Linköping Electronic Articles in
Computer and Information Science
Vol. 5(2000): nr 40

Qualitative trees applied to bicycle riding

Dorian Šuc and Ivan Bratko

Linköping University Electronic Press
Linköping, Sweden

http://www.ep.liu.se/ea/cis/2000/040/

Published on December 21, 2000 by
Linköping University Electronic Press
581 83 Linköping, Sweden

Linköping Electronic Articles in
Computer and Information Science
ISSN 1401-9841
Series editor: Erik Sandewall

Typeset by the author using LaTeX
Formatted using étendu *style*

Recommended citation:
<Author>. <Title>. Linköping Electronic Articles in Computer and Information Science, Vol. 5(2000): nr 40. http://www.ep.liu.se/ea/cis/2000/040/. *December 21, 2000.*

This URL will also contain a link to the author's home page.

Abstract

We present a method for learning qualitative constraint trees implemented in the program QUIN (qualitative induction). At difference with usual decision trees, the leaves of *qualitative* trees contain qualitative functional constraints as used in qualitative reasoning. We demonstrate the use of qualitative trees by their application to the difficult task of cloning the human skill of bicycle riding. The approach results in qualitative controllers that are straightforward to explain, and also enable optimization of corresponding quantitative controllers which carry out the task in the style similar to the human, but more effectively.

Authors' address

Faculty of Computer and Information Sciences
University of Ljubljana
Ljubljana, Slovenia
E-mail: {dorian.suc, ivan.bratko}@fri.uni-lj.si

1 Introduction

It has been shown in various studies [5, 2, 16], that for some tasks qualitative models are more suitable then classical quantitative (numerical) models. Qualitative reasoning about processes [6] and qualitative simulation [7] with Qualitative Differential Equations enable a kind of commonsense reasoning by abstracting numerical values into qualitative values and real functions into qualitative constraints.

In this paper we consider the problem of learning qualitative models of dynamics behaviours from measured data. In our case the motivation came for applications in behavioural cloning, where the task is to induce a controller for a dynamic system from performance traces of a skilled operator as for example in [11, 4].

In our previous experiments in behaviour cloning [13, 15] qualitative strategies based on qualitative proportionality constraints [6] proved to be very successful. These qualitative strategies were obtained as an abstraction of quantitative control strategies induced from the human operator's execution traces. In this paper we investigate a natural extension of that approach, the learning of qualitative strategies *directly* from numerical execution traces. We develop the program QUIN (QUalitative INduction) that induces trees with qualitative constraints (called *qualitative trees*) from numerical data, and apply it to a difficult task of cloning the skill of ridding bicycle.

In Section 2 we present a method for the induction of qualitative trees. In the following two sections we apply this method to qualitative cloning of bicycle riding skill. In Section 5 qualitative clones are transformed into quantitative controllers. The resulting controllers are experimentally shown to be successful and robust.

2 Qualitative induction

We consider the usual setting of classification learning problem, but in our case the hypothesis language is defined in terms of *qualitative constraints.*

Let there be N learning examples. Each example is described with $n+1$ continuous variables $X_1, \ldots, X_{n+1}$ with values $x_{i,1}, \ldots, x_{i,n+1}$, $i = 1, \ldots, N$. The variable X_{n+1} is called *class*, and the others are called *attributes.* Discrete ordinal attributes can also be used, but for simplicity we here assume that all the attributes are continuous. In the context of modeling dynamic systems, an example can be a state in the system's state space. Given the learning examples, our problem is to learn a hypothesis that separates the areas of state space which share a common qualitative behaviour of the class variable. We learn such hypotheses in the form of *qualitative trees.* A qualitative tree is a binary tree with internal nodes called splits and *qualitatively constrained functions* in the leaves. The splits define a partition of the state space into areas with common qualitative behaviour of the class variable. A split consists of a split attribute and a split value. Qualitatively constrained functions (abbreviated QCFs) in leaves define qualitative constraints on the class variable.

In the rest of this section we first define qualitatively constrained functions (QCF), describe how a QCF is induced from a set of examples and then give the qualitative tree learning algorithm QUIN. A more detailed description of this qualitative learning system is given in [12].

2.1 Qualitatively constrained functions

We use qualitatively constrained functions to define qualitative constraints on the class variable. A QCF constrains the qualitative change of the class variable in response to the qualitative changes of the attributes. A qualitatively constrained function $M^{s_1,\dots,s_m} : \Re^m \mapsto \Re$, $s_i \in \{+,-\}$ represents an arbitrary function with $m \leq n$ continuous attributes that respect the qualitative constraints given by signs s_i. The qualitative constraint given by sign $s_i = +$ ($s_i = -$) requires that the function is strictly increasing (decreasing) in its dependence on the i-th attribute. We say that the function is *positively related* (negatively related) to the i-th attribute. $M^{s_1,\dots,s_m}$ represents any function which is, for all $i = 1,\dots,m$ positively (negatively) related to the i-th argument, if $s_i = +$ ($s_i = -$). In the case of continuously derivable functions, $s_i = +$ ($s_i = -$) requires that the i-th partial derivative is strictly positive (negative):

$$M^{s_1,\dots,s_m} : \Re^m \mapsto \Re \ , \ s_i \in \{+,-\}$$
$$\frac{\partial M^{\dots,s_i,\dots}(\dots,x_i,\dots)}{\partial x_i} \begin{cases} > 0, \text{if } s_i = + \\ < 0, \text{if } s_i = - \end{cases}$$

