MULTI-RESOLUTION MODEL PLUS CORRECTION PARADIGM FOR TASK AND SKILL REFINEMENT ON AUTONOMOUS ROBOTS

by

Çetin Meriçli

B.S. in Computer Engineering, Marmara University, 2002

M.S. in Computer Engineering, Boğaziçi University, 2005

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2011

MULTI-RESOLUTION MODEL PLUS CORRECTION PARADIGM FOR TASK AND SKILL REFINEMENT ON AUTONOMOUS ROBOTS

APPROVED BY:

Prof. H. Levent Akın (Thesis Co-supervisor)	.....
Prof. Manuela Veloso (Thesis Co-supervisor)	.....
Prof. Ethem Alpaydın	.....
Asst. Prof. Hatice Köse Bağcı	.....
Assoc. Prof. Yağmur Denizhan	.....

DATE OF APPROVAL: 15.06.2011

ACKNOWLEDGEMENTS

I would like to thank many people who have supported me along the way and who have helped facilitate this work in different ways.

First and foremost, I would like to thank my advisors. I would like to thank H. Levent Akın for his way of broad thinking, and for always encouraging me to dare to think different. He has played a key role in my transition from a fresh graduate into a scientist over the past years. I would like to thank Manuela Veloso for her guidance and passion. This thesis would not have been possible without Manuela being a constant source of inspiration. Her supernatural ability to say "No" has helped enormously to keep me focused and on track through the inevitable research setbacks.

My sincere thanks go to my thesis committee: Ethem Alpaydın, Yağmur Denizhan, and Hatice Köse Bağcı. I have learned almost all I know about machine learning and experiment design from Prof. Alpaydın. Our conversations with Prof. Denizhan had always been very inspiring and sparkling with her broad knowledge, and her unconventional way of tackling research problems. Starting from the beginning of my graduate life, Prof. Bağcı had always been a good role model, and had influenced me in many ways along the prickly path of becoming a researcher.

Parts of this thesis study were supported by The Scientific and Technological Research Council of Turkey Programme 2214 and the Turkish State Planning Organization (DPT) under the TAM Project, number 2007K120610.

I am deeply grateful to the entire CmpE family for their support, friendship, camaraderie, and their unique notion of having fun. In particular, I would like to thank Itır Karaç, Aslı Uyar, Derya Çavdar, Gaye Genç, Yunus Emre Kara, Barış Kurt, Suzan Bayhan, Didem Gözüpek, and Gül Çalıklı for their invaluable support throughout the years.

I would like to thank the members of ÇOT! (a.k.a the Artistic Breakfast Club): Tekin Meriçli, Onur Dikmen, Yunus Durmuş, Ahmet Yıldırım, and İsmail Arı for the best times of my short-lived musical career, and for the best artistic breakfast sessions on the planet.

I would like to thank all the members of AILab, particularly to Serhan Danış, Akın Günay, Abuzer Yakaryılmaz, Reyhan Aydoğan, and Başak Aydemir for their invaluable support. Extra credit goes to Başak for her enormous help with the formatting issues of the manuscript.

I would like to thank the members of the Cerberus team: Tekin Meriçli, Kemal Kaplan, Buluç Çelik, Can Kavaklıoğlu, Barış Gökçe, and Ergin Özkucur for their friendship and for collectively contributing to the Cerberus code base that enabled the experiments of this thesis.

A big thank you goes to all the members of CORAL group for their invaluable support and constructive criticism throughout my thesis study. Somchaya Liemhetcharat, Junyun Tay, and Brian Coltin receive special thanks for their contributions to the CMurfs code base that played a key role in most of the experiments in this thesis, their friendship, and the constant mental pressure they put on me that kept me on track towards the end.

I would like to thank Joydeep Biswas for the mind-sizzling conversations and for his good musical taste, and Stephanie Rosenthal for her support, friendship, and proof-reading my manuscripts without a single complaint. I am grateful to Prof. Changjiu Zhou and Prof. Rodrigo Ventura for all their support and constructive feedback on my thesis defense presentation. Thank you all very much.

I would not be at where I am today if it was not for my family. I want to thank my mom Birgül and my dad İsmet for their endless love, and for having always supported all my decisions in life without questioning me. My brother Tekin Meriçli receives my heartful thanks for being my oldest collaborator on many subjects and

for his endless support. His frightful but constructive criticism has set the bar very high for me to meet his expectations.

Finally, my deepest thanks go to Bahar Karaoğlu for bearing with me all these years, and for her constant care and support that kept me sane throughout all the ups and downs of graduate school. Words cannot capture my gratitude.

ABSTRACT

MULTI-RESOLUTION MODEL PLUS CORRECTION PARADIGM FOR TASK AND SKILL REFINEMENT ON AUTONOMOUS ROBOTS

Robots need to be taught what type of tasks or skills they are expected to perform, and how to perform those particular tasks or skills. However, there is no universally accepted single approach for transferring the task and skill knowledge to a robot. Among several popular approaches, the most widely adopted method for transferring the task or skill knowledge to the robot is to develop an algorithm for performing the task or skill in question. Such a development requires a model of the system to be available. Moreover, despite that it usually is easier to develop a simple algorithm to handle trivial cases, it becomes a time consuming process to keep refining the algorithm by modifying the underlying model to handle more complex situations.

Learning from Demonstration (LfD) is another popular approach for transferring the task and skill knowledge to the robot. Instead of explicit programming, a teacher demonstrates the robot how to perform the task or skill and the robot records the demonstrated action together with the perceived state of the system at the time of demonstration. An execution policy is then derived out of the recorded demonstration data for reproducing the task or skill. Depending on the complexity of the task or skill in question and the robotic platform to be used, providing sufficient number of examples in order to be able to extract a generalized execution policy can be a very time consuming process.

This thesis contributes a novel complementary corrective demonstration paradigm called Model Plus Correction (M+C) for task and skill refinement on autonomous robots. The M+C approach strikes a balance between model-based and data-driven methods by combining them in a complementary manner. We assume the availability of an algorithm capable of performing the task or skill in question with limited success in terms of performance. Our approach utilizes a human teacher who observes the partially successful execution of the task, and corrects the action of the robot when the default algorithm is unable to select an appropriate action to be executed. The collected demonstration data stamped with the state of the system at the time of demonstration is then used to augment the default algorithm by modifying the action computed by the algorithm according to a correction reuse function, and the state of the system.

This thesis also introduces an algorithm for using the same complementary corrective demonstration approach at multiple detail resolutions. The Multi-Resolution Model Plus Correction (MRM+C) algorithm assumes that a set of detail levels are defined with different state and action representations together with a different model-based controller for each detail level are available at hand. The teacher provides demonstration for which detail resolution to use at a particular state of the system in addition to delivering corrective demonstration for the controller associated with the current detail resolution. Having multiple detail resolutions with different complexities allows the system to use more detailed state and action representations and more complex model-based controllers only when needed. Using a less detailed state and action representation with a simpler controller makes it possible to cover the solution space at a lower computational cost and using fewer number of demonstrations. The learned detail resolution selection policy favors the least detailed resolution by default and switches to a more detailed resolution if commanded to do so in a similar state before.

We present experiment results where the M+C approach is first applied to a complex biped walk stability improvement problem as an example to the skill refine-

ment, and to a ball dribbling problem in a robot soccer environment as an example to the task refinement. We also present experiment results where the MRM+C approach is applied to a humanoid obstacle avoidance task on a robot soccer field. Finally, we present an experimental analysis of the proposed algorithms in terms of their robustness against uncertainty and the cost analysis of using multiple detail resolutions over using a single detail resolution in a simulated version of the obstacle avoidance task.

ÖZET

ÖZERK ROBOTLAR ÜZERİNDE GÖREV VE BECERİ İYİLEŞTİRME İÇİN ÇOKLU-ÇÖZÜNÜRLÜKLÜ MODEL ARTI DÜZELTME PARADİGMASI

Robotlar kendilerinden hangi görev ve becerileri icra etmeleri beklendiği ve bu görev ve becerileri nasıl gerçekleştirecekleri konusunda bilgilendirmeye ihtiyaç duyarlar. Bu bilgilendirmenin nasıl yapılacağı konusunda üzerinde anlaşılmış evrensel bir metod henüz bulunmamakla birlikte popüler olarak kullanılan metodlar arasında en yaygın olanı ilgili görev ya da beceriyi gerçekleştirebilecek bir algoritmanın geliştirilmesidir. Böyle bir algoritma geliştirmek, sistemin bir modelinin bulunmasını gerektirir. Dahası, basit durumlar için görevi yerine getirecek bir algoritma geliştirmek kolay olsa da, algoritmanın varsaydığı modeli daha karmaşık durumları da kapsayabilecek şekilde güncellemeye devam etmek giderek daha çok zaman alan bir sürece dönüşmektedir.

Gösterimden öğrenme (GÖ), robotu programlamadan görev ve beceri bilgisini aktarmak için kullanılan bir yöntemdir. Bu yöntemde robotu programlamak yerine bir öğretmen görev ya da becerinin nasıl icra edileceğini robota gösterir ve robot bu gösterim-leri sistemin o anki durumu ile birlikte kaydeder. Bu işlemi takiben gösterilen görev ya da beceriyi tekrarlayabilmek için kaydedilen veri üzerinden bir icra politikası oluşturulur. Söz konusu görev ya da becerinin karmaşıklığına bağlı olarak düzgün genelleştirilmiş bir icra politikası oluşturabilmek için gereken sayıda gösterimi robota sunmak çok zaman alıcı bir süreç olabilir.

Bu tez, yeni bir tamamlayıcı düzeltici gösterim anlayışı olan Model Artı Düzeltme (M+D) paradigmasını bir görev ve beceri başarım iyileştirme yöntemi olarak sunmaktadır. M+D yöntemi model-tabanlı ve veri-güdümlü yaklaşımlar arasında bir denge kurarak bu yöntemleri birbirlerini tamamlayacak şekilde birleştirmektedir. Bu yöntemde, söz konusu görev ya da beceriyi sınırlı bir başarım ile gerçekleştirebilen bir algoritmanın var olduğunu varsayıyoruz. Yaklaşımımız, söz konusu görevi mevcut algoritma ile icra eden robotun eylemini algoritmanın yanlış bir karar alması halinde devreye girerek düzeltecek bir insan öğretmen kullanmaktadır. Sistemin o anki durumu ile damgalanarak saklanan gösterim bilgisi daha sonra bir düzeltim kullanımı fonksiyonu ve sistem durumuna gore varsayılan algoritmanın hesapladığı eylemin uygun bir şekilde değiştirilmesinde kullanılır.

Bu tez ayrıca aynı tamamlayıcı düzeltici gösterim yaklaşımının birden fazla detay çözünürlüğünde kullanılabilmesi için de bir algoritma sunmaktadır. Çoklu-Çözünürlüklü Model Artı Düzeltme (ÇÇM+D) algoritması her biri ayrı detayda durum ve eylem tanımlarına ve değişik karmaşıklıkta varsayılan algoritmalara sahip bir dizi detay çözünür-lüğü tanımlanmış olduğunu varsayar. Daha az detaylı bir durum ve eylem tanımı ve daha az karmaşık bir algoritmanın kullanılması, durum uzayının daha büyük bir kısmının daha az hesaplama maliyeti ile kapsanmasını sağlar. Gosterim sırasında öğretmen robota o anki detay çözünürlüğünde düzeltici gösterim yapmasının yanında hangi durumda hangi detay çözünürlüğünün kullanılması gerektiği konusunda da gösterimde bulunur. Farklı karmaşıklık seviyelerine sahip birden çok detay çözünürlüğünün bulunması, sistemin daha detaylı durum ve eylem tanımları ve daha karmaşık algoritmaları ancak gerektiğinde kullanabilmesini sağlar. Öğrenilen detay seçim politikası ön tanımlı olarak en düşük detay çözünürlüğünü kullanmaya çalışır ve daha yüksek bir detay çözünürlüğüne ancak daha önce benzer bir durumda öğretmen tarafından detay çözünürlüğünü arttırma komutu verilmişse geçer.

Sunduğumuz deney sonuçları M+D yönteminin önce beceri iyileştirmeye bir örnek olarak karmaşık bir iki ayaklı yürüme eyleminin dengesini iyileştirme problemine uygulanmasının, sonra da görev iyileştirmeye bir örnek olarak robot futbolu ortamında tanımlan-mış bir top sürme problemine uygulanmasının sonuçlarını içeriyor. Bunlara ek olarak, ÇÇM+D yönteminin bir insansı robotun bir robot futbolu sahasında engel savuşturması problemine uygulanması ile ilgili deney sonuçları da sunuyoruz. Son olarak, önerilen algoritmaların ortamdaki belirsizlikten ne kadar etkilendikleri ve birden çok detay çözünürlüğü kullanmanın tek bir çözünürlük kullanmaya göre hesaplamasal maliyet karşılaştırmaları üzerine bir deneysel analizi insansı robot engel savuşturması probleminin benzetim ortamında modellenmiş bir halini kullanarak sunuyoruz.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CBA	Confidence Based Autonomy
CBR	Case-Based Reasoning
CD	Corrective Demonstration
CPG	Central Pattern Generator
LfD	Learning from Demonstration
LWR	Locally Weighted Regression
M+C	Model Plus Correction
MDP	Markov Decision Process
MoG	Mixture of Gaussians
MRM+C	Multi-Resolution Model Plus Correction
MRTE	Multi-Resolution Task Execution
RL	Reinforcement Learning

1. INTRODUCTION

Transferring the knowledge of how to perform a certain task or skill to a robotic platform remains a challenging problem in robotics research with an increasing importance as robots start emerging from research laboratories into everyday life and interacting with ordinary people who are not robotics experts. Robots observe their environments through a set of sensors, and perform actions using their actuators. Performing a task or skill requires a mapping function from the observed state of the robot to the proper actions according to the task or skill definition. This mapping is called a *policy*.

A widely adopted method for transferring task knowledge to a robot for obtaining a policy is to develop an algorithm using a model (a set of assumptions about the system) for performing the task or skill, when such a model is available. Although it is usually relatively easier to develop an algorithm that can handle the trivial cases, handling more complex situations often requires substantial modifications on the algorithm and these modifications require the credit for erroneous execution to be assigned properly to the underlying model. Therefore, it becomes a tedious and time consuming process to ameliorate the controller and the underlying model as the complexity of the cases the robot is facing increases. Moreover, for some problems, the refined algorithm might still fail to cover all the cases no matter how many refinement iterations have been performed.

The Learning from Demonstration (LfD) paradigm is a data-driven approach that utilizes supervised learning for transferring task or skill knowledge to an autonomous robot without explicitly programming it. Instead of developing an algorithm for performing a task or skill, LfD methods make use of a teacher who demonstrates the robot how to perform the task or skill while the robot observes the demonstrations and synchronously records the demonstrated actions along with the perceived state of the system. The robot then uses the stored state-action pairs to

derive an execution policy for reproducing the demonstrated task or skill. Moreover, since the LfD approaches do not require the robot to be programmed explicitly, they are very suitable for cases where the task knowledge is available through a user who is an expert in the task domain but not in robotics.

Providing a way for humans to transfer task and skill knowledge to robots via natural interactions, the LfD approaches are also suitable for problems, where a complete analytical model for the task or skill is not available but a human teacher can tell which action to take in a particular situation. It is impractical to expect the teacher to give demonstrations for each and every possible case; therefore, the policy percolated from the gathered demonstration data should be able to generalize the received demonstrations to cover the states of which an implicit demonstration example has not been provided by the teacher. However, providing sufficient number of examples for good generalization is a very time consuming process when working with robots with highly complex body configurations; such as humanoids, and for sophisticated tasks with very high dimensional state and action spaces.

This thesis presents a novel paradigm for task and skill performance improvement by combining the algorithm-based methods and the learning from demonstration approach in a complementary manner to take advantage of the strong parts of both sides. We assume an algorithm for performing the task or skill in question is available but has limited success rate in terms of some domain dependent performance metrics. The human demonstrator observes the robot as it performs the task or skill using the available algorithm, and provides corrective feedback only when the robot makes a mistake. The received feedback data are stored by the robot along with the observed state of the system. During autonomous execution, the robot executes the action computed by the default algorithm unless there is a corrective demonstration example in the database which is given by the teacher in a similar situation. This idea of keeping the default algorithm as the primary source of the action and using the demonstration data only to make exceptions as needed reduces the number of demonstration examples required, and leads to a rapid performance improvement.

This thesis also introduces the concept of multiple detail resolutions for giving demonstrations at different detail levels, depending on the complexity of the situation the robot is facing. Assuming that:

- the underlying algorithms become more expensive in terms of computational power as their complexity increases,
- the more detailed the state and action representations get, the more demonstration examples it requires to have a good generalization in covering the state-action space, and
- not all of the cases for a task or skill require state and action representations of the same detail resolution, and algorithms to perform the task of equal complexities

our algorithm builds upon the complementary corrective demonstration approach combining an available algorithm with human demonstration and uses multiple instances of the complementary corrective demonstration system running at different detail resolutions in terms of state representation, action definition, and the complexity of the default controller. We introduce the concept of an *arbitrator* component to select which detail level to use in a particular situation using a selection function that maps the current observed state of the system to a certain detail resolution. In this multi-resolution scenario, the teacher provides corrective demonstration examples for the different detail resolutions, and he or she also provides demonstration for which detail level to use in a particular state.

Finally, this thesis contributes a formalization for the introduced *platform-independent* and *domain-independent* task and skill refinement frameworks for single and multiple detail resolutions. We present detailed experiment results for all algorithms. We present results from the application of the complementary corrective demonstration approach to a complex biped walk stability improvement problem, as an example to the skill refinement, application to a ball dribbling problem in a robot soccer environment as an example to the task refinement, and application of

the multi-resolution complementary corrective demonstration to a humanoid obstacle avoidance task. We also present a thorough analysis of both the single-resolution and multi-resolution complementary corrective demonstration approaches in terms of their robustness against the uncertainty in perception and action, and in terms of the execution cost.

1.1. Approach

This thesis seeks to answer the following questions:

- How can an existing algorithm for performing a task or a skill be augmented with corrective human demonstration to improve the system performance?
- How can the combination of the algorithm and human demonstration be extended in such a way to allow the demonstrator to correct the actions of the robot at multiple detail resolutions with different state and action representations and default algorithms or varying complexity?

We present our approach in answering the thesis questions by breaking them into several sub-questions.

1.1.1. Augmenting an Algorithm with Corrective Human Demonstration

1.1.1.1. How Can We Collect Corrective Human Demonstration?. Differing from the classical LfD methods, in our approach, the demonstrator does not start teaching the task or skill from scratch. The demonstration process comprises of the robot *performing* the task or skill using the default algorithm and the teacher observing the execution and stepping in to correct the robot only when the default algorithm takes a wrong action. We utilize the *Learning from Experience* method, where the teacher makes the robot perform an action by providing a demonstration using the action definitions of the robot via a custom interface. By doing so, it is guaranteed that all demonstration data coming from the teacher maps properly to the action capa-

bilities of the robot. Therefore, the so called *correspondence problem* does not hold for our approach. The corrective demonstration can be in the form of *modifications* over the action computed by the default algorithm, or *substitutions* for replacing the computed action with the demonstrated action. A distinguishing property of our approach over traditional LfD methods is interleaving execution and demonstration. Instead of waiting the robot to perform a complete execution of the task, the teacher gets involved at any time during the execution to correct the robot. We present examples for this execution-demonstration interleaving at its extreme in Chapter 4, where the demonstration occurs in real-time while the robot is performing a very complex biped walk motion, and in Chapter 5, where the demonstration is still interleaved with the execution, but is required only when the robot is in a state where it needs to compute the next action to be executed.

1.1.1.2. How Can the Robot Reuse the Corrections Together with the Algorithm?.

The demonstration process in our approach builds a demonstration database as in most other LfD approaches. The database consists of demonstration examples in the form of $< state, action >$ pairs where the *action* is the corrective demonstration given by the teacher, and *state* is the observed state of the system at the time of demonstration. The reuse of the collected demonstration data takes place during autonomous execution and according to a *correction reuse function*. The correction reuse function computes the final action to be executed by the robot using the actions coming from the default algorithm and the correction database, and the current observed state of the system. We present two types of correction reuse methods: modifying the default action with using the corrective demonstration action (Chapter 4), and replacing the default action with the demonstrated action (Chapter 5, Chapter 6). One of the important properties of the general complementary corrective demonstration idea is to use the action computed by the default algorithm as much as possible. Therefore, the collected demonstration data is very sparse and does not extend to cover the entire state-action space. In order to compute an action using the demonstration database, one needs a generalization feature. The proposed framework does not impose a constraint on the generalization method to be used. We present three different

generalization methods: fitting multiple normal distributions on the demonstration data and using the mean value of the distribution associated with the current state as the correction value (Chapter 4), using Locally Weighted Regression (Chapter 4), and through a domain dependent state similarity function to find the demonstration example received in the most similar state to the current observed state, and then using the found demonstration sample as the correction sample, is the similarity value is above a certain threshold (Chapter 5, Chapter 6).

1.1.2. Multi-Resolution Complementary Corrective Demonstration

1.1.2.1. How Can We Reduce the Task Execution Cost?. Handling complex cases for complicated tasks require sophisticated algorithms with complex underlying models and very detailed state and action representations. Using the most detailed state and action representations result in very high dimensional state-action spaces and running sophisticated algorithms for acting properly could be very expensive in terms of required computational power. However, not all of the cases for a task or skill require such detailed representation and complex algorithms. Moreover, it requires the teacher to provide demonstration at this highest detail level, covering a small part of the state-action space, and results in substantial increase in the required number of demonstrations to cover the state-action space, hence the demand for teacher attention. We introduce the concept of multiple detail resolutions in Chapter 6 for tackling this problem by allowing multiple instantiations of the complementary corrective demonstration system with state and action definitions at different detail resolutions, and with default algorithms of varying complexity. Using the proposed approach, the robot uses a less detailed state and action representation and simpler default algorithms as much of the time as possible, and the teacher gets to provide detailed demonstrations only when the lower detail resolutions fall short to compute a proper action.

1.1.2.2. How Can We Provide Demonstration at Different Detail Resolutions?. Obtaining the demonstration data for multiple detail resolutions is done in the same way

we collect the demonstration data for the single resolution since the multi-resolution system is a combination of several single-resolution systems and only one of those systems can be active at a time. Therefore, practically the teacher interacts with only a single detail resolution at a time. During the demonstration, the teacher either provides a corrective action, or makes the robot switch to a higher detail resolution if he or she thinks the current situation can not be handled at the current detail resolution. The received correction actions for each detail resolution are stored in a separate demonstration database, and the received detail resolution change commands so called *elaborate commands* are also stored in a separate *elaboration* database. Each detail resolution uses its own state representation for storing the received demonstration examples while the most detailed state representation is used for the elaboration database. Chapter 6 covers the collection of demonstration data for multiple detail resolutions in depth.

1.1.2.3. How Can the Robot Reuse the Multi-Resolution Corrective Demonstration?.

The reuse of the collected demonstration data at different detail resolutions, and the demonstration data for changing the detail levels are reused in a similar manner with the reuse presented in Chapter 5. During the autonomous execution of the robot, each time the robot needs to compute an action, it first looks for a demonstration example for the current detail resolution. The robot always starts looking from the lowest detail resolution with the least amount of detail. If a demonstration example received in a state that has a similarity value between that state and the current state of the system larger than a certain threshold, the corrective demonstration action is executed by the robot. If no such demonstrations are found, the robot looks for a demonstration example in the elaboration database for changing the detail level into a higher resolution. A separate state similarity function is used for different state representations of each detail resolution while the elaboration demonstration sharing the same state similarity function with the highest detail resolution. We thoroughly describe the reuse process in the multi-resolution corrective demonstration system in Chapter 6.

1.1.3. Evaluation

We used several real world and simulated domains for the experimental evaluation of the algorithms presented in this thesis. The robot soccer domain, which is the main application area that the evaluation domains have stemmed from is described and an overview of the software components for playing soccer in Appendix A. We give complete detailed definition of each evaluation domain in the relevant chapters.

1.2. Contributions

This thesis makes several major contributions to task and skill knowledge transfer through human demonstration. The contributions of this thesis are as follows:

- Model Plus Correction (M+C), a platform and domain independent hybrid paradigm combining an existing algorithm for performing a task or skill with corrective human demonstration. The M+C paradigm has three components. The *Model* component is an algorithm for performing the task or skill which is developed based on a model, or a set of assumptions about the system. The *Correction* component consists of corrective demonstration examples provided by a teacher when the robot computes an erroneous action using the Model component and the observed state of the system at demonstration time. The Correction component also features a generalization method, usually a classifier or a regressor trained on the collected demonstration data. The *Correction Reuse* component (the "Plus" part) decides how to combine the actions computed by the Model and the Correction components, or how to decide which one of the actions to execute based on the similarity of the current state of the system and the states associated with the received demonstration examples.
- Multi-Resolution Model Plus Correction (MRM+C), an extension to the M+C paradigm comprising multiple M+C instances and an arbitrator for selecting among those instances to become active. MRM+C paradigm that allows the teacher to correct the actions of the robot at different detail resolutions and to

teach the robot which detail resolution to use in a particular state.

- A formalization of the M+C and MRM+C approaches, describing how they extend the traditional LfD approach.
- A thorough experimental analysis of both the M+C and MRM+C approaches to evaluate their robustness against the uncertainty in the environment and utility analysis of using multiple detail resolutions instead of using the highest available detail resolution all the time.

1.3. Reader's Guide to the Thesis

The thesis is organized as follows:

- Chapter 1 - Introduction: We provide an introduction to the thesis. We outline the research questions that this thesis study seeks to answer. We summarize our major contributions, and we provide a document outline for the thesis.
- Chapter 2 - Background: We present a description of the LfD approach along with a formal model, and we outline some relevant work in the literature while pointing out the similarities and differences of our approach with respect to the existing work.
- Chapter 3 - Model Plus Correction (M+C) Paradigm: We introduce the key contributions of the thesis: Model Plus Correction (M+C) and Multi-resolution M+C (MRM+C) complementary corrective demonstration paradigms. We present formal models for both the M+C and MRM+C paradigms, and we describe each component of the M+C and MRM+C paradigms in detail.
- Chapter 4 - Skill Refinement Using M+C: We present a real world application of the M+C on a complex biped walking domain to improve the stability of an existing walk algorithm. We present a novel interface for providing real-time corrective demonstration feedback to the robot without physical contact.
- Chapter 5 - Task Refinement Using M+C: We present an application of the M+C to a complex ball dribbling task in robot soccer environment. We evaluate the performance of the proposed approach using official RoboCup Standard

Platform League rules.

