A Machine Learning System for Controlling a Rescue Robot

Timothy Wiley[1], Ivan Bratko[2], and Claude Sammut[1(✉)]

[1] School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia
{t.wiley,c.sammut}@unsw.edu.au
[2] Faculty of Computer and Information Science, The University of Ljubljana, Ljubljana, Slovenia
bratko@fri.uni-lj.si

Abstract. Many rescue robots are reconfigurable, having subtracks (or flippers) that can be adjusted to help the robot traverse different types of terrain. Knowing how to adjust them requires skill on the part of the operator. If the robot is intended to run autonomously, the control system must have an understanding of how the flippers affect the robot's interaction with the ground. We describe a system that first learns the effects of a robot's actions and then uses this knowledge to plan how to reconfigure the robot's tracks so that it can overcome different types of obstacles. The system is a hybrid of qualitative symbolic learning and reinforcement learning.

Keywords: Machine learning · Qualitative models · Rescue robots

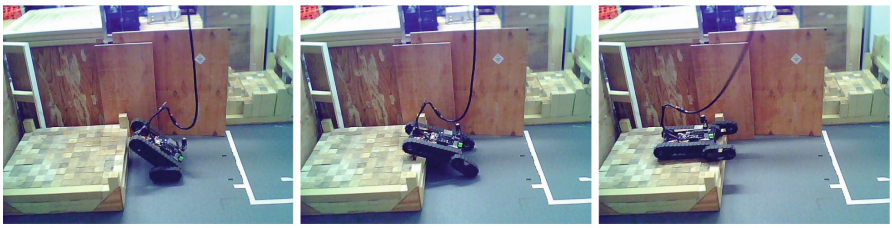
1 Introduction

Driving a wheeled robot on flat ground is relatively straightforward once the robot has a map of its environment. The operator only needs to control the steering and speed. Driving a tracked vehicle over rough terrain is much more difficult, especially if the vehicle has subtracks, or flippers, because the operator must make decisions about the configuration of the flippers, as well as steering and speed. Thus, subtracks give the robot greater terrain traversal capabilities at the expense of greater control complexity. Reconfigurable robots are used commonly in urban search and rescue, but are mostly tele-operated. However, remote control is impossible when there is a loss of communication. Therefore, rescue robots need at least enough autonomy to be able to navigate out of a radio dropout zone. The goal of this research is to develop an autonomous driving system for reconfigurable robots. Since the interactions of the robot with the terrain in a disaster site are extremely difficulty to predict, we use a learning system to build a model of how control actions, including changing flipper angles, affect the robot's state. Once we have the model, the driving system can plan a sequence of actions to achieve the desired goal state.

H. Akiyama et al. (Eds.): RoboCup 2017, LNAI 11175, pp. 108–119, 2018.
https://doi.org/10.1007/978-3-030-00308-1_9

An important requirement is that the learning system must be able to acquire the model in a small number of trials. A naive application of reinforcement learning [22], which is commonly used for such tasks, may need thousands of trials, which would be very slow and eventually break the robot. Therefore, a more economical approach is required. When humans learn a new skill, they are almost always guided by some knowledge of the domain. For example, when learning to drive a car with a manual gear shift, an instructor tells the student something like, "gradually depress the clutch while releasing the accelerator, then shift the gear lever and depress the accelerator, at the same time releasing the clutch". If the student was required to deduce this sequence by trial and error, this would take a very long time. However, armed with the instructor's hints, the student only needs to learn how to make this plan operational, by learning how to synchronise the actions so as not to stall the car.

In this scenario, the student has been given a plan the describes the correct sequence of actions, but leaves out the numerical details, which must still be learned through trial and error. However, knowing what actions to perform greatly reduces the number of trials, as now the student only needs to refine a set of parameters within a given envelop. We adopt a similar approach to building a control system for driving a rescue robot. A planner produces a sequence of actions and reinforcement learning system performs the parameter refinement.



