Nao Improvements for Standard Platform League

T. Cooijmans, T.J.N. Hendriks, D.M. Nunes,
N.D.A. Petrakis, M.E. Romeijn

January 24, 2013

Abstract

This paper represents the attempt made to realise a fully functional Nao soccer team. In order to do so, the skills of the individual soccer player were researched; goalkeeping, walking, kicking and vision. These research efforts have resulted in a Nao capable of locating the ball, walking towards it and shooting it towards a goal. A framework has also been made for further implementations such as a state controller and localization in the field. Furthermore, research was done to improve balance of the Nao gait. This research in locomotion has given rise to inspiration for further work in the area of reinforcement learning.

1 Introduction

The expected result of the project illustrated below was to create a robot controller that enables a group of robots to play soccer. In order to do so, four Aldebaran Nao robots were supplied. These were to be used in a Standard Platform League style soccer match, which is one of the leagues present the Robocup event.

1.1 Robocup

In order to promote artificial intelligence and robotics research, the first official Robocup was introduced in 1997 [12]. At this time there were already several different leagues, such as small-size league and expert-robot league. Thirteen years later the Standard Platform League was introduced, in order to let teams focus on programming expertise [18]. This is achieved by obliging all teams in the league to use the same robot. For the first seven years, the Sony AIBO was used. But after Sony stopped AIBO's production, Aldebaran's Nao became the robot of choice, playing it's first official match in 2008.

1.2 SPL Rules

The Standard Platform League rule-book [20] ensures matches are as standardized as possible.The field for example has to have certain dimensions, as well as a predefined center circle and penalty areas. The colors of the environment are also predefined; yellow goals, an orange ball and a green field. Lighting conditions however are not specified and can vary from match to match.
No modifications to the Nao hardware are allowed. Although teams are free to alter software as they please, all teams must implement the use of a game controller. This device acts as a referee, using UDP to transmit in which state the Nao players should find themselves. These states determine whether for example whether it is half-time, or whether a penalty has to be taken (see Figure 1).

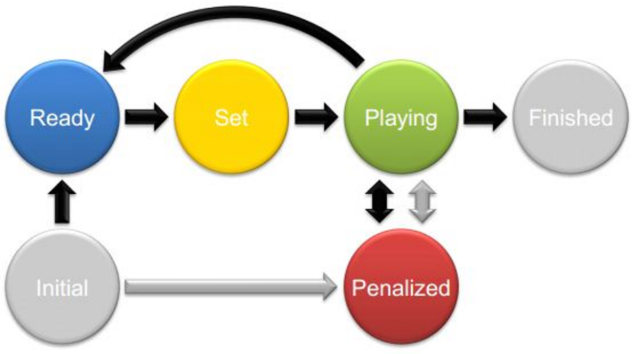


Figure 1: States during an SPL match, grey states can also be issued by pressing the chest-button.

In a normal match teams consist of four players including the goalkeeper. However, in this case the Naos were developed to participate in a two against two format. The team's goalkeeper is the only player allowed to touch the ball with its hands while in it's own penalty area, as well as being the only player of the team allowed there.

1.3 Nao

Standing at 0.57m with a weight of 4.5kg, the Nao is a humanoid robot designed with affordability, performance, modularity and open architecture as guidelines [8]. The main software running on the Nao is called NaoQi, containing many modules with basic functionality such as standing up, walking and accessing infor-

Figure 2: Aldebaran's Nao.

mation on the camera. The Nao software architecture is both cross-platform (it is possible to develop in Windows, Mac and Linux) and cross-language (software can be developed in multiple languages, such as C++ and Python). For the purposes of this project, the H25 academic version of Nao was used, updated to a NaoQi version of 1.14. It has a total of 25 degrees of freedom, and possesses two cameras located in it's head. In order to test in a safer, virtual world Nao can be tested in a special Webots for Nao simulator.

2 Strategy

At this point in time, Nao soccer matches are still very primitive in nature. Most of the play consists of finding the ball and kicking it towards the goal. Also, as mentioned earlier, teams were restricted to a size of two players. These two restrictions have led to the decision to focus on individual soccer skills of the Nao rather than team skills such as passing. Therefore the decision was made to dedicate one Nao to goalkeeping and having the other focus on field play. In order to comply to the aforementioned states, both players were designed to have a number of different behaviours:

- **Initial** The state reached after booting, in this state the Nao is only allowed to stand up.
- **Ready** In this state the robot is allowed to walk to it's kick-off position.
- **Set** The Nao waits for kick-off in this state.
- **Playing** The actual match takes place in this state.
- **Penalized** A punishment state that can for example be issued when a Nao blocks another player. The Nao is allowed no movement.
- **Finished** This state is reached at the end of each half.