- Chapter 6 - Task Refinement Using MRM+C: We present Multi-Resolution Model Plus Correction (MRM+C), an extension over the M+C approach by introducing the concept of multiple instances of M+C running at different detail resolutions. We evaluate the performance of the MRM+C approach in a humanoid obstacle avoidance domain and we demonstrate the effectiveness of the MRM+C algorithm against the single detail resolution approach.
- Chapter 7 - Experimental Analysis: We present extensive experimental evaluation of the M+C and MRM+C algorithms in a simulated version of the humanoid obstacle avoidance domain. We evaluate the robustness of both the M+C and MRM+C approaches against the level of uncertainty in the environment and we demonstrate the effectiveness of the MRM+C algorithm over M+C algorithm in terms of execution cost.
- Chapter 8 - Conclusion and Future Work: We conclude the thesis with a summary of the major contributions and with a discussion of the possible promising directions for future work.

2. BACKGROUND

In this chapter, we first present a description of the Learning from Demonstration approach along with a formal model. We then describe Corrective Demonstration, a form of LfD that the contributions of this thesis are based on. We present a brief description of the Advice Operators Policy Improvement (A-OPI) approach that is used in some of the applications of the methods this thesis introduces. In the last section of the chapter, we outline some relevant work in the literature, comparing our approach to the existing work.

2.1. Learning from Demonstration

As briefly introduced in Chapter 1, the Learning from Demonstration (LfD) paradigm is a supervised learning approach for transferring task or skill knowledge to an autonomous robot by means of the demonstrations of the task or skill execution. A human teacher provides the demonstrations for the task or skill. Depending on the implementation, the demonstrations can occur in two main categories:

- *Learning from observation:* In this category, the teacher performs the task or skill and the robot acquires the demonstration examples passively through observing the teacher. This type of demonstration requires the robot to be able to identify and map the teacher's actions to its own action set. This problem is also known as *the correspondence problem.*
- *Learning from experience:* In this category, the teacher makes the robot execute the task or skill by means of either manipulating the body parts of the robot or through instructing the robot using its own action set.

We define the learning from demonstration problem formally as a tuple $< S, A, \pi_{demo} >$. The world consists of states S, and A is the set of actions the robot can take. Transitions between states are defined with a probabilistic transition func-

tion $T(s'|s,a) : S \times A \times S \rightarrow [0,1]$. The state is not fully observable; instead, the robot has access to an observed state Z with the mapping $M : S \rightarrow Z$. A policy $\pi_{demo} : Z \rightarrow A$ is extracted from the demonstration dataset D consisting of teacher demonstrations $d \in D$ which are of the form $d =< z, a >, z \in Z, a \in A$. When executing the task autonomously, the robot uses π_{demo} for selecting the next action a based on the current observed state z. The execution model of generic learning from demonstration system is given in Figure 2.1.

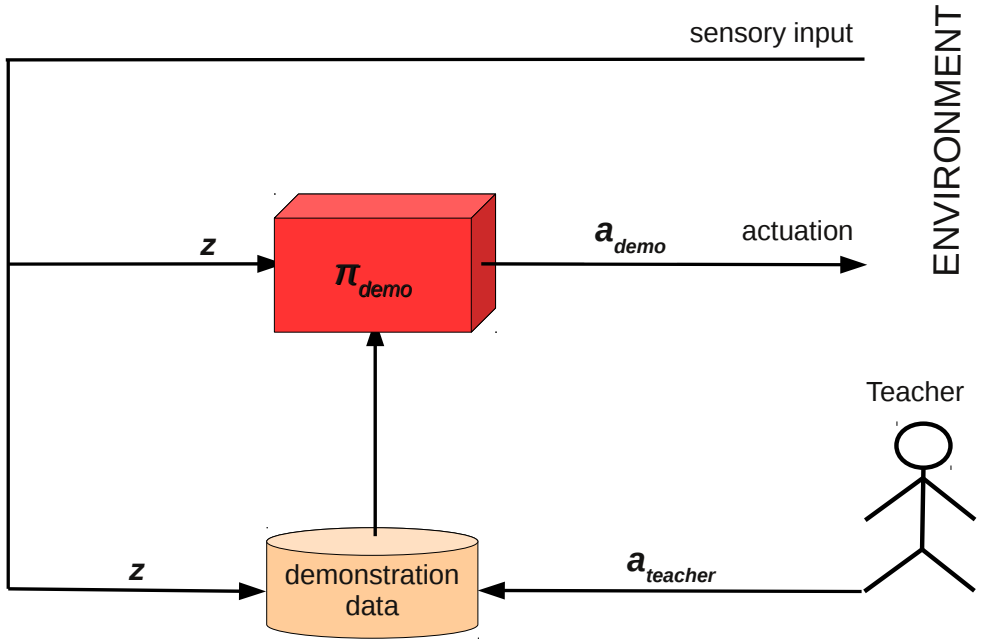


Figure 2.1. The schematic representation of the generic LfD system.

2.2. Corrective Demonstration

Corrective demonstration is a form of teacher demonstration focusing on correcting an action selected by the robot to be performed by proposing one of the following types of feedback:

- An alternative action to be executed in that state
- A modification to the selected action

The usual form of employing corrective demonstration is either through adding the corrective demonstration example to the demonstration dataset or replacing an example in the dataset with the corrective example, and re-deriving the action policy using the updated demonstration dataset.

However, re-deriving the execution policy each time a correction is received can be cumbersome if the total number of state-action pairs in the demonstration database is large. On the other hand, accumulating a number of corrective demonstration points and then re-deriving the execution policy may be misleading or inefficient since the demonstrator will not be able to see the effect of the provided corrective feedback immediately.

2.3. Advice Operators

Advice Operators Policy Improvement (A-OPI) is a corrective demonstration method for improving the execution performance of the robot in a human-robot learning from demonstration (LfD) setup [1]. Advice operators provide a *language* between the human teacher and the robot student, allowing the teacher to give *advice* as a mathematical function to be applied on the actions in the demonstration database and/or the observations corresponding to those actions. The resulting data is then used to re-derive the execution policy. More formally, for the defined advice operators $O = \{o_1, o_2, \ldots, o_N\}$, there is a set of corresponding mathematical functions

$$F = \{f_1(X_1), f_2(X_2), \ldots, f_N(X_N)\} \tag{2.1}$$

where $X_o =< x_1, x_2, \ldots, x_K >$ is the parameter vector for the advice operator o. For each received advice o along with its parameter vector X_o, the corresponding mathematical function $f_o(X_o)$ is applied on the observations Z and/or actions A

such that $Z' \leftarrow f_o(X_o, Z)$ and/or $A' \leftarrow f_o(X_o, A)$. Advice operators are especially useful in domains with continuous state/action spaces where the correction must be provided in continuous values.

2.4. Related Work

LfD based methods have been applied to many learning scenarios involving high level task and low level skill learning on different robotic platforms varying from wheeled and legged robots to autonomous helicopters. Here we present a few representative studies and strongly encourage the reader to resort to [2] for a comprehensive survey on LfD.

2.4.1. Task Learning

While learning to perform high level tasks, it is a common practice to assume that the low level skills required to perform the task are available to the robot. Task learning from demonstration have been studied in many different contexts.

Thomaz and Breazeal have proposed a method for utilizing human feedback as the reward signal for the Reinforcement Learning (RL) system [3]. They used a simulated kitchen environment modeled as a Markov Decision Process (MDP) where a robot tries to learn how to bake a cake. The human teacher observes the robot operating and provides a reward signal at any time without interrupting the operation. The authors have presented a user study. The notion of observing the robot executing the task and intervening to provide feedback bears a resemblance with our approach. However, they utilize the feedback as a reward signal to an action selected by the robot whereas in our approach the teacher provides actions and/or action corrections instead of quantitative evaluation of the action outcomes. The second main difference is that our approach makes use of the received feedback to improve the performance of an existing algorithm while their approach utilizes the received feedback for training a RL system.

Çakmak *et al.* investigated the issues arose when using active learning to speed up the learning and to improve the learning accuracy [4]. They evaluated three different ways of making queries where each of the methods differ in the conditions of when to ask teacher for a demonstration using an upper-torso humanoid robot in a concept learning task. They presented a user study where they evaluate the performance of the different ways of asking for feedback against each other and against a baseline supervised learning method.

Chernova and Veloso introduced an approach for learning behavior policies from human demonstration called "Confidence Based Autonomy (CBA)" [5]. The CBA approach utilizes a confidence calculation mechanism for assessing how confident the robot is about the action selected by its execution policy. If the confidence value is above a certain threshold, the robot proceeds with the execution of the selected action. Otherwise, it asks for teacher demonstration. The system builds a statistical model of the received demonstration examples and becomes more confident in situations where it has received a higher number of demonstrations. The CBA approach reduces the need for teacher attendance, hence it makes the teaching process less tedious and time consuming for the teacher. The goal of reducing the need for teacher attention is also shared by our approach in this thesis. However, instead of starting from scratch, our approach utilizes an existing algorithm as the baseline controller and needs teacher feedback only when the algorithm fails to compute a proper action to execute. The CBA approach is applied to a set of behavior learning problems for single robot such as humanoid obstacle avoidance [6], simulated car driving [7], and for multi-robot systems such as a simulated furniture-moving problem [8], and a humanoid ball sorting task [9].

2.4.2. Skill Learning

Several approaches to low level skill learning in the literature utilize LfD methods with different foci. Unlike task learning, most skill learning approaches deal with continuous domains where the robot learns to execute a sequence of low level actions

properly.

A recently populerized method for teaching low level skills that utilizes the "Learning from Experience" approach is named "Kinesthetic Teaching". In kinesthetic teaching methods, the teacher makes the robot perform the skill through tactile interaction. Hersch *et al.* proposed a method utilizing dynamic system control and statistical learning theory for acquiring goal-directed gestures. The authors have evaluated their approach using a humanoid robot in a reaching and grasping skill, and a skill for putting an object in a box [10].

Tactile interaction has also been utilized for skill refinement through tactile correction. Argall *et al.* have proposed a method for refining a demonstrated skill execution policy using kinesthetic feedback from the teacher during the execution of the skill using the execution policy extracted from the demonstration examples [11, 12, 13, 14]. In the "Tactile Policy Correction" approach, if tactile feedback is detected, the policy is modified according to the received corrective tactile feedback. This approach shares a similarity with our approach as both systems utilize teacher feedback interleaved with the skill execution. The main difference with the TPC method and our approach is that while the TPC method uses the received tactile feedback to modify the execution policy learned from demonstration, our approach keeps the received feedback commands separately and learns a *correction policy* out of the feedback commands to correct the actions of the underlying controller. A similar incremental skill refinement method is proposed by Calinon and Billard where they utilized different modalities like using motion sensors in addition to tactile correction for teaching a humanoid robot how to perform a bimanual grasping skill and for learning the affordances and effectivities of objects [15].

Nakanishi *et al.* have proposed a method for learning biped walking from human demonstration using motion primitives [16]. Their method utilizes dynamical motion primitives as a Central Pattern Generator (CPG) for generating cyclic walking patterns for a biped robot. The trajectories demonstrated by a human are learned

through motion primitives using Locally Weighted Regression (LWR). Motion primitives have also been utilized for skill learning by Bentivegna and Atkeson where the robot learns how to play air hockey [17]. In their approach, the robot learns the parameters of the motion primitives through the observation of other parties performing the same skill. Similarly, Bentivegna *et al.* have proposed an approach for learning how to select behavioral primitives and how to generate subgoals for the big task at hand. They evaluated the proposed approach on a robotic platform playing marble maze game as well as a simulated version of the system [18].

Several regression based approaches have been proposed for skill learning from demonstration. Grollman and Jenkins have proposed a learning framework named "Dogged Learning" to learn several low level skills for playing soccer [19, 20]. They evaluated the efficiency of their problem and platform independent framework on Sony AIBO robotic dogs and in robot soccer domain where the robot learns how to seek for the ball and how to mirror the movement of its tail with its head. Calinon *et al.* have proposed a probabilistic approach that utilizes Hidden Markov Models (HMM) along with regression for learning several low level skills with different characteristics. They evaluated the generalization ability of the proposed approach on several different skills to be learned such as a cyclic bimanual dancing motion on a highly articulated iCub humanoid robotic platform, a spoon-feeding skill with multiple simultaneous constraints on a HOAP-3 humanoid robot, and a ball hitting skill that can be performed in multiple ways on a Barrett WAM redundant robotic arm [21]. Gribovskaya *et al.* have proposed a method for being able to generalize non-linear multivariate motion dynamics of a certain low level skill from human demonstrations of the skill. Their approach utilizes Mixture of Gaussians (MoG) to estimate multivariate robot motions [22].

Interacting with the learner using high level abstract methods in low level skill learning problems has been proposed in different forms. Breazeal *et al.* have proposed a theoretical framework for human-robot collaboration using joint intention theory. In their approach, the teacher can interact with a highly expressive learner robot

using natural language and the robot communicates its inner state through a set of gestures and expressions [23]. Rybski *et al.* have proposed a method for interactive robot training through dialog using natural language. A set of behavior networks are learned from verbal teacher feedback where a rule-based behavior specification is dictated to the robot [24].

Another method for learning low level skills from human demonstration through high level communication methods is the Advice Operator Policy Improvement (A-OPI) approach proposed by Argall *et al.* [25, 1]. A set of defined verbal operators are associated with functional transformations for low level robot motion. The teacher provides feedback in the form of defined verbal operators and the corresponding transformations are applied on the specified portion of the demonstration database. A new execution policy is then re-derived out of the modified demonstration database. The A-OPI approach is evaluated on a trajectory learning task using a Segway RMP robot platform.

Several examples of learning from demonstration utilizing reinforcement learning methods have been proposed. Abbeel and Ng have proposed an inverse reinforcement learning method for teaching a robotic helicopter to perform several complex low level skills [26]. They assume a domain expert to be available for providing good examples of the skill execution. A proper reward function is then learned as to maximize the reward signal for the action sequence provided by the human teacher. In another reinforcement learning based approach, Atkeson and Schaal have proposed a method for learning how to perform a complex skill from a single demonsration [27, 28]. They applied their proposed approach to a pole balancing skill performed by a SARCOS robotic arm. Guenter *et al.* have presented a system for imitating constrained reaching tasks using reinforcement learning [29]. The proposed system is based on a dynamical system generator in combination with a reinforcement learning component for allowing the robot to adapt the trajectory learned through demonstration to novel situations such as avoiding obstacles along the way which were not present during the initial training [29]. Kolter *et al.* have proposed a hierarchical appren-

ticeship learning approach for learning complex skills which are non-trivial even for the domain experts [30]. They propose a method that allows the teacher to provide advice at different hierarchical levels as providing isolated advice for a smaller part of the skill is often easier for the teacher. This approach shares similarities with our multi-resolution task and skill refinement approach since one of the two key advantages of our multi-resolution approach is the ability to cover a larger portion of the state-action space with demonstration provided at a low detail resolution. Our approach differs from the hierarchical apprenticeship learning approach with its utilization of multiple algorithms having different computational complexities and running at different detail levels.

2.4.3. Case-Based Reasoning

Case-Based Reasoning (CBR) is a method for solving problems based on the solution of the similar problems encountered in the past [31]. CBR consists of four steps:

- Retrieve: In this step, similar cases are retrieved from the memory for a given case.
- Reuse: In this step, the retrieved solution for the most similar case is adapted for the new case at hand.
- Revise: In an iterative process between the Reuse and this steps, the adapted solution is tested against the new case and further revised as needed.
- Retain: Once the performance of the adapted solution to the new case is satisfactory, the new solution is added to the database of solutions along with the description of the new case.

The method of generalizing the received corrections over novel and unforeseen situations in our approach is similar to the retrieve and reuse steps of CBR-based systems. Using a domain-specific similarity measure, our approach also scans its database of corrections and fetches the correction that is received in a state most

similar to the current state of the system, if the similarity value is over a certain threshold. The main difference between our approach and CBR-based systems is that in CBR-based systems, it is not possible to employ a case-independent generic algorithmic solution to the problem for covering for handling most simple cases. Therefore, for non-trivial tasks of certain complexity performed by highly articulated robotic platforms, CBR-based systems require a high number of different cases in order to be able to find similar cases for a given new case. From this perspective, CBR-based systems suffer from the same scaling problem with the general LfD systems.

3. MODEL PLUS CORRECTION (M+C) PARADIGM

This chapter introduces the two complementary corrective demonstration paradigms which are the main contributions of this thesis. We start with our key contribution, the Model Plus Correction (M+C) paradigm. We first present a formal model of the M+C paradigm and then describe each of its components in detail. We then present the generalized version of M+C with multiple detail resolutions called Multi Resolution M+C (MRM+C). We present a formal model to the MRM+C paradigm as an extension over M+C, and then we describe the components of MRM+C. In the last part of this chapter, we describe how a system utilizing M+C and MRM+C is trained and how a trained system executes the task or skill in question along with algorithms for both training and autonomous execution.

3.1. Model Plus Correction

We define our complementary corrective demonstration approach by extending the LfD model given in Chapter 2. Since the distinguishing property of complementary corrective demonstration approach is that it uses an available model-based algorithm as the default controller and utilizes corrective human demonstration not to learn the task or the skill from scratch, but to refine the performance of the available controller algorithm, we name the approach as *Model Plus Correction (M+C)*.

We define the M+C system as a tuple $< S, A, \pi_{demo}, \pi_{model}, f_{reuse} >$. The LfD definition given in Chapter 2 is extended with a model-based controller, which can be considered as a hand-coded action policy $\pi_{model} : Z \rightarrow A$, and a correction reuse function $f_{reuse}(z, a_{demo}, a_{model}) : Z \times A \times A \rightarrow A$, where a_{demo} is the action computed by π_{demo}, and a_{model} is the action computed by π_{model}. The correction reuse function computes the final action to be executed by the robot as a function of the current observed state, and the actions computed by the model-based and the corrective demonstration policies. The schematic representation of the M+C system is given in

Figure 3.1.

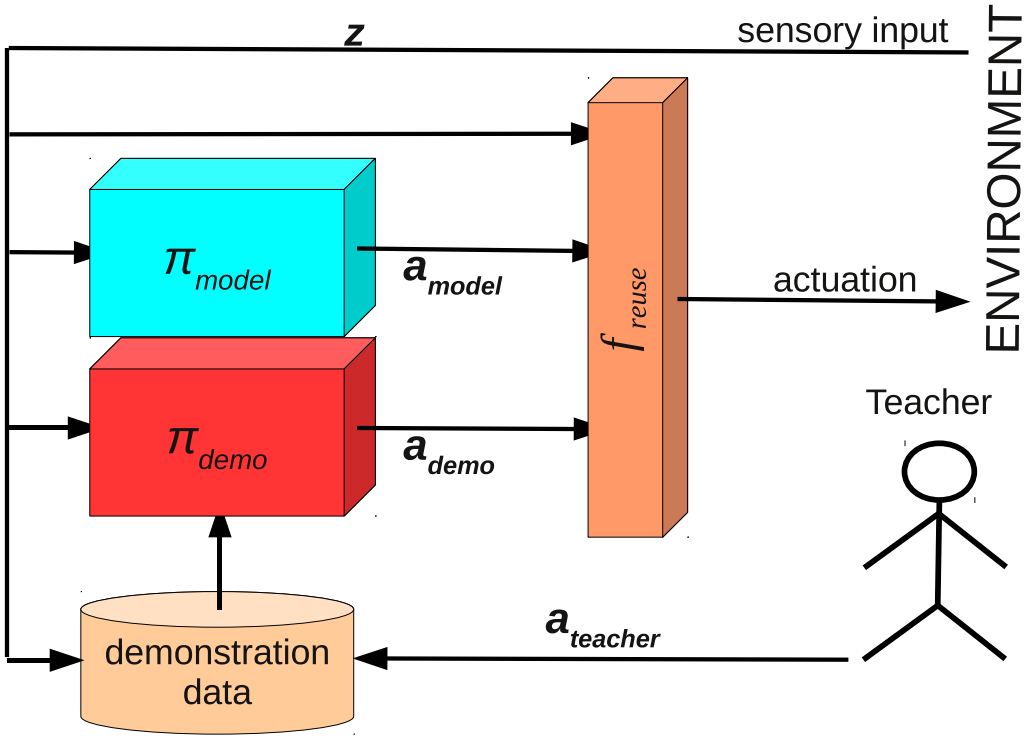


Figure 3.1. The schematic representation of the M+C system.

In the remainder of this chapter, we will rephrase the individual components of the M+C paradigm, and its generalized form, the MRM+C model.

3.2. The Model

The *model* part of the M+C paradigm aims to cover as much of the state space as possible with a model as simple as possible, either a mathematical model or a model derived out of some available data. More formally, the model is an action policy $\pi_{model} : Z \rightarrow A$. Another strong motivation behind the *model* component besides simplicity is to be able to use an available algorithm for the task or the skill as black-box. The model-based algorithm is able to perform the task for some simple cases; therefore, this makes it easier for the demonstrator and will require less attention as the teacher would only need to get involved when the algorithm fails.

3.3. The Correction

The *correction* part of the paradigm augments the *model* part with corrective human demonstration to improve the performance of the system by providing substitute actions for the states where the model is unable to compute a good action for that state. Just like the model, the correction is also an action policy $\pi_{correction} : Z \rightarrow A$ which uses the collected corrective demonstration database D. Depending on the application domain and the type of the task or skill, the policy can be extracted from the demonstration data and represented with a model, or, the demonstration data can be kept and used by the correction policy during the autonomous execution of the task or the skill.

3.4. Correction Reuse

The correction reuse component constitutes the *"plus"* part of the M+C paradigm. It functions as a glue between the model and the correction parts, and is responsible from delivering the final action to be executed by the robot. The final action is computed as a function of the actions provided by the model and the correction components, and the observed state of the system. Depending on the approach and whether the actions are discrete or not, the *correction reuse* part does one of the following:

(i) Decide which one of the model action and the correction action to be executed as the next action.

(ii) Combine the model and correction actions together by considering the correction action as a modification on the model action

In (i), a binary classifier that takes the current state of the system as input, and outputs a class label $c \in \{model, correction\}$ is employed. The final action to be executed is selected according to the output of the classified for the observed state of the system. In (ii), a function that takes the current state, the model action, and the

correction action as its input and outputs a corresponding action is employed. All the applications in this thesis assume a fixed correction reuse policy. However, since the correction reuse policy itself is also a function, how to select an action in case (i), and how to combine the actions in case (ii) can be learned either autonomously, or again from a human teacher through demonstration.

3.5. Multi-Resolution Model Plus Correction Approach

Most LfD approaches use a single fixed state and action representation and a single action policy extracted from the demonstration data. Finding efficient state representation and action definitions for complex tasks is considered a difficult problem without a widely accepted general solution. For complex tasks, using the most detailed state representation and action definitions available often requires a large number of demonstrations to be provided in order to be able to extract a sufficiently generalized policy. Similarly, a hand-coded algorithm using the most detailed state and action definitions might be computationally expensive and infeasible for being used as the sole action policy. On the other hand, using a very abstract state and action definition might fail to capture the complexity of the task. Depending on the nature of the task, different portions of the state-action space can be covered at lower detail resolutions, hence saving both computational power and space. A hand-coded algorithm running at a lower detail resolution would also be easier to implement and less demanding in terms of computational power requirements.

We use the introduced M+C model in the previous section and present a new model to include multiple instances of M+C operating at a set of different detail resolutions R. We define Multi-Resolution Model Plus Correction (MRM+C) as a tuple $< f_{refine}, \{r_1, r_2, ..., r_N\} >$, where $r_k \in R$ is an instance of a modified version of the M+C model defined for the detail resolution r_k, and $\pi_{arbitrator}(z) : Z \rightarrow R$ is the detail resolution arbitration policy. The extended M+C model is a tuple $< S_r, A_r, f_{state}, f_{action}, \pi_{demo}, \pi_{model}, f_{reuse} >$ where $f_{state} : S \rightarrow S_r$ is the function for mapping the global state to the state definition at the detail resolution r, and

$f_{action} : A_r \rightarrow A$ is the function for mapping the action computed at the detail resolution r into an action representation at the finest detail level. The arbitrator component decides which detail resolution to use for computing the next action to be executed based on a the observed state of the system.

In addition to the M+C definitions for each detail resolution, there is also a function for mapping the most detailed state definition to the state definition of each detail resolution and a function for mapping the actions given at that detail resolution to actions represented in most detailed action definition. A schematic diagram of the Multi-Resolution M+C framework is given in Figure 3.2.

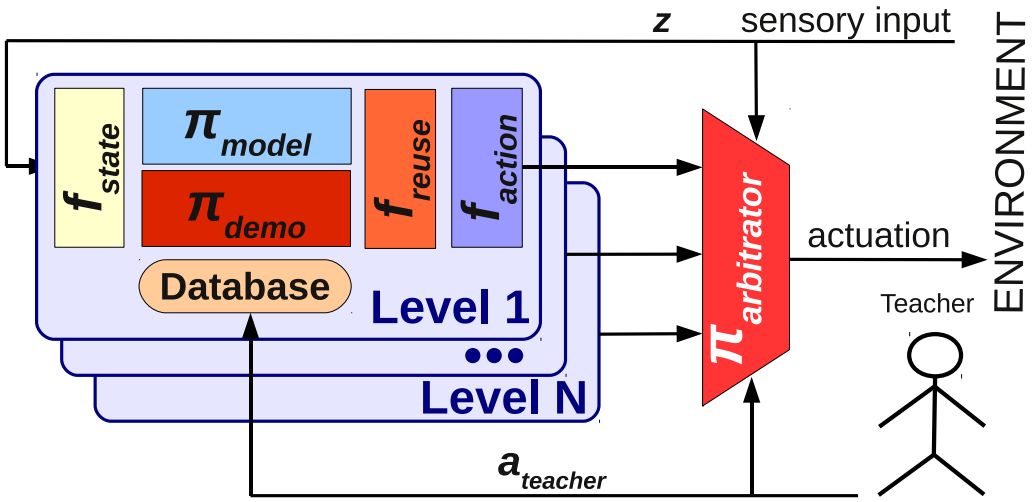


Figure 3.2. The schematic representation of the MRM+C framework.

3.5.1. State Definition

The *state definition* is a represented as a feature vector computed using the most detailed sensory and proprioceptive information available. A separate state mapping function is defined for each detail resolution to map the available sensory information at the finest detail resolution to the state representation defined for the current detail resolution. Defining a state representation at a lower detail level than the state representation for the finest detail resolution helps keeping the state space smaller. Having a smaller state space reduces the complexity of the computations needed to select an action to be performed for a given state vector in case of the

execution of the default algorithm, or leads the robot requiring smaller number of demonstrations to learn the exceptions, or "patches" to the default algorithm. The obvious downside of having a coarse state representation is the possibility of missing an important detail that could be caught with a more detailed state definition, and as a result, failing to take the proper action for that state of the system.