(a) (b) (c)

Fig. 1. The Negotiator robot reversing up a high step

The planner requires models of the actions the robot can perform. That is, it must know the preconditions and effects of each action. These may be given by an instructor, as above for learning to drive a car, or the models may, themselves, be learned. During a "pre-training" phase, the robot performs random actions, observing the robot's state before and after each action. These observations become training examples for a system that learns a qualitative model of the actions. A qualitative model is like the instructor's explanation. It describes each action at an abstract level but does not specify exact numerical values for any parameters. This two stage learning process, acquiring an approximate abstract model, followed by reinforcement learning to refine parameters, greatly reduces the overall search space, and therefore reduces the number of trial required to learn a new skill. However, it is possible that the learned qualitative model is

incorrect, in the sense that it does not provide the constraints needed to learn an operational behaviour. In this case, system acquires more training data to refine the qualitative model. Thus, it is a *closed-loop* learning system that can continuously improve its behaviour. Closed-loop learning is the main focus of this paper, but we must first explain the qualitative representation of actions and how these are used for planning. We then describe how the qualitative models are acquired by a symbolic learning system and how reinforcement learning refines action parameters. Experimental results are presented that demonstrate that by closing the loop in the learning system, errors from a single pass can be corrected.

2 Qualitative Representation of a Rescue Robot

The experimental platform that we use is an iRobot Negotiator, shown in Fig. 1 climbing a step. In this case, the step is too high for the robot to climb with the flippers forward, since they are not long enough lift to robot over the step. Instead, the robot reverses up to the step and uses the flippers to raise the body, which is long enough to reach over the edge of the step. The robot's planning system should be able to reason about the geometry of the vehicle and make appropriate decisions about what sequence of actions will achieve it's goal. To do so, the planner must have a model of how actions affect the robot and its environment. The model does not need to be highly accurate for the planner to get the right sequence. An approximate qualitative model will suffice.

The qualitative model is based on Kuiper's QSIM [8] but is extended so that it can be used for planning. QSIM represents the dynamics of a system by a set of *qualitative differential equations* (QDE). An example of a model for this domain is shown in Fig. 2. The graph shows the relationship of the angles of the robot's body to the floor, θ_b, and the flipper angle, relative to the body, θ_f. The relation, $M^+(\theta_f, \theta_b)$, indicates that if one of the arguments increases (θ_f), the other also increases (θ_b). The relation, $M^-(\theta_f, \theta_b)$, says the that when one variable increases, the other decreases, that is, they change in opposite directions. The $const(\theta_b, 0)$ relation states that θ_b remains steady at 0.

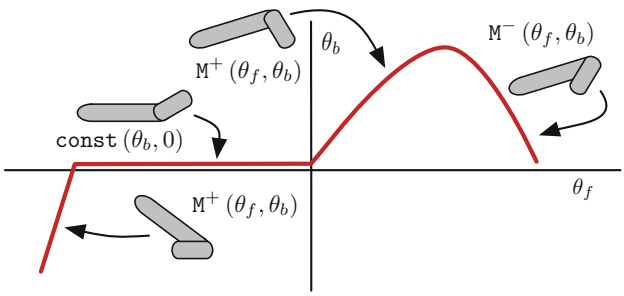


Fig. 2. Qualitative Models for the Negotiator Robot, describing the relationship between the angle of the flippers, θ_f, and base, θ_b, through four operating regions.

Each segment in the graph represents an *operating region*, that is, a region of the state space in which one model holds. As the flippers rotate clockwise through 360°, the body is raised and lowered, while the flippers are angled below the body, but have no effect when they are above the body. If the flippers are rotated anti-clockwise, they can raise the body. To accommodate different operating regions, we extend the QSIM notation so that each QDE has an associate *guard*, which is the condition under which that QDE holds.

$$Guard \rightarrow Constraint \tag{1}$$

Since a QDE does not specify a numerical relationship between variables, we regard a QDE as a *constraint* on the possible values of the variables. For example, in a particular region, if the flipper angle decreases, the body angle increases. QSIM was originally intended to perform qualitative simulation. That is, given an initial condition, QSIM estimates how the system's state evolves over time. A state in QSIM is represented by the qualitative values of the variables. However, a qualitative variable does not take on a numerical value. Instead it's value is a pair: $magnitude/direction$, where the magnitude may be a fixed landmark value or an interval, and the direction is one of increasing, decreasing or steady. For example, the robot's position may be given by $x = 0..x_{step}/inc$, which states that the x position of the robot is between its initial position and the position of the step, and its value is increasing. A set of transition rules specifies how one state evolves into the next. For example, when the robot reaches the step, the position becomes $x = x_{step}/std$. A detailed explanation is given by Wiley [26].

