AAAI Technical Report SS-12-02
Designing Intelligent Robots: Reintegrating AI

Autonomous Skills Creation and Integration in Robotics

Lorenzo Riano and T. M. McGinnity
Intelligent Systems Research Centre
University of Ulster
Londonderry, BT48 7JL
United Kingdom
l.riano@ulster.ac.uk, tm.mcginnity@ulster.ac.uk

Abstract

The fragmentation of research in AI and robotics has created a vast repertoire of skills a robot could be equipped with but that must be manually integrated to form a complex action. We propose a novel evolutionary algorithm that aims at autonomously integrating, adapting and creating new actions by re-using skills that are either externally provided or previously generated. Complex actions are created by instantiating a Finite State Automaton and new skills are created using fully recurrent neural networks. We validated our approach in two scenarios, i.e. exploration and moving to pre-grasp positions. Our experiments show that complex actions can be created by composing independently developed skills. The results have been applied and tested with a real robot in a variety of scenarios.

Introduction

A robot needs a huge variety of skills in order to effectively solve a task in a real-world scenario. Providing a robot with these skills has been the main activity of researchers in robotics or in related fields. Unfortunately this has created fragmentation of the efforts, as the creation of reliable skills requires a researcher to devote most of the time on a particular problem.

Fortunately there is an increasing number of high-quality programs or routines that researchers can experiment with or simply integrate for free. A well known example is the Robotics Operating System (ROS) (Quigley et al. 2009), which is an open source effort to allow people to easily share code that runs robot applications. Skills like mapping, motion control, object detection and planning are freely available to researchers that can therefore integrate them to build complex applications.

When combining skills to solve a particular problem, a roboticist has to answer the following questions:

- Which skills are needed and how to combine them.
- How skills should be modified to work in cooperation with others.
- Which skills are not available and need to be created.

In this paper we propose a novel approach to combine, in a unified framework, composition, adaptation and creation of new actions based on skills the robot is already provided with[1]. An action is performed by a Finite State Automaton (FSA, plural automata) (Hopcroft, Motwani, and Ullman 1979), whose nodes represent skills that are externally provided to the robot (or previously created actions) and whose transitions are the outcomes of the actions. Each action can have a set of parameters. The FSA are instantiated by an evolutionary process (Bäck 1996) that simultaneously evolves the topology of the FSA and the parameters of the actions. Therefore, during the evolutionary process, skills are adapted and combined together to solve a particular problem, and new skills are created whenever the old ones are not sufficient.

The creation of new skills is performed by instantiating fully recurrent neural networks (NN) (Beer and Gallagher 1992). A NN skill is therefore an "empty placeholder" for a generic numerical processor. The results of the computation are then used by other skills as an input to their programs. For example, neural networks could generate numbers that will be interpreted by a motion planner as a location to move the robot to. Each NN skill has a network with fixed topology but, as the output of one network can be chained to the input of another, arbitrary topologies can be achieved. We will exploit this feature in during our experiments, when chaining the output of a fixed topology network to the input of another network creates more complex topologies with a richer variety of results.

Our proposed approach does not depend on a particular implementation of a skill. Therefore if a new algorithm is provided that performs better than an old one, the corresponding node in the evolved FSA can be replaced with the new procedure without impairing the functionalities of the action. This allows for specific AI techniques to be developed and improved independently from the overall system, and then be autonomously combined at a later stage. Reuse of components thus becomes an important advantage of our proposed approach.

[1] We will use the word "skill" to indicate a program the robot is already provided with, while we will use the term "action" to indicate a newly created skill. The term skill should not be confused with its general usage in the Reinforcement Learning literature, where "skill" denotes a motor controller.

One of the main limitations of evolutionary algorithms in robotics is that they require a simulator to be effective. This is due to the requirement that hundreds if not thousands of trials have to be experimented in order to find a solution. In our approach this problem is avoided by concentrating on high-level actions and their effect, rather than the physics of the robot and its interactions with the environment. For example we deal with a grasping action in a high-level way: the result of *grasp(object_i)* is simply "*object_i is in the robot gripper*", while the particular application of grasping in use will take care of the details in the real robot. This is therefore an implementation of the "minimal simulator" approach described in (Jakobi 1997). Noise can be introduced by allowing actions to have a *failure* outcome based on pre-conditions not being met or even a random event.

