Building a behavior simulator for autonomous robot soccer

Octavian Neamtu

May 14, 2012

Contents

1 The problem **2**
- 1.1 Platform constraints . . . 2
- 1.2 Inefficient emergent behavior . . . 2
- 1.3 Unexpected situations . . . 3
- 1.4 Choosing the best behavior . . . 3

2 Proposed approach **4**
- 2.1 Overview . . . 4
- 2.2 Other simulators . . . 5
- 2.3 Simulator description . . . 5
  - 2.3.1 Simulator design . . . 6
  - 2.3.2 The world model . . . 6
  - 2.3.3 The cognitive controller . . . 7
  - 2.3.4 The motion controller . . . 8
- 2.4 Modeling data . . . 8
- 2.5 Assumptions and sources of error . . . 9

3 Setup and framework **10**
- 3.1 Roadmap . . . 10
- 3.2 Capturing the robot's cognitive state . . . 10
- 3.3 Interpreting and representing the robot's cognitive state . . . 12
- 3.4 The ground truth . . . 14
- 3.5 Bringing the agent up to par . . . 15

4 Results and future work **17**
- 4.1 Data recording benchmark . . . 17
- 4.2 Data recording tests and use in debugging . . . 18

5 Future work **19**
- 5.1 Future development . . . 19
- 5.2 The simulator as a debugger . . . 20
- 5.3 Applying machine learning . . . 20
- 5.4 Developing robot foresight . . . 20

6 Conclusion **21**

List of Figures

1 A typical Robocup situation – the robotic agent need to react quickly in order to get to the ball before their opponents have the chance to. 2
2 A robotic agent in an unexpected situation – an opponent blocking the path to the goal. 4
3 Webots in action. 6
4 Simplified simulator design for one robot. 7
5 The organization of the memory module and the logging module. The arrows show how the data is passed down from the cognition modules down to the actual output. 11
6 The flow of the robot state data through the system. It starts from the robot's cognitive modules and then it travels through a series of loggers and parsers to be stored remotely on disk. The stored data can then be inspected at a later time. 13
7 Tool viewer organization. View 1 and view 2 are representing memory object 1 in two different ways, and memory object 1 is represented by view 3 after it is converted to the needed intermediate model. 13
8 The tool showing a typical recorded log. The Nao camera image and the visual detections are shown in a picture on the left, while sensor data is shown in generic views on the right. 14
9 Image point cloud (left) and tool ground truth view (right). The orange circle represents the ball and the blue circle represents a robot. The green and red lines represent the original Kinect coordinate system. . 15
10 Recorded streams with individual message timestamps. By comparing the value each of the timestamps against each other we can find an absolute ordering for all of the messages which can be used to replicate the order of the states as they happened. 16
11 Graph of the ball distance data. The black dots represent ground truth distances, the red dots represent robot-estimated ball distances and the blue curve is the curve-fitted function. 19

Introduction

This project aims to analyze the difficulties which a human observer encounters when trying to build efficient robotic behavior in the context of the RoboCup Standard Platform League. A solution is then proposed to help eliminate some of these difficulties: a light-weight simulator platform that realistically simulates the robotic agent's environment.

Humanity's interest in Artificial Intelligence (AI) is well-known. With the advent of the first early computers, there was an explosion in AI research, and the community hoped to achieve true AI in a matter of years. Of course, that breakthrough did not occur, proving that nature's evolutionary achievements are not that easy to replicate. Early AI researchers believed in a top-down cognitive approach to AI in which an agent would first analyze its environment and store new learned information, and then act it would accordingly. In this way it could build an abstract map of its environment.

The arrival of the first robotic agents marked a new chapter in AI, but this time scientists approached the problem with a bottom-up philosophy. Such early robots were simple agents that had a light sensor and a means of locomotion and would try to stay away from the light or follow the light [Arkin, 1998, p. 10]. Such reflexes were very simple in design, but highly reactive, and together they lead to emergent behavior. Such highly reactive robotic systems gave way to what today is described as *behavior-based robotics*, a methodology that imposes a general, biologically inspired, bottom-up philosophy [Wilson and Keil, 2001, p. 74]. Behavior-based robotics provides a robust basis for real, physical robots because it only tries to achieve some limited number of goals by using direct input from its sensors and sending direct signals to its actuators.

RoboCup, an autonomous robotic soccer competition, is the perfect example of the success of behavior-based robotics. The game of soccer has a definite objective, which is to score as many goals as possible without being scored upon. The environment is also well known before every match, since most constraints such as field size and ball shape can be considered invariant. Players should respond as fast as possible and make informed decisions without too much analysis. A behavior-based system is very well suited to perform some of the tasks that lead to scoring a goal, such as kicking when the ball is close or scanning for the goal posts before attempting a kick. Figure 1 shows an example of such a situation where the player is expected to approach the ball and dribble it before the opponent has the chance to steal the ball. However, despite encoding seemingly logic basic reflexes for the agent, the emergent behavior might not always be desirable in a task as complex as soccer.

In four of the five RoboCup soccer leagues the teams construct their own robots, so in some cases the teams can find a hardware solution to behavior problem, such as adding new sensors that can provide more information which shapes the emergent behavior. However, in the Standard Platform League (SPL), each team uses standard



Figure 1: A typical Robocup situation – the robotic agent need to react quickly in order to get to the ball before their opponents have the chance to.

hardware, produced by a third-party provider, to which no modifications can be made. This introduces very specific constraints on how behavior can be shaped.

1 The problem

1.1 Platform constraints

The Standard Platform League uses the Aldebaran Nao as its hardware platform. The Nao Nextgen comes equipped with a 1600 MHz AMD Atom processor, which is roughly as much processing power as the average low-end smartphone. To be functional in real time, the system needs to run at 30 frames per second, or in other words, to process each image frame in 33 ms. After image processing, object detection, and localization, all of which are very processor-intensive tasks [Morrison, 2010], not much time is left to the AI decision agent. This is another reason thata behavior-based system is ideally suited for SPL needs. Besides having the advantage of being more reactive than other choices, a behavior-based system is also not as computationally intensive as other approaches to AI, which makes it the best choice for our purposes.

1.2 Inefficient emergent behavior

Imagine a human soccer player with a ball in front of him. If the goal is clear, the player will approach the ball, attempt to align with it, and then kick it in the direction of the goal. In a similar RoboCup situation, it would seem logical to have the robot

perform the same series of actions. This would hopefully lead to a successful *emergent* behavior - that is, a behavior that makes the agent appear intelligent in the sense that the robot would indeed score a goal. However, in most cases, it may turn out that having the robot kick the ball immediately in the general direction of the goal, rather than aligning first will increase the chance of the robot scoring. The second behavior would thus be more efficient than the first and would obviously be more desirable.