3 Planning with Qualitative Models

QSIM has no concept of an action, which is needed for planning. We extend the QSIM representation by distinguishing certain variables as *control variables*, whose values can be changed by the planner. A change in a control variable corresponds to an action. For example, changing the flipper angle, θ_f, corresponds to the motor action that moves the flipper. Like classical planning, given an initial state and a goal state, the qualitative planner searches for a sequence of actions that leads to the goal state. We briefly describe the planner below, but details of the implementation are given elsewhere [27,28].

Qualitative planning differs from classical planning and numerical simulation because the variables can specify a range of values. Therefore, a qualitative state can be thought of as defining constraints on regions of valid quantitative states, that is, where the variables take on specific values. For example, a qualitative state may be $x = 0..x_{step}/inc, \theta_b = 0/std, v = 0..max/std, \theta_f = 0..90/std$, which describes the robot driving up to the step. To find a sequence of actions the qualitative planner must propagate the constraints from the initial to the final state. Therefore, planning can be seen as a constraint satisfaction problem. We take advantage of this, translating the planning problem into an Answer Set Programming (ASP) problem [4] and using the Clingo-4 solver [5] to generate the plan, described in [27].

The search space for the qualitative planner is considerably smaller than the search space for the corresponding continuous domain. This can be seen from the previous observation that one qualitative state covers a region of quantitative states. Thus, qualitative planning is reasonably efficient in finding a sequence of actions. However, these actions are only approximate, in the same sense as the driving instructor's plan for changing gears. In this case of the robot, as it approaches an obstacle, the plan may say that the flippers should be raised, but not by how much. We will see in Sect. 5, that "how much" can be found by reinforcement learning, but for this to be efficient, i.e. require only a small number of trials, the planner must pass on its constraints to the reinforcement learning system.

In addition to the sequence of actions, the planner generates the state transitions caused by those actions:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \rightarrow \ldots \xrightarrow{a_n} s_g$$

Thus, for each action, we have the preceding and succeeding states. As these are qualitative states, they effectively specify the preconditions and postconditions of the action. Thus, when the reinforcement learning system searches for the angle to set for the flippers, it must only search with the constraints of the pre and post conditions. In the following sections we explain how the qualitative model is learned and then how reinforcement learning is used to find operational parameters for the actions generated by the planner.

4 Learning a Qualitative Model

To learn a qualitative model of the robot, the system must acquire samples of the robot's interaction with its environment. In the experiments described in Sect. 6, a human operator drives the robot, performing random actions. This could equally be done by the robot "playing" by itself. Each time an action is performed, the before and after states are recorded so that the changed effected by the action can be determined. An example of flipper actions is shown in Fig. 3. The figure also shows qualitative relations induced by Padé [29]. This systems uses a form of regression, called *tubed-regression*, to find regions of a graph where neighbouring points have the same qualitative relation. In Fig. 3, Padé has identified regions where the body angle increases with the flipper angle, decreases, or remains steady. In this case, the binary relation between the angles has been rewritten in functional form so that the body angle is dependent on the flipper angle. Note that this plot corresponds to the graph in Fig. 2.

The data show that there are several operating regions where these relations apply. Recall that we express the qualitative model as a set of rules, whose left hand side specifies the operating region and the right hand side give the qualitative constraints (Eq. 1). These rules can be automatically generated from the graph by applying a symbolic rule learning system or decision tree learner. In this case, we use Quinlan's C4.5 [18]. A problem with this is that classifier learning systems usually require negative examples, as well as positive examples.

