajectories that QSIM produces.

Table 3: Comparison between induced and hand-crafted qualitative models on the number of rules and size of the qualitative state space of each model.

	Induced		Hand-crafted	
Model	Rules	State Space	Rules	State Space
θ_f vs θ_b	13	1215	4	891
Flat ground	25	3.5×10^7	14	5.9×10^6
Pole-cart	25	1.9×10^9	7	1.7×10^9

Figure 4d further shows the trajectory from QSIM still may not follow the sampled data, as it is not enforced that the end-points of monotonic constraints are to be reached. However, monotonic constraints may specify *corresponding values*, pairs of landmarks where the qualitative variables are known to 'correspond'. Therefore, we add corresponding values to each monotonic constraint to create an extended model $M^{\mathbb{Q}}_{UC}$, denoted by subscript C. The corners of the bounding box of the qualitative state where a monotonic constraint applies, define corresponding values. Figure 4d shows an example QSIM trajectory where corresponding values have been added.

4.4 Preliminary Results

We have applied Padé with our extensions to learning a qualitative model of the Negotiator as it operates on flat ground. This model was compared to a qualitative model of the same domain that was hand-crafted by an expert familiar with the Negotiator. We also made the same comparison on the common pole-and-cart balancing domain. Table 3 compares the models in terms of the number of rules and the size of the qualitative state space that each approach produces.

In general, the learnt qualitative models use a larger number of rules, however only on the Negotiator domain did we observe a larger qualitative state space. There are two key reasons for these differences. First, experts introduce auxiliary variables, simplifying the specification of the model. Auxiliary variables do not correspond to sensor measurements, but are calculated from other variables in the system. For example, the hand-crafted model of Negotiator contains the sum of the angles of the flipper and base, that is $\theta_{fb} = \theta_f + \theta_b$. Padé does not introduce new variables. Instead C4.5 must use a large number of partitions in the model, and hence more rules. For example, 'long' $\mathtt{M}^+/\mathtt{M}^-$ relations are sub-divided in order to accurately partition around null-regions. Secondly, dynamics between different pairs of state-control variables may change at similar (but not exactly the same) points. An expert can identify these similarities and compress the model, whereas the pair-wise decomposition that we employ cannot infer such compressions.

5. Qualitative Planning

The second stage of the Multi-Strategy architecture takes a qualitative model, $M^{\mathbb{Q}}$, and plans a sequence of actions to solve a desired task. $M^{\mathbb{Q}}$ may be hand-crafted or learnt using the techniques from Section 4. In previous work (Wiley, Sammut, & Bratko, 2013; Wiley, Sammut, & Bratko, 2014a) we have detailed various aspects of qualitative planning of robotic tasks, and applied the planner to the step-climbing task on the Negotiator. This previous work used a hand-crafted model

Pre-Condition:	$var_i^{\mathbb{Q}} = \texttt{mag/roc} \mid\mid \ldots$
Effect (external event):	$cvar_i^{\mathbb{Q}} = \texttt{mag/roc}$
Parameters:	$\{var_i, \ldots\}$

Figure 6: STRIPS-like Qualitative Planning Action

of the Negotiator. We summarise this work, and integrate it with the first and third stages of the Multi-Strategy architecture.

The qualitative planner uses the Qualitative Simulation (QSIM) (Kuipers, 1986) algorithm, which predicts changes in the qualitative state of the robot over time. The prediction of QSIM is a sequence of qualitative states, $s_i^{\mathbb{Q}} \in S^{\mathbb{Q}}$ shown in (12), where each state transition occurs under the influence of an *external event*, which may be empty. An external event, $e_i^{\mathbb{Q}}$ shown in (13), forces the value of the specified qualitative variables to the specified value, and QSIM predicts the change to all other unspecified qualitative variables. QSIM is non-deterministic such that QSIM may predict multiple qualitative states as the outcome of an external event.

$$s_1^{\mathbb{Q}} \xrightarrow{e_1^{\mathbb{Q}}} s_2^{\mathbb{Q}} \xrightarrow{e_2^{\mathbb{Q}}} \ldots \rightarrow s_n^{\mathbb{Q}} \tag{12}$$

$$e_i^{\mathbb{Q}} = \left\{ var_j^{\mathbb{Q}} = \text{magnitude/direction of change}, \ldots \right\} \tag{13}$$