Therefore it might not always be apparent or intuitive to a human observer what the optimal behavior will be for any given subtask. There are so many inputs involved with even a simple behavior like kicking that it is difficult for a human observer to guess how to best use the inputs in order to produce an efficient behavior.

1.3 Unexpected situations

One of behavior-based robotics' greatest pitfalls is the lack of adaptability [Wilson and Keil, 2001, p. 75]. If unexpected constraints appear, then the emergent behavior might not be able to achieve its goal. For example, if a human player is blocked by an opponent, then he might consider passing the ball to a free teammate since trying to kick the ball straight at the goal would be inefficient. However, a robotic agent would not discover this alternative on its own. Put in a similar situation, a simple kicking agent would try to kick it into the blocking opponent (see Figure 2), which would reflect off the opponent with undesired consequences. Unless the robotic agent's behavior was built with a passing strategy in mind, the agent could not "learn" this behavior by itself. It is up to a human observer to identify these different situations and keep them in mind when building agent behavior. Of course, this presents a problem: anticipating all different situations that could occur in a soccer game is intractable.

1.4 Choosing the best behavior

Assume that a human observer makes changes to an existing behavior in order to improve it. Without extensive testing, it would be difficult for the observer to decide whether the new behavior is more effective than the old one. A good methodology may be having a robot player using the new behavior and a player using the old one play ten games; at the end of the games, the behavior that resulted in more goals can be considered more succesful. However, this method is actually completely unfeasible. Then the better behavior would be the one that the high-scoring robot is using. Taking into account that the average time required to play a RoboCup SPL game is around 30 minutes, it would take the observer five hours to run this benchmark. Coupled with the large number of choices in the behavior possibility space as detailed in sections 2.2 and 2.3, it is highly improbable that the observer will

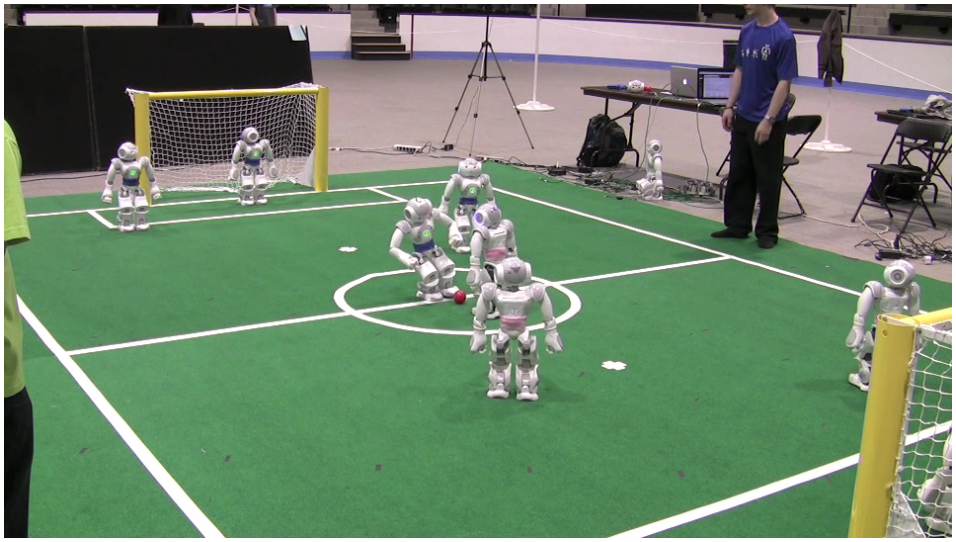


Figure 2: A robotic agent in an unexpected situation – an opponent blocking the path to the goal.

find an efficient behavior in any short period of time. Testing behaviors manually is thus inefficient and ineffective.

2 Proposed approach

2.1 Overview

In order to mitigate these problems, we propose a technique that will allow a human observer to more easily improve robotic behaviors. This will be accomplished by allowing the observer to test many possible behaviors in a simulated environment and quickly determine which one is the most appropriate.

Therefore we propose to build a platform that will be able to realistically simulate an entire Nao soccer game in a few milliseconds. Such a platform would allow one to compare the effectiveness of a new behavior against previous behaviors by analyzing the outcomes of a large number of simulated games. For example, if someone wants to determine whether a new kicking behavior improves the agent's soccer ability, they would simulate a set number of games where a team of robotic players is using the old behaviors and the other team is using the new behaviors . The better behavior would be chosen based on a metric, such as average number of goals per game.

In the rest of this section we will review other existing simulators, detail the design of our own simulator, and analyze the setup and framework needed to develop it. The next sections will then detail the steps taken to date in building the necessary framework, as well as results and future work.

2.2 Other simulators

The proposed simulator needs to be light weight but also true to real-world situations. The two major robotic simulators widely used for Naos are Cyberbotics' Webots, shown in action in Figure 3, which is available as a commercial software package, and SimRobot, an open-source academic package. Both of these simulators are focused on around representing the entire physical world and all the forces and interactions that occur in a real environment. Two major problems arise with this approach.

The first is that modeling all of the physical interactions, such as collisions, that happen in the real world is a very complex task. This translates into a high processor load and a simulation that is not much faster than the real world alternative.

The other problem is that the simulator will build a world that follows a generic set of rules and constraints that may not always stay true to the complex physical world. The simulation will act naturally according to the simulator's generic set of rules, but there will always be subtle differences between the simulation and the real world that the simulator cannot account for. For example, the simulated image in Webots will not take into account the distortion that results from using a rolling-shutter camera [Laue and Rofer, 2008]. Furthermore, none of these simulators models the motors that power the joint actuators well enough to mimic the robots' motions realistically [Hofmann et al., 2011].

The main use for the previously described simulators in the context of RoboCup SPL is usually in the early phases of developing motion systems. Teams can run such systems in a simulator until they are mature enough to be tested on the real robots so that they don't damage the robot's hardware with dangerous joint commands. However, once the motion systems reach maturity, the simulator's shortcomings usually prove to be a hindrance to further development. At this point it becomes necessary to continue development on a physical robot.

2.3 Simulator description

The main difference between the previously described simulators and the proposed simulator will be its level of action: our new simulator will act at a very abstract level and will simulate a robot's environment cognitively rather than physically. Instead of simulating a robot's actuators interacting in the real world, it will simulate the robot's generate motion commands ; instead of simulating the image a robot should see, it will simulate what objects the robot could detect in the image in a particular situation; and instead of simulating the hardware-level ethernet packets the robot would send, it will simulate what information the robot communicates.