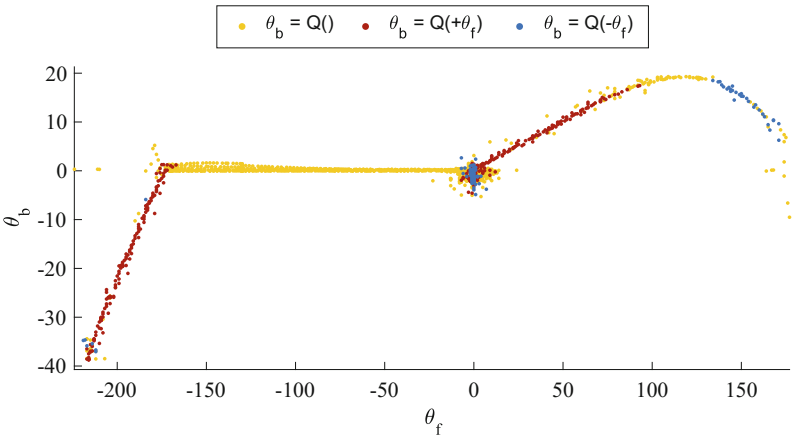


Fig. 3. Body angle versus flipper angle from actions

To accommodate this, we make a kind of closed world assumption where the space outside the sampled area is assumed to contain only negative examples. Thus, *null* values are randomly added to the C4.5's training data so that it can induce a decision tree in which the conditions in the intermediate nodes specify the operating region where the qualitative relation in the leaf node applies. Figure 4 shows the decision tree induced from the flipper data. Since the training data are noisy, the decision tree is not as clean as a model that a human might write.

With the qualitative model represented by the decision tree, the planner can determine the qualitative state of the system. QSIM's transition rules, mentioned in Sect. 2, tell the planner what possible next states are reachable depending on which action is applied in the current state. With this information, the planner can search for a sequence of actions that will achieve its goal.

5 Refining Actions by Reinforcement Learning

The actions generated by the planner are qualitative. For example, to climb a low step, the plan may indicate that the velocity should be forward, non-zero and the flipper angle should be between 0 and 90°. To find values of velocity and flipper angle that actually work, the robot must perform some trial and error learning. Figure 5 illustrates this process. For each action generated by the plan, the robot has many options for executing that action (e.g. selecting an angle between 0 and 90°). Through trial and error learning, it must discover the parameter settings that will result in the robot achieving its goal.

Parameter refinement is setup as a Semi-Markov Decision Process (SMDP) over Options [23]. In a SMDP, time is continuous and actions have a duration, which may be of variable length. The robot may try to select a set of options, that is, numerical control value settings, that will work, called a *satisficing* solution,

```
theta_b<=-5.072
|    theta_f<=-178.415: Q(+theta_f)
|    theta_f>-178.415: Q(null)
theta_b>-5.072
|    theta_b<=1.628
|    |    theta_f>17.279: Q(null)
|    |    theta_f<=17.279
|    |    |    theta_b<=-2.772: Q(+theta_f)
|    |    |    theta_b>-2.772: Q()
|    theta_b>1.628
|    |    theta_f<=133.524
|    |    |    theta_b>18.704: Q()
|    |    |    theta_b<=18.704
|    |    |    |    theta_f<=-10.273: Q(null)
|    |    |    |    theta_f>-10.273
|    |    |    |    |    theta_f<=98.500: Q(+theta_f)
|    |    |    |    |    theta_f>98.500: Q(null)
|    |    theta_f>133.524
|    |    |    theta_f<=154.558: Q(-theta_f)
|    |    |    theta_f>154.558: Q()
```

Fig. 4. The C4.5 decision tree

or it may try to find settings to achieve an optimal solution. Which type of solution is found depends on the reward function that is set. For a satisficing solution, the reward is simply 1 if the plan succeeds and 0 if it fails. To find an optimal solution, a cost function must be defined. Here, we only consider satisficing solutions, but optimal solutions are treated in [26]. The robot repeats trials, updating its policy according to its reward, until a solution is found.

The unique part of this process is how the SMDP is constructed from a plan. First, the ranges for all continuous variables must be discretised for the SMDP algorithm. This corresponds to the intervals in the qualitative model being split into a set of smaller intervals. So we know distinguish between a quantitative state description, where all the variables are continuous, a qualitative state description, as described in Sect. 2 and a *discretised* state representation, which is similar to the BOXES representation [11], commonly used in reinforcement learning. For each action, the qualitative states satisfying the pre-condition and post-condition are required. Options are formed from every combination of discretised states that are subsets of the qualitative states. QSIM constrains a variable's rate-of-change, as well as its magnitude. If the qualitative state satisfying the post-condition has an increasing rate-of-change, its quantitative value must increase during the option, likewise for decreasing or steady rates-of-change. Only options that satisfy the rates-of-change constraints are added to the SMDP.