Given the abstract level of the simulator, deploying an FSA to a real robot is a straightforward process, as we will illustrate with two experiments.

Related Work

The idea of sequencing a robot's behavior in several sub-actions has been explored several times in the past. In (Konidaris and Barto 2009) the authors propose a Reinforcement Learning algorithm with options that allows skills to be discovered and combined into options. Although it is possible that an option is externally provided, their effect on skill acquisition had not been investigated. While the previous work used a simulated problem, in (Konidaris et al. 2011) the same approach is used to have a real robot solve a complex task. The robot discovered new skills by extracting trajectories obtained from the composition of existing ones. Although we share the same goals with this work, the main differences with our proposed approach are in both the techniques we employed and in the level of abstraction of the skills our robot uses.

A related approach to skills building is in (Hart and Grupen 2011). Here the authors propose a framework where control policies are hierarchically combined to generate complex behaviours. Reinforcement learning is used to create the control policies. Policies need however to be expressed in terms of potential functions, which limit their application to more abstract actions like the ones we use in this paper.

Planning-based approaches (Beetz et al. 2010; Stulp and Beetz 2008) include the possibility to optimize the free parameters of two subsequent actions so that the overall execution of the plan is optimal. This optimization happens only when two actions have to be performed together, therefore it applies only to a limited set of scenarios. Plans are constructed on-line by using knowledge extracted from the web. A plan is then executed by invoking elementary program units, which are analogous to our idea of actions. The concept of elementary programs, or actions, is used also in (Kaelbling and Lozano-Pérez 2011) in the context of manipulation and geometric planning.

Our approach borrows ideas from the robotics planning literature, in that we share with it some of the goals outlined in the previous section. An FSA can be seen as the structure that instantiates and carries on the execution of a plan. However while a planning system requires a detailed description of every action's pre-conditions and post-conditions, training an FSA can be performed with data driven simulations. For example during our experiments the success of a grasping action is determined by a neural network trained on real data, instead of having been simulated. To the best of our knowledge this is not feasible in a planning system.

A second major difference between our proposed approach and classic planning is that we allow actions to be adapted to a particular problem by varying their parameters. Moreover, in our proposed approach adapting an action or creating a new one is performed in the same framework, while the same might not be as easy using classical planning approaches.

Techniques

Finite State Automata

Given our particular application, our formulation of the FSA differs from the classic one adopted for example in (Hopcroft, Motwani, and Ullman 1979). We define an FSA as a quadruple (A, O, δ, s), where:

- A is a finite, non-empty set of actions.
- O is a finite set of outcomes. Each outcome j of state a_i is denoted by $out_j[a_i]$.
- $\delta : A \times O \rightarrow A$ is the transition function.
- s is the initial state.

In addition to the above, all the states have a (potentially empty) set of real-valued parameters and they can send data to other states. An example of an FSA is given in Figure 2. The parameters are used to adapt an action to a particular problem.

The main difference with the model illustrated in (Hopcroft, Motwani, and Ullman 1979) is the lack of the input alphabet, or inputs. This means the the transition from one action to another depends on the action outcome only. However data passing replaces and enhances the input function. Although the main features and behavior of our proposed model still closely resembles the classic FSA's one, it is closer to a Turing Machine than to an automaton.

Evolutionary Algorithms

The evolutionary algorithm we used follows the general standard structure described for example in (Bäck 1996). The implementation made use of the library PyEvolve described in (Perone 2009). As the details of the proposed evolutionary algorithm have been described in a previous work (Riano and Mcginnity 2012), here we only summarise the main steps involved.

The genome is represented as a directed graph $G = (V, E)$ with parallel edges, where V is the set of the nodes and E is the set of edges. A node $v_i \in V$ is associated with a single skill type $a_j \in A$, and it has a (possibly empty) list of real-valued parameters $0 \leq \alpha_i \leq 1$ (as in our model of FSA described in the previous section). The meaning

of the parameters is skill-specific. As every skill a_j has a fixed number of outcomes, every node will have a specific number of outgoing edges, each of them representing the specific outcome of an skill. In addition to the nodes and edges, the genome encodes the FSA starting state. There is no restriction on the skill type the node can be associated to, and several nodes can have the same skill type. When using neural networks as a single skill, the weights of the network are co-evolved with the FSA structure, as they represent real-valued parameters of a generic node.