3.5.2. Action Definition

The *action definition* contains a set of actions that can be taken at that detail resolution. As in the state definition case, a mapping function should be provided to convert the actions expressed at a particular detail resolution to actions at the finest detail resolution. Having a smaller set of less detailed actions leads to a smaller action space. A natural result of this is in case of the demonstration, it becomes easier for the teacher to provide demonstration examples as using the more abstract actions. For the autonomous execution, similar to the state definition case, the actions can be computed by less complex and therefore computationally inexpensive algorithms. The possible major disadvantage of using more abstract action definitions is in a certain state a complex instance of the task which requires delicate and accurate actuation, the abstract action model might fail to match the required level of accuracy,

3.5.3. Default Controller

We use the *default controller* as the primary action policy to compute which action to take in a particular state expressed in the state definition for the current detail resolution of the system unless a demonstrated correction action states otherwise. The default controller can be implemented as a hand-coded algorithm employing a mathematical model for performing the task, or it can be in the form of an action policy learned using a machine learning method, such as reinforcement learning or genetic algorithm. Although there is not a restriction on the complexity of a default algorithm, it is expected that the default algorithm for a coarse detail resolution to be a simpler algorithm than the algorithm for a finer detail resolution. Similar to the

state and action definition cases, simpler algorithms are easier to develop or obtain and are often computationally inexpensive compared to their more complex counterparts. However, they share the same trade off with the state and action definition resolutions: simpler algorithms carry the risk of falling short on making complex decisions as needed in a particular state of the system. An unnecessarily complex algorithm, on the other hand, consumes the processing power without producing added value.

As we stated in the introduction part of this chapter, it is a non-trivial problem to strike a good balance in computational burden and functional efficiency by choosing the right complexity level for the algorithm. In fact, this is the very rationale behind this multi-resolution approach, that is to have different controllers and human correction databases for covering different parts of the state-action space for the task with different complexities.

3.5.4. Correction Reuse Algorithm

For each defined detail resolution, we build a separate corrective demonstration database consisting of the correction actions delivered by the teacher at this detail resolution of the system. We store the received demonstration actions in the form of state-action pairs without extracting an action policy out of the demonstrations. During the autonomous task execution, for a given state of the system represented at the current detail resolution, both the default algorithm and the corrective demonstration parts can compute an action. Therefore, the complementary corrective demonstration instance needs a mechanism to populate a single action out of two actions generated by the default algorithm, and by the corrective demonstration component. The mechanism can be a simple selection algorithm based on a criteria, e.g., selecting the demonstration action if the state similarity computed for the state that the system was in when the selected demonstration action was delivered by the teacher, or the correction reuse algorithm can combine the two actions to compute a third action (using demonstration action to modify the computed action by the

default algorithm). Since the correction reuse algorithm is in fact, another action selection policy which maps the current state of the system to an appropriate correction reuse action, it is possible to also learn this correction reuse policy either using self-exploratory methods like reinforcement learning, or again from human demonstration. In this chapter, we assume hand-coded fixed correction reuse algorithms and we leave the investigation of learning such reuse policy as a future work.

3.5.5. The Detail Resolution Arbitrator

The detail resolution arbitrator, or *the arbitrator* in short, is the outermost component of the MRM+C algorithm and its main duty is to select the appropriate detail resolution for a given state and executes the action computed by the M+C instance associated with that particular detail resolution. Similar to the model and the controller components, the arbitrator is also an action policy $\pi_{arbitrator} : Z \rightarrow R$ and can be implemented as a hand-coded algorithm, or can be learned. The detail resolution arbitrator acts as an active proxy during the task execution and activates the appropriate detail resolution depending on its policy and the current state of the system. As with the correction reuse algorithm, it is possible to use a fixed hand-coded policy or a learned policy for the detail resolution arbitrator component. The employed policy can decide to switch the system into a finer detail level when the system is at a more abstract detail resolution, or the policy can also decides to switch to a more abstract detail resolution for the sake of using a simpler and computationally inexpensive compared to the default algorithm defined at the current detail resolution. In our application problem using MRM+C, we used an arbitrator policy learned from human demonstration.

3.5.6. Correction Delivery: Training the System

The instantiated MRM+C system for performing a certain task is ready to perform the task but usually the task execution performance is sub-optimal. Various components of the MRM+C system, namely, the individual corrective demonstration

components for each of the defined state resolutions, and the detail resolution arbitrator component need to be trained through a set of corrective human demonstration sessions. The correction delivery and correction reuse stages can be configured to work in an interleaved manner. In other words, since we keep the received correction actions as raw data without generalizing a policy, the robot can start using the demonstration actions immediately. However, for the sake of simplicity, we explain the training procedure assuming the correction reuse is disabled. In the real world experimentation, the correction reuse is used in conjunction with the correction delivery to avoid redundant teacher corrections.

Along the course of a demonstration, the robot starts executing the task until it reaches a decision state where an action needs to be computed to be able to proceed with the task execution. At the beginning of each decision process, the MRM+C system switches to the most abstract detail resolution. The state vector according to the state definition for the most abstract detail resolution is computed and an action is selected by the default algorithm associated with that detail resolution. The robot then proceeds with the execution of the selected action. The teacher observes the robot as it executes the task, and intervenes by providing a feedback to the robot, if the action selected by the default algorithm is erroneous.

The teacher gives two types of feedback:

- The *elaborate command* to take the system to the next detail level with finer resolution. A new action is computed using the specified hand-coded controller associated with the new detail level.
- The *correct command*, issued with the specified replacement action to substitute the current action with another action defined for the same detail resolution.

If an *elaborate* command is received, the system checks if there is a finer detail resolution available. If such a resolution is found, the received elaborate command is stored with the current state of the system represented with the state definition

for the finest detail resolution available. The system then switches to that detail resolution and goes back to the action computation step. If a *correct* command is received, the provided substitute action is stored with the current state of the system represented with the state definition for the current detail resolution and the action to be executed by the robot is replaced with the received corrected action. The algorithm for MRM+C training is given in Figure 3.3.

```
1: resolution ← LOWEST
2: state ← computeState(resolution)
3: action ← computeAction(state)
4: executeAction(action)
5: if feedbackReceived() then
6:    feedback ← readFeedback()
7:    if feedback == ELABORATE then
8:       if resolution < HIGHEST then
9:          saveDetailDemonstration()
10:         increaseResolution()
11:         goto 2
12:      end if
13:   else if feedback == CORRECT then
14:      action ← readCorrection()
15:      saveCorrectionDemonstration()
16:      executeAction(action)
17:   end if
18: end if
```

Figure 3.3. The algorithm for training the MRM+C system.

The demonstration continues as long as the teacher observes a room for improvement in the system. Once the demonstration session is over, the MRM+C system has the learned individual correction policies as well as a detail resolution policy. We now present the algorithm for the autonomous task execution case where the robot performs the task using the hand-coded algorithms at different detail resolutions and augmented with complementary corrective demonstration.

3.5.7. Correction Reuse: Autonomous Execution

During the autonomous task execution using the MRM+C system, each time the robot reaches a decision point during the autonomous task execution, the MRM+C algorithm sets the system resolution to the most abstract detail resolution available and

computes an action using the algorithm associated with that detail resolution. Then, the system starts searching the correction and elaboration demonstration databases for correction actions and detail resolution change commands, in that particular order. In other words, the system tries to find:

- A correction sample in the corrective demonstration database for the current detail resolution
- An elaborate command in the elaboration demonstration database for switching to the next detail level with finer resolution.

If a correction sample is found in the corrective demonstration database for the current detail resolution which is received when the robot was in a state that is *similar enough* to the current state of the system, the action is selected as the next action and the execution continues with the announcement of the selected action. If no correction samples can be found in the corrective demonstration database for the current level but an elaborate command received in a state *similar* to the current state of the robot is found, the system changes its detail level to the level specified in the elaborate command and recomputes an action using the hand-coded algorithm specified for the new detail level. In both cases, a domain specific state similarity measure is employed and during the search, the entry with the highest similarity value for the current state of the system is found. If the similarity value for the located demonstration point is higher than a specified threshold, the demonstration point is executed instead of the default action. If the demonstration point is a *correct* action, the next action to be executed by the robot is replaced with the demonstration action. If the demonstration point is an *elaborate* action, the system switches to the next detail level and goes back to the action selection step. The algorithm for the autonomous MRM+C execution is given in Algorithm 3.4.

```
1: resolution ← LOWEST
2: currentState ← computeState(resolution)
3: mostSimilar ← ∅
4: maxSimilarity ← 0
5: for each demonstration ∈ correctionDatabase_resolution do
6:    similarity ← getSimilarity(currentState, demonstration(state)
7:    if similarity > maxSimilarity then
8:       maxSimilarity ← similarity
9:       mostSimilar ← demonstration
10:   end if
11: end for
12: threshold ← getCorrectionThreshold(resolution)
13: if maxSimilarity > threshold then
14:   action ← demonstration(action)
15: else
16:   mostSimilar ← ∅
17:   maxSimilarity ← 0
18:   for each demonstration ∈ elaborationDatabase do
19:      similarity ← getSimilarity(currentState, demonstration(state)
20:      if similarity > maxSimilarity then
21:         maxSimilarity ← similarity
22:         mostSimilar ← demonstration
23:      end if
24:   end for
25:   threshold ← getElaborationThreshold()
26:   if maxSimilarity > threshold then
27:      if resolution < HIGHEST then
28:         increaseResolution()
29:         goto 2
30:      else
31:         action ← computeAction(currentState)
32:      end if
33:   else
34:      action ← computeAction(currentState)
35:   end if
36: end if
37: executeAction(action)
```

Figure 3.4. The algorithm for the autonomous MRM+C execution.

3.6. Discussion

In this chapter, we presented a unified view on the complementary corrective demonstration approach. We introduced Model plus Correction (M+C) and Multi-Resolution Model plus Correction (MRM+C) algorithms as parts of a general-purpose paradigm for task and skill refinement using complementary corrective demonstration.

The M+C paradigm, and its generalized extension, the MRM+C algorithm are both provide a solid framework for task and skill refinement by having components with well defined input, output, and functionalities, hence making it easy to model complex tasks and skills to be refined in terms of execution performance. It is possible to use a plethora of learning algorithms of choice seamlessly within the individual components that involve learning, since no part of the proposed algorithms depend on any specific algorithm, hardware, application domain, or state and action representations.

4. SKILL REFINEMENT USING M+C

In this chapter, we present a multi-phase corrective demonstration approach for improving the biped walk stability on the Aldebaran Nao humanoid robot platform as an example application of the M+C paradigm. Being actively studied in humanoid robot research, biped walking is a challenging problem due to the high dimensional state and action space and the complex dynamics of the walking process. In our approach, we make use of an existing walk algorithm to obtain an initial open-loop walk cycle, and then we improve the stability of the walk in two corrective demonstration phases.

The phases of learning in our approach are as follows:

- An initial modeling of the walk motion by using the output of an existing walk algorithm
- Offline improvement of the obtained walk model via high level human advice
- Acquisition of a closed-loop gait via real time corrective human demonstration while the robot is walking using the open-loop walk obtained in the previous phases

The demonstration signals given using a comercially available wireless game controller are transmitted to the robot over a host computer via wireless network. This setup allows the demonstrator to closely follow the robot and deliver the corrective demonstration without tactilely interacting with it. The received correction signals are recorded together with the state of the robot in the form of sensory readings. A correction policy is then derived out of the recorded state-action pairs using a learning algorithm of choice. Finally, the learned correction policy is used to modify the open loop walk cycle in such a way to keep the robot balanced as it walks autonomously.

We present different types of correction and different methods for state-action association, policy derivation, and the application of the correction with different complexities. In particular, we present two different correction types (applying correction in the joint space or in the task space), two different state-action association methods (associating a single sensor to a correction value without taking the current position in the walk cycle into account or associating multiple sensors with a correction value while taking the current position in the walk cycle into account), two different policy extraction methods (fitting normal distributions on the received correction values in the discretized sensory reading space or using locally weighted regression with Gaussian kernel), and two methods for deciding when to apply correction to the system (applying the correction at each N^{th} timestep within the walk cycle or applying the correction only if the sensory readings go beyond the normal values according to a certain statistical definition of the *normal*). We present experiment results evaluating the performances of the different combinations of the aforementioned methods compared to each other and compared to the initial open-loop walk. Experiment results demonstrate an improvement in walk stability in all of the presented methods with an increase in the overall performance as the used method gets more complex.

The organization of the rest of the chapter is as follows. Section 4.2 presents a formal definition of biped walking, and covers how an open-loop walking behavior can be acquired from an existing walk algorithm and how the acquired walking behavior can be improved using human advice. We explain our real-time corrective demonstration approach thoroughly in Section 4.3. Section 4.4 describes how we combine the corrective demonstration with the state of the robot to obtain a closed-loop walk. We present experiment results and evaluate the performances of the proposed methods in Section 4.5.

4.1. Modeling as a M+C Instance

4.1.1. State and Action Definitions

For the biped walk problem we use two different correction methods:

- Applying the correction in the joint space
- Applying the correction in the task space

In the joint space case, we represent the state of the system as a vector containing the accelerometer readings: $\overrightarrow{S} =< Acc_X, Acc_Y >$ where Acc_X and Acc_Y are the accelerometer readings along the X axis and the Y axis, respectively. We define the action as a vector of real numbers, each member representing the target angle for a joint of the robot. More formally, $A = \{\overrightarrow{a_1}, \overrightarrow{a_2}, ...\}$ and $\overrightarrow{a} =< j_1, j_2, ...j_N >$.

In the task space case, we represent the state of the system as a vector containing the current position in the walk cycle in additon to the accelerometer readings: $\overrightarrow{S} =< t, Acc_X, Acc_Y >$. We define the action for the task space correction as a vector containing relative offsets of the feet on ground plane and with respect to the hip of the robot. Formally, $A = \{a_1, a_2, ...\}$ and $\overrightarrow{a} =< x^{left}, y^{left}, x^{right}, y^{right} >$

4.1.2. The Model

As the underlying algorithm, we use an open-loop walk algorithm which obtains a walk behavior through playing back a single walk cycle extracted out of a ZMP-based walk algorithm in a loop. The walk cycle is defined as $wc_j(t) = \mu(\overrightarrow{D_j}), j \in Joints, t \in [0, T)$.

4.1.3. The Correction

In the joint space correction case, we use corrections for four hip joints (pitch and roll joints for left and right legs) therefore the action definition for the joint

space correction is $\overrightarrow{a} = \{C^{left}_{roll}, C^{left}_{pitch}, C^{right}_{roll}, C^{right}_{pitch}\}$. Even though the original walk cycle wc contains joint commands for all the joints of the robot, the correction action addresses only a subset of those joints. For the correction policy, we associate a single sensor with a single joint. In other words, we learn multiple correction policies for each joint to be corrected, and we use a single sensor reading, either the accelerometer reading along the X axis or the reading along the Y axis as the state representation for that correction policy.

In the task space correction case, the corrections are delivered in the same form with the actions. For this case, we define the correction action as $\overrightarrow{C} =< C^{left}_{X}, C^{left}_{Y}, C^{right}_{X}, C^{right}_{Y} >$. Similar to the joint space correction case, we learn multiple correction policies for the correction offsets along each direction and for both feet. The state representation, however, remains the same and defined as $\overrightarrow{S} =< t, Acc_X, Acc_Y >$.

4.1.4. Correction Reuse

For the joint space correction case, we apply the correction at fixed frequency (at each N^{th} timestep) regardless of the system state so the state representation for the correction policy is $\overrightarrow{S} = \emptyset$.

For the task space correction case, we use a sensor model obtained by fitting a normal distribution over the recorded sensory data gathered from many examples of the robot walking without any balance loss. We fit a separate distribution for each timestep within the walk cycle over all sensory readings recorded at that particular timestep. During the autonomous execution, we apply the computed correction values whenever the current sensor values are not in the range $\mu_t \pm K\sigma_t$ for the current timestep t in the walk cycle. We define the state representation for the correction reuse policy of the task space correction as $\overrightarrow{S} =< t, Acc_X, Acc_Y >$.

In both correction cases, we compute the final action to be executed by the robot through adding the actions computed by the model and the correction parts as $\overrightarrow{a}_{final} = \overrightarrow{a}_{model} + \overrightarrow{a}_{correction}$

4.2. Open-loop Biped Walking

Biped walking is a periodic phenomenon consisting of consecutive walk cycles. A walk cycle (wc) is a motion segment that starts and ends with the same configuration of the joints. Each walk cycle consists of four phases:

- First single support phase (left)
- First double support phase
- Second single support phase (right)
- Second double support phase

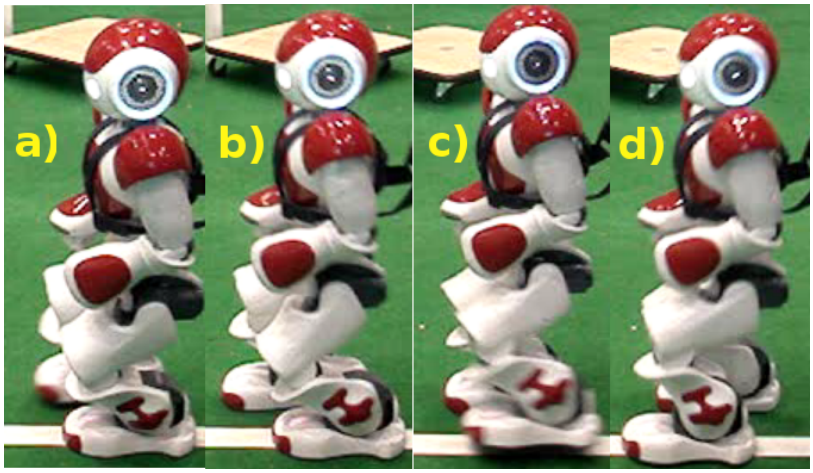


Figure 4.1. Walk cycle phases: a) first single support, b) first double support, c) second single support, and d) second double support.

During the first single support phase, the robot stands on its left foot, and swings the right leg forward. During the double support phases, both feet are on the ground, differing in the offsets along the X axis from the first double support phase to the second. During the second single support phase, the robot stands on its right foot to lift and swing the left leg forward as shown in Figure 4.1. The walk cycle has a duration of T timesteps, where $wc_j(t), t \in [0, T), j \in Joints$ is the *command* to the

joint j provided at timestep t.

In principle, if we could generate the correct joint command sequence for a walk cycle, it would then be possible to make the robot walk indefinitely by executing this cycle repeatedly in an open loop fashion. However, in reality, various sources of uncertainty associated with sensing, planning, and actuation affect biped walking.

- In *sensing*, the main source of uncertainty is the noise in the sensor readings due to the lack of precision/accuracy (e.g., high noise rate on the gyroscopes and the accelerometers and imprecise position sensing on the joints), or the environmental effects (e.g., electromagnetic fields affect compasses negatively). The Nao robot does not have a compass, therefore the environmental effects do not constitute a problem for us.
- In *planning*, the simplifications and assumptions that have been made while building the mathematical model of the system prevent the developed model from capturing all physical aspects of the real world.
- In *actuation*, several factors such as friction inside the gearboxes, the backlash in the gears, and unmodeled payload effects constitute the main sources of uncertainty.

As a result, the actual movement of the robot differs from the desired one as seen in Figure 4.2. Here, the plot with circles illustrates the joint commands, i.e., the desired trajectory, and the plot with triangles shows the actual trajectory followed by the joint. The section towards the end where the actual joint position significantly diverges from the desired trajectory corresponds to a moment where the robot is standing on its left foot in the first single support phase and the movement of the ankle joint is affected by the weight of the whole body.

Failing to follow the desired trajectory of the joint causes the robot to act differently than expected and this difference affects the balance negatively. This kind of unforeseen or poorly modeled sources of uncertainty are the typical drawback of

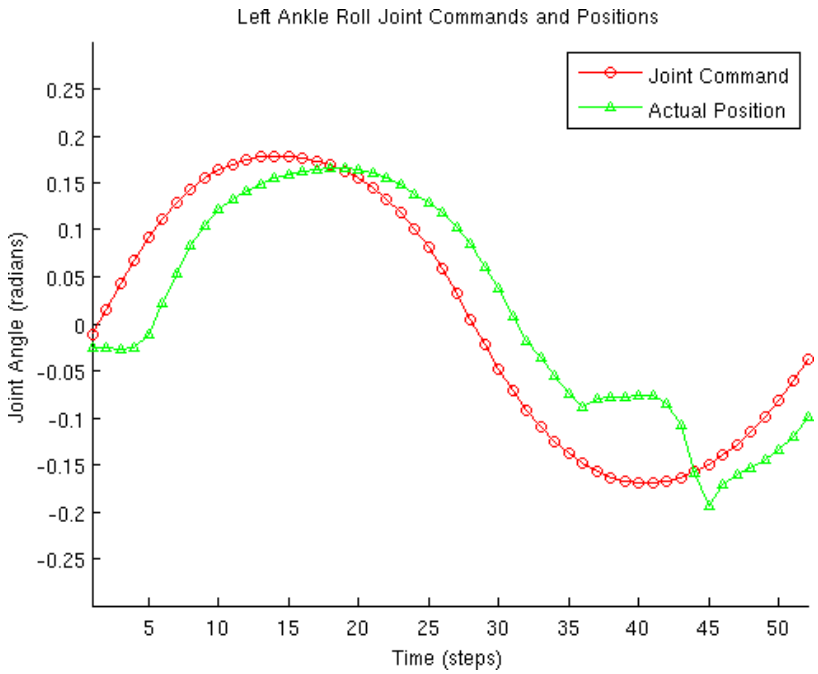


Figure 4.2. An example to the actuation error.

an open-loop controller, which is a type of controller that solely uses its model of the system to compute the next action and does not take any feedback from the environment into account to determine whether the desired state is achieved. A closed-loop controller, on the other hand, uses both its model of the system and the feedback received from the system to determine the next action. Similarly, an open-loop walk algorithm generates a set of joint angle commands at each execution timestep to form a walk pattern without taking the actual state of the robot (i.e., sensory readings) into account while a closed-loop walk algorithm incorporates the sensory feedback into the joint command generation process in such a way that the resulting walk motion keeps the robot balanced.

In the remainder of this section, we first present how an open-loop walk cycle can be captured by observing the output of an existing walk algorithm. We then present how the obtained open-loop walk can be further improved offline using the *Advice Operators Policy Improvement* method [1]. In the following sections, we present how a closed-loop walk can be built on top of the obtained open-loop walk.

4.2.1. Obtaining an Open-loop Walk

If a walk algorithm is readily available at hand, one way of obtaining a walking behavior without directly employing the algorithm is to observe the output of the algorithm and generate a single walk cycle out of those observations to be played back in a loop. To accomplish this, we use the existing walk algorithm as a black-box and record a number of walk sequences where the robot walks forwards for a fixed distance at a constant speed using the selected algorithm. We record the sequences in which the robot was able to travel the predetermined distance while maintaining its balance.

A set of examples of the robot walking without falling provide data D for each $t, t \in [0, T)$, in the form of the commands received by each joint $\overrightarrow{D_j}(t)$ and the corresponding sensory readings $\mathbf{S}(t)$ provided by the set of sensors $Sensors$. We obtain a single walk cycle wc using D as $wc_j(t) = \mu(\overrightarrow{D_j}), j \in Joints, t \in [0, T)$. In addition, we fit a normal distribution $N(\overrightarrow{\mu(t)}, \overrightarrow{\sigma(t)})$ to the readings of each sensor at each t, where $\mu_s(t)$ is the mean, and $\sigma_s(t)$ is the standard deviation for the readings of the sensor $s \in Sensors$ at time t in the walk cycle (Figure 4.3). In the figure, the middle line denotes the mean and the vertical lines denote +/- 3σ variance. The X axis is timesteps, and the Y axis is the sensor value.

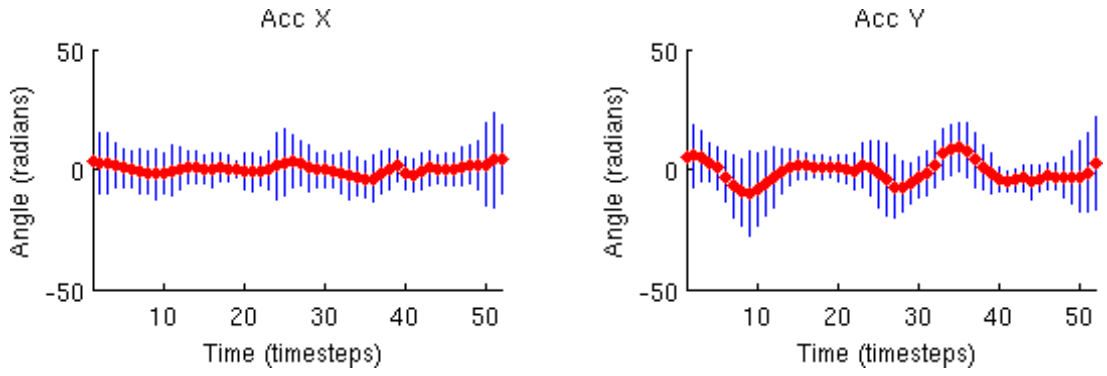


Figure 4.3. Distribution of the sensor values over the complete walk cycle for a stable walk sequence.

Sending the joint commands in the obtained walk cycle to the robot repetitively, hence playing back the captured walk cycle in a loop yields an open-loop walk be-

havior that performs similar to the original walking algorithm without employing the algorithm itself. Although the Nao robot has a total of 21 joints, for our experiments we utilize only 12 of them, which are all the leg joints except the shared hip yaw-pitch joint and the shoulder roll joints for the arms, constituting the set $Joints$.

4.2.2. Corrective Demonstration Using Advice Operators for Offline Improvement

We use A-OPI for correcting the obtained walk cycle in its open-loop form based on human observations of the executed walk behavior. We define three advice operators $O = \{ScaleSwing, ChangeFeetDistance, ChangeArms\}$ that are applied on the walk cycle:

- ScaleSwing(k): Scales the joint commands of the hip roll joints (along the X axis) in the walk cycle by a factor of k where $k \in [0, 1]$. The hip roll joints generate the lateral swinging motion while walking.
- ChangeFeetDistance(d): Applies an offset of d millimeters to the distance between the feet along the Y axis.
- ChangeArms($angle$): Raises or lowers the arms by $angle$ radians along the $Y - Z$ plane.

The algorithm for A-OPI is given in Figure 4.4. After a set of iterations consisting of the execution of the walk behavior, receiving advice from the teacher, and revising the walk cycle accordingly, an improvement is achieved. The initial and improved versions of hip roll joint values to generate lateral swinging motion are shown in Figure 4.5 as an example. Here, decreasing the amplitude of the hip roll joint signal causes the robot to swing less, which contributes to preservation of balance positively.

1: **while** the teacher sees room for improvement **do**
2: $executeWalk(wc)$
3: $o, X_o \leftarrow getAdviceFromTeacher()$
4: $wc' \leftarrow f_o(X_o, wc)$
5: **end while**

Figure 4.4. Advice Operator Improvement (A-OPI) algorithm.