Once the system has found the available options, it can perform its trial and error learning to turn the planning actions into operational motor commands.

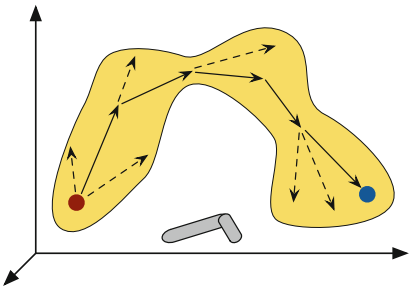


Fig. 5. The planner gives an approximation of actions, (yellow region) that must be refined by trial and error learning into precise motor actions (arrows). (Color figure online)

However, the quality of the plan depends on the quality of the model that was induced from the original training data, which was randomly sampled in the beginning. If the sampling does not yield sufficient training data in the regions of the state space relevant to the task, then the model may be poor, resulting in inefficient plans. This problem can be reduced by closed-loop learning.

6 Closed-Loop Learning and Experiments

The learning process begins with the collection of training examples for learning the qualitative model. In these experiments, the data were generated by a human operator instructing the robot to perform mostly random actions. Experiments were performed for several tasks using the Negotiator robot [26,28]. These included driving over different height steps and climbing a staircase. Of these, climbing a high step, which requires the reversing manoeuvre, was the most difficult and is the one we focus on in this discussion. The number of examples for each task is shown in Table 1. Because the high step requires a complex plan, we obtained one training set of random examples and one handcrafted set. As explained below, the handcrafted data set was created because the random set resulted in very long plans. However, closed-loop learning helps avoid sampling problems.

Table 1. Training data sets used in the experiments.

Data set	Number of examples	Duration (seconds)	Sampling frequency (seconds per sample)
Low step	1290	129.1	0.1
High step	2740	280.2	0.1
High step (Generated)	250	5.3	0.1
Staircase	765	160.1	0.2

In closed-loop learning, the system can repeat the entire learning process to improve its performance. The first column in Table 2 gives statistics for learning to climb a high step. Over three repetitions, the average number of trials needed to learn to climb the step was 104 and the average amount of time needed to complete the task is 32.7 s. The actions and their effects are recorded so that they can be added to the training examples for the model learner.

The second column in Table 2 gives the statistics for learning to climb a high step after the data from the first pass are added. Over five repetitions, the average number of trials needed to learn to climb the step has been reduced to 31 and the average amount of time needed to complete the task is 25.1 s, indicating that a better plan was produced.

Table 2. Results for climbing a high step

Climbing high step	1st pass	2nd pass
Repeats	3	5
Min. trials	102	22
Max. trials	105	41
Avg. trials	104	31
Avg. time	32.7 (s)	25.1 (s)

The systems performs better because the training data for closed-loop learning contains more training examples in areas that the human operator failed to properly sample. In fact, the difference is stronger than the table indicates. In the first pass, the planner was helped by using the handcrafted training examples. Had this not been done, the number of trials and times would have been significantly greater.

7 Related Work

Madani, Hanks and Condon [9] show that when deterministic planning is possible, it has the advantage of having lower computational complexity than nondeterministic or stochastic planning. When the domain is continuous or noisy, any plan generated by a classical planner can only be a rough approximation to an optimal solution. Stochastic planning is able to generate solutions that are robust and somewhat closer to optimal when uncertainty is present. However, to achieve high performance in both learning and execution, they are usually heavily engineered to suit a specific problem domain. Abbeel, *et al.* [1] and Stulp, *et al.* [21] are excellent examples of systems that achieve very impressive performance in their respective domains of helicopter flight and humanoid robot walking. In both cases, much of this performance is achieved by priming the learning system with models of helicopter dynamics or gait trajectories. Thus learning is largely a matter of tuning parameters.