In our proposed evolutionary algorithm we developed both mutation and crossover operators. Each of these operators has to guarantee that every node in the graph has the same number of outgoing edges as the number of outcomes of the associated skill. Moreover every node which is not reachable from the starting node will be removed from the graph. This is to ensure that the evolutionary search is not wasted in areas that do not contribute to the overall fitness function.

Mutation happens both at the graph-level and at the node level. At the graph level nodes can be added or removed, or the starting node can be changed. At the node level, parameters can be mutated by adding a normally distributed random number, an outgoing edge can be re-routed to a different node or the skill associated with a node can be changed.

Crossover is performed between two parents genomes, g_1 and g_2, to create two children graphs c_1 and c_2. Crossover has to ensure that groups of nodes that are potentially working together will not be broken in the process. In this work we assume that nodes whose distance is small (as the path length in the graph) are more likely to be working together than nodes that are far away. This allows for sub-solutions to be developed and maintained by subgraphs and to be preserved over generations if they contribute positively to the fitness function. Potential candidates for crossover are therefore searched using a breadth-first search starting from randomly selected nodes.

Provided Skills

In this work we assume that a robot is equipped with skills that are the product of research conducted in the relative field. These are:

- Detecting and grasping unknown objects using $3D$ information (Hsiao et al. 2010).
- Planning and executing a collision-free trajectory with an artificial arm (Cohen, Chitta, and Likhachev 2010).
- Navigation and obstacle avoidance using an omni-directional base (Marder-Eppstein et al. 2010).

These skills are freely available as part of ROS[2].

Experimental Results

In this section we describe two experiments we conducted to validate our proposed approach. In the first experiment we wanted to develop an FSA for an exploration strategy.

Figure 1: The PR2 robotics platform we used in our experiments.

The robot is placed in an unknown environment and its task is to locate an object. Therefore a new action is built to move the robot so that it will cover as much space as possible.

The second experiment makes use of the exploration action. The robot has to locate the object and then move to a position from where it can grasp it. As the details of the grasping are abstracted by the simulator, we used a previously trained classifier to evaluate if an object is graspable.

As a result of the mutation and crossover the FSA grow or shrink in size. It is not uncommon to observe more than 50 states during an experiment. As many states can be associated with the same skill, we appended numbers to their name to distinguish between them. This is shown in the figures in the next sections.

Experimental Setup

In our experiments we assumed that the robot is located in a squared environment whose edges length varies between 3 and 5 meters. In the environment a table with an object on top is randomly placed. Both the size of the table and the location of the object on the table are randomly selected. The robot is placed so that the table will always be with a positive x coordinate (in the robot's frame of reference), but at the beginning of the experiment the table is not visible by the robot. We assume that the robot can detect objects within $\pi/2$ degrees in front of it and with a maximum range of 1.5 meters. To evaluate the performance of an action the FSA is tested in 300 randomly generated environments. To avoid infinite loops a FSA is allowed only a finite number of transitions before terminating with failure.

The robotic platform we used is a mobile manipulator PR2 robot manufactured by Willow Garage[3] (Figure 1). It is two-armed with an omni-directional driving system. Each arm has 7 degrees of freedom. The torso has an additional degree of freedom as it can move vertically. The PR2 has a variety of sensors, among them a tilting laser mounted in the upper body, two stereo cameras (with narrow and wide field

[2] www.ros.org

[3] http://www.willowgarage.com

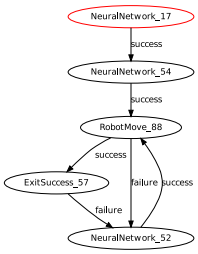

Figure 2: The Explorer FSA. The numbers at the end of state's names are used to uniquely identify them. The red circle is the starting state.

of view) and a laser scanner mounted on the base which is used for mapping and navigation.

Exploration

For the exploration task the robot started with the following skills:

- **ExitSuccess**: The robot uses the stereo camera in front of it to find the $3D$ location of the object. This skill's outcome is either *success* if the object is within the field of view of the robot or *failure*. If successful the FSA terminates.
- **RobotMove**: The robot moves to the x, y location with orientation θ provided as an input message. This movement is collision-aware, i.e. it will fail if the path ends in collision with either the table or the environment boundaries. The movement goal is provided by the neural network described below. This skill's outcome is either *success* if the movement was completed or *failure*.
- **NeuralNetwork**: This skill is a fully recurrent neural network with 3 inputs, 3 outputs and no hidden neurons. The network's input is the output or whatever neural network had previously been activated, or it is random if this is the first time this skill is active. To avoid random fluctuations the network is allowed to settle for 30 steps before generating the output. This skill has only a *success* outcome.