Therefore, rather than trying to model the physics of the real world, the simulator will try to break down the behavioral interactions of the robot with the real world into discrete events like walking, kicking, moving the head, sighting an object and communicating. It will then attempt to realistically simulate the outcome of each

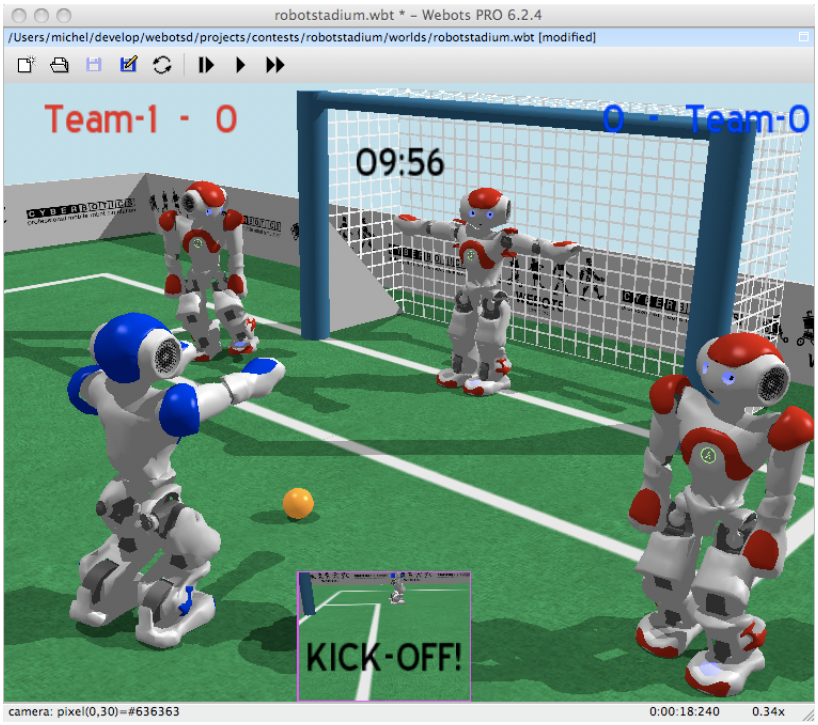


Figure 3: Webots in action.

of these events and how they change the state of the environment without complex physical calculations.

2.3.1 Simulator design

The simulator will be designed as an extended model-view-controller (MVC) system where the simulated state of the world constitutes the primary model, as detailed in Figure 4. A cognitive controller then transcribes the state of the world into cognitive models of what each of the robotic agents should perceive. Each agent's behavior controller then makes decisions based on its cognitive input and sends motion commands based on those decisions back to the motion controller. The motion controller aggregates all of the motion commands it receives and translates them into world model movements. The view depicts the state of the world and is synchronized with the world model; it helps in tracking the state of the simulator and provides a visual aid but is entirely optional.

2.3.2 The world model

The world model will contain the current arrangement of the simulated robots and the ball on the field as well as other intrinsic information about their states. Each of the robots is represented by its position on the field, its heading, and its head

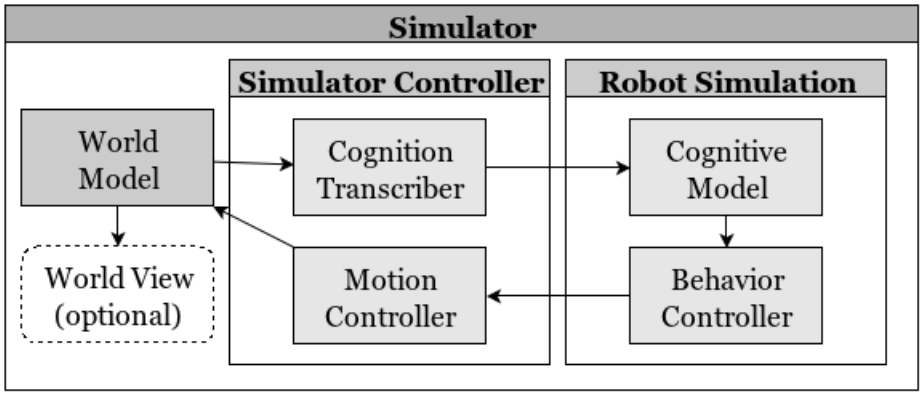


Figure 4: Simplified simulator design for one robot.

position. The ball is represented by its position on the field, its current velocity, and its current acceleration.

2.3.3 The cognitive controller

The cognitive controller has the task of simulating the robot's perception process. In a real-world situation the robot will use its sensor inputs to build up an accurate description of its environment. From the perfect world model, the cognitive controller will try to reproduce what the robot's cognitive state would be if the robot was actually in that particular real-world situation.

To make this simulation feasible, the cognitive controller will simulate each part of the robot's aggregate belief state separately. Furthermore, only the aspects needed directly by the behavior controller will be simulated.

The list of simulated aspects will thus be restricted to the following:

Ball Belief

-
- Velocity in the robot's relative frame of reference
- Position uncertainty
- Whether the ball is visible to the agent or not
- "Heat" value - whether an opposing robot is likely to kick the ball or not

Localization Belief

- A position represented by an (x, y) point in the field frame of reference
- A heading representing the direction the robot is facing in the field frame of reference

- Position and heading uncertainty

Communication Belief

- Teammates' localization information
- Teammates' ball information

2.3.4 The motion controller

The motion controller has the task of translating the robot's motion commands into real world movement. The the controller will need to simulate the following list of commands:

Walk Commands

- **Speed** walk commands, which provides the robot a walk vector that the robot is expected to follow at maximum speed
- **Destination** walk commands, which tell the robot to walk to a given destination in its relative frame of reference at a given speed

Head Commands

- **Head angle** commands which tell the robot to position its head at certain joint angles
- **Look-to** commands which tell the robot to position its head in order to look at a given point in the robot's relative frame of reference

Kick Commands

- **Kick** commands that specify one of the various kick motion primitives that are available

Besides the motion commands, the controller will also have to update the ball model within the world model according to its acceleration values.

2.4 Modeling data

In order to simulate the various motion commands and agent beliefs, the controllers will utilize probabilistic models of each of these events' and observations' possible outcomes. Thus, in order to create an accurate simulator we will need to derive these probabilistic models from typical robotic agent interaction with its environment.

For example, in order to simulate a robot kicking the ball straight, we would first need to know the usual outcome of the robot kicking the ball straight. This means that we would need to observe the robot kicking the ball straight a certain number of times and recording how far the ball traveled relative to its original position and the

delay between the robot making the kick decision and the kick execution. Based on such data, we would derive a probabilistic model that accurately simulates a possible state of the world after the robot kicks the ball straight.