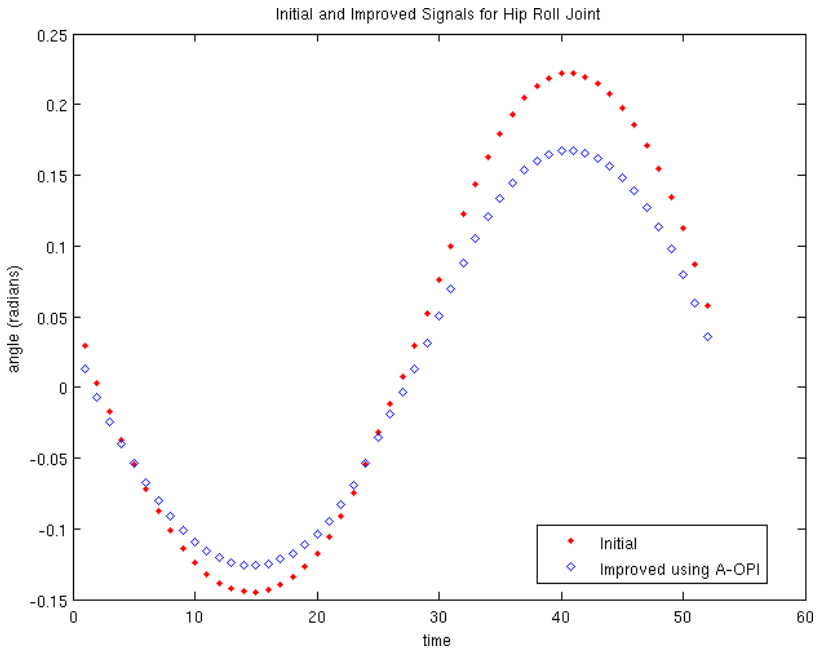


Figure 4.5. Initial and improved joint commands for hip roll joints generating swinging motion while walking.

4.3. Real-Time Corrective Demonstration

Without any corrections, an open-loop playback mechanism by itself is usually not enough to maintain the robot's balance while walking. Therefore, the next step after obtaining an open-loop playback walk and improving it offline using A-OPI method is to close the loop by adding a mechanism to modify the open-loop walk cycle during autonomous execution according to the feedback received from the system. The changes in sensor readings when the robot is about to lose its balance (Figure 4.6) are used to derive a correction policy by mapping these changes to corrective feedback signals. The right plot in the figure depicts an unstable walk sequence where the robot starts losing its balance after around 200^{th} timestep. We use real-time human corrective demonstration to learn a correction policy, which is a function that maps the sensory readings of the robot to the proper correction signals as it walks using the initial or the improved open-loop walk cycle.

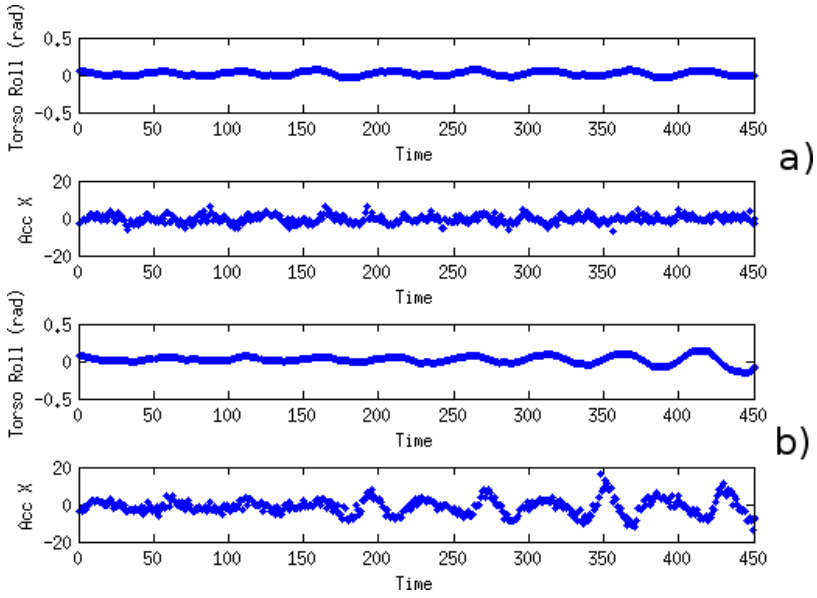


Figure 4.6. Sample torso orientation and accelerometer readings: a) a stable walk sequence, and b) an unstable walk sequence.

Due to the noisy nature of the sensors, fluctuations may occur in the sensor readings and that may result in jerky motions that lead to loss of balance when the correction values calculated as a function of the sensor readings are applied to the

joints directly. Therefore, the readings need to be filtered. Running mean and median smoothers are widely used methods for filtering noisy data. In running smoothers, the data point in the middle of a running window of size N is replaced with the mean or the median of the data points lying within that window. The filtered signal gets smoother as the window size increases. The delicate trade-off in filtering lies in the selection of an appropriate window size for smoothing the data just enough to filter out the noise without rendering the patterns in the data hard to detect.

We evaluated the running mean and median smoothers with window sizes 5 and 10 (Figure 4.7), and decided to use a running mean filter with window size 5 since it filters out the noise reasonably well and is computationally cheaper than the running median filter. Also, considering our sensor sampling rate is 50 Hz, we can still detect a significant change in the sensor readings in at most $1/10^{th}$ of a second. In the figure, a), b), c), d), and e) are the raw data, the output of the median smoother with window size 5, the output of the median smoother with window size 10, the output of the mean smoother with window size 5, and the output of the mean smoother with window size 10.

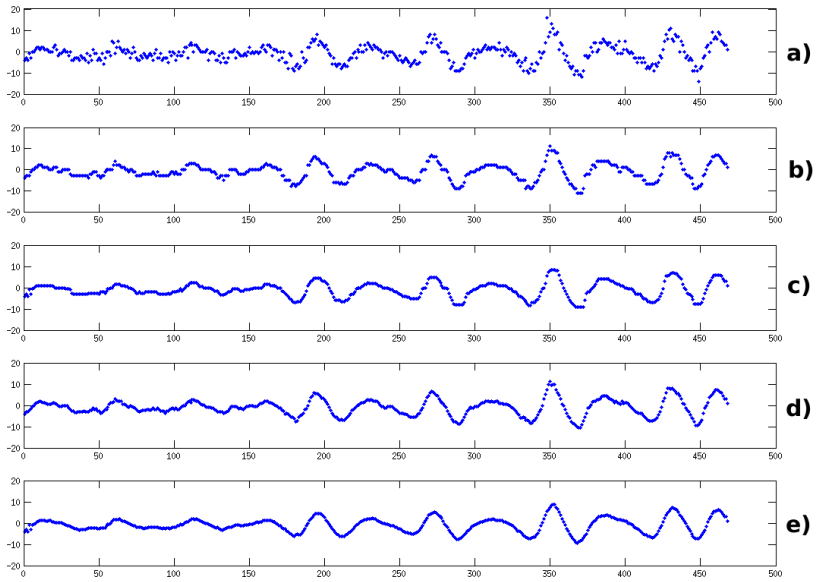


Figure 4.7. Results of applying various smoothers on an example accelerometer data.

In the remainder of this section, we first present our corrective demonstration setup, elaborating on the implementation details. We then present two different correction methods for forward walking along with a simplified inverse kinematics model for the Nao.

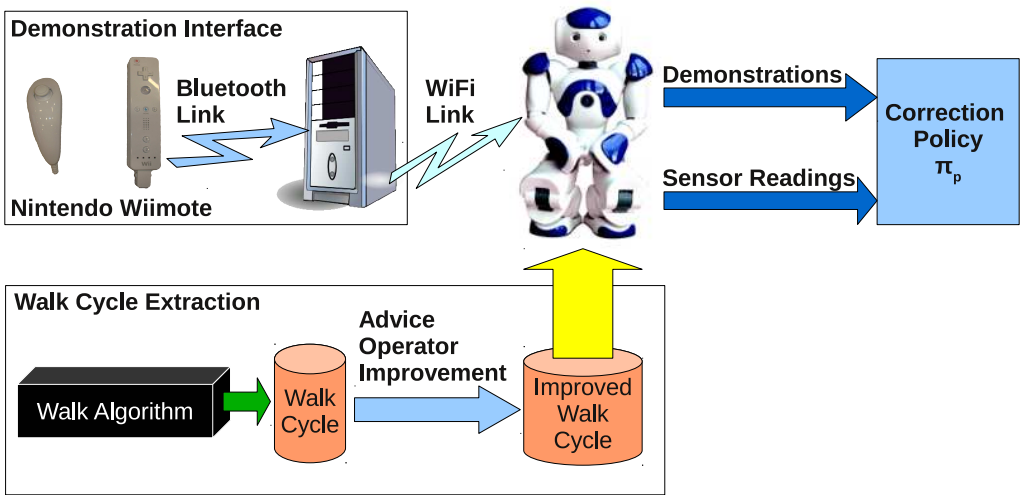


Figure 4.8. The diagram for the real-time demonstration framework for policy extraction.

4.3.1. Corrective Demonstration Setup

A major challenge in providing corrective demonstration for the biped walking process is to find a proper way of delivering the demonstration as fast as possible without physically contacting the robot. Fast delivery is needed because biped walk is such a delicate dynamic process that it might be too late to recover from a balance loss if the robot receives the provided correction signal with a significant delay. Another problem with late delivery is that in such a case the received demonstration is associated with the wrong set of sensory data; hence, it results in an erroneous association of the demonstration points with the state information in the policy generation process. The necessity of delivering the demonstration without touching the robot also stems from the delicate dynamics of the biped walking process since interfering with those dynamics of the robot affects the learned policy negatively.

We utilize a wireless control interface using the commercially available *Nintendo Wiimote* game controller (http://www.nintendo.com/wii/what/controllers) to deliver corrective demonstration to the robot. Both the Wiimote controller and its *Nunchuk* extension are equipped with accelerometers measuring the acceleration of the controllers as well as allowing their absolute roll and pitch orientations to be computed. Therefore, the Wiimote with its extension has four measurable axes allowing four different correction signals to be delivered simultaneously. The computed roll and the pitch angles are in radians and they use the right-hand frame of reference.

We use a custom developed software framework for delivering the correction signal received from the Wiimote by the host computer to the robot over wireless Ethernet connection as fast as possible (Figure 4.8). The custom software also provides the demonstrator an interface to define scaling and shifting operators on the received signals from the Wiimote before transmission to the robot, allowing the demonstration signal to be scaled up or down. By scaling down the demonstration signals, it is possible to reduce the undesirable noise factors like trembling hands of the demonstrator. The position of the Wiimote which is connected to the host computer over a Bluetooth connection is sampled and the processed demonstration signals are transmitted to the robot over a UDP connection via wireless network at a frequency of 1KHz; therefore, even if some of the UDP packets are dropped due to the network conditions, we can still deliver the demonstration signal packets at around 50Hz, which is the update frequency for the sensors and the actuators of the robot. A snapshot of the custom software is given in Figure 4.9.

The custom user interface allows a joint-wise or the defined correction signal wise definition of scaling coefficients and fixed offsets to make the demonstration process easier for the teacher. The teacher can associate a correction source for each correction signal. In this study, we only used Wiimote signals as the correction sources but the interface supports all generic joysticks and game pads to be used as the correction source. Defining a scaling coefficient smaller than 1.0 allows the user to reduce the noise injected by the shaking hands or the changes in the Wiimote handles

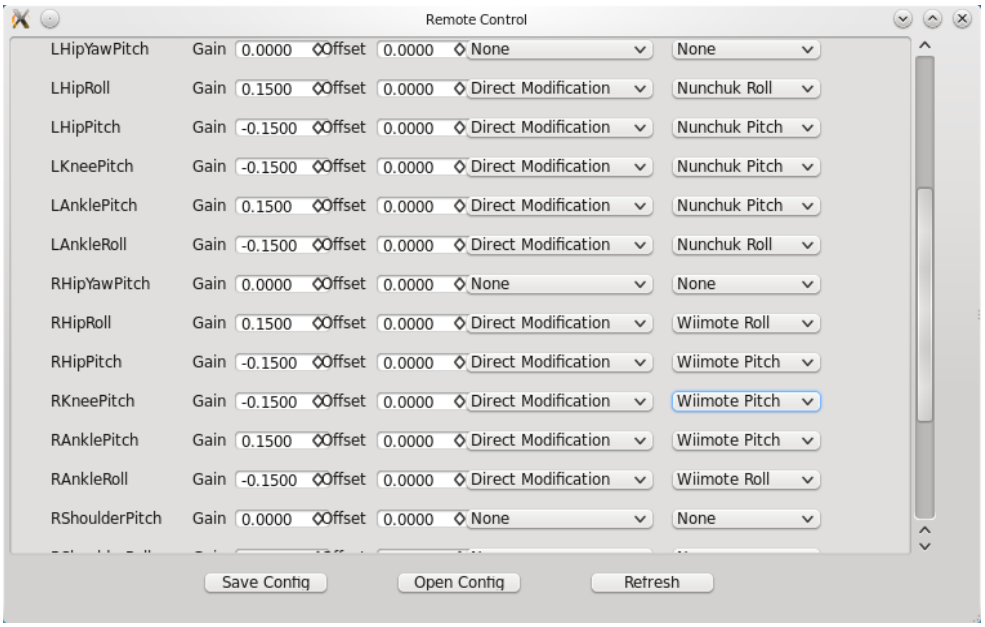


Figure 4.9. A snapshot from the software developed for delivering real-time corrective demonstration to the robot.

due to the teacher walking while chasing the robot. All coefficient and correction source associations are stored in a config file and the software supports multiple config files to make it possible for the teacher to try and evaluate different correction configurations easily.

We define two different methods for converting the Wiimote signals into the correction signals to be applied on the robot:

- Applying correction signals in the joint space by means of direct modifications to the joint commands.
- Applying correction signals in the task space by means of feet position displacements.

The demonstrator delivers the corrective demonstration signals to the robot by changing the orientations of the Wiimote and the Nunchuk controllers in real time while the robot is walking using the open-loop walk cycle. We record the received correction signals during the demonstrations synchronously with the rest of the sensor readings at 50Hz. The Nunchuk extension and the Wiimote control the left and the

right side corrections on the robot, respectively (Figure 4.10). A loose baby harness is used to prevent possible hardware damage in case of a fall. The harness neither affects the motions of the robot nor lifts it as long as the robot is in an upright position.

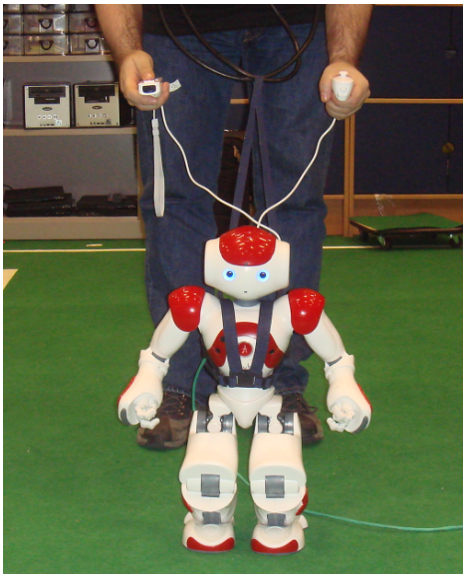


Figure 4.10. A snapshot from a demonstration session.

4.3.2. Applying Correction in the Joint Space

In this correction method, we associate the four correction signals received from the demonstrator to the four individual joints on the hip. Namely, we use the hip roll and the hip pitch joints to apply the correction signals. To provide a finer control ability to the demonstrator, a scaling factor γ is applied on the Wiimote readings using the interface described above for scaling the demonstration signals before they are transmitted to the robot. We used $\gamma = 0.1$ in our implementation. The received roll corrections are applied on the hip roll joints and the received pitch corrections are applied on the hip pitch joints. To keep the feet parallel to the ground, the following correction values are applied on the ankle roll and the ankle pitch joints:

$$C_{AnkleRoll} = -C_{HipRoll} \tag{4.1}$$

$$C_{AnklePitch} = -C_{HipPitch} \tag{4.2}$$

At each timestep, we compute the correction values for all joints $j \in Joints$ using the defined correction functions. We then add the calculated values to the joint command values in the walk cycle for that timestep before sending the joint commands to the robot. The correction is applied to the system at each m^{th} timestep where $1 \leq m \leq T$ where T is the length of the walk cycle in timesteps.

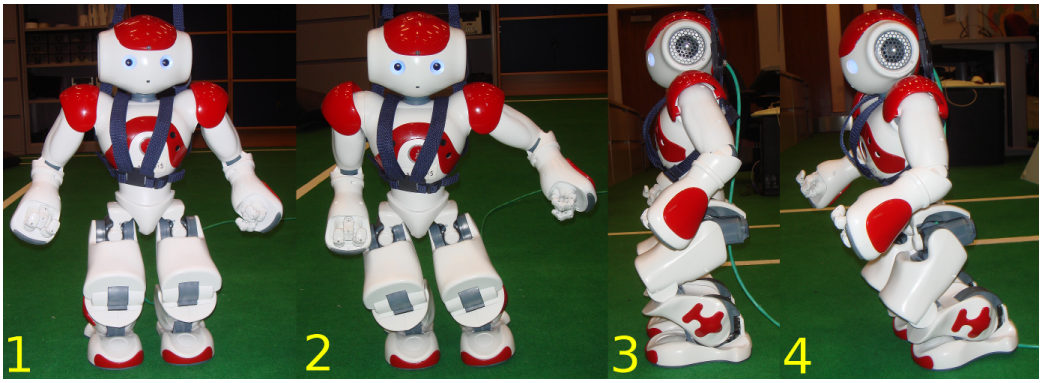


Figure 4.11. Applying correction in the joint space.

In Figure 4.11, rolling the Wiimote to the right transitions the robot from its neutral posture (1) to a posture bent along the Y axis (2). Similarly, tilting the Wiimote forward transitions the robot from its neutral posture (3) to a posture bent along the X axis (4).

4.3.3. Applying Correction in the Task Space

In this correction method, we modify the feet positions in the 3D space with respect to the torso center by mapping the received correction values to the offsets along the X-Y plane instead of applying the correction signals directly to the joints. At each timestep of playback, the vector of joint command angles for that timestep is used to calculate relative positions of the feet in 3D task space with respect to the torso using forward kinematics. The calculated corrections (in the autonomous mode), or the received corrections (during the demonstration) are applied on the feet positions in 3D space and the resulting feet positions are converted back into a vector of joint command angles using inverse kinematics and sent to the robot (Figure 4.12). Here, rolling the Wiimote to the right takes the right leg of the robot from its neutral posture (a) to a modified posture along the Y axis (b). Similarly, tilting the Wiimote

forward brings the right leg of the robot from its neutral posture (c) to a modified posture along the X axis (d).

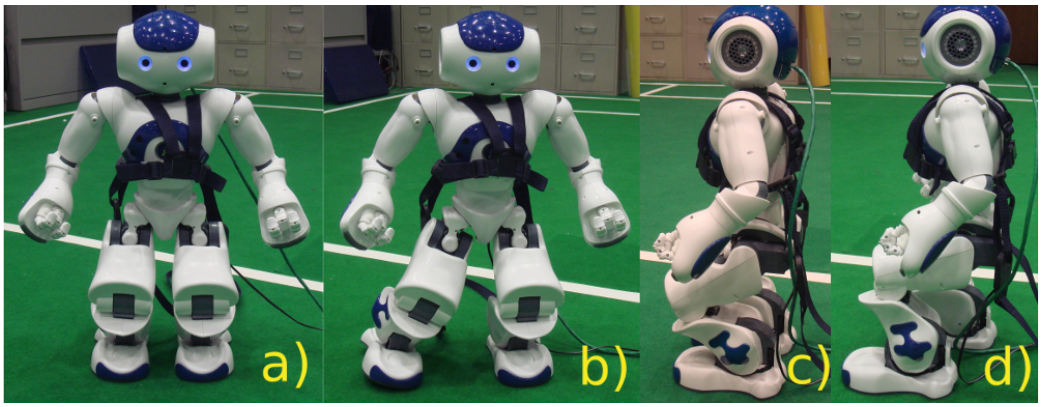


Figure 4.12. Applying correction in the task space as feet position displacement.

Due to the physically coupled hip-yaw joints of the Nao, inverse kinematics for feet positions cannot be calculated independently for each foot. Graf *et al.* propose an analytical solution to inverse kinematics of the Nao, presenting a practical workaround for the coupled hip-yaw pitch joints constraint [32]. We used a simplified version of this approach by assuming the hip-yaw joints to be fixed at 0 degrees for the straight walk. The desired position *Pos* of the foot with respect to the hip joints is given in the form of a homogeneous transformation matrix.

We assume a stick figure model for the feet as shown in Figure 4.13. The thigh and the tibia (upper and lower leg parts) form a triangle with the imaginary edge d_{foot} which represents the distance of the foot from the hip. This distance equals the magnitude of translation vector $\mathbf{t}$ and can easily be calculated as $d_{foot} = |\mathbf{t}|$. The angle β between the upper and lower leg parts can be calculated using the law of cosines.

$$d_{foot}^2 = l_{thigh}^2 + l_{tibia}^2 + 2 l_{thigh} l_{tibia} \cos\beta \tag{4.3}$$

$$\beta = \arccos \frac{l_{thigh}^2 + l_{tibia}^2 - d_{foot}^2}{2 l_{thigh} l_{tibia}} \tag{4.4}$$

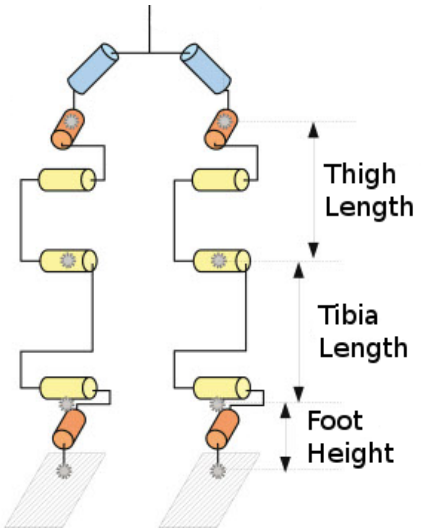


Figure 4.13. Kinematic configurations for the legs of the Nao robot.

where l_{thigh} and l_{tibia} are the length of the thigh and the tibia, respectively. When the leg is fully extended, the knee pitch joint angle $\alpha_{KneePitch} = 0$; therefore, the resulting angle for knee pitch is calculated as $\alpha_{KneePitch} = \pi - \beta$. The angle between lower leg and foot plane constitutes the first part of the final ankle pitch angle and can be computed by the law of cosines.

$$\gamma = \arccos \frac{l_{tibia}^2 + d_{foot}^2 - l_{thigh}^2}{2 l_{tibia} d_{foot}} \tag{4.5}$$

The second part of the ankle pitch angle is calculated using the components of the translation vector

$$\theta = atan2(t_x, \sqrt{t_y^2 + t_z^2}) \tag{4.6}$$

where, $atan2(y, x)$ calculates the angle between the X axis, and the point (x, y).

The final ankle pitch angle value is the sum of its two components; that is, $\alpha_{AnklePitch} = \gamma + \theta$. The hip roll angle value is also calculated using the translation vector. Similar to the hip pitch joint, its final angle value is equal to the exterior angle value; that is, $\alpha_{HipRoll} = \pi - atan2(t_y, t_z)$. The value of the ankle roll angle is the difference between the desired absolute orientation of the foot along the X axis (calculated using the rotation matrix part of Pos), and the calculated hip roll joint angle value

$$\alpha_{AnkleRoll} = \arcsin(p_{32}) - \alpha_{HipRoll} \tag{4.7}$$

where p_{32} is the third row and the second column of Pos. Finally, the hip pitch angle value is calculated as

$$\alpha_{HipPitch} = -(\alpha_{KneePitch} + \alpha_{AnklePitch}) \tag{4.8}$$

Any given valid joint command vector satisfying the assumptions stated at the beginning of this subsection can be converted into the relative positions of the feet in the 3D task space using the method described above.

4.4. Closed-Loop Walking Using Playback And Corrective Demonstration

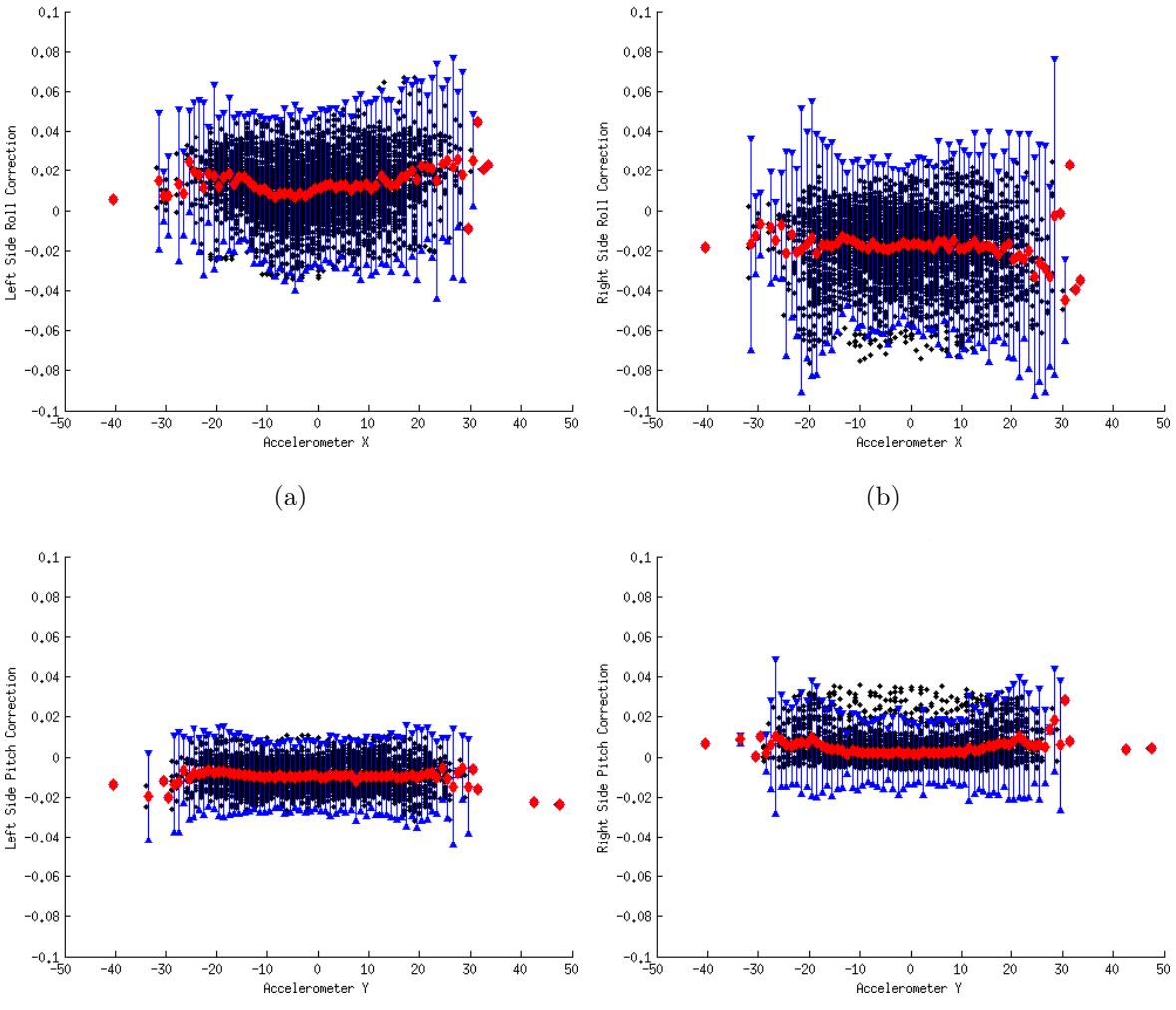
With the correction methods described in the previous section, we can collect demonstration data consisting of the sensory readings representing the state of the system as it is perceived by the robot, and the correction values provided by

the demonstrator based on his/her observation of the state of the robot. To obtain a closed-loop gait, we need a function that maps the sensory readings to the corresponding demonstration values so that we can use that mapping to infer the appropriate correction values to be applied for a given sensory reading. We present two different association methods:

- Associating a single sensor with joint space correction
- Associating multiple sensors with task space correction

4.4.1. Associating a Single Sensor with Joint Space Correction

In this method, we apply the correction on individual joints, and we define the correction value for a joint as a function of a single sensor reading. We use the accelerometer readings along the X and Y axes as the sensory input. Each point in the resulting demonstration dataset is a tuple $< \overrightarrow{S}, \overrightarrow{C} >$ where $\overrightarrow{S} = \{Acc_X, Acc_Y\}$ is the vector of accelerometer readings, and $\overrightarrow{C} = \{C_{roll}^{left}, C_{pitch}^{left}, C_{roll}^{right}, C_{pitch}^{right}\}$ is the vector of received correction values for the left hip roll, the left hip pitch, the right hip roll, and the right hip pitch joints, respectively. The accelerometers on the Nao can measure accelerations in the range $[-2g, 2g]$ where g is the standard gravity and their readings are integer values in the interval $[-128, 127]$. To model the noise associated with the demonstration data, we fit a normal distribution on the correction data points received for all 256 values of the accelerometer. The resulting distributions versus the accelerometer readings populated using approximately 6000 correction points out of about 30000 points recorded in a single demonstration session of roughly 10 minutes are given in Figure 4.14. In the figure, a), b), c), and d) shows Acc. X vs. left side roll, Acc. X vs. right side roll, Acc. Y vs. left side pitch, and Acc. Y vs. right side pitch, respectively. In each subfigure, the bold points in the middle denote the mean, and vertical lines denote the variance of the normal distribution fit on that sensor value interval.