Model-based reinforcement learning methods provide structure to a problem to reduce the search required in both learning and planning. Structure can also be provided in the form of algebraic decision diagrams as in SPUDD [7] or by performing dimensionality reduction [10]. Hierarchical learning systems reduce search complexity by breaking a problem into layers. In some cases, each layer employs the same learning mechanism applied to different levels of abstractions, such as Dietterich [3], and Hengst [6]. Others use different representations and algorithms for each layer. For example, Powers and Balch [17] have a layered architecture for controlling a vehicle in a simulated world. A deliberative layer represents the world as a grid and performs path planning. The actual control is performed by a lower, reactive layer, which handles uncertainty in a continuous world. Our approach is similar in spirit to Powers and Balch [17] except that we aim for greater domain independence by combining a general purpose planner, for constructing a sequence of abstract actions, with a parameter optimisation method to learn the implementation of those actions in a non-deterministic environment. The action models of the planner are extended to include qualitative models for reasoning about continuous variables and their qualitative relations, and are learned through active discovery by the robot.

This research builds on our own previous work on combining qualitative and quantitative methods in learning and planning. Potts [16] used machine learning to perform system identification. Ryan [19] created a planner that generates a sequence of actions whose implementation is not given but which can be learned by reinforcement learning. Ryan's method worked well in discrete domains but does not scale well to continuous domains. This problem was addressed by Sammut and Yik [20], who developed a system for learning the gait for a bipedal robot. Here, a planner produces a sequence of parameterised qualitative actions. Subsequent trial-and-error learning determines the values of those parameters so that the qualitative plan is made operational. Brown [2] used Inductive Logic Programming to learn action models for a robot planning system.

In work related to locomotion in rescue robots, Ohno *et al.* [14] manually specify control rules before learning precise actuator movements. Mihankhah [12] follow a similar pre-programmed approach with fuzzy controllers, and Tseng [24] explicitly model a robot slipping. Vincent and Sun [25] use reinforcement learning to teach a robot, in simulation, to climb over a box, and Mourikis [13] train multiple PID controllers to climb a staircase. These approaches all focus on learning the actuator movements. This is equivalent to only our parameter refinement stage, whereas, this work also builds a model of the robot and plans the robot's actions.

8 Conclusion

We have demonstrated a hybrid learning system that combines learning qualitative models with reinforcement learning. The result is a system that learns complex plans in a relatively small number of trials, supporting the claim that high-level knowledge about the task can reduce the attempts needed to learn a new skill.

There are several ways in which the present system can be improved. It relies on existing base learning methods, where Padé is used to induce qualitative relations that are then used by C4.5 to organise them by operating region. The use of these base learners has limitations. Padé can only induce M^+ and M^- relations for a limited number of variables. C4.5 requires the artificial introduction of negative training examples. In the future, we would like to explore different methods for learning qualitative models, such as [15]. It would also be preferable to find alternatives to C4.5 that can learn from positive only data or a form of model tree learning that can induce the qualitative model as it builds the classifier for the operating region.

Acknowledgements. We thank Jure Žabkar, the University of Ljubljana, for his help in using the Padé software, and Torsten Schaub and Max Ostrowski, the University of Potsdam, for their assistance with the Clingo-4 ASP solver, which are used in our symbolic planner. This research was supported by the Australian Research Council grant DP130102351 and an Australian Postgraduate Award.

References

1. Abbeel, P., Coates, A., Ng, A.Y.: Autonomous helicopter aerobatics through apprenticeship learning. Int. J. Robot. Res. **29**, 1608–1639 (2010)
2. Brown, S., Sammut, C.: Learning tool use in robots. In: Langley, P. (ed.) Advances in Cognitive Systems: Papers from the AAAI Fall Symposium, pp. 58–65. AAAI Press, Menlo Park (2011)
3. Dietterich, T.G.: The MAXQ method for hierarchical reinforcement learning. In: 15th International Conference on Machine Learning, pp. 118–126 (1998)
4. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer set solving in practice. Synthesis Lectures of Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, San Rafael (2013)
5. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: the Potsdam answer set solving collection. AI Commun. **24**(2), 107–124 (2011)
6. Hengst, B.: Discovering hierarchy in reinforcement learning with HEXQ. In: 19th International Conference on Machine Learning, pp. 243–250. Morgan Kaufmann (2002)
7. Hoey, J., St-Aubin, R., Hu, A., Boutilier, C.: SPUDD: stochastic planning using decision diagrams. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, pp. 279–288. Morgan Kaufmann (1999)
8. Kuipers, B.J.: Qualitative simulation. Artif. Intell. **29**(3), 289–338 (1986)
9. Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and related stochastic optimization problems. Artif. Intell. **147**(1–2), 5–34 (2003)
10. Mahadevan, S.: Learning representation and control in markov decision processes: new frontiers. Found. Trends Mach. Learn. **1**(4), 403–565 (2009)
11. Michie, D., Chambers, R.A.: BOXES: an experiment in adaptive control. Mach. Intell. **2**(2), 137–152 (1968)
12. Mihankhah, E., Kalantari, A., Aboosaeedan, E., Taghirad, H.D., Moosavian, S.A.A.: Autonomous staircase detection and stair climbing for a tracked mobile robot using fuzzy controller. In: Proceedings of the 2008 IEEE International Conference on Robotics and Biometrics, pp. 1980–1985 (2008)