The key to this method is recording real data about a robot's cognitive perception synchronized with data about the true state of its environment. In the example above we'd rely on knowing the distance the ball traveled, which we can infer from knowing the true state of the environment, and also knowing the moment the robot perceives as executing kick, which we can infer from knowing the cognitive state of the robot. Once we have a large amount of data for a particular event, such as a robot kicking or sighting an object, we can extrapolate the usual outcome from our environment data. The extrapolation can be accomplished with either a curve-fitting algorithm or a simple linear interpolation. For example, depending on the distance and the angle of between the camera and the ball, the robot will perceive the ball as either being visible or not in sight.

The true state of the environment can be accurately described by simply recording the position of all the objects in the environment. The true position of an object on the field is known as its "ground-truth" position because the object is assumed to be resting on the ground. Ground-truth detection systems have been used in the RoboCup SPL extensively, and one recently-developed low-cost system is especially practical [Khandelwal and Stone, 2012]. This alternative uses a Microsoft Kinect sensor coupled with custom software based on the popular Robot Operating System (ROS) to track the real positions of a robot and the ball on the field. It is especially attractive because it does not require any specialized hardware; the Kinect is readily available in most electronics stores, and it does not require any special installation for use in the laboratory.

2.5 Assumptions and sources of error

The proposed simulator will not exactly model the real world due to the simplifying assumptions made from the onset of this project.

The first assumption is that we can accurately simulate the interaction of the agent with the environment by simulating only a restricted subset of its beliefs and motions and ignoring the effects of any interaction not in this subset. One important interaction that is ignored in the simulator is the robot falling down. For the purposes of the simulator, we assume that the robot would not fall at any time. Modeling a robot fall is not feasible because it is a highly unpredictable event with a large number of possible outcomes.

The next assumption is that each event's outcome depends on a small number of measurable parameters and that its outcome can be described based on the analysis of a large sample of event data. In actuality, there will also be events that may happen in a real game situation that the simulator will not take into account. For example, the simulator may not consider the possibility of unexpected objects obstructing the

field, even though this is a situation that can occur in SPL games since there is a referee moving on the field.

Another source of error is introduced by the ground-truth system, whose estimation errors are known [Khandelwal and Stone, 2012]. This will affect the accuracy of event modeling.

3 Setup and framework

3.1 Roadmap

In the previous section we came to the conclusion that we need to compare the robot's cognitive state against the ground truth state in order for our simulator to model events accurately.

To that end we decided to first build a framework that could easily record the robot's cognitive state. Since the ability to inspect the robot's cognitive state remotely while the robot is playing is essential to debugging the robotic platform as a whole, we also decided we needed to incorporate that capability into our design.

The second step in our roadmap was to implement a ground truth system and integrate it with our recording framework design.

As we began testing the logging framework we noticed that the robot's walk was not stable enough for us to capture significant amount of data uninterrupted because the robot kept falling down. This posed such a serious problem to our setup that in order to make further progress we needed to improve the robot's motion system.

The rest of this section is dedicated to detailing each of the development phases mentioned above with regards to the design, implementation, and challenges faced in developing the framework. Results and future challenges are then discussed in the following section.

3.2 Capturing the robot's cognitive state

The first step in this project was developing a fast and reliable way to record the robot's cognitive state. To that end we have developed memory and logging modules for the Northern Bites platform.

The logging system uses the Google Protocol Buffer [Google, 2011] framework, which Google uses internally for passing messages between systems. This choice was made because of its empirically proven reliability and efficiency. Every Google search query is transmitted using a protocol buffer.

The memory module provides a series of "memory objects," simple data structures which describe one aspect of the robot's cognition, such as image data, sensor data or perceived visual objects. These objects listen to the robot's different cognitive systems, update themselves to reflect the new data when a cognition state changes, and also update their timestamp at the time of the change. A memory object has methods for producing a serialized representation of itself in the form of a string of bytes and for de-serializing such a string back into the original memory object. It is also designed with thread-safety in mind; while the object is in the process of updating its state, it guards against a different thread attempting to access the object's serialized form.

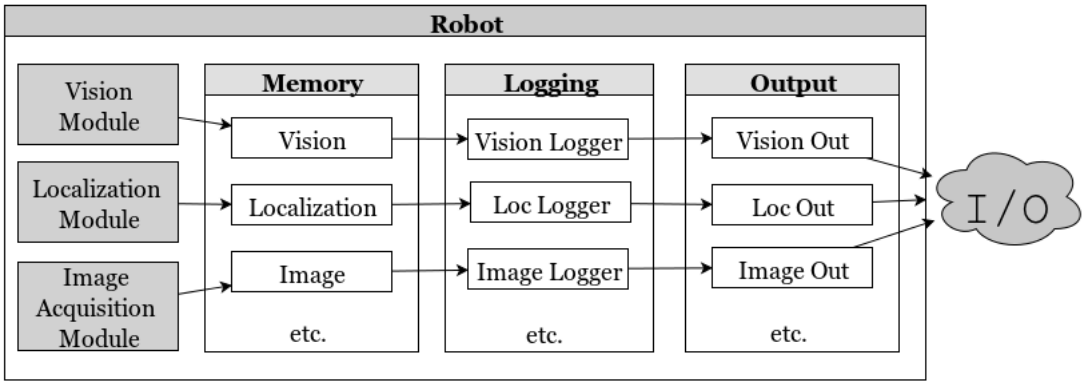


Figure 5: The organization of the memory module and the logging module. The arrows show how the data is passed down from the cognition modules down to the actual output.

The logging module provides logger and parser type objects that can attach to a memory object. They can either use the memory object serialization to write the object states into a stream or use the de-serialization to read a stream of saved states back into a memory object from various data sources.

A logger first writes a header to the stream. Then, each time its memory object updates, it will retrieve the serialized string form of the object and write the size of the string and the string itself to the stream. By having the size of each message in the stream, the parser knows how the messages are delimited. The parser simply needs to first read in the log header then sequentially read each string size and string, which can be used to de-serialize the memory object into its original form. Each parser and each logger runs on a separate thread in order to minimize their interference with the rest of the system.

Having each memory object captured in its own stream is important to the modular design of the memory system. New memory objects can easily be added without interfering with the handling of other memory objects. It also makes it easy to selectively decide capture certain memory objects and not others.

Each stream is handled by special objects called input providers and output providers.

All input providers implement an *InProvider* interface, and all output providers implement an *OutProvider* interface. This is an important layer of encapsulation that separates the physical implementation of the input-output (IO) stream from the generic stream interface exposed to the loggers and parsers. For each different IO device we have a pair of such providers; for example, for network IO we have a socket input provider and a socket output provider; and for regular file IO we have a file input provider and a file output provider.

As seen in Figure 5, the logging module's central interface is an association object that keeps a map of the memory objects and their assigned logger or parser. This provides encapsulation and and separates the memory module, which only represents the current cognitive snapshot of the robot, from the logging module, which concerns itself with streaming or saving the data from the memory module.