The only state where behaviours differ per Nao is the playing state. In order to improve feasibility both were elected to be relatively simple: the field player looks for the ball and kicks it toward the goal on every attempt. To find the ball the Nao first utilizes the head to search for the ball. If it is not within the field of view, a turn is made to check other areas. The goalkeeper scans for the ball and positions itself in between it and the goal.

3 Software Architecture

In order to research the various aspects of Nao soccer, an architecture was first created. It is centered around a mind module. The mind constantly checks for changes in states, player type, and other match properties in the state controller. From there, it decides on the state to run. The mind also decides whether the Nao should act as the goalkeeper or a field player.
Depending on the current player type and state, the Nao executes the appropriate phase. The use of the different phases described below represent the distinct problems that can be solved independently of each other. This enhances modularity; any one of the phases can be improved without having to make any adjustment to the others.
For the field player the playing state consists of the phases *ballFound*, *ballNotFound* and *kick* (see Figure **??**). These phases are executed in succession, each phase going through it's actions before calling the next (kicking entails the ball isn't found). The goalkeeper utilizes a slightly different variation of phases; *ballFoundKeeper* and *ballNotFoundKeeper*.

4 Vision

The duo of cameras the Nao comes equipped with are both located in the head; the top camera slightly above the eyes, the bottom camera at what appears as it's mouth (see Figure 7). Since switching between these cameras can take at least 2 seconds [15], it was opted to use one exclusively. Using the top camera is problematic, since anything directly before the Nao's feet becomes undetectable even with the camera in it's bottommost position. Because this ball position is a crucial element in Nao soccer all observations from here on after are made exclusively with the bottom camera. One downside of this approach is having to rotate the head in rather large circles in order to see far away balls.
There are three crucial aspects to Nao vision for Robocup; finding the ball, the opposition's goal and the field lines. In order to overcome these problems, Nao software supports OpenCV [16].
Short for Open Source Computer Vision Library, OpenCV is a freely downloadable open source library commonly used for all vision related tasks. For all three of the aforementioned aspects, OpenCV is mainly used as a tool for Object recognition. It has Interfaces for Python, C++, C and Java. For the purposes of this project the Python interface was used.
The decision was made to focus research on ball-locating, as it was considered the most fundamental aspect of vision when attempting to realize a soccer playing Nao. Finding the ball consists of a couple of issues which need to be tackled in succession. Firstly, the ball itself has to be recognized from the camera. Secondly, the position of the ball relative to the Nao must be modelled.

4.1 Red ball tracker

One way to tackle both problems at once is to use the NaoQi *ALRedBallTracker* module, which is already provided by the NaoQi API. For the Webots simulator this approach works well, immediately providing coordinates relative to the Nao's position to walk towards. However, using it in the real world presented some problems. Because of the restrictions by the Robocup association mentioned earlier, the ball used in the game is orange and cannot be detected with the red ball tracker. Furthermore, since red ball tracker is a closed source method it cannot be improved upon. Hence, in order to locate the ball in a Robocup match a color detection method was researched. Color detection is done by specifying a range of colors, and filtering them from the image returned by the camera.

4.2 Color detection

Snapshots made by the Nao are returned using the normal RGB (Red, Green, Blue) model. As the name suggests, every pixel contains three values ranging from 0 to 255, indicating how much of each color is included. For color detection however,the HSV model is preferable [13]. HSV consists of the three values Hue, Saturation and Value. The Hue value represents the color of the object, the Saturation attribute returns the (inverse) amount of grey in the color and the value gives the intensity of a color. HSV provides some useful properties to color detection. In order to correct color only the Hue value needs to be altered. Moreover, slight variations in lighting conditions can be corrected by changing only the S and V channels.
After changing the image to an HSV model a thresholding algorithm was attempted, which involves converting an image to a binary black and white [26]. This was achieved by converting pixels with colors in several ranges to white, and leaving all others black. The results are shown in Figure 5. In order to get the coordinates of the middle of the ball, the topmost, rightmost, bottommost and leftmost pixel coordinates are retrieved.