A *Qualitative Action* (Figure 6) used for planning, is a symbolic representation of moving an actuator on the robot and is modelled on the *parametrised* STRIPS-like actions of Sammut and Yik (2010). For an action, the pre-conditions define the qualitative states where the action may be selected. The effects are a QSIM external event, which sets the value of one control variable. The parameters are a list of one or more qualitative variables that the action 'targets', that is, the action should be executed on the robot until the parameters reach desired values. The parameters may include the action's control variable. For example, on the Negotiator, qualitative actions for velocity (v) have the position of the robot (pos_x and pos_y) as parameters, while actions for moving the flipper (θ_f) are also parameterised by θ_f. Multiple qualitative actions may be executed in parallel. A *Parallel Qualitative Action*, $ap_i^{\mathbb{Q}}$, is a set of qualitative actions where the effects (external events) of the multiple qualitative actions do not conflict.

The effect of choosing a parallel qualitative action is calculated by QSIM. Figure 7 illustrates this relationship. For planning, a qualitative state is used to seed QSIM, and the chosen parallel qualitative action sets the external events of QSIM. QSIM produces a sequence of one or more qualitative states with the enforced external event, and the final qualitative state is the result of performing the parallel qualitative action.

A *Qualitative Plan*, $P^{\mathbb{Q}}$ shown in (14), is a sequence of qualitative state-action pairs from an initial state $s_1^{\mathbb{Q}}$ to a goal state $s_g^{\mathbb{Q}}$. A *stage* of the plan, $p_i^{\mathbb{Q}} \in P^{\mathbb{Q}}$ shown in (15), is one state-action transition. Importantly, in order to handle the non-determinism of QSIM, the qualitative plan does not record the precise sequence of states QSIM predicted for each parallel action. Additionally,

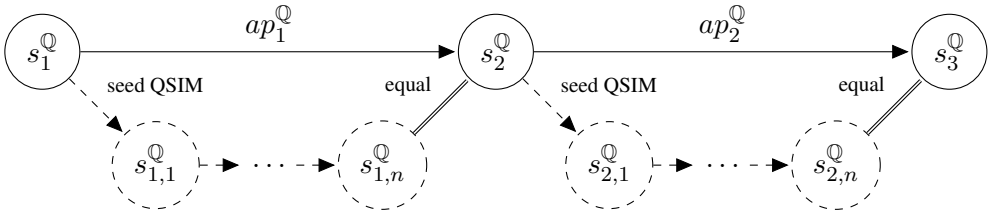


Figure 7: Qualitative Planning using QSIM to calculate successor states

the parameters of an action $ap_i^{\mathbb{Q}}$ are *constrained* by the qualitative values in $s_{i+1}^{\mathbb{Q}}$. Therefore, each stage $p_i^{\mathbb{Q}}$ states that: starting in $s_i^{\mathbb{Q}}$, the robot traversing any sequence of qualitative states (not necessarily those predicted by QSIM) while maintaining the control value settings of $ap_i^{\mathbb{Q}}$ until the parameters of $ap_i^{\mathbb{Q}}$ reach their target values as required by $s_{i+1}^{\mathbb{Q}}$.

$$P^{\mathbb{Q}} = s_1^{\mathbb{Q}} \xrightarrow{ap_1^{\mathbb{Q}}} s_2^{\mathbb{Q}} \xrightarrow{ap_2^{\mathbb{Q}}} s_2^{\mathbb{Q}} \rightarrow \ldots \xrightarrow{app_n} s_g^{\mathbb{Q}} \tag{14}$$

$$p_i^{\mathbb{Q}} = \left(s_i^{\mathbb{Q}}, ap_i^{\mathbb{Q}}, s_{i+1}^{\mathbb{Q}}\right) \in P^{\mathbb{Q}} \tag{15}$$

Any form of classical state-action planning may be used to find a plan. We use two implementations. The first is implemented in Prolog (Wiley, Sammut, & Bratko, 2013) using an A* search, and the second is implemented in ASP (Wiley, Sammut, & Bratko, 2014b) using the Clingo-4 solver (Gebser et al., 2011). The ASP implementation is more efficient and can handle larger qualitative models. However, the ASP implementation, unlike the Prolog implementation, is unable to reason about quantitative constraints of the physical robotic system (Wiley, Sammut, & Bratko, 2014a). In some circumstances without reasoning about quantitative constraints, the planner may not be able to rule out physically impossible qualitative states.