3.3 Interpreting and representing the robot's cognitive state

Even though we are able to log the robot's cognitive state, this is not useful unless we have a way of manipulating and representing the log. To that end we developed a graphical user interface (GUI) tool to aid in debugging, manipulating, and visualizing the streams recorded by the logging module. The tool was developed using the Qt framework, a "a cross-platform application and UI framework with APIs for C++ programming" [Nokia, 2011]. Qt was chosen for its UI creation capabilities, the ease of interfacing with the existing C++, codebase and the strong performance of a system written in a traditional compiled language such as C++.

The tool is based around an offline instance of the robot memory. To populate the offline memory it uses a parsing module that is the complement of the robot's logging module. The offline memory can represent either a snapshot of the robot's brain stored on disk, or a live snapshot from a remote robot. Since the physical streams and their handlers are abstracted into generic streams, the tool can operate without needing to have any further information about the stream type.

The two most useful datastream flows are shown in Figure 6. The tool can either parse archived streams stored to files on disk or connect to a live stream from a remote robot. In the latter situation, a logging module can be attached to the offline memory in the tool in order to log the streamed data to disk. In this way, the robot's cognitive stream can be archived and synchronized with ground truth data for easy offline processing later on.

The offline robot memory module, along with the parsing module and the logging module, are all controlled by the data manager, which provides a transparent interface to access the memory. It also provides methods for advancing or rewinding the logs.

The tool is made up of a collection of single-purpose modules that use and manipulate the data in robot memory to different ends. Each module uses a shared collection of GUI widget classes to represent data from memory objects. These classes use

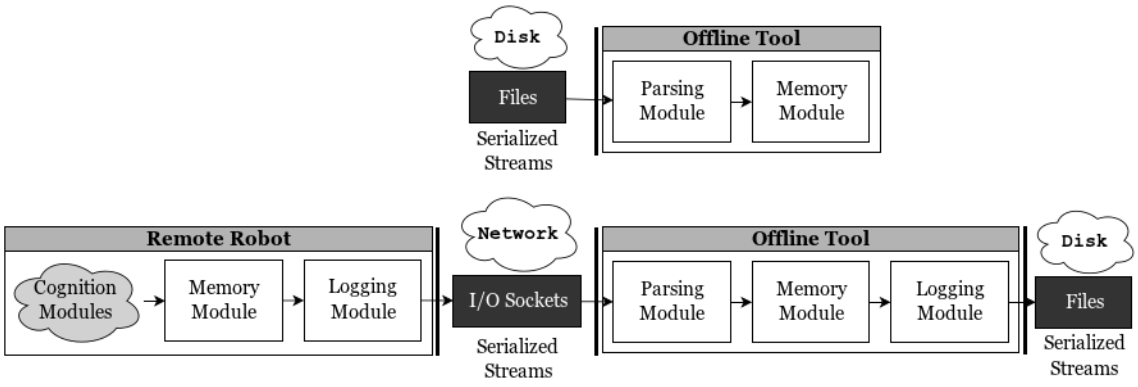


Figure 6: The flow of the robot state data through the system. It starts from the robot's cognitive modules and then it travels through a series of loggers and parsers to be stored remotely on disk. The stored data can then be inspected at a later time.

an altered model-view design to represent the data from the memory objects. It is not a traditional model-view pattern because sometimes the model's data format is different from the view's expectation and in that case another model layer is added between the view and the memory object model for conversion. This addition was especially useful for converting the robot image data, which is stored in the YUV422 format, into the more convenient RGB32 format.

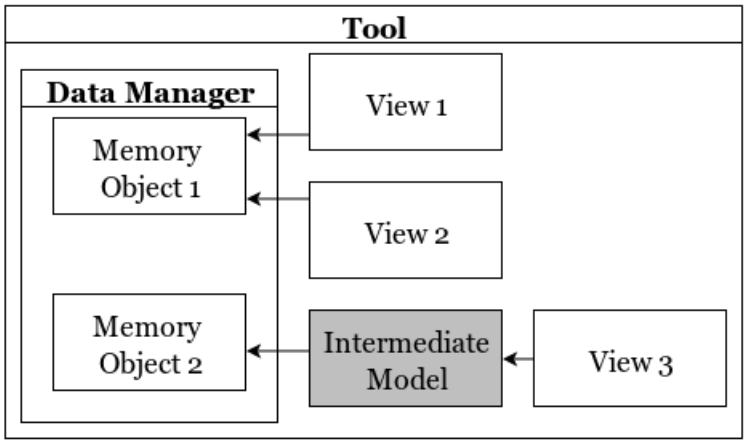


Figure 7: Tool viewer organization. View 1 and view 2 are representing memory object 1 in two different ways, and memory object 1 is represented by view 3 after it is converted to the needed intermediate model.

In order to update the view synchronously with its underlying model, a publisher-subscriber design was implemented. The data manager acts as a publisher to all of the views, and each of the views can subscribe to updates from their respective memory objects. When new data enters the offline memory, the data manager propagates update signals to all of its subscribers so that they can react accordingly. A simplified

representation of this design is shown in Figure 7.

The collection of views thus allows a human observer to easily inspect the cognitive state of the robotic agent. An example of GUI showing some of these views is shown in Figure 8. Since protocol buffer messages are used to store data in the memory objects, we were able to create a generic memory object view that takes advantage of the reflective capabilities of protocol buffer messages in order to display data from the memory object in a tree-like hierarchy. Specialized views exist for memory objects that contain data that have relevant visual representations. For example, robot location belief is better represented visually, as a dot drawn on a field, than as a tuple of numbers.

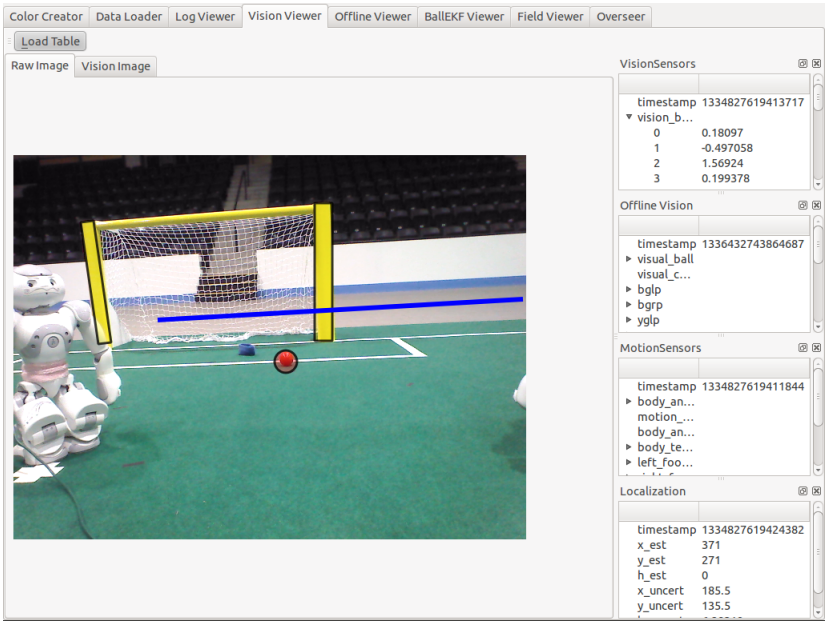


Figure 8: The tool showing a typical recorded log. The Nao camera image and the visual detections are shown in a picture on the left, while sensor data is shown in generic views on the right.