This notation is similar to the qualitative proportionality predicates Q_+ and Q_- as defined by Forbus [6]. It is also a generalization of the well known qualitative constraint $M^+(x)$, as used in [7].

Note that the qualitative constraint given by sign $s_i = +$ only states that when the i-th attribute increases, the QCF will also increase, *barring other changes.* It can happen that the QCF with the constraint $s_i = +$ decreases even if the i-th attribute increases, because of a change in another attribute. For example, consider the behaviour of gas pressure in a container: $Pres \times Vol/Temp = const$. We can express the qualitative behaviour of gas by QCF $Pres = M^{+,-}(Temp, Vol)$. This constraint allows that the pressure decreases even if the temperature increases, because of a change in the volume. Notice however, that the qualitative behaviour of gas is not consistent with the constraint $Pres = M^+(Temp)$.

QCFs are concerned with qualitative changes and qualitative change vectors. *Qualitative change* q_j is the sign of change in continuous variable X_j where $q_j \in \{pos, neg, zero\}$, corresponding to *positive*, *negative* or *zero* change. A *qualitative change vector* is a vector of qualitative changes of the variables. For example, when heating the gas in the container, the observed qualitative change vector could be (q_{Temp}=pos,q_{Vol}=$zero$,q_{Pres}=pos) describing the increasing temperature, steady volume and increasing pressure.

We define *QCF-prediction* $P(s_i, q_i)$ as:

$$P(s_i,q_i) = \begin{cases} \text{pos, if } (s_i{=}+ \wedge q_i{=}pos) \vee (s_i{=}- \wedge q_i{=}neg) \\ \text{neg, if } (s_i{=}+ \wedge q_i{=}neg) \vee (s_i{=}- \wedge q_i{=}pos) \\ \text{zero, otherwise} \end{cases}$$

Qualitative change vector $(q_1,\dots,q_{n+1})$ is *consistent* with a given QCF $M^{s_1,\dots,s_m}$ ($m \leq n$), if the QCF possibly explains the qualitative change of the class variable, that is, if there exists an attribute whose QCF-prediction is equal to the class's nonzero qualitative change: $\exists i \in \{1,\dots,m\} : P(s_i,q_i) = q_{n+1}$. This definition reflects an interesting property of the QCF, that is *qualitative ambiguity.* QCF does not always uniquely predict the class's qualitative change given the qualitative changes of the attributes. Qualitative ambiguity, i.e. ambiguity in the class's qualitative change appears

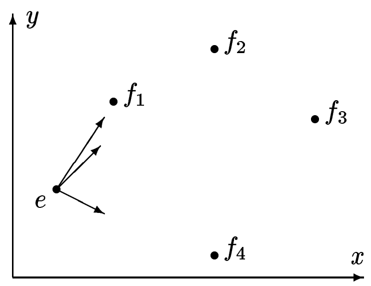

Figure 1: Computing estimates at the point e with $k = 3$ (number of nearest q-neighbors used): the arrows illustrate qualitative change vectors and the lengths of the arrows indicate their weights.

whenever there exist both positive and negative QCF-predictions. In this case any qualitative class change is consistent with the QCF.

2.2 Learning qualitatively constrained functions

Here we describe how QCF is learned from a set of examples. First every pair of examples e and $f \neq e$ is used to form a qualitative change vector with qualitative changes $q_{(e,f),j} \in \{pos, neg, zero\}$, $j = 1, \dots, n+1$ defined as:

$$q_{(e,f),j} = \begin{cases} \text{pos, if } x_{f,j} > x_{e,j} + Tzero_j \\ \text{neg, if } x_{f,j} < x_{e,j} - Tzero_j \\ \text{zero, otherwise} \end{cases} \tag{1}$$

where $Tzero_j$ denotes a user-defined steady threshold defining negligible changes of j-th attribute. Typically, many pairs of examples map into the same qualitative change vector. A qualitative change vector is either consistent or not consistent with a given QCF. Note that a consistent qualitative change vector can also be ambiguous for a given QCF as discussed in Section 2.1.

To find the "most consistent" QCF we define the *error-cost* of a QCF (defined later in eq. 4) that guides the learning. The error-cost of a QCF penalizes inconsistent qualitative change vectors at every example according to their confidence estimates (defined later in eq. 3). We experimented with different algorithms for the induction of QCFs [12], but here we use just a simple exhaustive search algorithm that forms all possible QCFs and selects the one with the smallest error-cost.