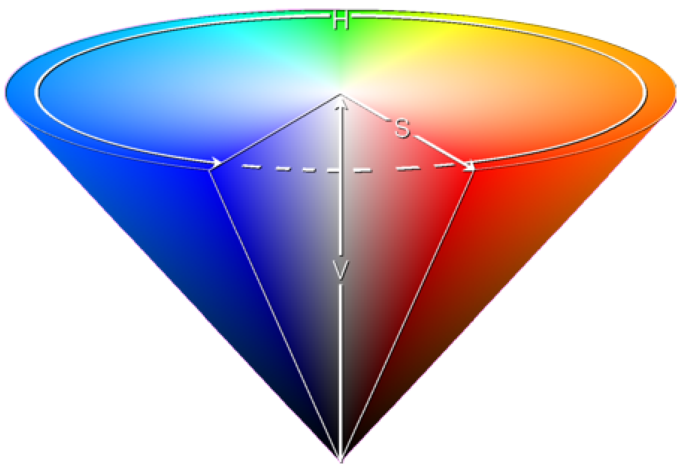


Figure 3: the HSV model.

Retrieving the coordinates in such a manner is problematic, since a single pixel of noise can greatly distort the outcome of the detection. In order to remove all noise many solutions can be utilized.

Figure 4: Noise after thresholding.

4.3 Locating the ball

In Figure 4 for example, configuring an averaging filter with a high number of pixels will insure the shoe is not detected. The use of these filters comes with some disadvantages however. When the object is small (the ball

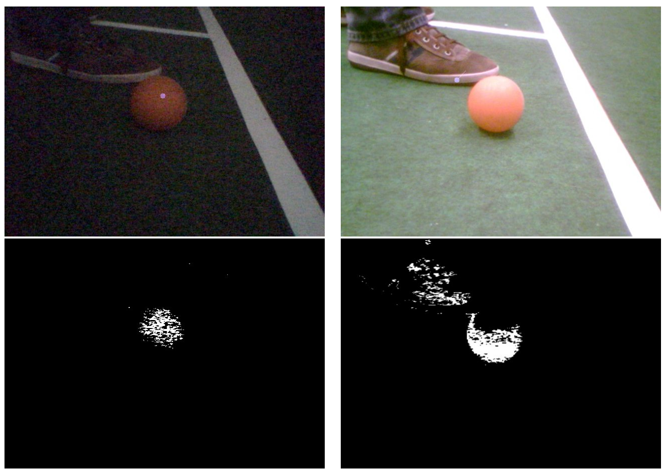


Figure 5: Considerable recognition differences due to lighting.

is far away) necessary data may be deleted making the detection worse. When calculating for a high number of pixels this is also a slow way of filtering. Therefore, in this case, it was opted to use a blurring method similar to box filtering, which takes the thresholded image and averages the pixel values over a specified region [14].
Calibrating the color detection to the surroundings is important when detecting objects. A predefined range of colors can result in detecting completely different objects depending on environment variables such as lighting conditions, color of the room and objects in the room. Especially the lighting conditions proved to be influential.

As Figure 5 shows, without alterations to the range the Nao detects different objects depending on whether the environment is well lit or not. On the left side, detection with proper lighting is displayed. It can be observed that the dark part of the foot is detected by the color detection. This is due to the color range being too dark for the environment. Because of reflection on the ball, only the darker lower part of the ball is recognized. The right side illustrated the effect of a darker environment. The detection is improved and better distinction of the colors is achieved, thereby erasing all noise.

4.4 Shape detection

In order to reduce sensitivity to noise, a third solution, namely shape detection was therefore investigated. In order to implement this, candidate circles are generated by using an alteration of a Hough transform [5]; first, the image is converted to a greyscale, then circles are detected by applying a canny edge variant. The resultant circles in the environment shall include the ball. Some of these can immediately be filtered out by having an unreasonably small or large radius.