3.4 The ground truth

The second step in our roadmap was to set up a ground truth system in our laboratory. Over time, various systems and setups have been used to efficiently keep track of ground truth in a RoboCup setting. For the aforementioned reasons, the solution that seemed to provide the flexibility needed for this project was that proposed by Khandelwal and Stone [2012]. This alternative uses a Microsoft Kinect sensor to capture point-distance data, represented visually in Figure 9, from which location estimates are extracted.

Using the code release accompanying the paper released by Khandelwal and Stone [2012] as a basis, we created a similar setup in our own laboratory. In order to enable

a human observer to easily inspect the ground truth data and also to synchronize it with the collected agent cognitive data, we imbued the existing system with server-streaming capabilities.

We built a simple TCP server that continuously transmits the ground truth information to a connected client streaming at the maximum framerate of detection . The protocol itself is an abstraction of the one used for the memory and logging module and uses the same combination of Google Protocol Buffers and logging and parsing streams. The ease of reusing the same approach in this new situation is a testament to the modularity and generality of the system.

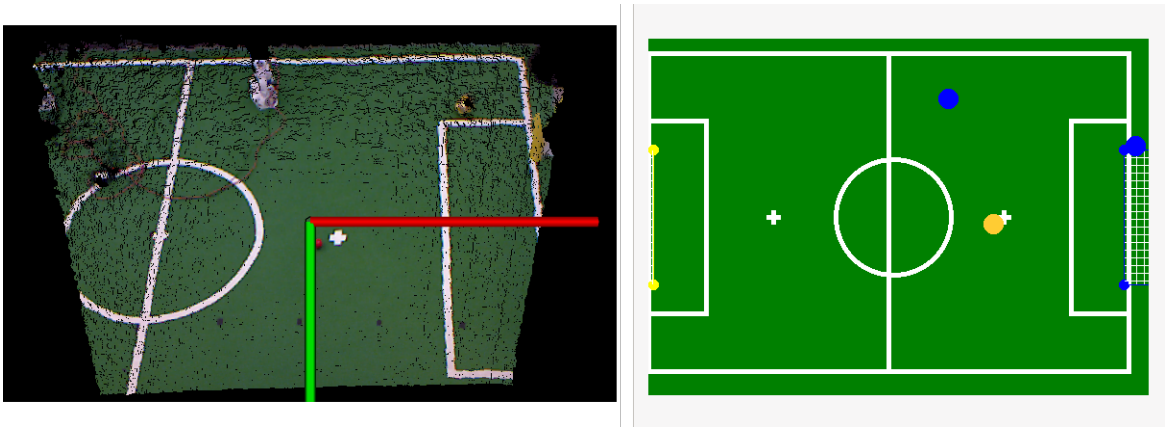


Figure 9: Image point cloud (left) and tool ground truth view (right). The orange circle represents the ball and the blue circle represents a robot. The green and red lines represent the original Kinect coordinate system.

Synchronizing the two is a well-known problem in the context of RoboCup SPL [Niemller et al., 2011]. Having both the agent cognitive data and the ground truth data streaming to a centralized tool makes the problem of much easier to tackle. Our approach was to synchronize the time on the robot system and the ground truth system using the network time protocol (NTP). As shown in Figure 10, one can then parse the messages in each stream according to their timestamp t_i in order to get an accurate timeline of the ground truth and the cognitive state of the agent.

The advantage of using this approach is its inherent simplicity, with the downside being that small differences of even a few seconds in synchronization can be disastrous. In order to make the system robust in the face of these kinds of deviations, we check if the timestamps from the two different systems match up within a certain threshold. This guards against instances where the systems are considerably out of sync.

3.5 Bringing the agent up to par

When attempting to record combined agent state and ground state logs we encountered a major problem in that the robot's walk was not stable enough to sustainably

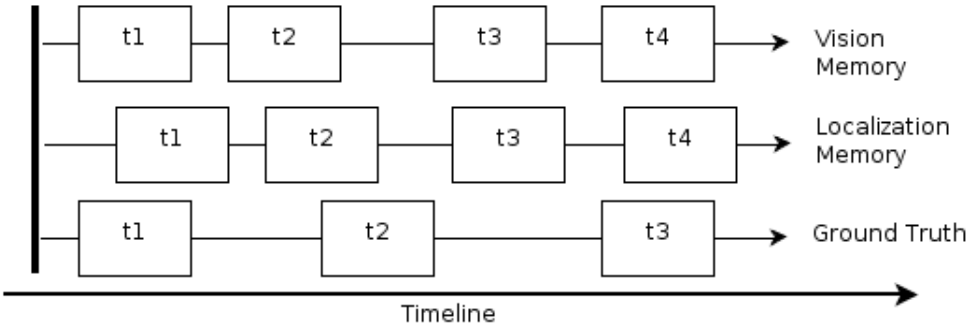


Figure 10: Recorded streams with individual message timestamps. By comparing the value each of the timestamps against each other we can find an absolute ordering for all of the messages which can be used to replicate the order of the states as they happened.

record significant amount of data before falling down. Besides the physical damage to the robot, falling compromises the recorded data by placing the agent in a situation that the simulator assumes impossible.

The goal of the simulator is to realistically synthesize the motions of a robotic soccer match, yet there are aspects that it cannot easily simulate because of their highly random and chaotic nature. The most important of these, and the one which most often occurs in RoboCup SPL matches, is the robot falling off of its feet.

This event is highly random: it could happen because of small bumps in the playing field; it could happen while executing a motion primitive around another robot; or it could happen because of weaker joints due to joint strain. The outcome is also highly unpredictable since the robot can fall in any direction, can become a potential obstacle while in its fallen state, and will more than likely lose its correct localization belief. Falls also happen often enough within our platform that not factoring them into the simulator would pose a serious limitation.

Since there is no feasible way to make the simulator predict this event realistically, we instead decided to minimize the probability of the robot falling down. We singled out our walking engine as the major cause of a large portion of the falls.