In the next paragraphs we first define the confidence estimates of qualitative change vectors, and then define the error-cost of a QCF.

2.2.1 Estimating qualitative change vectors

Our learning algorithm uses heuristic confidence estimates of qualitative change vectors at example e. We refer to confidence estimates simply as estimates. These estimates take into account the locality and consistency of a qualitative change vector.

We say that example f is a q-neighbor of example e if the change vector $q_{(e,f)}$ is equal to q in all the qualitative attribute changes (but not necessarily in the qualitative class change). Let $\mathcal{N}_e(q)$ denote the set of all q-neighbors of e. Then $\mathcal{N}_e(q) = \{f | \forall j = 1, \dots, n : q_{(e,f),j} = q_j\}$.

The confidence estimate $w_e(q)$ of the qualitative change vector q at example e takes into account the proximity of the q-neighbors and the consistency of qualitative class change w.r.t. the q-neighbors. To estimate the qualitative change vector q at e, k nearest q-neighbors are taken into account, where k is a user-defined parameter with the default value 5. The proximity of the examples e and f is evaluated by Gaussian weight $w(e,f)$ using Gaussian kernel K and distance considering only the attributes of the two examples:

$$
\begin{aligned}
w(e,f) &= e^{\frac{-d(e,f)}{2K^2}},\ d(e,f) = \sum_{j=1}^{n} d_j^2(e,f) \\
d_j(e,f) &= \begin{cases} |x_{e,j} - x_{f,j}|, \text{ if } q_{(e,f),j} = \text{zero} \\ |x_{e,j} - x_{f,j}| - Tzero_j, \text{ otherwise} \end{cases}
\end{aligned} \tag{2}
$$

The formula to compute the estimates $w_e(q)$ is:

$$
\begin{aligned}
\mathcal{C}_e(q) &= \{f | \forall j = 1, \ldots, n+1 : q_{(e,f),j} = q_j\} \\
w_e(q) &= \frac{1}{k} \sum_{f\ \in \mathcal{C}_e^k(q)} w(e,f)
\end{aligned} \tag{3}
$$

where $\mathcal{C}_e^k(q)$ denotes the set of k nearest q-neighbors of e with the qualitative class change equal as in q. Note that k is also the maximal possible sum of the weights $w(e,f)$ over all nearest q-neighbors of e. Using k in denominator penalizes estimate $w_e(q_i)$ if nearest q-neighbors of e differ in qualitative class change.

We illustrate this procedure by the example in Fig. 1. In Fig. 1 y is the class variable and x is the only attribute. Each pair of neighbor examples $(e,\ f_i)$ defines a qualitative change vector at the point e. The pairs (e,f_i), i=1,2,3 have the same qualitative change vector p=(q_x,q_y)=(pos,pos), pair (e,f_4) gives r=(q_x,q_y)=(pos,neg). Suppose the user set $k = 3$, so the nearest p-neighbors of e are f_1, f_2 and f_4. The nearest r-neighbors of e are the same, since p and r differ only in qualitative class change. Now the weights $w(e,f_i)$, i=1,2,4 are computed. The estimates of the qualitative change vectors p and r at e are: $w_e(p) = (w(e,f_1) + w(e,f_2))/3$ and $w_e(r) = w(e,f_4)/3$.

2.2.2 Error-cost of a QCF

The error-cost of a given QCF is based on the minimum description length principle [10, 8]. So the idea of our error-cost is to estimate the number of bits needed to code the QCF plus the number of bits to code the inconsistent qualitative change vectors, where the change vectors q are weighted according to their confidence estimates $w_e(q)$ (eq. 3). The error-cost $E(g)$ of a QCF g that mentions m out of all n attributes is:

$$
\begin{aligned}
E(g) = &\log_2 n + m(\log_2 n + 1) + \\
&\log_2 w_{nonamb} + w_{wrong}(\log_2 w_{nonamb})
\end{aligned} \tag{4}
$$

Here w_{wrong} denotes the sum (over all examples e) of estimates $w_e(q)$ of qualitative change vectors q that are not consistent with g. Similarly w_{nonamb} denotes the sum of estimates $w_e(q)$ of vectors q that are not ambiguous for g.

2.3 Learning a qualitative tree

The algorithm in QUIN that induces a qualitative tree from a set of examples, uses top-down greedy approach, similar as in the ID3 algorithm. Given the examples, it chooses the best split by comparing the partitions of the examples they generate: for every possible split, it splits the qualitative states into two subsets (according to the split), finds the corresponding QCF in both subsets, and selects the split which minimizes the *tree error-cost* (defined below). It puts the best split in the root of the tree and recursively finds both subtrees for the corresponding example subsets, until the stopping condition is satisfied, i.e. until using the best split does not improve the tree error-cost.