The reward function is 1.0 whenever the robot was able to discover the object, 0.0 otherwise. The reward is averaged over 300 trials with randomly generated environments.

After around 400 generations our algorithm discovered the FSA in Figure 2. We double-tested this action in 5000 randomly generated environments obtaining a success rate of 93%. We believe that many failures were due to environments where the object was not discoverable, e.g. it was in a corner of the room at the very end of the table, thus too far to be seen by the robot.

We tested the same FSA on the real robot in a 4×4 environment. The experiment has been repeated 50 times with a randomly placed table and object. The robot obtained a success rate of 91%. All the failures where due to the neural networks generating poses that were unreachable by the robot.

The behavior of the FSA in Figure 2 can be described as:

1. *NeuralNetwork_17* generates an output triple using a random input.
2. *NeuralNetwork_54* uses the output from *NeuralNetwork_17* to generate another triple.
3. *RobotMove_88* uses the output from *NeuralNetwork_54* to move the robot to a an x, y, θ location. If it fails then control switches to *NeuralNetwork_52*. Otherwise control switches to *ExitSuccess_57*.
4. *ExitSuccess_57* checks if the object is visible. If it is not, control switches to *NeuralNetwork_52*, otherwise the FSA terminates with success.
5. *NeuralNetwork_52* uses any previously generated output as input for its computation. This includes the output it generated 2 or 3 steps ago. Control then switches to *RobotMove_88*.

Although we did not use hidden nodes in the neural networks, *NeuralNetwork_52* has its input chained with its output (via other states), therefore it is mimicking the behavior of an hidden layer.

This experiment showed that the attractors in fully recurrent neural networks are capable of blindly search for an object in a variety of environments.

Move to Pre-Grasp Position

In this second experiment we made use of the exploration action previously developed to have the robot first search for an object then move to a pose from where it is possible to grasp it.

Many robotics applications require the robot to be able to manipulate objects. The approach we use to grasp an object (Hsiao et al. 2010) works only if the object is reachable by the robot. However during our experiments we found that an object is often hard to reach, even if it is close to the robot. This is due to physical constraints of the robot's arms that are not easy to analytically model. Several approaches have been proposed (Stulp, Fedrizzi, and Beetz 2009; Berenson, Kuffner, and Choset 2008) to deal with this problem, and the results prove that this is still a hard benchmark for robotics algorithms.

Representing a grasping scenario requires an accurate simulation of the environment and the robot. As this will slow down the evolutionary process, a fast computation of an action's outcome is required. In this experiment we decided to use a Radial Basis Function neural network (RBFNN) (Poggio and Girosi 1990) to classify whether an object is reachable or not. We collected training and validation data over a full day of experimentations, where the robot was moving randomly with respect to an object on a table. To generate more random positions the robot had to try to push the object instead of grasping it. During our experiments we found that if the robot can push an object then it can grasp it as well. The opposite is not necessarily true, so pushing is a harder task than grasping[4]. For each tentative push we

[4]Pushing an object requires the robot to be able to reach several positions around the object, while grasping requires only one. Therefore if the robot can push an object, it can grasp it as well.

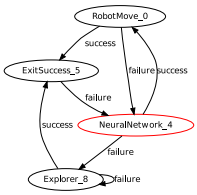

Figure 3: Pre-Grasp The numbers at the end of state's names are used to uniquely identify them. The red circle is the starting state.

recorded the position of the object in the robot's frame of reference, the robot's torso height and which arm was used to push the object, or if the object is unreachable. Overall we collected 872 data points: of these 500 points were used for training and 372 for validation. We then trained a RBFNN with 5 inputs, the x, y, z position of the object in the robot frame of reference, the angle θ between the robot and the object on the $x - y$ plane and the robot's torso height h. The RBFNN classifies the input into three classes, 0 if the object is unreachable, 1 if it is reachable with the left arm and 2 if it is reachable with the right arm. After training the network had a performance of 92% correct classifications over the validation data set. We have thus obtained a fast way to determine if an object is reachable from a given position: this can be used in a simulator without resorting to a computationally expensive analysis.