This walking engine is a product of the research of Strom et al. [2010] and has been used by the Northern Bites since 2009. Unfortunately the robustness of the system has declined over the years due to a lack of maintanence, which has severely impacted the stability of the robot's walk [Chown et al., 2011].

In order to minimize the probability of the robot falling we decided to adopt a more robust walking engine that is better suited to deal with random external perturbations. Thus we ported team BHuman's walking engine and integrated it with our own platform. Their closed-loop gait system is well-known for being the fastest and stablest Nao walking system in the context of RoboCup SPL [Laue and Rofer, 2010].

4 Results and future work

4.1 Data recording benchmark

One measure of the success of this system is whether or not there are any frames dropped in the system from acquiring the data from the cognition modules down to recording it remotely. A large number of dropped frames would mean the system wouldn't be reliable enough for our purposes. In order to benchmark the streaming and recording system we decided to compare the frequency at which the streamed data was being recorded versus the frequency it was processed with on the robot. If the system was losing messages or was not transmitting messages fast enough then the recording frequency would be expected to be lower. For the experiment we measured the recording frequency for the vision stream direct to the robot disk and remotely on two machines M_1 and M_2. M_1 was a 2010 Mac Pro with an 2.40 GHz Intel Xeon E5620 with 8 GB of RAM and M_2 was a 2008 Macbook Pro with an 2.40 GHz Intel Core2 Duo P8600 CPU and 2 GB of RAM. The streaming tests were run over standard ethernet. Each test was run for about a minute. The normal test was run with image streaming turned off and the heavy test was run with image streaming turned on. Fractions were rounded up to the nearest integer. Results are measured in frames per second (fps) and are summarized in Table 1.

	Normal Test	Heavy Test
Disk	29 fps	29 fps
M_1	29 fps	26 fps
M_2	29 fps	22 fps

Table 1: Frequency measurements for streaming vision data.

Since the robotic vision runs in sync with the camera sensor on the robot, the vision stream is expected to run at 30 Hz. The results therefore show that streaming and recording the cognitive state of the robot is lossless. The heavy test was included as a test of how well the streaming framework can reliably transmit even large amounts of data remotely; streaming and recording images, as invaluable as it is for other purposes, is not part of the data we need to record for simulation purposes. Even with image streaming on the performance does not suffer significantly, as we only see a loss of a few frames per second.

The system also is robust enough that it does not need to use any special network application protocol for establishing and managing the client-server connection. Exceptions such as connection being closed abruptly are handled in such a manner that the system can quickly recover.

The ground truth system has been found to be equally as reliable in transmitting data. The system was in continuous use for an uptime of two weeks without any significant reliablity issues. We ran the system on two machines M_3 and M_4 to test its performance. M_3 was a 2008 Macbook with an Intel Core2 P7200 CPU @ 2.00

GHz and 1 GB of RAM and M_4 was a 2008 Mac Pro with an Intel Xeon 5150 CPU @ 2.66 GHz and 2 GB of RAM. We ran the tests with 1 client, 5 clients and 10 clients connected over ethernet for a minute or so. The results are shown in 2.

	M_3	M_4
1 client	1 fps	9 fps
5 clients	1 fps	11 fps
10 clients	1 fps	10 fps

Table 2: Frequency measurements for streaming ground truth data.

The differences in number of clients connected does not influence the frequency of the system. Because ground truth detection is very processor-intensive, the only bottleneck in the system is the speed of the processor. The random fluctuations in frequency for M_4 are a result of random fluctuations in the processor load caused by other programs than the ground truth detection. We found that a frequency of 10 Hz provides a good resolution for ground truth data. Since the robot can move at most at 30 cm/s, in between two ground truth frames at 10 Hz it could have only moved about 3 cms, which is well within the ground truth system error. Thus even if the system could be made to run at a higher frequency, it wouldn't add enough precision to be worth investing the resources to do so.

4.2 Data recording tests and use in debugging

In all of the data recording test sessions we noticed that streaming cognitive data from the robot did not impact the agent's functionality in any noticeable way thanks to the non-intrusive design of the logging module. A short benchmark also revealed that the platform as a whole was not running any slower while streaming.

Thanks to the steps taken to remedy the issue of the robot falling down, we managed to record continuous logs of a few minutes in length. More research needs to be done to explore the potential issues of recording an entire RoboCup SPL half, which is about 10 minutes in length.

The utility of the framework we developed is also undeniable in debugging the entire robotic platform. For example, using some of the data recorded so far we managed to both test how accurate our current ball distance estimates are and improve them. We managed to extract a set of 460 data points of the form $(b_{pix}, \hat{d}, d)$ where b_{pix} is the ball's visual size in the image in pixels, $\hat{d}$ is the robot's ball distance belief and d is the ground truth ball distance. We discovered that the average error between the belief and the truth was 10 cm so we decided to switch the ball distance estimation system from a kinematics-based approach; instead we used the recorded data to curve-fit a function that will approximate the distance based on the size of the ball in the image as shown in Figure 11.

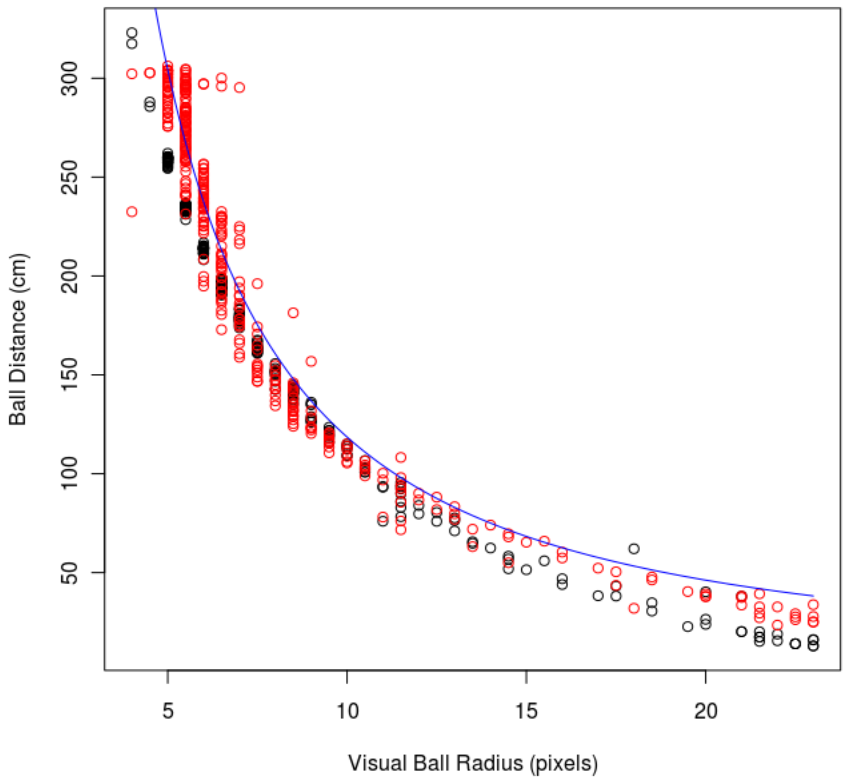


Figure 11: Graph of the ball distance data. The black dots represent ground truth distances, the red dots represent robot-estimated ball distances and the blue curve is the curve-fitted function.