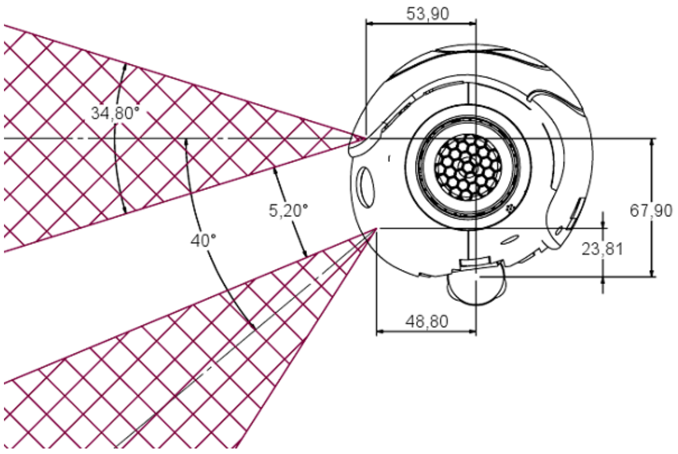


Figure 6: Nao camera angles.

4.5 Modelling the ball

After having received the position of the the ball in a snapshot, it becomes necessary to model the position of the ball relative to the Nao. In order to do this, distances to the ball were calculated by using the angles returned by the Nao camera.
The first step towards achieving this is calculating the difference between where the Nao's gaze is centered, and the position of the ball in the snapshot. Since the number of pixels in both directions between the center and the ball is known, this is a matter of converting the pixels to the number of radians the head should turn. This can be done by using the angle in degrees of the camera picture [1].

As shown in Figure 7, The built-in Nao *getPosition* [1] method returns all dimensions and angles relative to (in this case) the frame of the robot. In order to get the angle towards the ball, the current pitch must be added to the vertical distance from the middle of the snapshot. The resultant angle θ is shown in Figure 8. In combination with the height difference of the camera and the middle of the ball, the distance x to the plane the ball is in is given by:

$$\frac{camera\,height -\ ball\,radius}{\tan(camera\,pitch +\ angle\,change)}$$

Thus far all calculations assume the robot frame is in alignment with the head, which contains the camera. In order to retrieve the location coordinates of the ball relative to the robot frame the yaw of the head and the aforementioned distance x are required, see Figure 9.

$$y = \tan(xAngle) * x$$

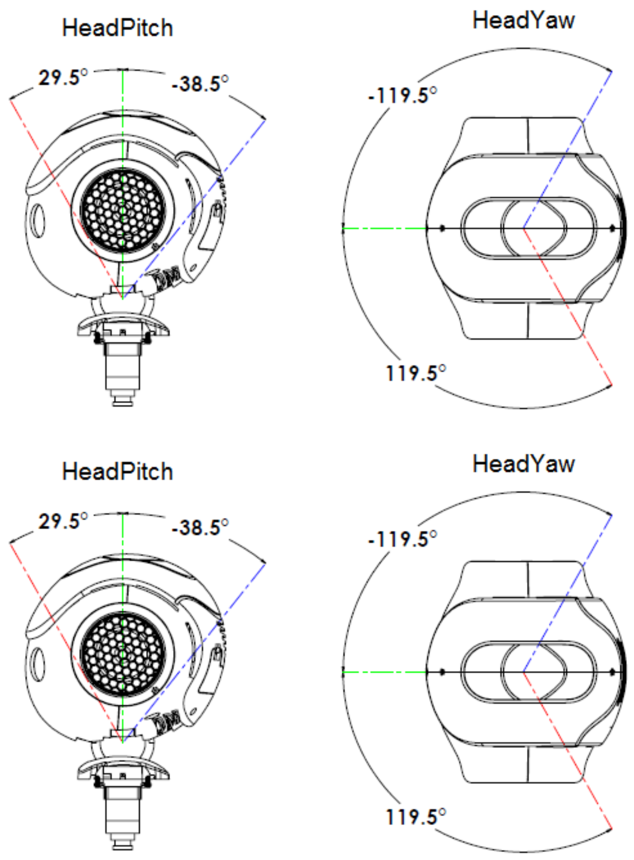


Figure 7: Pitch and yaw ranges of the Nao's head.

$$x\prime = \cos(yaw) * x + -\sin(yaw) * y$$

$$y\prime = \sin(yaw) * x + \cos(yaw) * y$$

Where $x\prime$ and $y\prime$ represent the coordinates in a coordinate system relative to the robot frame.