5.1 Example Qualitative Plan for the step climbing task

In previous works (Wiley et al. 2013, 2014a, 2014b) we have discussed various theoretical and performance aspects of our qualitative planner and applied the planner to solving terrain traversal tasks on the Negotiator. These works used a qualitative model of the Negotiator that was handcrafted by an expert. For the purposes of this paper, we consider finding a plan for the Negotiator to solve the step climbing task (Figure 3), where the height of the step can vary. Table 4 shows one suitable plan consisting of four stages, where the height of the step is low enough for the task to be solved following Approach 1. The combined effect (external QSIM event) and constrained parameters of each parallel action are also listed. The plan states the robot should drive towards the step while raising the flippers, then the step is traversed without moving the flippers, next the robot continues to drive forward as the flippers are lowered until, in the final action, the goal is reached and all actuator movements are halted. Plans following Approach 2 are found in previous work.

The plan in Table 4 highlights two important aspects of the qualitative planner. First, the same control variable (such as θ_f) may be used for both the effect and parameter of a STRIPS-like qualitative action (Figure 6). For θ_f, we interpret the actuator movement to the the direction-of-change

Table 4: Sequence of actions for solving the step climbing task following Approach 1.

Plan Stage	Effect (External Event)		Parameter Constraints	
	v	θ_f	pos_x	θ_f
1	$v_{max}/\texttt{std}$	$-\frac{\pi}{2}\ldots 0/\texttt{dec}$	$x_0 \ldots x_{step}$	$-\frac{\pi}{2}\ldots 0$
2	$v_{max}/\texttt{std}$	$-\frac{\pi}{2}\ldots 0/\texttt{std}$	$x_{step} \ldots x_{goal}$	$-\frac{\pi}{2}\ldots 0$
3	$v_{max}/\texttt{std}$	$-\frac{\pi}{2}\ldots 0/\texttt{inc}$	$x_{step} \ldots x_{goal}$	$-\frac{\pi}{2}\ldots 0$
4	$0/\texttt{std}$	$-\frac{\pi}{2}\ldots 0/\texttt{std}$	x_{goal}	0

of the variable, and the magnitude as the constrained region of the parameters. Secondly, in some situations the parameters may be constrained to exact values. This plan will be used in Section 6.2.

6. Trial-and-Error Parameter Refinement

The third stage of the Multi-Strategy architecture takes a qualitative plan, $P^{\mathbb{Q}}$, and refines (or optimizes) the constraints of the parameters of the plan using an on-line trial-and-error learner. The parameters of the plan are under-constrained, such that some combinations of their quantitative values will not allow the plan to be successfully executed. Further, only specific quantitative values for the parameters will give an optimal execution of the plan.

The parameter refinement of a qualitative plan, $P^{\mathbb{Q}}$, is phrased as a Semi-Markov Decision Problem (SMDP) using Options. (Sutton, Precup, & Singh, 1999). An Option is a temporally extended (or abstract) action which takes a variable length of time to execute. The SMDP is defined in Equation 16 where, X is the search space of the SMDP, O is the set of Options, T is the Transition Function, and R the Reward Function. The SMDP search space, X, is defined across the *quantitative* states of the robot, and stages of the given plan.

$$SMDP := \langle X, O, T, R \rangle \qquad \text{where } X := S^{\mathbb{T}} \times P^{\mathbb{Q}} \tag{16}$$

The Quantitative State Space, $S^{\mathbb{T}}$ (Equation 17), is the discretised quantitative mirror of the qualitative state space, $S^{\mathbb{Q}}$, used in planning. Each qualitative variable is discretised into equal-width bins bounded by the minimal and maximal landmarks of each variable[1].