(a) (b)

(c) (d)

Figure 4.14. The normal distributions fit on the received correction data versus the accelerometer readings for the single sensor - joint space correction association.

Any discontinuity or a sudden change in the correction signal causes a jerky movement of the robot and further contributes to the loss of balance. To deal with it, the correction is modified to be a mapping from the sensory reading to the mean of each joint command to be corrected, namely, the left hip roll, the left hip pitch, the right pitch, and the right hip roll. During autonomous execution, given the perceived sensory data, the corresponding mean value is added to the walk cycle commands. The computed correction values are applied to the walk cycle commands at each N^{th} timestep, where N is a predefined value that does not change during the execution. The pseudo-code of that process is given in Figure 4.15.

```
1: t ← 0
2: loop
3:    S⃗ ← readSensors()
4:    S⃗ ← smoothen(S⃗)
5:    for all j ∈ Joints do
6:       if timestep MOD correctioninterval = 0 then
7:          C_j = Correction(S⃗, j)
8:       else
9:          C_j = 0
10:      end if
11:      NextAction_j ← wc_j(t) + C_j
12:   end for
13:   t ← t + 1   (mod T)
14: end loop
```

Figure 4.15. Algorithm for closed-loop walking using single sensor-joint space correction association.

In addition, we defined a hand-tuned simple linear function to be used as a benchmark closed-loop gait in our experiments. We use the roll and the pitch angles of the torso, calculated by the inertial measurement unit as the sensor readings and associate them with the hip roll and the hip pitch joints. The inertial measurement unit returns the roll and pitch orientations of the torso in radians with respect to the ground. The used linear coupling functions are of the form $C = AX + B$ where A is a gain value, B is an offset value, X is the sensor reading, and C is the calculated correction value. For the four hip joints to be corrected, we have four functions with individually set A and B values. We hand-tuned the parameters of these four functions using expert knowledge and visual observation of the robot walking. The resulting hand-tuned policy provided an improvement over the initial open-loop walk.

Details of the results are given in Section 4.5.

4.4.2. Associating Multiple Sensors with Task Space Correction

In this method, the correction values received during the demonstration are recorded synchronously with the sensory readings, tagged with the current position in the walk cycle. Each point in the resulting demonstration dataset is a tuple $< \overrightarrow{S}, \overrightarrow{C} >$, where

$$\overrightarrow{S} =< t, Acc_X, Acc_Y >$$

is the state vector of consisting of the position in the walk cycle at the time when this correction is received, and accelerometer readings, and

$$\overrightarrow{C} =< C_X^{left}, C_Y^{left}, C_X^{right}, C_Y^{right} >$$

is the vector of received correction values for the left foot along the X axis, the left foot along the Y axis, the right foot along the X axis, and the right foot along the Y axis, respectively.

We utilize locally weighted regression with a Gaussian kernel [33] for generalizing a policy using the recorded correction and sensor values. For each received sensor reading vector $\overrightarrow{S}$, we calculate the correction vector $\overrightarrow{C}$ as follows:

$$d_i = e^{-\sqrt{(\overrightarrow{S}-\overrightarrow{S_i}(t))^T \Sigma^{-1} (\overrightarrow{S}-\overrightarrow{S_i}(t))}} \tag{4.9}$$

$$\overrightarrow{C} = \frac{\sum_i d_i \overrightarrow{C_i}(t)}{\sum_i d_i} \tag{4.10}$$

where Σ is the covariance matrix of the sensory readings in the demonstration set, $\overrightarrow{C_i}(t)$ is the i^{th} received correction signal for the walk cycle position t, $\overrightarrow{S_i}(t)$ is the i^{th} sensory reading for the walk cycle position t, $\overrightarrow{S}(t)$ is the current sensory reading, $\overrightarrow{C}$ is the calculated correction value to be applied, and t is the current position in the walk cycle.

The calculated correction values are applied only if any of the sensor values are not in the range $\mu_t \pm K\sigma_t$ (i.e., if an abnormal value is read from that sensor, meaning that the robot is losing its balance) where K is a coefficient, and t is the current position in the walk cycle. In our implementation, we chose $K = 3$ so the correction values are applied only if the current sensory readings are outside the range $\mu_s(t) \mp 3\sigma_s(t)$, corresponding to the %99 of the variance of the initial sensory model given in Section 4.2.1. The pseudo-code for multiple sensors - feet position displacement association is given in Figure 4.16. Here, Pos_{left} and Pos_{right} are the positions of the feet in 3D space.

```
1: t ← 0
2: loop
3:    S(t)⃗ ← readSensors()
4:    S(t)⃗ ← smoothen(S(t)⃗)
5:    Pos_left, Pos_right ← forwardKine(wc(t))
6:    if (μ_s(t) − Kσ_s(t) ≤ S_s(t) ≤ μ_s(t) + Kσ_s(t)) then
7:       C_left, C_right ← 0
8:    else
9:       C_left, C_right ← correction(S(t)⃗)
10:   end if
11:   Pos_left ← Pos_left + C_left
12:   Pos_right ← Pos_right + C_right
13:   NextAction ← inverseKine(Pos_left, Pos_right)
14:   t ← t + 1  (mod T)
15: end loop
```

Figure 4.16. Algorithm for closed-loop walking using multiple sensors-task space correction association.

4.5. Experimental Evaluation

To evaluate the performance of the proposed methods, we conducted a set of walking experiments on a flat surface covered with carpet. We used the walking

algorithm proposed by Liu and Veloso as the black-box algorithm [34]. The duration of the extracted walk cycle is 52 individual timesteps, approximately corresponding to one second.

We evaluated different combinations of the proposed correction, sensory association, and policy derivation methods as follows:

- *Case (a) :* Initial open-loop playback walk (OL).
- *Case (b) :* Closed-loop playback walk with the joint space correction policy (JS) using hand-tuned (HT) single sensor-correction association (SS) on top of the original open loop walk cycle (OL), and the fixed frequency application of the correction (FC) twice a walk cycle ($N = 26$).
- *Case (c) :* Closed-loop playback walk with the joint space correction policy (JS) using single sensor-correction association (SS), normal distribution fit (NF) on top of the original open loop walk cycle (OL), and the fixed frequency application of the correction (FC) twice a walk cycle ($N = 26$).
- *Case (d) :* Open-loop playback walk cycle (OL) after offline improvement using advice operators (AO).
- *Case (e) :* Closed-loop playback walk with the task space correction policy (TS) using multiple sensors - correction association (MS), locally weighted regression (LWR) as the policy extraction method on top of the advice improved walk cycle (OL+AO), and the application of the correction under state anomaly (AC), in other words, when the sensory readings go beyond the $\pm 3\sigma$ of the normal sensory readings (Figure 4.3).

We used two benchmark combinations (Case (a) and Case (b)), the former being the base case and the latter being a simple closed-loop method with hand tuned parameters as described in Section 4.4.1. For each combination, we performed 10 runs and measured the distance traveled before falling. The results are given in Figure 4.17 as boxplots, where the lines within the boxes mark the mean, the marks at both ends of boxes indicate minimum and maximum distances, and the left and

right edges of boxes mark 25^{th} and 75^{th} percentiles, respectively.

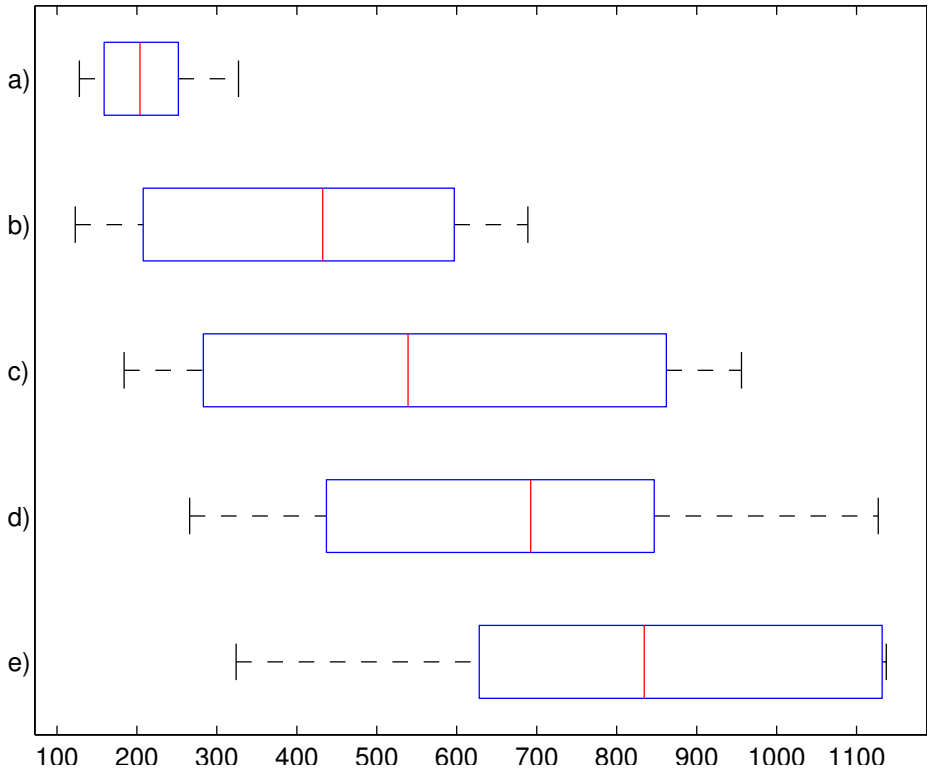


Figure 4.17. Performance evaluation results for the biped walk improvement problem.

During three demonstration sessions of 28 minutes, a total of about 83000 demonstration points are recorded for both joint space and task space corrections, and 25428 of them corresponding to about 489 walk cycles are selected as good examples of corrective demonstration by visually inspecting the demonstration data based on the changes in the sensory readings towards the recovery of balance.

The mean and maximum distances that the robot could travel using the initial open loop benchmark walk (Case (a)) were 203 and 327 centimeters, respectively, while the mean and the maximum distances the robot was able to travel using the closed loop benchmark walk (Case (b)) were 432 and 689 centimeters, respectively. The performance difference between the two benchmark cases stems from the fundamental difference between the open-loop and the closed-loop control paradigms under the presence of uncertainty and noise in the environment.

All of the combinations involving proposed methods outperformed the benchmark cases Case (a) and Case (b). Case (c), which is directly comparable to Case (b) demonstrated considerable improvement over the latter, reaching a maximum traveled distance of 956 centimeters with a mean traveled distance of 539 centimeters[1] . The improvement in the performance could be accounted for the non-linear relationship between the computed means for the received correction and the accelerometer readings (as seen in Figure 4.14) of which the assumed linear relation function in the hand tuned case was unable to capture appropriately.

The open-loop walk improved with advice operators (Case (d)) performed surprisingly well and outperformed the closed-loop Case (c), reaching a maximum traveled distance of 1127 centimeters with a mean traveled distance of 692 centimeters. During the advice operator improvement, the teacher continuously observes the robot and gives high level advice which corresponds to a systematic correction to the walk cycle. Taking a closer look at the mean correction values in Figure 4.14, we see that the mean values are off from the zero position by a fixed offset in addition to the nonlinear relation of the sensory readings to the received correction value. These offsets are results of the implicit high level correction of the same systematic error by the demonstrator. An explanation for why the improved open loop walk did better compared to Case (c) could be that it is easier to focus on the *"big picture"* and hence to spot the systematic error when the teacher is solely observing the robot rather than being actively involved in delivering real-time correction to the robot.

Despite the fact that the application of the advice operators on the walk cycle resulted in a considerably improved walk performance with the maximum and mean traveled distances of 1137 and 834 centimeters, respectively, the last case (Case (e)) shows that there is still room for improvement with the real-time corrective demonstration over the improved open-loop walk. The maximum distance of 1137 centimeters was the length of the available experimentation area and the Case (d) combination was able to reach this limit three times out of 10 runs.

[1]The open-loop walk performance was comparable to the performance of the original ZMP-based walk, which was not available to be accounted for in this empirical comparison.

4.6. Discussion

In this chapter, we presented an approach for learning a correction policy for improving the walk stability of the Nao humanoid robot using corrective human demonstration. We analyzed the Nao robot in terms of the variations of joint commands and sensor readings. The key question we tried to answer was whether it would be possible to improve the performance of an existing controller for performing a complex skill on a complex robotic platform without knowing the underlying technical details of the existing controller. We tackled the problem by making use of a human teacher who is able to externally observe the robot performing the skill using the existing controller. We utilized corrective human demonstration given in two phases (first offline and then in real-time) to learn a policy for modifying the joint commands in the open-loop walk cycle during the autonomous execution in such a way to keep the robot balanced.

Although the results suggest that more complex options for the correction type (task space correction instead of joint space correction), sensor-correction association (multiple sensors - task space correction association instead of single sensor - joint space correction association), policy derivation method (locally weighted regression for the individual timesteps within the walk cycle instead of fitting normal distributions for the whole walk cycle), and the application of the correction (applying correction only if the current perceived state differs from the normal values instead of applying correction at each N^{th} timestep regardless of the sensory readings) yielded better performance, we do not possess enough experimental evidence to claim such superiority.

5. TASK REFINEMENT USING M+C

In this chapter, we present an application of the M+C approach for task execution refinement where a hand-coded algorithm for performing the task exists but is inadequate in handling complex cases. The human demonstrator observes the robot carry out the task by executing the hand-coded algorithm and provides corrective feedback when the hand-coded controller computes a wrong action. The received demonstration actions are stored along with the state of the robot at the time of correction as complements (or "patches") to the base hand-coded algorithm. During autonomous execution, the robot substitutes the action computed by the hand-coded algorithm with the demonstrated action if the corrective demonstration history database contains a demonstration provided in a similar state. The key idea is to keep the base controller algorithm as the primary source of the action policy, and use the demonstration data as exceptions only when needed instead of deriving the entire policy out of the demonstrations and the output of the controller algorithm. We applied this approach to a complex ball dribbling task in the humanoid robot soccer domain.

5.1. Problem Definition

Technical challenges are held as a complementary part of the RoboCup SPL competitions with the aim of creating a research incentive on complex soccer playing skills that will help leverage the quality of the games and enable the league to gradually approach the level of real soccer games both in terms of the field setup and the game rules. Each year, the technical challenges are determined accordingly by the Technical Committee of RoboCup SPL.

Our application and evaluation domain, the "Dribbling Challenge", was one of the three technical challenges of the 2010 SPL competitions. In that challenge, an attacker robot is expected to score a goal in three minutes without having itself or the

ball touching any of the three stationary defender robots that are placed on the field in such a way to block the direct shot paths. The positions of the obstacle robots are not known beforehand; therefore, the robot has to detect the opponent robots, model the free space on the field, and plan its actions accordingly. An sample scenario is illustrated in Figure 5.1.

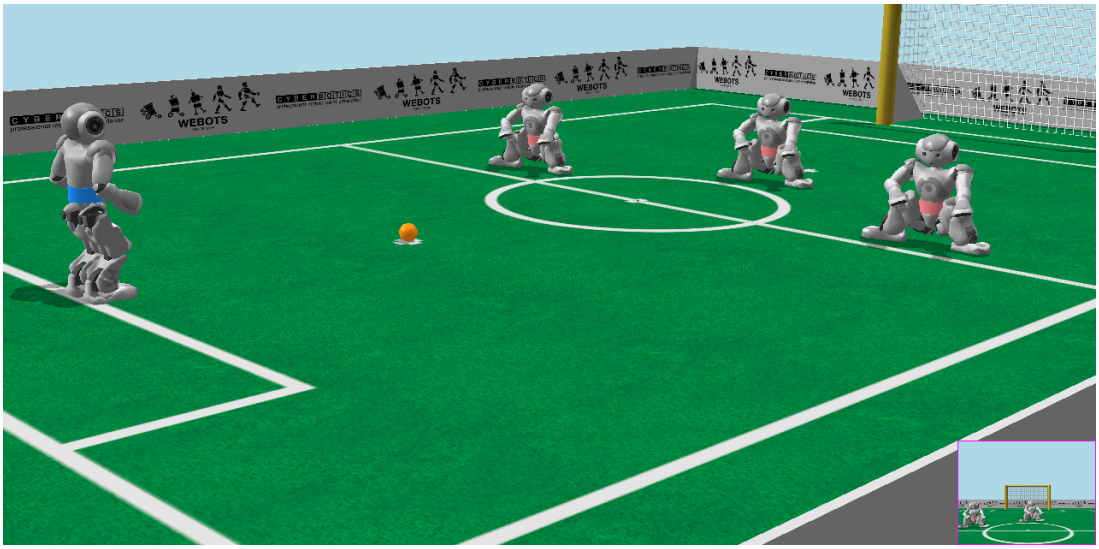


Figure 5.1. An example scenario for the dribbling challenge.

5.2. Modeling as a M+C Instance

5.2.1. State and Action Definitions

In the ball dribbling task, we use the free-space model built by processing a number of consecutive camera images taken during the robot scanning the field with a pan motion. We condense the perceived free-space information about the immediate surroundings of the 180^o area in front of the robot into 15 slots, each covering 12^o. We represent the state as a vector of 15 integers, each representing the distance to the nearest obstacle detected along that free-space slot, or the maximum free distance if no obstacle is detected. We also keep a boolean flag indicating whether it is facing towards the opponent goal or not. As a result, we define the state vector as $\overrightarrow{S} =< dist_1, dist_2, ..., dist_{15}, goal_1, goal_2, ..., goal_{15} >$.

The action definition has two stages. First, we define the first level actions as $A_{first} = \{Shoot, Dribble\}$. For the dribble action, we have a second set of actions for representing the dribble direction. We define the second level actions as $A_{second} = \{dir_1, dir_2, ..., dir_{15}\}$. We assume a fixed dribble distance of 100 cm.

5.2.2. The Model

For the model component, we employ two simple algorithms. The first algorithm is for deciding whether to take a direct shot on goal, or to dribble the ball to a location on the field more suitable for a direct shot. This algorithm simply calculates the differences between the distances of the slots facing towards the opponent goal and that of the robot to the goal. If the average distance is below a certain threshold, $A = Shoot$ is selected. If the distance is above that threshold, or none of the free-space slots face the opponent goal, $A = Dribble$ is selected.

The second algorithm is utilized whenever $A = Dribble$ is selected. For dribble direction selection, we go over each slot and we compute a weighted average distance for each. If none of the free-space slots face towards the goal, the slot with the closest direction difference from the goal is selected regardless of the occupancy status of that slot.

5.2.3. The Correction

During the demonstration session, corrective demonstration examples in the form of state-action pairs are collected and stored in a database individually. We do not learn a model for the gathered demonstration data. The teacher can correct the actions at both the first level and the second level. In case of correcting an erroneous *Shoot* action with a *Dribble* action, the teacher can either provide a dribble direction as well, or can leave it to the dribble direction selection algorithm in the *model* component.

5.2.4. Correction Reuse

The correction reuse component determines the next action to be performed by selecting an action among the actions computed by the model and the correction components. During the autonomous execution of the task, the robot looks for a replacement action whenever it reaches a decision state and computes an action using the algorithms of the *model* component. To find a proper replacement action, we search through the demonstration database to see if there is a demonstration in the database that is received when the system was in a state *similar* to the current state of the system. We use a domain specific, hand-coded state similarity measure that computes the similarity by overlapping the goal slots and then computes an average of absolute distances. We then apply a Gaussian kernel to the computed raw distance measure to compute the state similarity value. The state representation for the correction reuse component is the same representation used in the other components of the system.

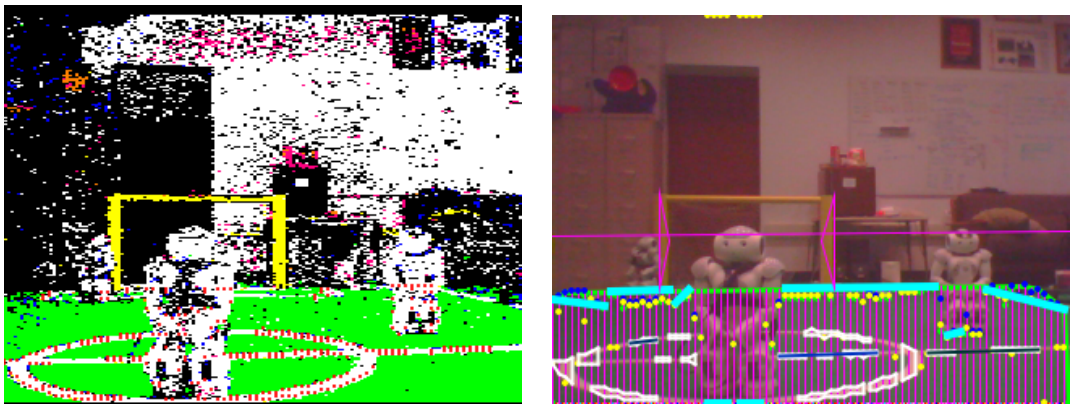
5.3. Free Space Modeling using Vision

Instead of trying to detect the defender robots and avoid them, our attacker robot detects the free space in front of it and builds a free space model of its surroundings to decide which direction is the best to dribble the ball towards. The soccer field is a green carpet with white field lines on it. The robots are also white and gray, and they wear pink or blue waist bands as uniforms (Figure A.1(b), Figure 5.1). Therefore, anything that is non-green and lying on the field can be considered as an obstacle, except for the detected field lines. We utilize a simplified version of the *Visual Sonar* algorithm by Lenser and Veloso [35] and the algorithm by Hoffmann *et al.* [36]. We scan the pixels on the image along evenly spaced vertical lines called *scanlines*, starting from the bottom end and continue until we encounter a certain number of non-green pixels. Although the exact distance of a certain pixel from the robot is a function of the position of the camera, in general the distance to a pixel increases as we ascend from the bottom of the image to the top, assuming all the pixels

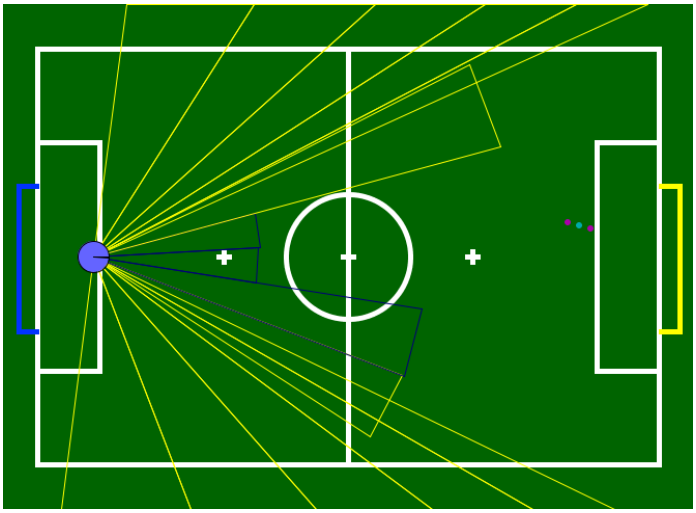
lie on the ground plane. If we do not encounter any green pixels along a scanline, we consider that scanline as fully occupied. Otherwise, the point where the non-green block starts is marked as the end of the free space towards that direction. To further save some computation time, we do not process every vertical line on the image. Instead, we process the lines along every fifth pixel and every other pixel along those lines. As a result, we effectively process only $1/10^{th}$ of the image (Figure 5.2(b)). The pixels denoting the end of the free space are then projected onto the ground to have a rough estimate of the distance of the corresponding obstacle in the direction of the scanned line. In order to cover the entire 180^o space in front of it, the robot pans its head from side to side. As the head moves, the computed free space end points are combined and divided into 15 slots, each covering an arc of 12^o in front of the robot. In the mean time, each free space slot is tagged with a flag indicating whether that slot points towards the opponent goal or not based on the location of the opponent goal in the world model, or the estimated location and orientation of the robot on the field (Figure 5.2(c)). Here, the dark triangles indicate the free space slots pointing towards the opponent goal.