The *tree error-cost* is computed as follows. The tree error-cost of a leaf is the error-cost $E(g)$ (eq. 4) of the QCF g that is induced from the examples in the leaf. The tree error-cost of an internal node is the sum of error-costs of both subsets plus the cost of the split, i.e. the number of bits needed to encode the split.

3 The bike domain

In this section we describe the challenging control task that requires maintaining the balance on the bike and driving it to the goal.

We consider the bike's speed is constant. Then the state of the bike is described by six variables: the tilt ω angle of the bicycle from vertical and its velocity $\dot{\omega}$, the angle θ between the front wheel direction and the bicycle direction due to the deflection of the handlebars and its velocity $\dot{\theta}$, the distance D_g from the goal and the angle Ψ_g of the bike's frame relative to the goal. The system is controlled by two actions: the torque τ applied to the handlebars and the displacement d_{CM} of the center of mass from the vertical. The control task is to maintain the balance on the bike and drive from the start position to the goal position with the constant speed of 10 km/h. We used the simulator with the parameters as in [9].

Maintaining the balance on the bike is a very hard task and requires a proper adjustment of the front wheel direction by applying a torque to the handlebars and/or a proper displacement of the center of mass. The requirement to approach the goal makes the task even harder, since the balancing and the goal-aiming are to be performed simultaneously.

In previous experiments [9] with reinforcement learning in this domain a technique called shaping was used to speed up the learning. Here, the agent was first given the task to learn to balance the bike and after that the task to reach the goal while using the skill of balancing. Although it succeeds to learn a successful policy, the approach requires manually formulating the subproblems to face the learner and extensive experimentation with the simulator. Another approach used in this domain called Iterated Phantom Induction [3] required less experimentation with the system for the price of substantial qualitative domain knowledge to bias the learning.

In our approach a qualitative control strategy is induced from a human execution trace. We show that this strategy gives good insight into the operator's control strategy and defines a space for controller optimization that results in a set of very successful quantitative strategies.

3.1 Human learning

The learning examples for behaviour cloning were control traces from an experiment where six students learned to manually control the bike simulator and complete the control task. They were required to balance the bike and drive it to the goal that was at 100 m from the start position, with the initial frame orientation of the bike $-\pi/2$ from the goal direction. More precisely: the control task was to drive a bike from the start position ($\omega = \dot{\omega} = \theta = \dot{\theta} = 0, D_g = 100, \Psi_g = -\pi/2$) to the 5 m radius of the goal ($D_g < 5$). A trial was accepted as successful if the bike reached the 5 m radius of the goal within 100 seconds and did not fall over on the way. The measure of the operator's success is the time needed to complete the task.

Each of the operators performed (a) 100 trials by using only torque control (no weight-shifting), plus (b) 50 trials using both displacement and torque control. First they learned to balance the bike using only torque control. When they were allowed to also use the displacement control, they usually used it only for the initial turn towards the goal, but this did not significantly improve their performance.

The operators found the task very difficult, since maintaining the balance requires precise adjustment of the handlebars and/or displacement of the center of mass. Losing the concentration for just a moment results in large tilt angle velocity $\dot{\omega}$ and falling of the bike. The requirement to approach the goal makes the task even harder, since the balancing and the goal-aiming are to be performed simultaneously. For this reason all the operators needed more than 25 unsuccessful trials before completing the control task for the first time. Usually the first successful trials of an operator were far from optimal: they turned to the goal slowly and sometimes even drove in circles before reaching the goal position. Afterwards the operators improved their performance, so most of the operators were able to complete the task in less than 45 sec. The most successful operator needed 37 sec. to complete the task, which is very close to the minimal possible time.

4 Learned qualitative strategy

To learn the operator's qualitative strategy, we used our qualitative induction system QUIN to induce a qualitative tree from an operator's execution trace. First we give some details of the approach used to learn the operators's qualitative strategy. Then we present an example of a qualitative strategy induced from an operator's trace and show that such a strategy gives a good insight into the operator's sub-cognitive control skill. In Section 5 we show that it is also very general and successful, and defines a space for the controller optimization.

When learning the operator's strategy one possibility would be to induce constraints on operator's control action, but we prefer to induce constraints among state variables defining what we call a *generalized operator's trajectory*. There are two advantages of learning a generalized operator's trajectory compared to learning directly the operator's actions. First, a strategy in the form of a generalized trajectory is usually closer to human thinking about control strategies and easier to understand. Second, learning the generalized trajectory yields more successful and robust controllers, as shown in the crane [13] and the acrobot [15] domains. A comparison of the two approaches is studied in [14]. A similar idea of decomposing the cloning problem into two earning problems appears also in [1]. They use abduction to learn effects of control actions and decision trees to learn subgoals. However, they do not report on experiments to investigate benefits regarding the success of controllers constructed in this way.