$$S^{\mathbb{T}} := svar_1^{\mathbb{T}} \times \ldots \times svar_m^{\mathbb{T}} \times cvar_1^{\mathbb{T}} \times \ldots \times cvar_n^{\mathbb{T}} \tag{17}$$

The Transition Function, T (Equation 18), is a deterministic mapping, for each Option, between a quantitative state and a stage of the plan. The stage of the plan is included in T as it is vital in correctly defining each Option.

$$T : X \times O \rightarrow X \tag{18}$$

An Option, O, is an abstract action defined by the tuple in Equation 19 where, I is the set of initiation states where the Option can be chosen, π is an action policy while the Option is in force, and β is a termination function defining the probability of termination of the Option in each

1. Typically each variable is discretised according to the error margin of the corresponding sensor or actuator. Additionally, minimal or maximal landmarks of infinity are replaced by reasonable, domain specific values.

state. We define Options (Equations 20 to 22) such that the policy π is the parallel action, $ap_i^{\mathbb{Q}}$ as specified by the stage of the plan, p_i, of the Option's Initialisation State, I, and β tests if a specific set of quantitative target values for the parameters of $ap_i^{\mathbb{Q}}$ have be reached and terminates the Option accordingly. One Option is defined for each combination of quantitative target values.

$$O := \langle I, \pi, \beta \rangle \tag{19}$$

$$I = X \tag{20}$$

$$\pi = ap_i^{\mathbb{Q}} \tag{21}$$

$$\beta(x_i \in X) = \begin{cases} 1 & : var_i^{\mathbb{T}} = \text{target}\left(var_i^{\mathbb{T}}\right) \quad \forall\, var_i \in \text{params}\left(ap_i^{\mathbb{Q}}\right) \\ 0 & : \text{otherwise} \end{cases} \tag{22}$$

We are only interested in a *satisficing* solution. That is, a refinement of the parameter values to a range which guarantees that the plan will be successfully executed. We define the reward function, $R(x_i \in X, o_j \in O)$ in three parts. $R = 1$ for a successful execution of the *entire* plan, $R = -1$ if from state x_i the Option o_j fails to terminate, and $R = 0$ otherwise. Given the plan is known to terminate, we define the Value Function, V (Equation 23), as the un-discounted sum of future rewards (Mitchell, 1997). We have a satisficing solution as soon as, $V(x_1) = 1$, that is, the Value Function of the state of the SMDP corresponding to the first stage of the plan indicates a successful plan execution. For an optimal solution, we need a measure of the cost of executing each Option from its initiation state and set the reward for each stage such that $R(x_i \in X, o_j \in O)$ is the negation of the *cost* of Option o_j when started in the initiation state x_i.

$$V\left(x_i \in X\right) = \max_{\forall o_j \in O} \left[R\left(x_i, o_j\right) + V\left(\text{successor of } x_i \text{ given } o_j\right)\right] \tag{23}$$

6.1 Deterministic Transition Function Assumption

We have chosen to use a deterministic transition function in our formulation of the SMDP, as typically, this requires a fewer number of trials to converge to a solution. For on-line learning, reducing the number of trials is critical. However, moving an actuators on a physical robot is probabilistic. However, we execute parallel qualitative actions on the robot using a PID controller for each actuator, as the desired target value of each parameter is approached. We therefore assume that, the PID controller will, deterministically, reach the targeted parameter values. This assumption permits the use of a deterministic transition function.

6.2 Sample SMDP results for the step climbing task

We demonstrate the SMDP on the Negotiator and the step-climbing task following the qualitative plan from Table 4, that solves the task using Approach 1. The Options are summarised in Table 5.

We conducted sample experiments on the Negotiator to find a satisficing solution for this plan. For the purpose of the experiments: we represent distances in centimetres and angles in degrees; we used the quantitative values for important landmarks as: $x_0 = 0$, $x_{step} = 100$, and $x_{goal} = 200$. In each experiment, trials were run until a satisficing solution was found. For each trial, a sequence of Options were chosen, leading from the starting state to the goal state. The sequence of Options

Table 5: Options for stages of the qualitative plan in Table 4.