5.4. Ball Dribbling Behavior

We use a Finite State Machine (FSM) based behavior system for developing the ball dribbling behavior. The FSM structure of the ball dribbling behavior is depicted in Figure 5.3. The robot starts with "searching for the ball" by panning its head from side to side several times using both cameras. If it cannot find the ball at the end of this initial scan, it starts turning in place while tilting its head up and down, and this cycle continues until the ball is detected. Once the ball is located on the field, "approach the ball" behavior gets activated and the robot starts walking towards the ball. Utilizing the omni-directional walk, it is guaranteed that the robot faces the ball when the "approach the ball" behavior is executed and completed successfully. After reaching the ball, the robot pans its head one more time to gather information about the free space around it, calculates its current state, selects an action that matches its state, and finally kicks the ball towards a target point computed according to the



(a) (b)



(c)

Figure 5.2. The environment as perceived by the robot: a) the color segmented image, b) the computed perceived free space segments, and c) the resulting free space model.

selected action. If the robot loses the ball at any instant of this process, it goes back to the "search for the ball" state.

Except for the lightly colored *select action* and *select dribble direction* states shown on the state diagram in Figure 5.3, each state in the given FSM corresponds to a low level skill. We use the existing low level skills in our robot soccer system without any modifications; namely, looking for the ball, approaching the ball, lining up for a kick, and kicking the ball to a specified point relative to the robot by selecting an appropriate kick from the portfolio of available kicks.

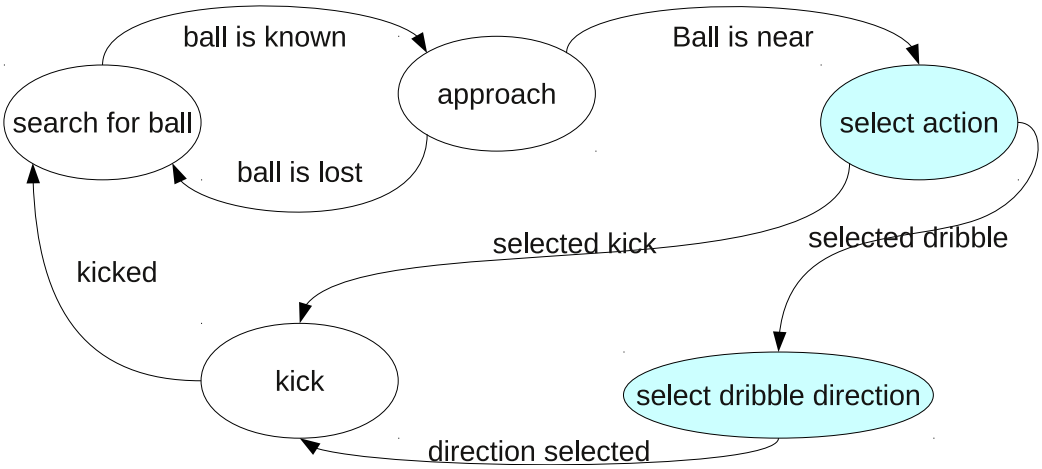


Figure 5.3. The state diagram of the ball dribbling behavior.

5.5. Action and Dribble Direction Selection

The *select action* and the *select dribble direction* states constitute the main decision points of the system we aim to improve using corrective demonstrations. The hand-coded algorithms for both the action and the dribble direction selection parts utilize the free space model in front of the robot. After lining up with the ball properly, the robot selects one of the following two actions:

- The *shoot* action corresponds to kicking the ball directly towards the opponent goal using a powerful and long range kick.
- The *dribble* corresponds to dribbling the ball towards a more convenient location on the field using a weaker and shorter range kick.

When the robot reaches the decision point; that is, after it aligns itself with the ball and scans the environment for free space modeling, the action selection algorithm checks if any of the free space slots pointing towards the opponent goal has a distance less than a certain fraction of the distance to the goal. If so, the path to the opponent goal is considered *"occupied"* and the *dribble* action is selected in that situation. Otherwise, the path is considered *"clear"* and the *shoot* action targeting the center of the opponent goal is selected. The pseudo-code of the action selection algorithm is given in Figure 5.4. In the algorithm, $\Gamma \in [0, 1]$ is a coefficient

for specifying the maximum distance to be considered as free space in terms of the distance of the goal. In our implementation, we use $\Gamma = 0.5$.

```
1: goalDist ← getGoalDist()
2: goalAngle ← getGoalAngle()
3: if goalAngle < −π/2 or goalAngle > π/2 then
4:    return dribble
5: else
6:    for all i ∈ getGoalSlots() do
7:       distDiff ← |goalDist − dist_i|
8:       if distDiff > Γ goalDist then
9:          return dribble
10:      end if
11:   end for
12: end if
13: return shoot
```

Figure 5.4. Action selection algorithm for the ball dribbling task.

If the action selection algorithm deduces that the path to the opponent goal is blocked and subsequently selects the dribbling action, a second algorithm steps in to determine the best way to dribble the ball. All slots in the free space model are examined and assigned a score computed as the weighted sum of the distance values of the slot itself and its left and right neighbors. The free space slot with the maximum score is selected as the dribble direction. The algorithm for dribble direction selection is given in Figure 5.5. In the algorithm, N denotes the number of free space slots.

Using the two algorithms explained above for the two action selection states in the behavior FSM, the robot is able to perform the ball dribbling task and score a goal with limited success. We define the success metric for this task to be the time it takes for the robot to score a goal. The performance evaluation results for the hand-coded action selection algorithms are provided in Section 5.7. In the following section, we present the corrective demonstration system developed as a complement to the hand-coded action selection algorithms for refining the task performance.

```
1: goalAngle ← getGoalAngle()
2: if goalAngle < −π/2 or goalAngle > π/2 then
3:    if |angle_0 − goalAngle| < |angle_{N−1} − goalAngle| then
4:       dribbleAngle ← angle_0
5:    else
6:       dribbleAngle ← angle_{N−1}
7:    end if
8: else
9:    maxDist ← 0
10:   for slot ← 1; slot < N − 1; slot ← slot + 1 do
11:      distance ← 0.25dist_{slot−1} + 0.5dist_{slot} + 0.25dist_{slot+1}
12:      if distance > maxDist then
13:         maxDist ← distance
14:         maxSlot ← slot
15:      end if
16:   end for
17:   dribbleAngle ← angle_{maxSlot}
18: end if
19: return dribbleAngle
```

Figure 5.5. Dribble direction selection algorithm for the ball dribbling task.

5.6. Corrective Demonstration

In our approach, we store the collected corrective demonstration points separately from the hand-coded controller, and utilize a reuse algorithm to decide when to use correction. In the following subsections, we first describe how the corrective demonstration is delivered to the robot, and then we explain how the stored corrections are used during autonomous execution.

5.6.1. Correction Delivery

The teacher uses a custom developed software to provide corrective feedback to the robot. The user interface visualizes the state of the system as it is observed through the sensors of the robot. The teacher observes the robot both physically and on the visualized state observation while executing the task, and intervenes the execution if the robot miscalculates the next action to be executed. The teacher generates a corrective feedback signal by pressing appropriate buttons on the user interface. The generated feedback signal is then transmitted to the robot over wireless Ethernet connection. The robot replaces the next action to be executed with the

corrected action received from the demonstration interface and stores the corrective feedback signal stamped with the observed state of the system.

5.6.2. Correction Reuse

By the end of the demonstration session, the robot has built a demonstration database of state-action pairs denoting what action is provided by the teacher as a replacement of the action computed by the hand-coded algorithms and what was the robot's state when that correction is received. During autonomous execution, the decision of when to execute the action selected by the hand-coded algorithms and when to use corrective demonstration samples is made by a correction reuse system based on the similarity of the current state of the robot to the states in which the demonstration samples were collected.

We define the observed state of the robot as

$$Z =< slotDist_0, ..., slotDist_{N-1}, goal_0, ..., goal_{N-1} >$$

where $slotDist_i$ is the distance to the nearest obstacle inside slot i, and $goal_i \in \{true, false\}$ is a Boolean flag which is set to $true$ if the slot i intersects with the goal, and set to $false$ otherwise.

Since the robot is expected to kick/dribble the ball into the opponent goal, rather than only the position of the robot on the field, the distribution of the free space with respect to the direction towards the goal needs to be taken into account. Therefore, we calculate the sum of the absolute differences of the free space slots using the slot pointing towards the center of the goal as the origin if the goal is in sight. If the goal is not somewhere within the 180^o in front of the robot, we calculate the sum of absolute differences of the free space slots using the rightmost slot as the origin. The similarity value in the range $[0, 1]$ is then calculated as

$$similarity = e^{-K\,diff^2} \tag{5.1}$$

where K is a coefficient for shaping the similarity function, and *diff* is the calculated sum of absolute differences of the slot distances. In our implementation, we selected $K = 5$. The algorithm for similarity calculation is given in Figure 5.6.

```
1: dist_curr ← getSlotDist(Z_curr)
2: dist_demo ← getSlotDist(Z_demo)
3: diff ← 0
4: if goalAngle < −π/2 or goalAngle > π/2 then
5:    for slot ← 0; slot < N; slot ← slot + 1 do
6:       diff ← diff + |dist_curr(slot) − dist_demo(slot)|
7:    end for
8:    diff ← diff/N
9: else
10:   goalSlot_curr ← getGoalSlot(Z_curr)
11:   goalSlot_demo ← getGoalSlot(Z_demo)
12:   num ← 0
13:   s_1 ← goalSlot_curr, s_2 ← goalSlot_demo
14:   while s_1 < N and s_2 < N do
15:      diff ← diff + |dist_curr(s_1) − dist_demo(s_2)|
16:      num ← num + 1, s_1 ← s_1 + 1, s_2 ← s_2 + 1
17:   end while
18:   s_1 ← goalSlot_curr, s_2 ← goalSlot_demo
19:   while s_1 >= 0 and s_2 >= 0 do
20:      diff ← diff + |dist_curr(s_1) − dist_demo(s_2)|
21:      num ← num + 1, s_1 ← s_1 − 1, s_2 ← s_2 − 1
22:   end while
23:   diff ← diff/num
24: end if
25: similarity ← e^{−K diff^2}
26: return similarity
```

Figure 5.6. The algorithm for computing the similarity of two given state vectors.

During autonomous execution, when the robot reaches the action selection or dribble direction selection states, it first checks its demonstration database and fetches the demonstration sample with the highest similarity to the current state. If the similarity value is higher than a threshold value τ, the robot executes the demonstrated action instead of the action computed by the hand-coded algorithm. In our implementation, we use $\tau = 0.9$. The algorithm for autonomous execution using corrective

demonstration is given in Figure 5.7.

```
1: currentState ← computeState(resolution)
2: mostSimilar ← ∅
3: maxSimilarity ← 0
4: for each demonstration ∈ correctionDatabase_resolution do
5:    similarity ← getSimilarity(currentState, demonstration(state)
6:    if similarity > maxSimilarity then
7:       maxSimilarity ← similarity
8:       mostSimilar ← demonstration
9:    end if
10: end for
11: threshold ← getCorrectionThreshold(resolution)
12: if maxSimilarity > threshold then
13:    action ← demonstration(action)
14: else
15:    action ← computeAction(currentState)
16: end if
17: executeAction(action)
```

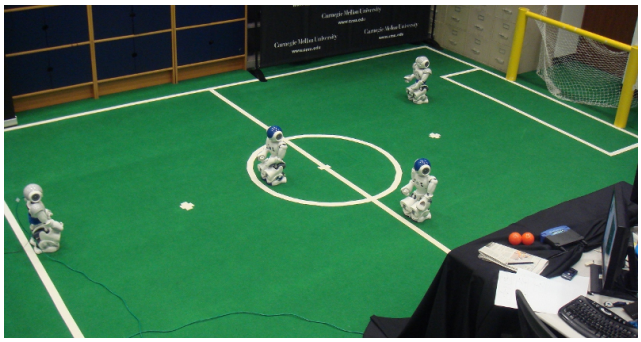
Figure 5.7. The algorithm for autonomous task execution using corrective demonstration.

5.7. Experimental Evaluation

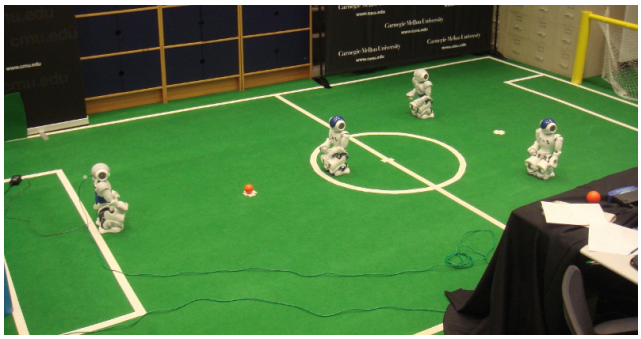
We evaluated the efficiency of the complementary corrective demonstration using three instances of the ball dribbling task with different opponent robot placements in each of them. The test cases were designed in such a way that the robot using the hand-coded action selection algorithm would be able to complete the task, but not through following an optimal sequence of actions (Figure 5.8). The following criteria were kept in mind while designing the test cases:

- Case 1: In this scenario, we place two robots on the periphery of the center circle, leaving a narrow, but passable corridor. The third robot is placed on the virtual intersection of the opponent penalty mark and the left corner of the opponent penalty box. The hand-coded behavior computes the corridor between the two center robots to be too narrow to pass. Therefore, the robot tries to avoid the two robots at the center and mostly chooses a right dribbling direction to avoid the third robot as well. During the demonstration, we advised the robot to take a direct shot between the two robots at the center (Figure 5.9(b)). This scenario

(a)



(b)



(c)

Figure 5.8. Three different configurations used in the experimental evaluation of the M+C system on the ball dribbling task. a) Case 1, b) Case 2, and c) Case 3.

was a good showcase for illustrating how to refine the otherwise imprecise output of a very simple algorithm; no additional complexity were introduced to the algorithm and a limited number of demonstrations were provided only when the robot tried dribbling the ball whereas it could take a direct shot.

- Case 2: In this case, a direct shot is not possible from the initial position, and the robots are placed asymmetrically on the field in such a way that dribbling the ball towards the robot placed further away is advantageous. During the

demonstration, the given advice was to first dribble the ball to the left, and then take a direct shot towards the goal (Figure 5.9(e)). The hand-coded algorithms tend to choose the right action by dribbling the ball to the left, but then the robot decides to advance the ball through a series of dribbles before kicking it into the goal instead of taking a direct shot.

- Case 3: This case was also designed to emphasize the ability of the proposed algorithm to reshape the behavioral response in addition to correcting mistakes. Similar to Case 2, a direct shot is not possible from the initial position, and the robots are placed symmetrically so no clear advantage of choosing one initial dribbling direction over another exists. During the demonstration, we gave a very similar advice to the one we gave in Case 2 to investigate whether we can create a bias towards a specific action in certain cases (Figure 5.9(h)).

We gathered corrective demonstration data from all three cases and formed a common database. A total of 42 action selection and 21 dribble direction selection demonstration points were collected in a roughly 30 minutes long demonstration session. The time required to score a goal being the success measurement metric, we then evaluated the performance of the system with and without the use of the corrective demonstration database.

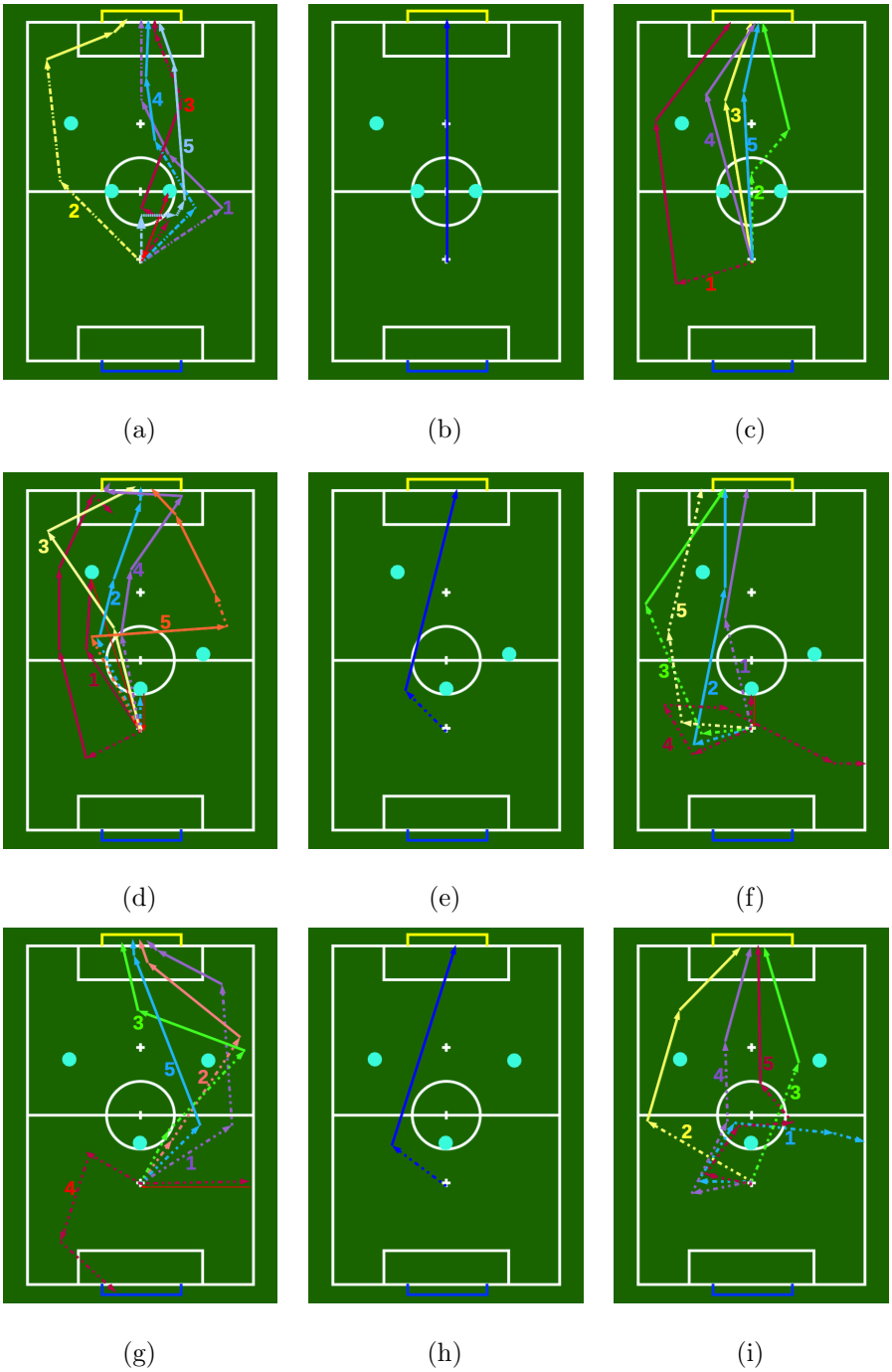
We ran 10 trials for each case, 5 with the hand-coded action and dribble direction selection algorithms (Model), and another 5 trials with the corrective demonstration data (Correction) in addition to the Model (M+C). The sequence of actions taken by the robot at each trial are depicted in Figure 5.9, and the timing information is presented in Table 5.1. In the figures, a dashed line indicates dribble action, a solid line indicates a shoot action, and a thin line indicates the replacement of the ball to the initial position after committing a foul. In the table, “out” means that the robot kicked the ball out of bounds from the sides, “missed” means that the robot chose the right actions but the ball did not roll into the goal due to imperfect actuation, and “own goal” means that the robot accidentally kicked the ball into its own goal. The failed attempts are excluded from the given mean and standard deviation values.

The failures were mostly due to the imperfection of the lower level skills like aligning with the ball, and the high variance in both the kick distance and the kick direction. In the figure, the rows represents (from top) Case 1, Case 2, and Case 3, respectively. The columns represent the *Model*, the *Correction*, and the *Model+Correction*, respectively, where Model stands for hand-coded algorithm and Correction stands for corrective demonstration. Model+Correction shows the cases where the robot is in autonomous mode using both hand-coded algorithm and the corrective demonstration database. In each subfigure, different colors denote different runs. For each run, a dashed line represents a dribble and a solid line represents a kick.

Table 5.1. Elapsed times during trials for the Ball Dribbling Task.

	Case 1		**Case 2**		**Case 3**	
Trial	**M**	**M+C**	**M**	**M+C**	**M**	**M+C**
1	158	95	Ball Out	102	130	Ball Out
2	147	109	211	107	151	117
3	108	92	122	128	144	63
4	156	87	232	Ball Out	Own Goal	113
5	237	91	176	Missed Goal	114	172
mean	**161**	**94**	**185**	**112**	**134**	**116**

The decrease in the timings in all three test cases when using (M+C) compared to the system using the hand coded action selection algorithms (Model) alone shows an improvement in the overall performance since according to the problem definition, the shorter completion times are considered more successful. In Case 1, where the average completion time is reduced by around one minute, the improvement in the task performance was mostly due to the bias created by the corrective demonstration which favors taking direct shots as opposed to the dribbling action computed by the hand-coded algorithm as given in Figure 5.9(c). In Case 2, the complementary corrective demonstration was able to correct the wrong decision made by the hand-coded algorithm on taking a second dribble action instead of a direct shot after dribbling the ball to the left. As a result, the average task completion time was



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 5.9. The illustrations of the performance evaluation runs for the ball dribbling task using M+C approach.

reduced almost to the half of the time it took on average when only the hand-coded algorithm (Model) was used. The effectiveness of corrective demonstration in Case 2 is presented in Figure 5.9(f). In Case 3, the corrective demonstration was again proven to be effective in creating a bias in situations where it is not analytically possible to prefer an action over another. Presenting a preference for dribbling to the left (Figure 5.9(h)), the corrective demonstration was able to change the initial response of the hand-coded algorithm from dribbling the ball to the right (Figure 5.9(g)) to dribbling the ball to the left (Figure 5.9(i)).

6. TASK REFINEMENT USING MRM+C

In this chapter, we present an application of the Multi-Resolution Model Plus Correction (MRM+C) framework to a humanoid obstacle avoidance problem. The MRM+C algorithm allows the teacher to deliver corrective demonstration at different detail resolution levels with each resolution level having its own state representation, action definition, and a default hand-coded algorithm for providing a state-action mapping policy at that detail level. Over the course of a demonstration period, the system builds up individual corrective demonstration databases for each detail level in addition to a system-wide correction reuse database for deciding which resolution level to use in a particular state. During the autonomous execution of the task, the robot chooses the right detail resolution level and the action to be performed in a given state of the robot at the current level. We present performance evaluation for the proposed approach on an obstacle avoidance task performed by a humanoid robot on a robot soccer field where the robot starts from its own goal area and tries to reach the opponent goal area as fast as possible without bumping into the unknown obstacles placed on the field at unknown locations.

6.1. Humanoid Obstacle Avoidance using MRM+C

We define the obstacle avoidance task for a humanoid soccer robot as the problem of walking to a specified point on the field without bumping into the various obstacles placed on the field. The robot starts in its own goal area and the aim of the task is to reach within 1 meter distance of the opponent goal. The numbers, shapes, and locations of the obstacles on the field are not known to the robot so the robot has to detect the obstacles, position itself on the field, and follow a safe trajectory towards the opponent goal that will both prevent the robot from hitting the obstacles and keep the total time to reach the target as short as possible. In our evaluation study, we use the regular field of the RoboCup Standard Platform League as the experiment field, and Aldebaran Nao robot as the humanoid robot platform.

Figure 6.1 presents a sample instance of the humanoid obstacle avoidance task with three obstacles. The dashed lines represent an example traversal of the course by the robot with the yellow circles denoting the destination points selected by the robot.

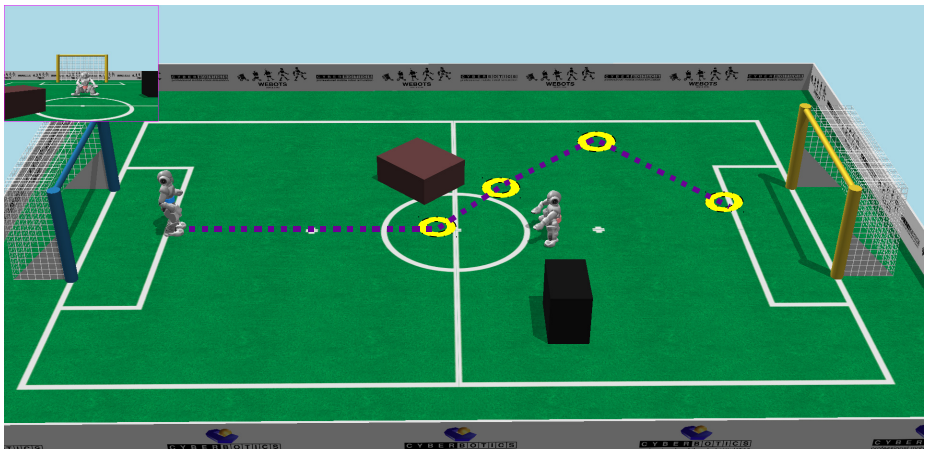


Figure 6.1. An example instance of the humanoid obstacle avoidance task with an example solution in a configuration where two box-shaped obstacles and another humanoid robot placed on the field.

Following the same Visual Sonar approach explained in detail in Section 5.3, instead of recognizing obstacles, we process the visual information gathered using the color cameras of the robot, and we build a free-space model of the area in front of the robot. We represent the state of the system at different detail resolutions using variations of the free-space information, and the position of the opponent goal with respect to the robot.

The most detailed state definition represents the free-space model as a vector of size 15, with each member of the vector being an integer number representing the distance in centimeters to the nearest perceived obstacle along the direction of that free-space slot. At the highest detail resolution, the action of the robot is represented as the (X, Y) coordinates on the field in centimeters with the center of the field being $(0, 0)$, the positive X axis pointing towards the opponent goal from the center point, and the positive Y axis pointing towards the left direction of the X axis (Figure 6.2).