The robot started with the following skills:

- **ExitSuccess**: The robot uses RBFNN to determine if the object is graspable.This skill's outcome is either *success* or *failure*. If successful the FSA terminates.
- **RobotMove**: This skill is analogous to the one presented in the previous experiment, with the difference that it will also take an input representing the robot torso height.
- **NeuralNetwork**: This skill is a fully recurrent neural network with 6 inputs, 4 outputs and 3 hidden neurons. The network's input is the x, y location of the object, the x, y location of the table and the width, length of the table. The network's outputs represent the x, y, θ location to move the robot to plus the robot torso height. This skill has a *failure* outcome if the object has not been discovered, or *success* otherwise.
- **Explorer**: This skill is the FSA developed in the previous section. It has both the *success* and *failure* outcomes.

The reward function is either 1.0 if the robot moves to a position where it can grasp the object, or 0 otherwise.

This experiment proved to be more difficult than the previous one, as only a few positions are good candidates to grasp an object. It took our algorithm 2300 generations to achieve a success rate of 88%. The resulting FSA is shown in Figure 3. In order to appreciate the complexity of the overall system we show in Figure 4 the complete FSA with the *Explorer* skill expanded.

We tested the PR2 in 50 different scenarios, obtaining a success rate of 77%. A third of the failures was due to the explorer not being able to find the object. A second cause of

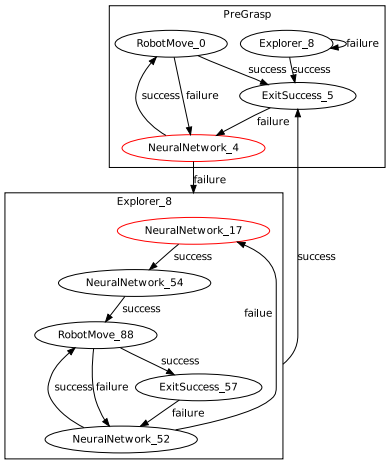

Figure 4: The Pre-Grasp FSA with the explorer state expanded. Confront with Figure 2 and Figure fig:pre grasp automaton.

failures was due to the errors induced by the approximation of the pre-grasp poses by the RBFNN, that predicted poses from which the robot could not reach the object.

The structure of the FSA in Figure 3 shows that the the topology is close to be optimal. The explorer is used only at the beginning when the neural network fails (the object has not been discovered yet), but after that it is not activated any more. The control switches to the upper part of the FSA where the usual cycle of generate a position - move - check if it is successful happens. The main duty is performed by *NeuralNetwork_4* which does not use any chaining, as the number of inputs differs from the number of outputs. Its role is therefore to generate candidate positions from where to approach the object.

Discussion and Conclusions

The goal of this work is to illustrate an algorithm where externally provided skills are automatically combined to generate complex actions. A researcher willing to integrate the results obtained in other fields with his/her own work only has to write a simple simulator, define a reward function and let the evolutionary algorithm find a solution to the problem. Moreover, by using generic computational devices like recurrent neural networks, our proposed algorithm is able to fill the gaps where the provided skills are not sufficient. This is the main distinction between our approach and other planning-based approaches.

A number of questions arises from this work. As the number of skills the robot is equipped with increases, choosing the right ones for a particular problem becomes crucial. In the experiments above we provided the evolutionary algorithm the right set of actions to work with. Although this choice has certainly helped to obtain results in a shorter number of generations, we do not believe this has a significant influence over our results. Our proposed algorithms deals with genomes whose size increases and decreases as the evolutionary process unfolds over time.

For example the node **RobotMove_88** suggests that at one point during evolution there were at least 88 nodes in the FSA. Many of these nodes have been found to decrease the genome's fitness and they have been evolved out. Therefore, as discussed above, the limited number of transitions an FSA is allowed during one execution acts as a force that pushes the evolutionary algorithm to look only for the nodes that positively contribute to the fitness. This in turn will automatically remove actions that do not contribute to solve a problem.