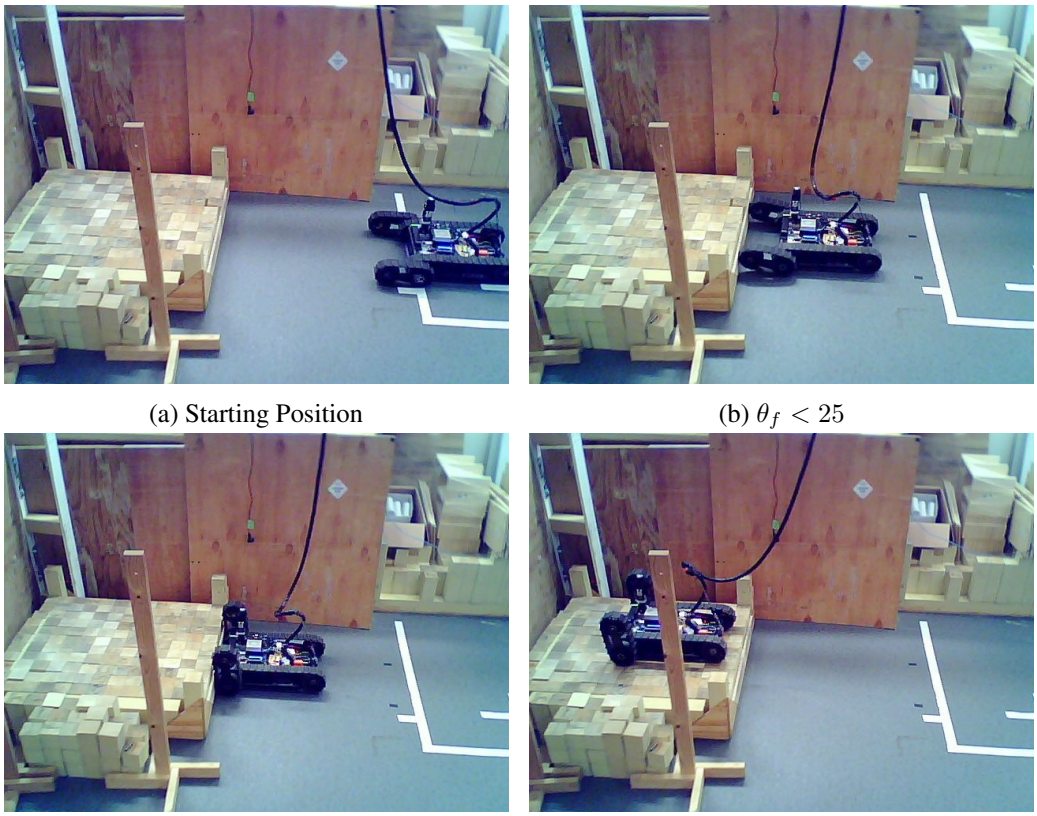
	I	π	β target values	
Stage	$s_i^{\mathbb{T}} \in S^{\mathbb{T}}$		pos_x	θ_f
1	$pos_x = 0, \theta_f = 0$	$v_{max}/\texttt{std}$, decrease θ_f	5	5
1	$pos_x = 0, \theta_f = 0$	$v_{max}/\texttt{std}$, decrease θ_f	5	15
1	$pos_x = 0, \theta_f = 0$	$v_{max}/\texttt{std}$, decrease θ_f	5	25
...	...	...	...	
1	$pos_x = 0, \theta_f = 0$	$v_{max}/\texttt{std}$, decrease θ_f	15	5
...	...	...	...	
1	$pos_x = 0, \theta_f = 0$	$v_{max}/\texttt{std}$, decrease θ_f	95	-85
2	$pos_x = 5, \theta_f = 5$	$v_{max}/\texttt{std}$, steady θ_f	105	5
2	$pos_x = 5, \theta_f = 5$	$v_{max}/\texttt{std}$, steady θ_f	115	5
...	...	...	...	
2	$pos_x = 5, \theta_f = 15$	$v_{max}/\texttt{std}$, steady θ_f	105	-15
...	...	...	...	
...	...	...	...	
3	$pos_x = 105, \theta_f = 15$	$v_{max}/\texttt{std}$, increase θ_f	185	5
...	...	...	...	
4	$pos_x = 200, \theta_f = 0$	$0/\texttt{std}$, steady θ_f	200	0

constituted a proposed satisficing set of quantitative parameters. The Options where chosen to maximise the number of Options whose reward function had not been observed in previous trials of the experiment. The robot was placed at the initial state. The robot attempts to execute the proposed sequence of Options until either one Option failed to terminate or the goal state was reached. If the goal state was reached, a satisficing solution was found, and the experiment concluded.