In our experiments we used traces where operators controlled the front wheel angle θ by applying torque control to the handlebars. ¿From an operator's trace we induced constraints between the operator's desired front wheel angle θ_{des} and the rest of the state variables. Here θ_{des} was computed as θ in the next state from the execution trace: $\theta_{des}(t) = \theta(t + dt)$. In the experiments dt was 0.1 sec. The induced constraints define the generalized operator's trajectory w.r.t. θ_{des}. Note that it would also be possible to learn the generalized trajectory of $\dot{\theta}_{des}$ or τ_{des}, but the θ_{des} trajectory is closer to human reasoning about ridding a bike and therefore easier to understand.

The following qualitative tree was learned from one of the most successful execution traces:

$$\begin{array}{l}
\Psi_g < 0.015 \\
\mid \quad \Psi_g < -0.027 \\
\mid \quad \mid \quad \theta_{des} = M^{+,+,-}(\omega, \dot{\omega}, \Psi_g) \\
\mid \quad \Psi_g > -0.027 \\
\mid \quad \mid \quad \theta_{des} = M^{+,+}(\omega, \dot{\omega}) \\
\Psi_g > 0.015 \\
\mid \quad \theta_{des} = M^{+,+,-}(\omega, \dot{\omega}, \Psi_g)
\end{array}$$

By introducing the qualitative term $NearZero(\Psi_g) \equiv (-0.027 < \Psi_g < 0.015)$ this tree can be written as the qualitative rule:

$$\begin{array}{l}
\text{if } NearZero(\Psi_g) \text{ then} \\
\quad \theta_{des} = M^{+,+}(\omega, \dot{\omega}) \\
\text{else } \theta_{des} = M^{+,+,-}(\omega, \dot{\omega}, \Psi_g)
\end{array} \tag{5}$$

Although this rule is very simple, it gives very good insight into the operator's control strategy. The operator uses the simpler qualitative rule, when the bike is oriented towards the goal (when $NearZero(\Psi_g)$ is true) and the more complex rule when the bike is not oriented towards the goal and has to turn towards the goal.

When the bike is oriented towards the goal, the operator just maintains the balance by the rule $\theta_{des} = M^{+,+}(\omega, \dot{\omega})$. The rule is very intuitive and says that the front wheel angle θ should be bigger at bigger tilt angle ω. Bigger ω requires bigger centrifugal force F_c, which acts from the centre of the turn and prevents the fall of the bike towards the centre of the turn. Bigger F_c is achieved by bigger θ, since the radius of the turn is smaller. The rule also says to increase (decrease) θ when the tilt velocity $\dot{\omega}$ is increasing (decreasing). Therefore, if the bike starts falling over, the rider should turn the front wheel in the direction of the fall, which increases the restoring centrifugal force. Since increasing $|\theta|$ causes increasing $|F_c|$, the change of θ causes the opposite change of $\dot{\omega}$ and the rule aims at bounding the magnitude of $\dot{\omega}$.

More surprising and possibly counter-intuitive is the induced rule $\theta_{des} = M^{+,+,-}(\omega, \dot{\omega}, \Psi_g)$. This rule balances the bike similar as the previous rule and also turns it towards the goal by turning the front wheel in the *opposite direction, away from the goal.* One might naturally expect that turning to the goal is achieved by turning the front wheel in the direction of the goal by the rule $\theta_{des} = M^{+}(\Psi_g)$. But this gives exactly the opposite result.

Suppose the bike drives straight with $\omega = \dot{\omega} = \theta = \dot{\theta} = 0, \Psi_g = -\pi/2$ and then suddenly increases θ towards the goal. This decreases $\dot{\omega}$ and ω. To prevent the fall, θ must be decreased away from the goal and once the balance is restored, ω is negative. So negative θ maintains the balance and the bike is turning away from the goal. To really achieve a turn towards the goal, we need to change the tilt ω (not θ) towards the goal by first decreasing θ away from the goal. Only after that, θ_{des} increases since ω has increased. That is exactly what happens if the induced rule is used.

4.1 Notes on experiments

To control the system, torque control is needed. We could learn the system dynamics by a function approximator and then compute the desired torque from the desired θ_{des} and the current state. Instead the torque control was computed by conventional PD controller: $\tau = P \times (\theta_{des} - \theta) - D \times \dot{\theta}$. We use $P = 50$ and $D = 5$, exactly the same as in [3]. Other approximate control (following θ_{des}) would also do since the strategy being goal directed, is robust [13].

When experimenting with different traces, an interesting observation was that similar trees were learned from all of the successful traces of the same operator. Also, most of the trees induced from a successful trace have the "default" rule $\theta_{des} = M^{+,+,-}(\omega, \dot{\omega}, \Psi_g)$ in the root of the tree.