A major issue with this work is to write a correct simulator for our experiment. Many skills can change the robot state in unpredictable ways, rendering the results obtained in simulation not usable on the real robot. For example, although the result of grasping is either the object is in the robot gripper or it is not, there are other not modeled results like the changed position of the robot's end effectors. This changes could in turn influence other actions, e.g. the robot can not safely move if the arms will collide with the environment. Obviously these effects could be modeled by a more complex simulator, but this will both increase the efforts of writing it and it will increase the required computational time, reducing some of the benefits of our proposed approach. We are currently working on automated ways to learn the effect of the actions, so that the robot will have a good estimate of the world's state changes after having performed an action. This will be an extension to the RBFNN for pre-grasping poses classification we used in the second experiment.

Another major issue is that, with the increasing complexity of the actions, co-evolving an FSA's topology with neural network parameters becomes a huge task. This in turn might hinder the scalability of our approach to solve complex tasks. We addressed this problem by using a "scaffolding" approach (Bongard 2008), i.e. presenting the robot with problems of increasing complexity and the re-using the results obtained before. This requires a big effort to correctly segment the task into feasible steps. We are currently investigating a double-step evolutionary approach, where first an FSA topology is found, then neural networks are generated to fill the missing skills. This will factorize the solution space rendering the search for a solution feasible.

References

Bäck, T. 1996. *Evolutionary algorithms in theory and practice*. Oxford University Press New York.

Beer, R. D., and Gallagher, J. C. 1992. Evolving Dynamical Neural Networks for Adaptive Behavior. *Adaptive Behavior* 1(1):91–122.

Beetz, M.; Jain, D.; Mösenlechner, L.; and Tenorth, M. 2010. Towards performing everyday manipulation activities. *Robotics and Autonomous Systems* 58(9):1085–1095.

Berenson, D.; Kuffner, J.; and Choset, H. 2008. An optimization approach to planning for mobile manipulation. *2008 IEEE International Conference on Robotics and Automation* 1187–1192.

Bongard, J. 2008. Behavior chaining: Incremental behavior integration for evolutionary robotics. *Artificial Life* 11:64.

Cohen, B.; Chitta, S.; and Likhachev, M. 2010. Search-based planning for manipulation with motion primitives. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2902–2908. IEEE.

Hart, S., and Grupen, R. 2011. Learning Generalizable Control Programs. *IEEE Transactions on Autonomous Mental Development* 3(3):216–231.

Hopcroft, J.; Motwani, R.; and Ullman, J. 1979. *Introduction to automata theory, languages, and computation*. Addison-wesley Reading, MA.

Hsiao, K.; Chitta, S.; Ciocarlie, M.; and Jones, E. 2010. Contact-reactive grasping of objects with partial shape information. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 1228–1235. IEEE.

Jakobi, N. 1997. Half-baked, ad-hoc and noisy: Minimal simulations for evolutionary robotics. In *Fourth European Conference on Artificial Life*, 348–357. The MIT Press.

Kaelbling, L., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *IEEE Conference on Robotics and Automation (ICRA)*.

Konidaris, G., and Barto, A. 2009. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in Neural Information Processing Systems* 22:1015–1023.

Konidaris, G.; Kuindersma, S.; Grupen, R.; and Barto, A. 2011. Autonomous Skill Acquisition on a Mobile Manipulator. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*.

Marder-Eppstein, E.; Berger, E.; Foote, T.; Gerkey, B. P.; and Konolige, K. 2010. The Office Marathon: Robust Navigation in an Indoor Office Environment. In *International Conference on Robotics and Automation*.

Perone, C. S. 2009. Pyevolve: a Python open-source framework for genetic algorithms. *SIGEVOlution* 4(1):12–20.

Poggio, T., and Girosi, F. 1990. Networks for approximation and learning. *Proceedings of the IEEE* 78(9):1481–1497.

Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; and Ng, A. 2009. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Riano, L., and Mcginnity, T. M. 2012. Automatically Composing and Parameterizing Skills by Evolving Finite State Automata. *Robotics and Autonomous Systems* (In Press, available at http://dx.doi.org/10.1016/j.robot.2012.01.002).

Stulp, F., and Beetz, M. 2008. Refining the execution of abstract actions with learned action models. *Journal of Artificial Intelligence Research* 32(1):487–523.

Stulp, F.; Fedrizzi, A.; and Beetz, M. 2009. Action-related place-based mobile manipulation. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems* 3115–3120.