We define three detail resolutions for the humanoid obstacle avoidance task: low,

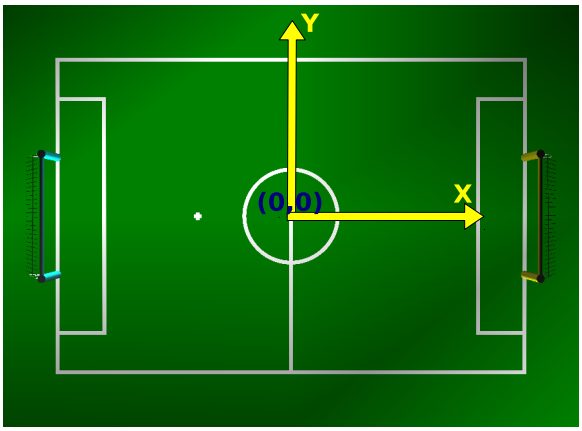
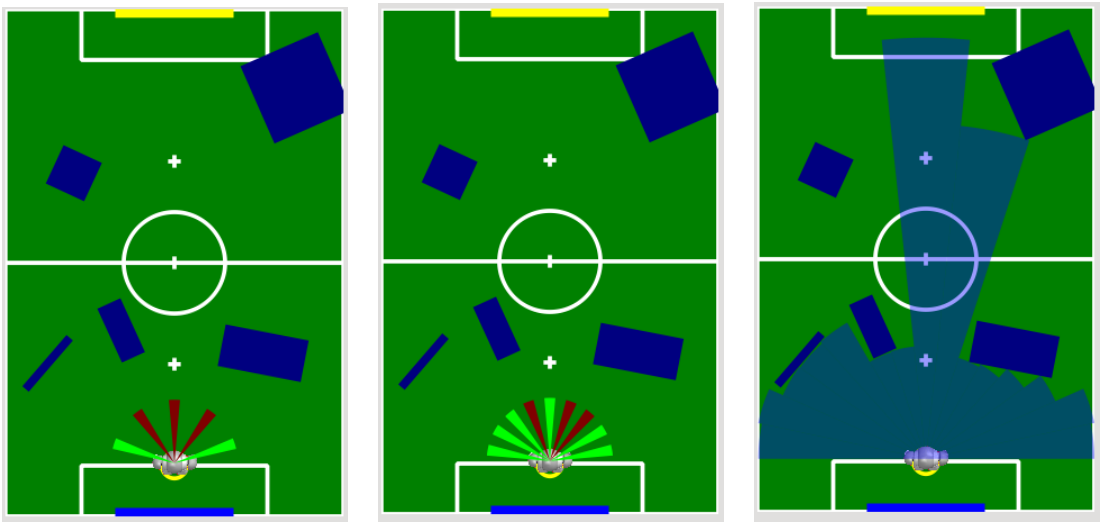


Figure 6.2. The coordinate system used in representing destination point on the field.

medium, and high. In the remainder of this section, we explain the state and action definitions, and the destination point selection algorithms for each detail resolution as well as the corrective demonstration setup for delivering the teacher feedback to the robot during the training sessions.

6.1.1. Low Detail Resolution Case

In the case of low detail resolution, the 180^o space in front of the robot is divided into five equal arcs of 36^o each. The existence of an obstacle along a free space slot is represented with a boolean value in the state vector where **true** indicates the slot is occupied with an obstacle. If the average distance of the most detailed free space representation slots that falls within a free space slot at this level is less than a certain threshold, that slot is marked as occupied. In our implementation, the threshold for considering a free-space slot as occluded is 120 centimeters, if the slot does not point towards the opponent goal, and is $min(120, 0.7 \times dist_{goal})$, if the slot is facing towards the opponent goal. The visualization of the state for the low detail resolution case is given in Figure 6.3(a). Here, for the low and medium level resolutions, a green slot means no obstacle towards that direction, and a red slot means this direction is occluded by an obstacle.



(a) (b) (c)

Figure 6.3. The example visualizations of the state representations. a) low detail resolution, b) medium detail resolution, and c) high detail resolution.

At this detail resolution, the destination point can be selected from among the five free space slot directions with a distance of 120 centimeters. However, the hand-coded algorithm for this resolution only selects from the three dribbling directions: *forward*, *left*, or *right*. If the middle slot (slot number 2) is free, the algorithm selects the forward direction, otherwise it checks the right and left slots to decide. The algorithm also favors the left direction over the right direction, if the leftmost free-space slot (the slot number 4) is free. The destination point selection algorithm for the first detail level is given in Figure 6.4.

6.1.2. Medium Detail Resolution Case

The state representation for the medium detail resolution case uses the same principles as the low detail resolution state representation with the exception of using nine slots instead of five. An example visualization of a medium detail resolution state representation is given in Figure 6.3(b).

The hand-coded algorithm for this resolution goes over each free-space slot and selects the direction of the closest available slot to the opponent goal as the destination

```
1: slot ← −1
2: booleanState ← getBooleanState(LOW)
3: if ¬ booleanState(2) then
4:    slot ← 2
5: else
6:    if ¬ booleanState(0) then
7:       slot ← 0
8:    else
9:       slot ← 4
10:   end if
11: end if
12: destAngle ← calculateDirection(slot)
13: destDistance ← 120
14: return calculateGlobalPoint(destAngle, destDistance)
```

Figure 6.4. Destination point selection algorithm for the low detail resolution.

direction, again using a fixed walking distance of 120 centimeters. The destination point selection algorithm for the medium detail resolution is given in Figure 6.5.

```
1: booleanState ← getBooleanState(MEDIUM)
2: goal ← getGoalSlot()
3: closestSlot ← 0
4: minDistance ← 9
5: for slot ← 0; slot < 9; i ← slot + 1 do
6:    if |goal − slot| ≤ minDistance and ¬ booleanState(slot) then
7:       minDistance ← |goal − slot|
8:       closestSlot ← slot
9:    end if
10: end for
11: destAngle ← calculateDirection(closestSlot)
12: destDistance ← 120
13: return calculateGlobalPoint(destAngle, destDistance)
```

Figure 6.5. Destination point selection algorithm for the medium detail resolution.

6.1.3. High Detail Resolution Case

At the finest detail resolution, the free space is represented with the distance values for 15 equally divided slots in centimeters and represented as integer values. The distance value of a slot denotes the distance of the nearest detected obstacle lying within the coverage of that particular free-space slot. The Figure 6.3(c) shows an example visualization of the state representation for the high detail resolution.

Contrary to the algorithms for the lower detail resolutions, the destination distance is also selected by the algorithm in addition to the destination direction. We go over each free-space slot and for each slot we compute a weighted distance value using a sliding window of size three with the weights 0.25 at both ends and 0.5 for the center. Finally, the direction of the free-space slot with highest weighted distance is selected as the dribble direction and the computed weighted distance is assigned as the dribble distance. The destination point selection algorithm for the high detail resolution is given in Figure 6.5.

```
1:  goalAngle ← getGoalAngle()
2:  if goalAngle < −π/2 or goalAngle > π/2 then
3:     if |angle_0 − goalAngle| < |angle_{N−1} − goalAngle| then
4:        destAngle ← angle_0
5:     else
6:        destAngle ← angle_{N−1}
7:     end if
8:     destDistance ← 120
9:  else
10:    maxDist ← 0
11:    for i ← 1; i < N − 1; i ← i + 1 do
12:       distance ← 0.25dist_{i−1} + 0.5dist_i + 0.25dist_{i+1}
13:       if distance > maxDist then
14:          maxDist ← distance
15:          maxSlot ← i
16:       end if
17:    end for
18:    destAngle ← angle_{maxSlot}
19:    destDistance ← maxDist
20: end if
21: return calculateGlobalPoint(destAngle, destDistance)
```

Figure 6.6. Destination point selection algorithm for the high detail resolution.

6.1.4. Corrective Demonstration Setup

During the demonstration sessions, the teacher uses a custom developed software running on a host computer to access the current detail level as well as the internal state of the robot. The same user interface is also used for delivering the action corrections and issuing detail resolution refinement commands. The host computer communicates with the robot over wireless Ethernet connection. The robot broadcasts its computed state, the current detail resolution, and the current destination point back to the host computer at each step. The robot also uses a text-to-speech

software system to announce the inferred state of the system and the action selected to be executed (Figure 6.8). The received state information of the robot is then visualized on the display. This visualization includes the perceived free-space information, the position of the robot on the field, and the current selected destination point that the robot walks to. A snapshot from the developed software is given in Figure 6.7.

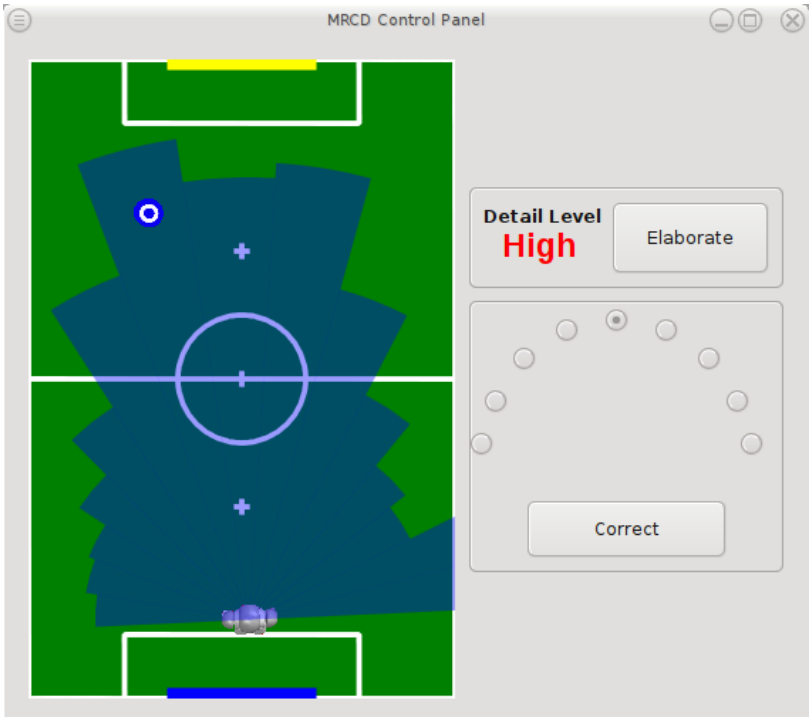


Figure 6.7. The user interface for delivering corrective demonstration to the robot.

The teacher uses the *Elaborate* button to issue a detail resolution refinement command. The current detail resolution is also displayed on the screen. If the current detail resolution is either *Low* or *Medium*, the teacher uses the radio buttons located on the bottom-right part of the interface. At the *High* detail resolution, the user specifies the destination point by clicking on the field visualization on the interface. There are 9 radio buttons placed on an arc, each representing a free space slot. For the *Medium* detail resolution, all radio buttons are enabled. For the *Low* detail resolution, every other button is enabled, reducing the number of enabled buttons to 5. At the *High* detail resolution, all radio buttons are disabled as the system expects a correction in the form of a global point on the field. Similarly, at the *Medium* and

Low detail resolutions, it is not possible to specify a destination point by clicking on the visualized field.

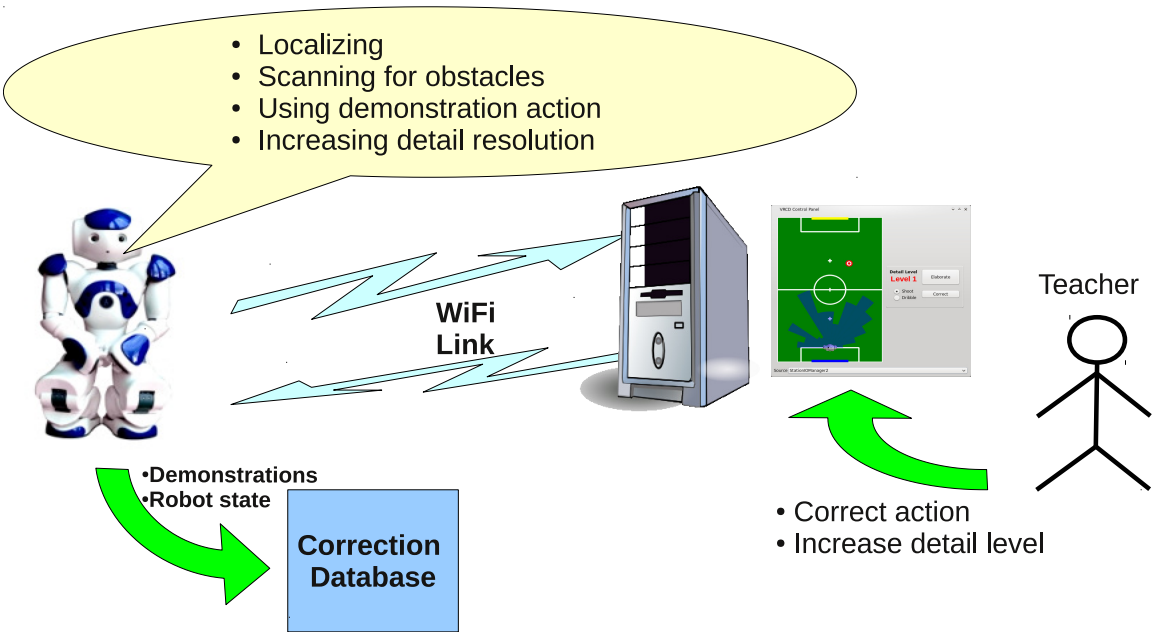


Figure 6.8. The corrective demonstration setup for the obstacle avoidance task.

6.2. Experimental Evaluation

We evaluated the performance of MRM+C approach against the hand-coded controllers at the lowest and the highest detail resolutions on the obstacle avoidance task using two different obstacle configurations, and an empty field as the base case (Figure 6.9).

We used the task completion time as the performance measure for the cases the robot was able to complete the task. The results are given in Table 6.1. We ran 5 trials per method for each configuration. The **Rate** column presents the success rate. The **Time** shows the average time it took the robot to complete the task for the successful trials. The units for the rate and the average time columns are percentages and seconds, respectively.

An examination of the results yields that the success rate drops as the average time increases as the number of obstacles increase which is an expected result. For the

(a)

(b)

(c)

Figure 6.9. The obstacle configurations used in the experimental evaluation. a) empty field, b) a single obstacle placed on the center of the field, and c) three obstacles placed around the center circle.

empty field configuration, all algorithms performed well in terms of success rate, while the hand coded algorithm for the high detail resolution outperformed the others. The main reason behind this result is since the high resolution algorithm uses free space slot distances to compute the destination point, it selects a destination point very close to the opponent goal and the task ends once the robot reach the destination so the robot does not lose any time in localizing itself and scanning the field for free space modeling. The performance of MRM+C was better than the low detail resolution algorithm but was worse than high detail resolution algorithm mainly due to the number of field scans it has executed.

For the single obstacle case, the performance of the low detail resolution algorithm degraded considerably but the high detail resolution algorithm and MRM+C were able to achieve high success rates. The high detail resolution algorithm outperformed the MRM+C since it uses the most detailed state representation and computes long distance destination points, yielding a smaller number of field scans.

For the three obstacles case, the low detail resolution algorithm was too simple to handle the case, and the high detail resolution algorithm was not able to compute the propoer actions in most of the times. Combining the use of simpler algorithms when the current obstacle model does not yield the need for very detailed actions, and the corrective demonstration actions provided by the teacher, the MRM+C algorithm outperformed both hand-coded algorithms despite a considerable performance degradation compared to the previous configurations.

In 8 out of 15 failed trials, the failure was mostly due to the poor self localization data. The destination points computed by the algorithms are in global world coordinates; therefore, the performance gets heavily affected by the error in the estimated position.

Table 6.1. Performance evaluation results for the MRM+C approach in Humanoid Obstacle Avoidance domain.

	Empty Field		**1 Obstacle**		**3 Obstacles**	
Method	**Rate (%)**	**Time (sec.)**	**Rate**	**Time**	**Rate**	**Time**
Low Detail Resolution	80	115	60	195	0	N/A
High Detail Resolution	100	59	80	94	40	133
MRM+C	80	96	100	103	60	182

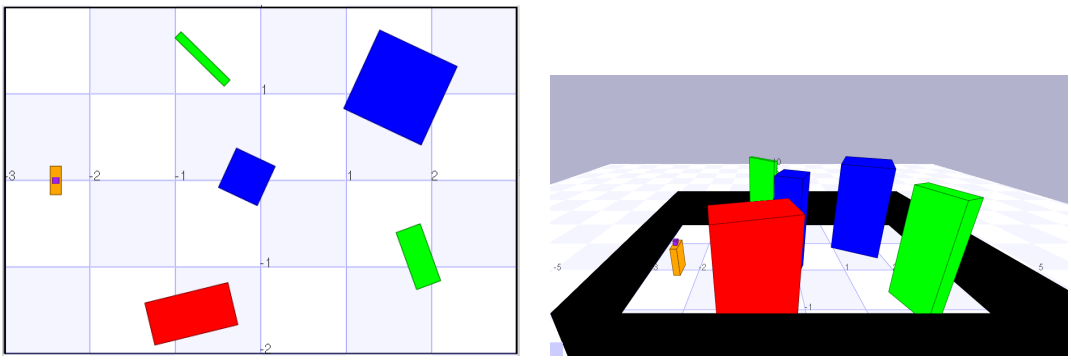
7. EXPERIMENTAL ANALYSIS

In this chapter, we present an experimental analysis of the M+C and MRM+C algorithms in terms of robustness against uncertainty in both perception and action, and in terms of execution cost imposed by the computational complexities of the default algorithms used. We use a simulated version of the humanoid obstacle avoidance problem as our experimental test bed.

7.1. Simulation Environment

We modeled a simulated version of the humanoid obstacle avoidance task as the experimental testbed. We use the Player/Stage framework [37] to model the environment in 2D. We model the Nao humanoid robot with an omnidirectional wheeled robot base, and we use a laser range finder to emulate the vision-based free space perception used on the real Nao robots. The laser range finder readings are processed and converted into the same format as the free-space detection module on the real Nao provides. The omnidirectional walk of Nao is modeled as a holonomic motion on 2D ground plane and the speed of the wheeled robot is limited to 10 cm/s, which is roughly the speed of a real Nao robot. We use a Monte-Carlo Localization based method for self-localization on the real Naos. We imitate the self-localization information in the simulation with a global positioning system distorted with a specified amount of white noise. By imitating the perceptual and action abilities of the robot accordingly, we are able to run the same software for the M+C and MRM+C we used in Chapter 6. A snapshot from the simulator is given in Figure 7.1.

Both for delivering the corrective demonstration and performing other automated experiments, the teacher uses a modified version of the user interface presented in Chapter 6 which provides the visualization of the true positions of the obstacles in the environment, the waypoints computed by the robot along the course of its execution, and the final path the robot traversed. The user interface also allows the



(a) (b)

Figure 7.1. A snapshot from the Stage simulator for the humanoid obstacle avoidance task: a) 2D view, and b) 3D view. The leftmost rectangular prism represents the Nao robot where the small cube on top of the robot is the laser range finder imitating the visual system of the robot.

teacher to modify a set of simulation parameters. A snapshot of the user interface is given in Figure 7.2.

7.2. Robustness Against Uncertainty

7.2.1. Uncertainty in Perception

The perceptual subsystem of the robot uses color cameras to process the visual information around the robot for inferring the system state. As in most real world sensing devices, the cameras of the robot are error-prone due to their sensitivity to even the slightest change in the lighting characteristics of the environment. The cameras are mounted on the head of the robot, which is the end effector of a highly complex manipulator chain, formed by the skeleton of the robot. The position sensing devices on the joints of the robot are also error-prone and small errors in the position readings of the joints accumulates through the kinematic chain of the robot. We consider two different uncertainty problems in the perception system:

- Uncertainty in free-space detection
- Uncertainty in self-localization

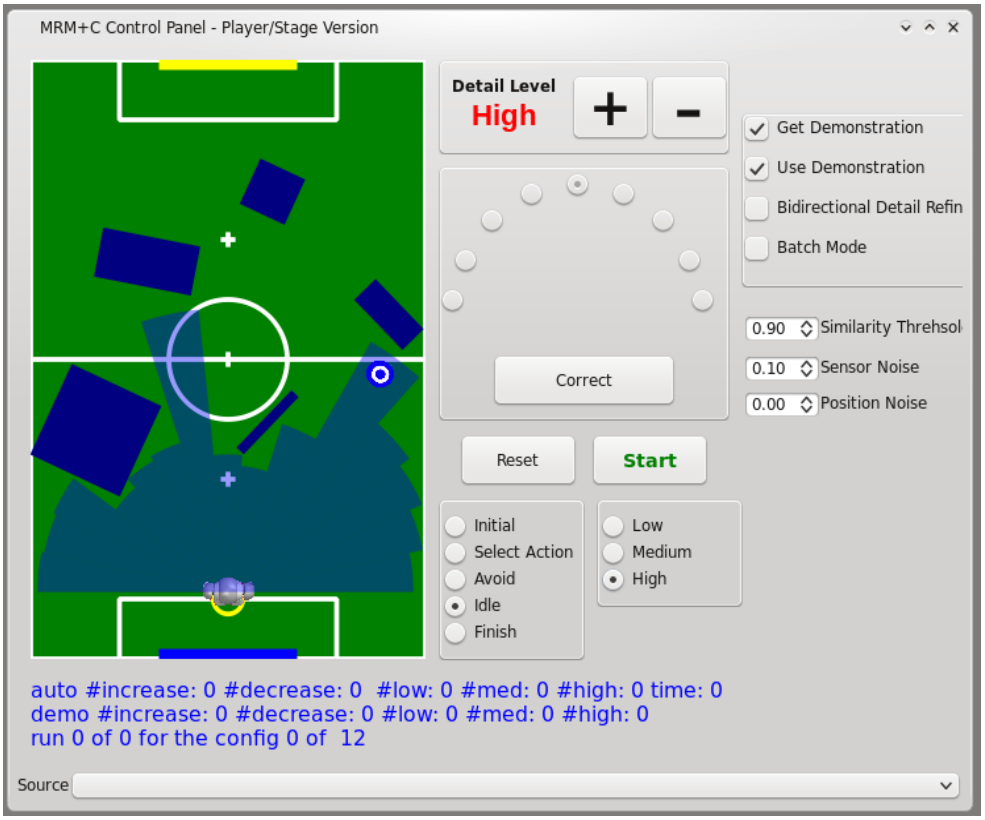


Figure 7.2. The modified user interface for delivering corrective demonstration to the simulated robot and managing the simulation

7.2.1.1. Uncertainty in Free-space Detection. As we described in detail in Chapter 5, we use our closed-world assumption about the colorized robot soccer world and we treat any non-green object on the field as an obstacle. We use the current posture information of the robot to calculate the position of the camera in 3D and then we use the camera position information to project recognized obstacle regions on the image onto the ground plane to have a relative position and distance estimation for that obstacle.

There are two sources of uncertainty in the calculation of the free-space model around the robot:

- Confusing the field lines with obstacles
- Erroneous ground projection due to imperfect joint position sensing

The field lines are marked with white tape and have the same shade of white color with the robots. We use a set of sanity checks including the size constraints of the ground-projected region but due to the changes in the lighting, the perception of lines and obstacle information get distorted and this results in an erroneous free-space model.

We calculate the position of the camera in 3D space using forward kinematics and the kinematic chain information of the robot. Every link in the kinematic chain is a servo motor with gears and both the imperfect sensing abilities of the position sensor, and the backlash caused by the gears, each joint has a slight error in position sensing and actuation (as previously presented in Figure 4.2).

The laser range finder readings provided by the simulation are impeccant; therefore, we apply an artificial noise to approximate the sensing error for the free-space detection in the real world, which has fairly complicated characteristics. For the sake of simplicity, we approximate the sensor noise with uniform random distribution of varying magnitude to test the robustness of the system against various levels of

uncertainty in the sensing.

7.2.1.2. Uncertainty in Self-localization. Knowing its position in its environment is of utmost importance for a mobile robot. We use a variant of Monte Carlo Localization called Sensor Resetting Localization [38] in our robot soccer setup. We combine the visual information extracted from camera images with the odometry estimation of the robot to form a belief on the robot's whereabouts. Both the visual landmark extraction and odometry estimation parts are imperfect and constitute the main sources of uncertainty with the position estimation.

We use the distance and orientation information of the goal posts, and line intersections. The robot starts with an initial randomly distributed belief distribution, encoded as a set of particles each representing a candidate position on the field. We compare the actual distance and orientation information for the perceived landmarks to the distance and orientation information for each particle that would have been perceived if the robot was on the position represented by that particle. A weight is calculated for each particle based on the similarity of the actual final pose estimation is computed using this weighted particle set.

Color classification, which is the process of assigning a pixel to a small set of colors (i.e., blue and yellow for goals, green and white for the field, red for the ball, etc.) is very sensitive to the changes in the lighting conditions. Due to the misclassification of the colors, the shapes of the lines and the goal posts might not be perceived as accurately as needed and this leads to either a wrong set of distance and orientation estimations, or incomplete information. In either case, the self-localization module gets negatively affected as this erroneous/incomplete information causes inaccurate similarity values to be computed for the particle set.

The robot uses the reported posture of the robot and a set of other pieces of information like the number of steps executed by the walking algorithm to compute an estimated displacement with respect to the starting position. In addition to the

imperfect sensing on the joint positions, external factors like a slippery floor or uneven carpet surface have an adverse effect on the accuracy of the odometry estimation. As a consequence, the resulting pose estimation of the robot on the field contains the uncertainty coming both from the odometry calculation and the visual perception.

Similar to the sensing error in the free-space detection, we model the error in the pose estimation with a uniform distribution.

7.2.2. Uncertainty in Action

Uncertainty in action is mainly due to the imperfect mechanical construction and imprecise and erroneous position sensing. In addition to these issues, the walking dynamics of the robot also affects the resulting motion. Although the action error is considerably small in Aldebaran Nao robots, we ran simulated experiments with high amount of action noise for testing purposes.

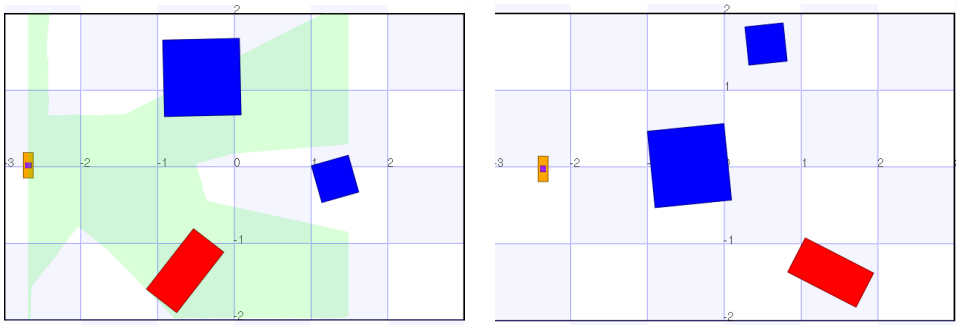
7.3. Experiment Results

We used two different obstacle configurations with three obstacles placed at different positions in each of them. We defined five uncertainty levels with different amounts of uniform noise applied on the free space model, the estimated position of the robot on the field, and the motion of the robot as follows:

- No Uncertainty: No noise on the free space model, self position, or motion.
- Low Uncertainty: 10 percent noise on the free space model, 2 percent noise on the self position, 3 percent noise on the motion.
- Medium Uncertainty: 20 percent noise on the free space model, 4 percent noise on the self position, 6 percent noise on the motion.
- Moderate Uncertainty: 30 percent noise on the free space model, 6 percent noise on the self position, 9 percent noise on the motion.
- Heavy Uncertainty: 40 percent noise on the free space model, 8 percent noise

on the self position, 12 percent noise on the motion.

The obstacle configurations used in the experiments are given in Figure 7.3.



(a) (b)

Figure 7.3. The obstacle configurations used in the performance evaluation experiments of the simulated obstacle avoidance system under uncertainty.