5 Transforming the induced qualitative strategy into operational quantitative strategies

5.1 From qualitative to quantitative control strategies

The induced qualitative strategy given by Rule 5 defines a set of quantitative strategies with free qualitative term $NearZero$ and qualitatively constrained functions $M^{+,+}(\omega, \dot{\omega})$ and $M^{+,+,-}(\omega, \dot{\omega}, \Psi_g)$. By transforming the induced qualitative strategy into a set of executable quantitative strategies we will show that it is general (not very sensitive to the exact setting of the qualitative parameters) and very successful. To transform the qualitative strategy into an executable quantitative strategy, the qualitative parameters have to be concretised into quantitative (real valued) functions. First we show how QCFs can be generated at random and then we present the experimental results.

We use the simpler rule $M^{+,+,-}(\omega, \dot{\omega}, \Psi_g)$ as an example to illustrate how a qualitative strategy is transformed into executable quantitative strategies. When Rule 5 was used instead of $M^{+,+,-}(\omega, \dot{\omega}, \Psi_g)$, the results were similar.

To transform QCFs into real functions, some domain knowledge and/or the information from the operator's trace can be used. Let f denote a function $[-\omega_{mx}, \omega_{mx}] \times [-\dot{\omega}_{mx}, \dot{\omega}_{mx}] \times [-\Psi_{mx}, \Psi_{mx}] \mapsto [\theta_{mn}, \theta_{mx}]$ satisfying qualitative constraints $M^{+,+,-}(\omega, \dot{\omega}, \Psi_g)$. The maximal values from the operator's trace are used to set the definition area of function f: $\omega_{mx} = \pi/36$, $\dot{\omega}_{mx} = 0.12$, $\theta_{mx} = \pi/6$, $\Psi_{g_{mx}} = \pi/2$. Three simple facts about the bike control task define additional constraints on f:

1. *maximal front wheel deflection*: maximal $|\theta_{des}|$ is between (depending on the operator) $\pi/10$ and $\pi/3$ ($\pi/10 \le |\theta_{mn}|, |\theta_{mx}| \le \pi/3$).
2. *drive straight if bike is aiming at the goal*: $f(0, 0, 0) = 0$

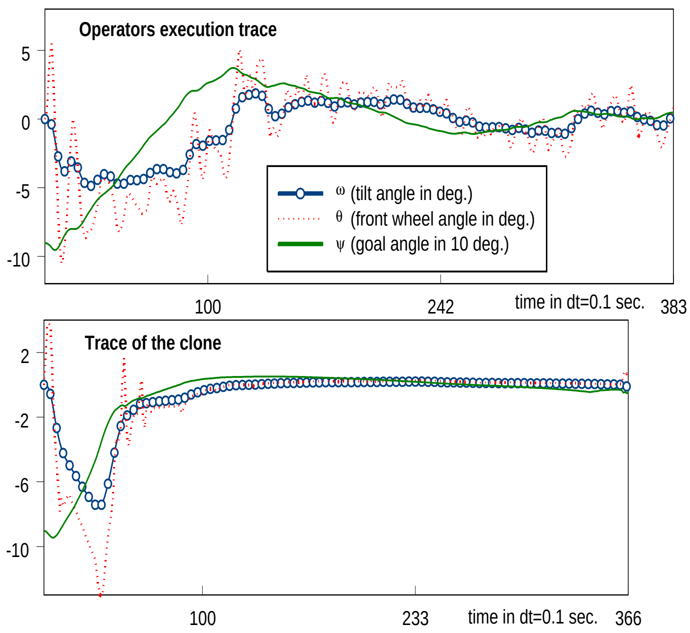

Figure 2: Operator's trace and its clone: Operator completes the task in 38 sec., while the clone needs approximately 36 sec. They both use the same qualitative strategy, but the clone makes the initial turn faster and is more precise in aiming at the goal afterwards.

3. *balancing is more important then aiming at the goal*: when ω and $\dot{\omega}$ both change from their minimal to their maximal values, θ_{des} changes $c > 1$ times of the change when Ψ_g changes from its minimal to its maximal value. Considering the constraints of the QCF $M^{+,+,-}(\omega, \dot{\omega}, \Psi_g)$ this gives:

$$
\begin{aligned}
\Delta_\theta &= f(\omega_{mx}, \dot{\omega}_{mx}, 0) - f(-\omega_{mx}, -\dot{\omega}_{mx}, 0)/c \\
&= |f(0, 0, \Psi_{mx}) - f(0, 0, -\Psi_{mx})| \\
&= |f(\omega_{mx}, \dot{\omega}_{mx}, \Psi_{mx}) - f(\omega_{mx}, \dot{\omega}_{mx}, -\Psi_{mx})| \\
&= |f(-\omega_{mx}, -\dot{\omega}_{mx}, \Psi_{mx}) - f(-\omega_{mx}, -\dot{\omega}_{mx}, -\Psi_{mx})| \qquad (6) \\
f(-\omega_{mx}, -\dot{\omega}_{mx}, \Psi_{mx}) &= \theta_{mx} \\
f(\omega_{mx}, \dot{\omega}_{mx}, -\Psi_{mx}) &= \theta_{mn} \\
\theta_{ma} - \theta_{mn} &= (c+1)\,\Delta_\theta
\end{aligned}
$$