The process of testing ball distances would have normally taken a few days of careful laboratory work, with elaborate setups and painstaking data collecting. With the new framework all of that was condensed into a few hours.

5 Future work

5.1 Future development

We are currently working on implementing the simulator system where each player has a perfect knowledge of its environment. After the simulator system is developed, we can incrementally and selectively make the simulation more realistic by making the beliefs and motions closer to what we expect them to be. The most important beliefs and motions then need to be modeled realistically; to that end we need to find the best modeling approach using collected recorded data.

5.2 The simulator as a debugger

One of the major motivations for building the proposed simulator is its usefulness in debugging the decisions that the agent makes. It is virtually impossible to debug the decisions that a robot makes in a real-world situation because pausing a robot to inspect its internal state would actually affect the way the robot will interact with the environment; the environment could change, momentum would be lost, or the pause could change the internal state of the robot. In other words one can pause the robot but cannot pause the environment; however pausing and even rewinding the states of a simulated environment can be easily accomplished in a simulator. Using such a platform, a human observer could step through the logic of a faulty decision and hopefully single out the source of the problem.

5.3 Applying machine learning

Reinforcement learning has been previously used successfully to optimize behavior in multi-agent simulated soccer environments [Stone and Veloso, 2000b]. The drawback of reinforcement learning in robot soccer is that it uses a trial-and-error process. In order to converge to an optimal solution, it takes an indefinite, often large, number of iterations. Because running each iteration would require playing a soccer game of some fixed duration, which would be impossibly time-consuming and would also wear the hardware down, it is clear that using reinforcement learning with physical robots is unfeasible. However, it is feasible with a simulation, and therefore the proposed simulator would enable the use of machine learning to improve behavior effectiveness.

The behavior hypothesis space for an agent playing soccer is virtually infinite, which makes applying machine learning to map the cognitive inputs of the robot to the outputs virtually impossible. In order to make the problem manageable, we propose the use of layered learning, a method that has been applied successfully in a similar situation: that of simulated AI agents playing soccer cooperatively [Stone and Veloso, 2000a]. In layered learning, the main task is broken up into subtasks and then the agent is trained on each of the subtasks. In our case, the main task is playing soccer, while the subtasks could be kicking, passing, or following the ball. The machine learning algorithm itself would use reinforcement learning to optimize the decision tree of the agent in completing each of these subtasks.

5.4 Developing robot foresight

Another opportunity that this simulator would present is that a robot could use it during a game to "foresee" what the outcome of switching to a different strategy might be. This would be done by initializing a simulation that mirrors the current game situation as closely as possible and then simulating the rest of the game a number

of times with different behaviors. Then the agent could determine which behavior would yield the best results.

6 Conclusion

The framework we have developed so far has provided us with a sturdy basis for advancing to the next stage of research and development. It has not only helped us advance our goal of building a realistic robotic soccer simulator, but has greatly benefited the Northern Bites robotic platform as a whole with easy access to advanced debugging and development machinery.

This paper also advances the concept of a robotic memory module that centralizes the entire cognitive state of the robotic agent. This centralization makes it easy to extend the memory module with a logging module, and adds to its reusability both on the robot and offline.

The framework design presented is also general enough to be used as a basis for other systems, and we hope our design will act as an inspirations to others trying to pursue similar goals.

References

Ronald C. Arkin. *Behavior-based robotics*. MIT Press, 1998.

Eric Chown, Jack Morrison, Nathan Merritt, Octavian Neamtu, Wils Dawson, Dani McAvoy, Elizabeth Mamantov, Edward Googins, Josh Zallinger, and Ellis Ratner. The northern bites 2011 standard platform league team, 2011.

Google. Google protocol buffers: Googles data interchange format, 2011. URL `http://code.google.com/p/protobuf/`. Documentation and open source release.

Matthias Hofmann, Soren Kerner, Ingmar Schwarz, Stefan Tasse, and Oliver Urbann. Nao devils dortmund - team description for robocup 2011, 2011.

Piyush Khandelwal and Peter Stone. A low cost ground truth detection system using the kinect. In Thomas Roefer, Norbert Michael Mayer, Jesus Savage, and Uluc Saranli, editors, *RoboCup-2011: Robot Soccer World Cup XV*, Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin, 2012. To appear.

Tim Laue and Thomas Rofer. Simrobot development and applications. *Workshop Proceedings of SIMPAR 2008*, 2008.

Tim Laue and Thomas Rofer. A closed-loop 3d-lipm gait for the robocup standard platform league humanoid, 2010.

John G. Morrison. Robotic vision with the hough transform. 2010.

Tim Niemller, Alexander Ferrein, Gerhard Eckel, David Pirro, Patrick Podbregar, Tobias Kellner, Christof Rath, and Gerald Steinbauer. Providing ground-truth data for the nao robot platform. In Javier Ruiz-del Solar, Eric Chown, and Paul Plger, editors, *RoboCup 2010: Robot Soccer World Cup XIV*, volume 6556 of *Lecture Notes in Computer Science*, pages 133–144. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-20216-2.

Nokia. Qt - a cross-platform application and ui framework, 2011. URL `http://qt.nokia.com/products/`.

Peter Stone and Manuela Veloso. Layered learning. In Ramon Lpez de Mntaras and Enric Plaza, editors, *Machine Learning: ECML 2000*, volume 1810 of *Lecture Notes in Computer Science*, pages 369–381. Springer Berlin / Heidelberg, 2000a. ISBN 978-3-540-67602-7.

Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8:345–383, 2000b. ISSN 0929-5593.

Johannes Strom, George Slavov, and Eric Chown. Omnidirectional walking using zmp and preview control for the nao humanoid robot. In Jacky Baltes, Michail Lagoudakis, Tadashi Naruse, and Saeed Ghidary, editors, *RoboCup 2009: Robot Soccer World Cup XIII*, volume 5949 of *Lecture Notes in Computer Science*, pages 378–389. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-11875-3.

Robert A. Wilson and Frank C. Keil. *MIT Encyclopedia of the Cognitive Sciences*. MIT Press, first edition, 2001.