We ran 20 experiments to find a satisficing solution. Across the experiments, a minimum of 1 trial, a maximum of 9 trials, and an average of 3 trials were required before a satisficing solution was found. Importantly, the experiments demonstrated that invalid sets Options are ruled out. Figure 8 shows the three key cases for the step climbing task. Each trial started at the same location shown in Figure 8a. In Figure 8b, the Option for stage 1 of the plan used a flipper angle is too low and prevents the robot ascending the step. In Figure 8c, the flipper angle is too high and also prevents the robot ascending the step. Further if an Option used a target value of pos_x that is too close to the step, the robot dangerously “launches" itself forward coming close to tipping over backwards when executing the next Option. In Figure 8d, the values of θ_f and pos_x for all Options formed a satisficing solution.

7. Future Work

The experiments that we have demonstrated do not constitute an end-to-end application of the Multi-Strategy Architecture. We have only learnt a qualitative model of the Negotiator on flat ground and thus only applied the qualitative planner to basic tasks. To solve the task of climbing a step, we have used a hand-crafted model, which was subsequently used by the qualitative planning and parameter refinement stages. Therefore, our key research challenge is apply the end-to-end Multi-Strategy



(a) Starting Position (b) $\theta_f < 25$

(c) $\theta_f > 75$ (d) Satisficing θ_f and pos_x

Figure 8: Trial stages showing aspects of finding a satisficing solution to the step climbing task.

architecture to not only solving the step-climbing task, but more complex tasks in Urban Search and Rescue, such as traversing staircases or rubble.

8. Conclusion

We have proposed a three-stage Multi-Strategy Architecture for learning robotic behaviours for control problems. For each stage of the architecture, we have presented a formalisation of the stage which were demonstrated on a multi-tracked robot intended for urban search and rescue.

Acknowledgements

We thank Jure Žabkar from the University of Ljubljana for his help in using the Padé software. We thank Dr. Torsten Schaub and Max Ostrowski from the University of Potsdam for their assistance with the Clingo-4 ASP solver.

References

Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, P. D., & Slack, M. G. (1997). Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence*, *9*, 237–256.

de Kleer, J., & Brown, J. S. (1984). A qualitative physics based on confluences. *Artificial Intelligence*, *24*, 7–83.

DeJong, G. F. (1994). Learning to Plan in Continuous Domains. *Artificial Intelligence*, *65*, 71–141.

Drabble, B. (1993). EXCALIBUR: A Program for Planning and Reasoning with Process. *Artificial Intelligence*, *62*, 1–50.

El-Fakdi, A., & Carreras, M. (2013). Two-step gradient-based reinforcement learning for underwater robotics behavior learning. *Robotics and Autonomous Systems*, *61*, 271–282.

Fayyad, U. M., & Irani, K. B. (1993). Multi-interval discretization of continuous valued attributes for classification learning. *Artificial Intelligence, 13th International Joint Conference on (IJCAI)* (pp. 1022–1029). Chambery, France.

Forbus, K. D. (1984). Qualitative Process Theory. *Artificial Intelligence*, *24*, 85–168.

Forbus, K. D. (1989). Introducing Actions into Qualitative Simulation. *Artificial Intelligence (IJCAI), 11th International Joint Conference on* (pp. 1273–1278).

Gat, E. (1998). On Three-Layer Architectures. *Artificial Intelligence and Mobile Robotics*, 195–210.

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., & Schneider, M. (2011). Potassco: The Potsdam Answer Set Solving Collection. *AI Communications*, *24*, 107–124.