13. Mourikis, A., Trawny, N., Roumeliotis, S.I., Helmick, D.M., Matthies, L.: Autonomous stair climbing for tracked vehicles **26**(7), 737–758 (2007)
14. Ohno, K., Morimura, S., Tadokoro, S., Koyanagi, E., Yoshida, T.: Semi-autonomous control system of rescue crawler robot having flippers for getting over unknown-steps. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3012–3018 (2007)
15. Pang, W., Coghill, G.M.: QML-Morven: a novel framework for learning qualitative differential equation models using both symbolic and evolutionary approaches. J. Comput. Sci. **5**(5), 795–808 (2014)
16. Potts, D., Sammut, C.: Incremental learning of linear model trees. Mach. Learn. **6**(1–3), 5–48 (2005)
17. Powers, M., Balch, T.: A learning approach to integration of layers of a hybrid control architecture. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009, pp. 893–898 (2009)
18. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Manteo (1993)
19. Ryan, M.R.K.: Using abstract models of behaviours to automatically generate reinforcement learning hierarchies. In: Sammut, C., Hoffmann, A. (eds.) Proceedings of the Nineteenth International Conference on Machine Learning, pp. 522–529. Morgan Kaufmann Publishers Inc., Sydney (2002)
20. Sammut, C., Yik, T.F.: Multistrategy learning for robot behaviours. In: Koronacki, J., Raś, Z., Wierzchon, S., Kacprzyk, J. (eds.) Advances in Machine Learning I. SCI, vol. 262, pp. 457–476. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-05177-7_23
21. Stulp, F., Buchli, J., Theodorou, E., Schaal, S.: Reinforcement learning of full-body humanoid motor skills. In: IEEE-RAS International Conference on Humanoid Robots, pp. 405–410 (2010)
22. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, 1st edn. MIT Press, Cambridge (1998)
23. Sutton, R.S., Precup, D., Singh, S.P.: Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. Artif. Intell. **112**(1–2), 181–211 (1999)
24. Tseng, C.K., Li, I.H., Chien, Y.H., Chen, M.C., Wang, W.Y.: Autonomous stair detection and climbing systems for a tracked robot. In: IEEE International Conference on System Science and Engineering, pp. 201–204 (2013)
25. Vincent, I., Sun, Q.: A combined reactive and reinforcement learning controller for an autonomous tracked vehicle. Robot. Auton. Syst. **60**(4), 599–608 (2012)
26. Wiley, T.: A Planning and Learning Hierarchy for the Online Acquisition of Robot Behaviours. Ph.D. thesis, School of Computer Science and Engineering, University of New South Wales (2017)
27. Wiley, T., Sammut, C., Bratko, I.: Qualitative simulation with answer set programming. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) Proceedings of the Twenty-First European Conference on Artificial Intelligence, pp. 915–920. IOS Press, Prague, August 2014
28. Wiley, T., Sammut, C., Hengst, B., Bratko, I.: A planning and learning hierarchy using qualitative reasoning for the on-line acquisition of robotic behaviors. Adv. Cogn. Syst. **4**, 93–112 (2016)
29. Žabkar, J., Mozina, M., Bratko, I., Demsar, J.: Learning qualitative models from numerical data. Artif. Intell. **175**(9–10), 1604–1619 (2011)