We used the procedure described in Section 5.2 to generate at random above described functions f. Every generated function defines a quantitative strategy consistent with the induced qualitative strategy and the constraints imposed by the three simple facts of the bike control task. To generate the functions we used the grid $C = 8 \times 8 \times 8$, $c = 2, 3, 4, 5$ and random $|\theta_{mn}|$, $|\theta_{ma}|$ from $[\pi/10, \pi/3]$. Out of 400 randomly generated quantitative strategies, 237 (59.2 %) completed the task with the mean time 42.1 sec. and best time 35.7 sec. Many of the quantitative strategies are faster than any human operator's trace. The remaining 163 strategies failed to maintain a balance or were not accurate enough in aiming at the goal. An execution trace of one of the clones and the operator's trace are given in Fig. 2. In Fig. 3 the clone's balancing strategy is illustrated by the generated $f(\omega, \dot{\omega}, \Psi_g = 0)$.

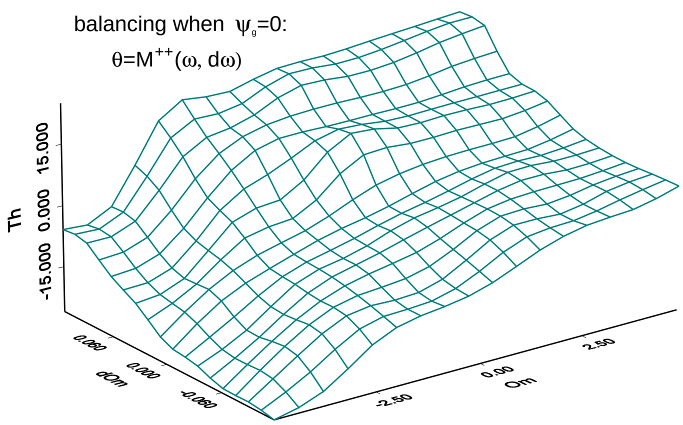

Figure 3: Balancing strategy used in Fig. 2. The Th-axis is the desired front wheel angle θ_{des} as a function of ω and $\dot{\omega}$ when Ψ_g=0. The balancing strategy satisfies the induced qualitative rule $\theta_{des} = M^{+,+}(\omega, \dot{\omega})$ and was generated at random.

Another important property of such controllers is their *robustness.* We performed two sets of experiments with:

1. *change in the start state:* the starting distance of the goal D_g was modified by 20%, that is from D_g=100 in the original control task to D_g=80 and D_g=120.
2. *random displacement of the bicyclist form the centre of mass:* at each time step a random $d_{noise} \in [-0.002, 0.002]$ was added to the current displacement from the mass centre. Note that in all previous experiments displacement from the mass centre was zero.

As in the above described experiments we randomly generated 100 quantitative strategies, corresponding to 100 controllers. Every controller was tested with both changes in the start state and with random displacement of centre of mass. With D_g=80, out of 100 controllers 54 were successful. 60 controllers were successful with D_g=120. Comparing this with 59.2 % controllers that are successful for the original task, we can conclude that a change in the start state does not effect the controller much.

When using random displacement from mass centre. 26 out of 100 controllers were successful. Considering the fact, that random displacement makes a control task considerably harder, the results are very good.

Some controllers were also tested with considerably different start states, for example with D_g=200 and changed starting orientation of the bike. Most of the controllers that were successful with the original task, were also successful with altered tasks.

Theses results show significant improvement in controllers robustness comparing to the previous results with behaviour cloning where the operator's actions were induced as a direct mapping from states to actions.

5.2 Generating QCFs at random

Here we describe how piece-wise linear QCFs can be generated at random. We define a set of functions $F^{s_1,\ldots,s_m}$, $s_i \in \{+,-\}$ defined on $[0,1]^m$ and mapping to $[0,1]$, such that any function $f^{s_1,\ldots,s_m}$ from this set is monotonically increasing (decreasing) in all the arguments i, if $s_i = +$ ($s_i = -$).

Different piece-wise linear functions $f^{s_1,\ldots,s_m}$ can be generated by splitting the space $[0,1]^m$ into $C = \prod_{i=1}^{m} R_i$ (R_i is the number of slices in the i-th dimension) cells and assigning each cell (with address $a = (a_1, \ldots, a_m)$, $a_i \in \{0, \ldots, R_i - 1\}$) an appropriate value v_a in its left-bottom corner. The values at $x_i = 1$ are addressed by $a_i = R_i$. The values at other points in a cell are linearly interpolated by considering the values at m neighbors of the cell. The value at a point x, with the cell address $a = (a_1 \ldots, a_m)$, $a_i = \lfloor x_i R_i \rfloor$ can be computed as:

$$f^{s_1,\ldots,s_m}(x) = v_a + \sum_{i=1}^{m} d_i(a)(x_i R_i - \lfloor x_i R_i \rfloor)$$

Here $d_i(a) = v_{\ldots,1+a_i,\ldots} - v_{\ldots,a_i,\ldots}$ denotes the i-th neighbour difference. To satisfy the qualitative constraints, $sign(d_i(a)) = s_i$.