Gonzalez, R., Fiacchini, M., Alamo, T., Guzman, J. L., & Rodriguez, F. (2010). Adaptive Control for a Mobile Robot Under Slip Conditions Using an LMI-Based Approach. *European Journal of Control*, *16*, 144–155.

Hogge, J. C. (1987). Compiling Plan Operators from Domains Expressed in Qualitative recess Theory. *Artificial Intelligence (AAAI), 6th National Conference on* (pp. 229–233).

Kuipers, B. J. (1986). Qualitative Simulation. *Artificial Intelligence*, *29*, 289–338.

Michie, D., Bain, M., & Hayes-Michie, J. (1990). Cognitive models from subcognitive skills. *Knowledge-base Systems in Industrial Control*.

Michie, D., & Chambers, R. A. (1968). BOXES: An Experiment in Adaptive Control. *Machine intelligence*, *2*, 137–152.

Mitchell, T. M. (1997). *Machine Learning*. Carnegie Mellon University: McGraw-Hill.

Mugan, J., & Kuipers, B. J. (2012). Autonomous Learning of High-Level States and Actions in Continuous Environments . *Autonomous Mental Development, IEEE Transactions on*, *4*, 70–86.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., & Liang, E. (2006). Autonomous inverted helicopter flight via reinforcement learning. *Experimental Robotics IX*, 363–372.

Quinlan, J. R. (1993). *C4.5: programs for machine learning*. San Manteo, CA: Morgan Kaufmann.

Ryan, M. R. K. (2002). Using Abstract Models of behaviours to automatically generate reinforcement learning hierarchies. *Machine Learning (ICML), 19th International Conference on* (pp. 522–529). Morgan Kaufmann.

Sammut, C., & Yik, T. F. (2010). Multistrategy Learning for Robot Behaviours. In J. Koronacki, Z. Raś, S. Wierzchon, & J. Kacprzyk (Eds.), *Advances in machine learning i*, 457–476. Springer Berlin / Heidelberg.

Šuc, D., & Bratko, I. (1999). Symbolic and qualitative reconstruction of control skill. *Electronic Transactions on Artificial Intelligence*, *3*, 1–22.

Šuc, D., Bratko, I., & Sammut, C. (2004). Learning to fly simple and robust. *Machine Learning (ECML), European Conference on* (pp. 407–418). Pisa.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press. 1st edition.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Theodorou, E. A., Buchli, J., & Schaal, S. (2010). A Generalized Path Integral Control Approach to Reinforcement Learning. *Journal of Machine Learning Research*, *11*, 3137–3181.

Troha, M., & Bratko, I. (2011). Qualitative learning of object pushing by a robot. *Qualitative Reasoning (QR), 25th International Workshop on* (pp. 175–180).

Tseng, C.-K., Li, I.-H., Chien, Y.-H., Chen, M.-C., & Wang, W.-Y. (2013). Autonomous Stair Detection and Climbing Systems for a Tracked Robot. *System Science and Engineering (ICSSE), IEEE International Conference on* (pp. 201–204).

Vincent, I., & Sun, Q. (2012). A combined reactive and reinforcement learning controller for an autonomous tracked vehicle. *Robotics and Autonomous Systems*, *60*, 599–608.

Watkins, C. J. C. H. (1989). *Learning with Delayed Rewards*. Doctoral dissertation, University of Cambridge.

Wiley, T., Sammut, C., & Bratko, I. (2013). Planning with Qualitative Models for Robotic Domains. *Advances in Cognitive Systems (Poster Collection), Second Annual Conference on* (pp. 251–266). Baltimore, USA.

Wiley, T., Sammut, C., & Bratko, I. (2014a). Qualitative Planning with Quantitative Constraints for Online Learning of Robotic Behaviours. *Artificial Intelligence (AAAI), 28th AAAI Conference on* (pp. 2578–2584). Quebec City, Canada.

Wiley, T., Sammut, C., & Bratko, I. (2014b). Qualitative Simulation with Answer Set Programming. *Artificial Intelligence, 21st European Conference on* (pp. 915–920). Prague, Czech Republic.

Žabkar, J., Mozina, M., Bratko, I., & Demsar, J. (2011). Learning qualitative models from numerical data. *Artificial Intelligence*, *175*, 1604–1619.