We evaluated the following algorithms:

- Low detail resolution algorithm
- Low detail M+C
- Medium detail resolution algorithm
- Medium detail M+C
- High detail resolution algorithm
- High detail M+C
- Multi-resolution task execution (MRTE) using only the hand coded algorithms
- MRM+C

During the training session, 23 low level, 8 medium level, and 14 high level demonstrations have been collected for the corrective demonstration part, and 22 demonstrations have been recorded for the detail resolution arbitrator component. For each algorithm and uncertainty configuration, we performed 10 trials with each obstacle configuration, 5 with fixed obstacles and 5 with randomly distorted obstacles.

Figure 7.4 shows the success rates of the algorithms. The blue bar in the Multi Resolution group is the success rate for the MRTE algorithm, and the red bar in the same group is the success rate for the MRM+C algorithm. As expected, the success rate of the algorithms increase as the algorithm gets more complex and runs at a higher detail resolution. In all four configurations (three detail resolutions, and the multi-resolution execution), the M+C instances outbested the algorithms alone, and the MRM+C algorithm outperformed the MRTE algorithm. The performances of the multi-resolution algorithms are close to the algorithm for the high detail level (73% vs. 72% for algorithm only, and 78% vs. 76% for M+C) despite that the robot was not executing the high detail resolution algorithm in all cases. The composition of executed actions per evaluated algorithm is given in Table 7.1 and visualized in Figure 7.5. In both MRTE and MRM+C evaluations, the majority of the executed actions were computed by the low detail resolution algorithm with and low detail resolution demonstration database, yet, the success rates for the MRTE and MRM+C

algorithms are better than the low and medium level algorithms, and close to the high level algorithm. The results show that the MRM+C approach uses less detailed actions for most of the time yet it demonstrates a performance level comparable to the highest detail resolution M+C instance.

Table 7.1. The average number of actions executed in the succeeded runs.

	Low Res.		**Medium Res.**		**High Res.**		
Action	**M**	**M+C**	**M**	**M+C**	**M**	**M+C**	**MRM+C**
Low Model	9.47	4.33	0	0	0	0	3.37
Low Correction	0	3.44	0	0	0	0	2.32
Med. Model	0	0	6.11	4.89	0	0	0.95
Med. Correction	0	0	0	1.26	0	0	0.16
High Model	0	0	0	0	3.47	2.89	0.18
High Corrections	0	0	0	0	0	1.58	0.04

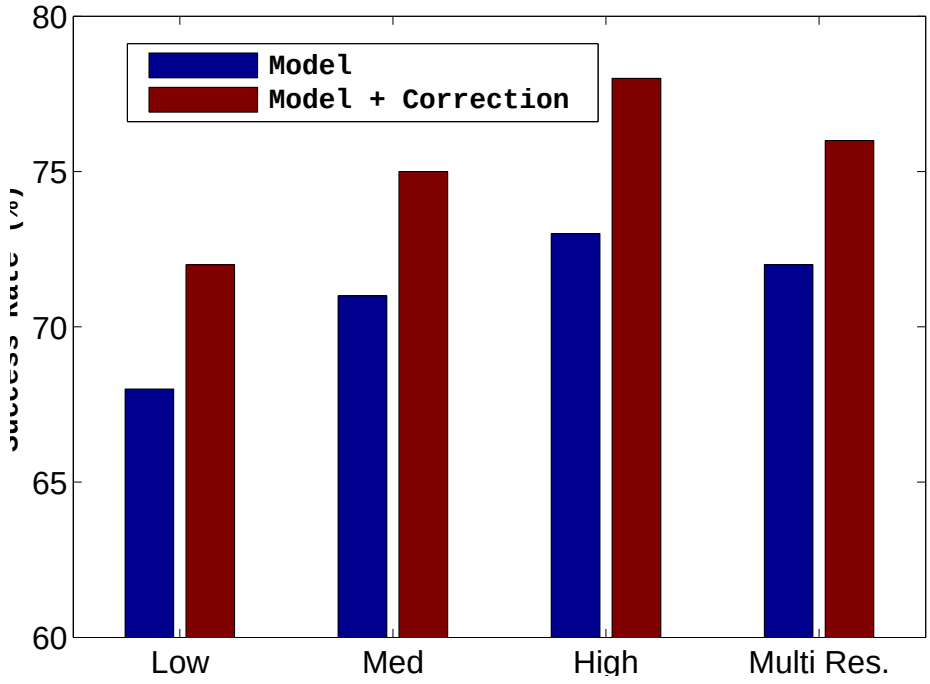


Figure 7.4. The overall performance results for the individual algorithms and M+C instances for each detail resolution, along with the multi resolution performances without (MRTE) and with (MRM+C) corrective demonstration.

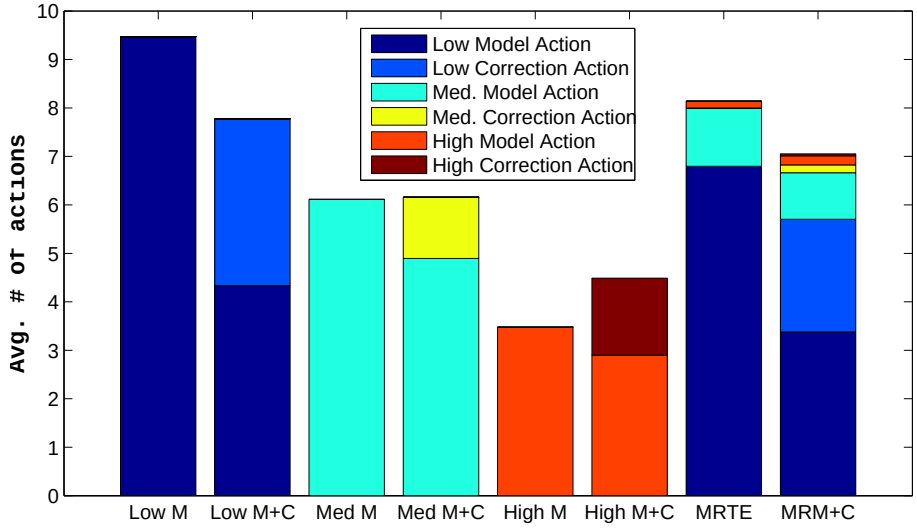


Figure 7.5. The average number of actions executed per individual algorithms, M+C instances, MRTE system, and MRM+C system.

8. CONCLUSION AND FUTURE WORK

This chapter summarizes the scientific contributions of this thesis and presents several promising future research directions that build up on this dissertation.

8.1. Contributions

This dissertation makes the following scientific contributions:

- Model Plus Correction (M+C) Paradigm: This is a hybrid approach to skill and task refinement on autonomous robots. The M+C paradigm makes use of the advantageous properties of the traditional algorithm-based controllers for task and skill execution and the human demonstration to provide a rapid performance improvement using a small number of demonstrations and hence, demanding less attention from the demonstrator. The M+C paradigm has three components. The *Model* component is an algorithm implementation for performing the task or skill. This algorithm uses a model of the system or a set of assumptions about the system. The *Correction* component contains the corrective demonstration database consisting of corrective actions delivered by the teacher when the Model component computes an erroneous action, and a generalization method for being able to compute the correction action for any given system state out of the collected sparse demonstration data. The *Correction Reuse* component for using the actions computed by the Model and the Correction components to compute the final action to be executed by the robot.
- Multi-Resolution Model Plus Correction (MRM+C): This is a framework consisting of a set of components running at different detail granularities to be used in situations with different complexities during the execution. Founded on M+C paradigm, the MRM+C approach extends it to allow the teacher to deliver corrective feedback at different detail resolutions, as required by the

complexity of the situation the robot is facing. The MRM+C approach consists of multiple M+C instances, each using state and action representations at varying detail levels, and associated with default algorithms of various complexities. The M+C approach features an arbitrator component to decide which M+C instance becomes active at a particular state. Using the MRM+C approach, the teacher does not have to provide corrections at the most detailed level if the correction requirement for the current situation can be handled at a less detail resolution, and the system does not have to run the most complex default default algorithm to cope with simple situations. Assuming an algorithm gets computationally more expensive as it gets more complex, using simpler algorithms whenever possible reduces the total execution cost. Moreover, allowing the teacher to provide corrections at lower detail resolutions reduces the number of demonstrations needed to cover parts of the state-action space corresponding to simpler situations, and reduces the demand for constant teacher attention.

- Formal models for the M+C and MRM+C approaches: This thesis presents a formalization for the proposed M+C and MRM+C approaches, describing each component of the approaches, and how these two approaches are related with each other, and with the classical learning from demonstration approach.
- Experimental analysis of the M+C and MRM+C approaches: A detailed experimental analysis of the M+C and MRM+C approaches to evaluate their robustness against the uncertainty in the environment is presented. A utility analysis of using multiple detail resolutions against using the defined detail resolutions individually is also presented.
- Extensive Evaluation of M+C and MRM+C approaches: This thesis presents detailed evaluations of both M+C and MRM+C approaches in several real world and simulated domains using a complex humanoid robot as the test platform.

8.2. Future Directions

- Teacher and Demonstration Quality Evaluation: The real-time corrective demonstration in Chapter 4 showed us that especially in the skill refinement case, if the

execution of a complex task occurs so fast to leave little time for the teacher to decide on a correction, the quality of the demonstration data decreases drastically. One approach to tackle this problem would be to try to develop a method for examining the demonstration data as it is being received from the teacher and try to identify the portions of the data not complying with the rest and mark those portions as noise to exclude them from the correction calculations.

- Adding Self-Exploration: The demonstration database collected for M+C approach is sparse, as the teacher only provides demonstration when the underlying default algorithm falls short on acting properly. The sparsity of the database necessitates a need for a good generalization in order to be able to use the correction during the autonomous execution. By adding a self-exploration feature for the robot to experiment with self-generated corrections based on the corrections given by the teacher and to evaluate the performance of the synthesized corrections, the robot can gain the ability to grow the correction database with the synthesized corrections.
- Open-Ended Learning: The M+C idea stores all the corrective demonstration samples without deriving a policy and dismissing the demonstration data afterwards. This leads to a very large amount of correction data to be accumulated over long periods of time and hence will affect the correction reuse computations negatively as most of the correction reuse methods presented in this thesis makes use of the demonstration data itself. One possible approach would be to examine portions of the demonstration data continuously and replace the data with a model if the model is able to represent the data accurately enough. This approach would make it possible to keep the correction database small enough to be processed efficiently during the task execution.

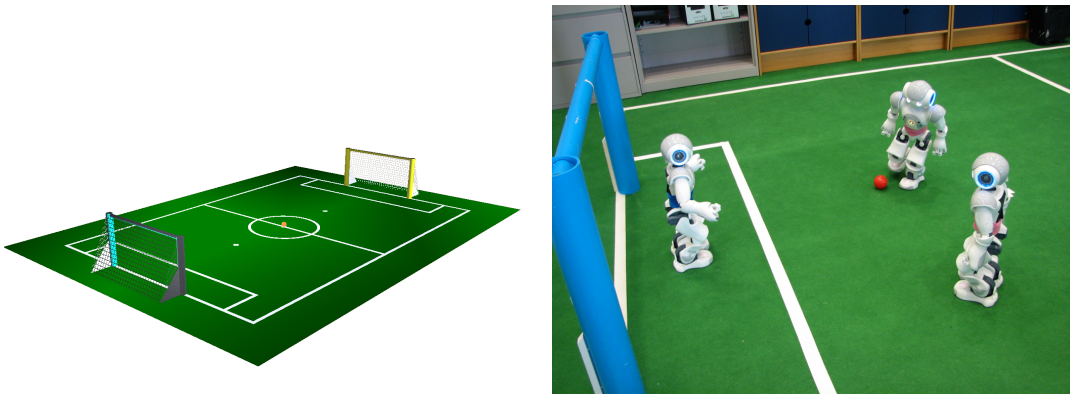
APPENDIX A: ROBOT SOCCER DOMAIN

The RoboCup Standard Platform League robot soccer is used as the application domain for task refinement using complementary corrective demonstration (Chapter 5) and multi-resolution complementary corrective demonstration (Chapter 6) evaluations.

RoboCup is an international research initiative that aims to foster research in the fields of artificial intelligence and robotics by providing standard problems to be tackled from different points of view; such as, software development, hardware design, and systems integration (http://www.robocup.org). Soccer was selected by the RoboCup Federation as the primary standard problem due to its inherently complex and dynamic nature, allowing scientists to conduct research on many different sub-problems ranging from multi-robot task allocation to image processing, and from biped walking to self-localization. With its various categories focusing on different challenges in the soccer domain; such as, playing soccer in simulated environments (the 2D and 3D Simulation Leagues) and physical environments using wheeled platforms (the Small Size League and the Middle Size League), humanoid robots of different sizes and capabilities (the Humanoid League), and a standard hardware platform (the Standard Platform League), the ultimate goal of RoboCup is to develop, by 2050, a team of 11 fully autonomous humanoid robots that can beat the human world champion soccer team in a game that will be played on a regular soccer field complying with the official FIFA rules.

In the Standard Platform League (SPL) of RoboCup (http://www.tzi.de/spl), teams of 3 autonomous humanoid robots play soccer on a 6 meters by 4 meters green carpeted field (Figure A.1(a)). The league started in 1998 as an embodied software competition with a common and standard hardware platform, hence the name. Sony AIBO robot dogs had been used as the standard robot platform of the league until 2008, and the Aldebaran Nao humanoid robot was decided to be the new standard

platform thereafter. A snapshot showing the Nao robots playing soccer is given in Figure A.1(b).



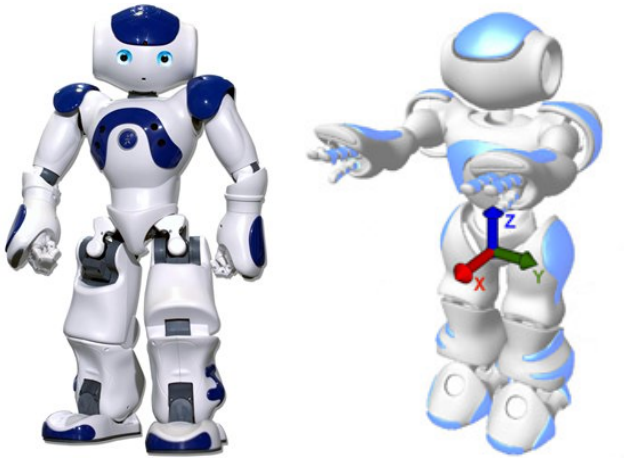
(a) (b)

Figure A.1. a) The field setup for the RoboCup Standard Platform League (SPL), and b) a snapshot from an SPL game showing the Nao robots playing soccer.

A.1. Hardware Platform

The Aldebaran Nao humanoid robot is used across all real world evaluations. The Nao (Figure A.2(a)), is a 4.5 Kg heavy, 58 cm tall humanoid robot with 21 degrees of freedom (http://www.aldebaran-robotics.com/). It is equipped with an on-board processor running at 500MHz, and a variety of sensors including a 3-axis accelerometer, a 2-axis (Roll-Pitch) gyroscope, and a special circuitry for computing the absolute torso (upper body of the robot) orientation using the accelerometer and gyroscope data. The torso angle estimator, the accelerometer, and the gyroscope sensors use a right-hand frame of reference (FigureA.2(b)). As opposed to most other humanoid robot designs, Nao does not have separate hip yaw joints for each leg [39], instead, the two legs have mechanically coupled hip yaw-pitch joints that are perpendicular to each other along the $Y - Z$ plane and driven by a single motor (Figure 4.13).

Nao runs a Linux-based operating system and has a software framework called *NaoQi*, which allows users to develop their own controller software and access the sensors and actuators of the robot. The internal controller software of the robot



(a) (b)

Figure A.2. a) The Nao robot. b) The frame of reference for sensors.

runs at 50Hz; therefore, it is possible to read new sensor values and send actuator commands every 20ms[2] .

A.2. Software Overview

Being able to play soccer requires several complex software modules (i.e., image processing, self localization, motion generation, planning, communication, etc.) to be designed, implemented, and seamlessly integrated with each other. In this section of the paper, we present a brief overview of the software infrastructure developed for the RoboCup SPL competitions and also used in this study.

A.2.1. Image Processing

The Nao humanoid robots perceive their environment via their sensors, namely the two color cameras, the ultrasound distance sensors, the gyroscope, and the accelerometer. All the important objects in the game environment (i.e., the field, the goals, the ball, and the robots) are color coded to facilitate object recognition. However, perception of the environment remains the most challenging problem primarily due to the extremely limited on-board processing power that prevents the use of in-

[2]The mentioned frequency is for the Nao V2 model which was the platform used in this study. The internal control software on the more recent V3 model runs at 100Hz.

tensive and sophisticated computer vision algorithms. The very narrow fields of view (FoV) of the robot's cameras ($\approx 58^o$ diagonal) and their sensitivity to changes in light characteristics like the temperature and luminance levels are among the other contributing factors to the perception problem.

The job of the image processing module is to extract the relative distances and bearings of the objects detected in the camera image. In addition to the position information, the image processing module also reports confidence scores indicating the likelihood of those objects being actually present in the camera image. The image processing module consists of two main stages: the low level vision processing, and the high level object detection. The first stage uses the CMVision [40] library to perform color segmentation, i.e., labelling each pixel on the image with one of the following color codes: green, white, pink, blue, yellow, orange, or none (Figure 5.2(a)). After the color segmentation, a connected component analysis is performed on the image to extract colored reqions in the form of the bounding box and centroid of each region. A set of object-specific detectors are then fed with the list of extracted regions and they report the relative position of the detected objects using the position of region on the image, and the position of the robot's camera.

A.2.2. Self Localization and World Modeling

These modules are responsible for determining the location of the robot as well as the locations of the other important objects (e.g. the ball) on the field. Our system uses a variation of Monte Carlo Localization (MCL) called Sensor Resetting Localization [38] for estimating the position of the robot on the field. For calculating and tracking the global positions of the other objects, we employ a modeling approach which treats objects based on their individual motion models defined in terms of their dynamics [41, 42].

A.2.3. Planning and Behavior Control

Our planning and behavior generation module is built using a hierarchical Finite State Machine (FSM) based multi-robot control formalism called Skills, Tactics, and Plays (STP) [43]. Plays are multi-robot formations where each robot is executing a tactic consisting of several skills. Skills can be stand-alone or formed via a hierarchical combination of other skills.

A.2.4. Motion Generation

The motion generation module is responsible for all types of movement on the field including biped walking, ball manipulation (e.g., kicking), and some other motions such as getting back upright after a fall. For the biped walking, we use the omni-directional walk algorithm provided by Aldebaran. For kicking the ball and the other motions, we use predefined actions in the form of sequences of keyframes, each of which define a vector of joint angles and a duration value for the interpolation between the previous pose and the current one. Two variations (strong and weak) of three types of kick (side kick to the left, side kick to the right, and forward kick) are implemented to be used in the games.

REFERENCES

1. Argall, B., B. Browning and M. Veloso, "Learning Robot Motion Control with Demonstration and Advice-Operators", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

2. Argall, B. D., S. Chernova, M. Veloso and B. Browning, "A Survey of Robot Learning from Demonstration", *Robotics and Automation Systems*, Vol. 57, No. 5, pp. 469–483, 2009.

3. Thomaz, A. L. and C. Breazeal, "Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance", *Proceedings of the 21st National Conference on Artificial Intelligence*, 2006.

4. Cakmak, M., C. Chao and A. Thomaz, "Designing Interactions for Robot Active Learners", *Autonomous Mental Development, IEEE Transactions on*, Vol. 2, No. 2, pp. 108 –118, 2010.

5. Chernova, S. and M. Veloso, "Confidence-Based Policy Learning from Demonstration using Gaussian Mixture Models", *In Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2007.

6. Chernova, S. and M. Veloso, "Learning Equivalent Action Choices from Demonstration", *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

7. Chernova, S. and M. Veloso, "Interactive Policy Learning through Confidence-Based Autonomy", *Journal of Artificial Intelligence Research*, Vol. 34, 2009.

8. Chernova, S. and M. Veloso, "Multiagent Collaborative Task Learning through Imitation", *In Proceedings of the 4th International Symposium on Imitation in*

Animals and Artifacts, 2007.

9. Chernova, S. and M. Veloso, "Teaching Collaborative Multirobot Tasks through Demonstration", *In Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2008.

10. Hersch, M., F. Guenter, S. Calinon and A. Billard, "Dynamical System Modulation for Robot Learning via Kinesthetic Demonstrations", *IEEE Transactions on Robotics*, Vol. 24, No. 6, pp. 1463–1467, 2008.

11. Argall, B., E. Sauser and A. Billard, "Tactile Feedback for Policy Refinement and Reuse", *In Proceedings of the 9th IEEE International Conference on Development and Learning*, 2010.

12. Argall, B., E. Sauser and A. Billard, "Policy Adaptation through Tactile Correction", *In Proceedings of the 36th Annual Convention of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, 2010.

13. Argall, B., E. Sauser and A. Billard, "Tactile Correction and Multiple Training Data Sources for Robot Motion Control", *In NIPS 2009 Workshop on Learning from Multiple Sources with Application to Robotics*, 2010.

14. Argall, B. D., E. Sauser and A. Billard, "Tactile Guidance for Policy Adaptation", *Foundations and Trends in Robotics*, Vol. 1(2), pp. 79–133, 2010.

15. Calinon, S. and A. Billard, "What is the Teacher's Role in Robot Programming by Demonstration? - Toward Benchmarks for Improved Learning", *Interaction Studies. Special Issue on Psychological Benchmarks in Human-Robot Interaction*, Vol. 8, No. 3, 2007.

16. Nakanishi, J., J. Morimoto, G. Endo, G. Cheng, S. Schaal and M. Kawato, "Learning from Demonstration and Adaptation of Biped Locomotion", *Robotics and Autonomous Systems*, Vol. 47, No. 2-3, pp. 79 – 91, 2004.

17. Bentivegna, D. and C. G. Atkeson, "Using primitives in learning from observation", *First IEEE-RAS International Conference on Humanoid Robots*, 2000.

18. Bentivegna, D. C., C. G. Atkeson and G. Cheng, "Learning Similar Tasks from Observation and Practice", *in Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

19. Grollman, D. and O. Jenkins, "Dogged Learning for Robots", *International Conference on Robotics and Automation*, 2007.

20. Grollman, D. and O. Jenkins, "Learning Elements of Robot Soccer from Demonstration", *International Conference on Development and Learning*, 2007.

21. Calinon, S., F. D'halluin, E. Sauser, D. Caldwell and A. Billard, "Learning and Reproduction of Gestures by Imitation: An approach based on Hidden Markov Model and Gaussian Mixture Regression", *IEEE Robotics and Automation Magazine*, Vol. 17, No. 2, pp. 44–54, 2010.

22. Gribovskaya, E., K. Zadeh, S. Mohammad and A. Billard, "Learning Nonlinear Multivariate Dynamics of Motion in Robotic Manipulators", *International Journal of Robotics Research*, 2010.

23. Breazeal, C., G. Hoffman and A. Lockerd, "Teaching and Working with Robots as a Collaboration", *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004.

24. Rybski, P. E., K. Yoon, J. Stolarz and M. M. Veloso, "Interactive robot task training through dialog and demonstration", *In Proceedings of the 2007 ACM/IEEE International Conference on Human-Robot Interaction*, 2007.

25. Argall, B., B. Browning and M. Veloso, "Learning from Demonstration with the Critique of a Human Teacher", *Second Annual Conference on Human-Robot Interactions*, 2007.

26. Abbeel, P. and A. Y. Ng, "Apprenticeship Learning via Inverse Reinforcement Learning", *In Proceedings of the Twenty-first International Conference on Machine Learning*, 2004.

27. Atkeson, C. G. and S. Schaal, "Robot Learning from Demonstration", *Proceedings of the Fourteenth International Conference on Machine Learning*, 1997.

28. Atkeson, C. G. and S. Schaal, "Learning Tasks from a Single Demonstration", *IEEE International Conference on Robotics and Automation*, 1997.

29. Guenter, F., M. Hersch, S. Calinon and A. Billard, "Reinforcement Learning for Imitating Constrained Reaching Movements", *RSJ Advanced Robotics, Special Issue on Imitative Robots*, Vol. 21, No. 13, pp. 1521–1544, 2007.

30. Kolter, J. Z., P. Abbeel and A. Y. Ng, "Hierarchical Apprenticeship Learning with Application to Quadruped Locomotion", *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*, 2007.

31. Aamodt, A. and E. Plaza, "Case-Based Reasoning; Foundational Issues, Methodological Variations, and System Approaches", *AI Communications*, Vol. 7, No. 1, pp. 39–59, 1994.

32. Graf, C., A. Härtl, T. Röfer and T. Laue, "A Robust Closed-Loop Gait for the Standard Platform League Humanoid", *Proceedings of the Fourth Workshop on Humanoid Soccer Robots in conjunction with the 2009 IEEE-RAS International Conference on Humanoid Robots*, 2009.

33. Atkeson, C., A. Moore and S. Schaal, "Locally Weighted Learning", *AI Review*, Vol. 11, pp. 11–73, April 1997.

34. Liu, J. and M. Veloso, "Online ZMP Sampling Search for Biped Walking Planning", *Proceedings of the IEEE/RSJ International Conference on Intelligent*

Robots and Systems, 2008.

35. Lenser, S. and M. Veloso, "Visual Sonar: Fast Obstacle Avoidance Using Monocular Vision", *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 886–891, 2003.

36. Hoffmann, J., M. Jüngel and M. Lötzsch, "A Vision Based System for Goal-Directed Obstacle Avoidance Used in the RC'03 Obstacle Avoidance Challenge", *In 8th International Workshop on RoboCup*, 2004.

37. Gerkey, B. P., R. T. Vaughan and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems", *In Proc. of the Intl. Conf. on Advanced Robotics*, 2003.

38. Lenser, S. and M. Veloso, "Sensor Resetting Localization for Poorly Modelled Mobile Robots", *International Conference on Robotics and Automation*, 2000.

39. Gouaillier, D., V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. L. 0002, B. Marnier, J. Serre and B. Maisonnier, "Mechatronic design of NAO humanoid", *International Conference on Robotics and Automation*, 2009.

40. Bruce, J., T. Balch and M. Veloso, "Fast and Inexpensive Color Image Segmentation for Interactive Robots", *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Japan, October 2000.

41. Coltin, B., S. Liemhetcharat, Ç. Meriçli and M. Veloso, "Challenges of Multi-Robot World Modelling in Dynamic and Adversarial Domains", *Workshop on Practical Cognitive Agents and Robots, 9th International Conference on Autonomous Agents and Multiagent Systems*, 2010.

42. Coltin, B., S. Liemhetcharat, Ç. Meriçli, J. Tay and M. Veloso, "Multi-Humanoid World Modeling in Standard Platform Robot Soccer", *Proceedings of 2010 IEEE-RAS International Conference on Humanoid Robots*, 2010.

43. Browning, B., J. Bruce, M. Bowling and M. Veloso, "STP: Skills, Tactics and Plays for Multi-Robot Control in Adversarial Environments", *IEEE Journal of Control and Systems Engineering*, Vol. 219, pp. 33–52, 2005.