Suppose we want to generate such random functions f^{s_1,s_2} on a 2×2 grid, and the values of the edge cells are already stored as $v_{0,0}$, $v_{0,2}$, $v_{2,0}$ and $v_{2,2}$. We first set the values $v_{1,0}$ and $v_{1,2}$ by taking two "suitable" random values from $(v_{0,0} \ldots v_{2,0})$ and $(v_{0,2} \ldots v_{2,2})$. These two values are suitable if their differences respect the constraints given by the signs s_1, s_2. Similarly we set the values $v_{0,1}$ and $v_{2,1}$. The value of the middle cell $v_{1,1}$ is set to a random number from $(v_{1,0} \ldots v_{1,2}) \cap (v_{0,1} \ldots v_{2,1})$. Note that the intersection is not empty, if the edge values are suitable. Similarly, random functions $f^{s_1,\ldots,s_m}$ can be generated by dividing the space into m^2 subregions, generating suitable values for the edge cells in every region, and recursively set the cell values in every subregion.

6 Conclusions

We presented a method for learning qualitative trees implemented in program QUIN. QUIN was applied to difficult learning task of cloning the human skill of bicycle riding. Qualitative trees were induced which define constraints on the operator's control trajectory. These constraints were transformed into actual, quantitative controllers for the riding task by random search of the space of functions satisfying the constraints. The results showed that the induced qualitative strategy is general and successful. Some points of interest are:

- In this paper we applied induction of qualitative trees to bicycle riding. On the basis of experience in the crane control problem and in synthetic domains [12] we believe this approach is applicable to a wide range of domains.

- We believe that the success and robustness of our controllers are due to (a) using a *generalized operator's trajectory* (as opposed to direct mapping from states to actions), and (b) the induced trajectory is *qualitative*.

- We used some bike domain knowledge (three simple facts) and some quantitative constraints from the operator's trace to constrain the search among the qualitative family of controllers. We could use just constraints from one or more traces or just the domain knowledge. This approach enables easy inclusion of additional constraints which further improves the percentage of successful controllers [13].

References

[1] M. Bain and C. Sammut. A framework for behavioural cloning. In K. Furukawa, D. Michie, and Muggleton S., editors, *Machine Intelligence 15*, pages 103–129. Oxford University Press, 1999.

[2] I. Bratko, I. Mozetič, and N. Lavrač. *KARDIO: a Study in Deep and Qualitative Knowledge for Expert Systems.* MIT Press, 1989.

[3] M. Brodie and G.F. DeJong. Learning to ride a bicycle using iterated phantom induction. In I. Bratko and S. Džeroski, editors, *Proceedings of the 16th International Conference on Machine Learning*, pages 57–66. Morgan Kaufmann, 1999.

[4] R. Camacho. Using machine learning to extract models of human control skill. In *Proceedings of AIT'95*, 1995. Brno, Czech Republic.

[5] B. Falkenheiner and K. Forbus. Self explananatory simulations: an integration of qualitative and quantitative knowledge. In *Proceedings of the 4th International Workshop on Qualitative Physics*, 1990. Lugano, July 1990.

[6] K. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[7] J. Kuipers, B. Qualitative simulation. *Artificial Intelligence*, 29:289–338, 1986.

[8] J. Quinlan and R. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

[9] J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning*, pages 463–471. Morgan Kaufmann, 1998.

[10] J. Rissanen. Modelling by shortest data description. *Automatica*, 14:465–471, 1978.

[11] C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *Proceedings of the 9th International Workshop on Machine Learning*, pages 385–393. Morgan Kaufmann, 1992.

[12] D. Šuc. Qualitative induction. Technical report, Artificial Intelligence Lab., Faculty of Computer and Information Sc., University of Ljubljana, Slovenia, 1999.

[13] D. Šuc and I. Bratko. Symbolic and qualitative reconstruction of control skill. *Linköing Electronic Articles in Computer and Information Science*, 4(033):1–22, 1999. http://www.ep.liu.se/ea/cis/1999/033/.

[14] D. Šuc and I. Bratko. Problem decomposition for behavioural cloning. In R.L. Mántaras and E. Plaza, editors, *Proceedings of the European Conference on Machine Learning*, pages 382–391. Springer-Verlag, 2000.

[15] D. Šuc and I. Bratko. Skill modelling through symbolic reconstruction of operator's trajectories. *IEEE Transaction on Systems, Man and Cybernetics, Part A*, 2000. In publishing.

[16] B. Williams. Interaction-based invention: designing devices from first principles. In *Proceedings of the 4th International Workshop on Qualitative Physics*, 1990. Lugano, July 1990.