5 Locomotion

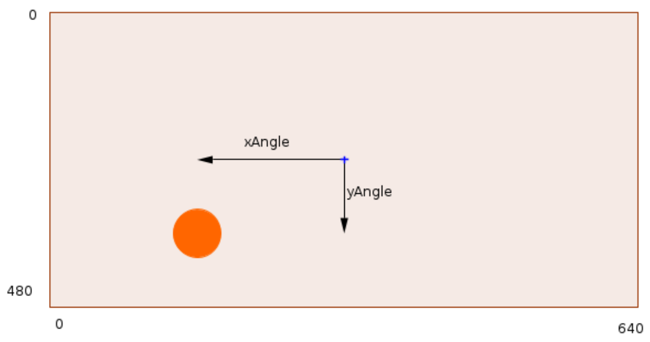
This section describes research that was done toward the end of teaching the Nao robot to perform the episodic task of walking toward a given target location. However, due to time constraints, no actual learning was carried out.

5.1 Prerequisites

In order to understand the general approach taken, some prerequisites are to be understood. This section covers them.

Reinforcement Learning

The task can be formalized as a discrete-time Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ is the set of states the system can be in, $\mathcal{A}$ is the set of actions available for influencing the system. $P(s, s', a) \in [0, 1]$ represents the probability of transitioning to state $s' \in \mathcal{S}$ at timestep $t + 1$ after taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ at timestep t. $R(s, s', a) \in \mathbb{R}$ is the expected immediate reward associated with such a transition [19]. Learning to perform the task then amounts to finding a generally stochastic policy $\pi^* : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that defines a state-conditional probability distribution over actions that maximizes long-term expected reward.



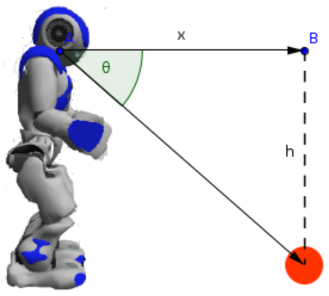


Figure 8: Calculating the distance to the ball.

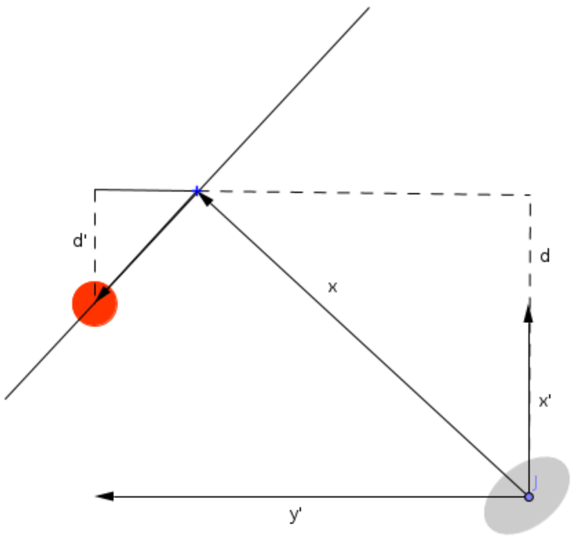


Figure 9: Calculating the distance to the ball.

If the MDP is known, then the policy can be found by solving the Bellman equations (typically using Dynamic Programming methods) [3]. In real-world problems, however, the MDP is seldom known. So-called model-free approaches exist, such as Q-learning [29] and policy gradient search [27].

Q-learning iteratively approximates a function $Q^*(s,a)$ which gives the maximum long-term expected reward that can be obtained after taking action a in state s. Given this function, the optimal policy is to always choose the action that maximizes it. This works well if the state and action spaces are discrete or low-dimensional enough that it is feasible to discretize them. Neither is true for the problem of bipedal locomotion; the state and action spaces are high-dimensional and continuous. This requires that a function approximator be used to represent candidates Q. The choice of parameterization tailors the approximator to a specific class of functions, which biases the learning algorithm that makes use of the approximations.

With policy gradient algorithms, it is the policy that is represented by a function approximator[1], and the optimal policy is directly searched for using the gradient of the long-term expected reward with respect to the function approximator parameters θ. Peters and Schaal [17] performed a survey of policy gradient methods for robotics.

In robotics, stability of the policy is highly desirable, for the safety of both the robot and the people around it. Since in Q-learning the policy is a discontinuous function of Q (computing the policy involves finding the argument of a maximum), small changes in Q can lead to large changes in the policy. Policy gradient algorithms, on the other hand, perform a gradient search in the policy space, hence the magnitude of the change in the policy can be controlled by the learning rate.

Dimensionality Reduction

Working with high-dimensional data is infamously difficult. One of the problems is that as the dimensionality of a space increases, exponentially larger datasets are needed to get equally reliable coverage of the space. Another problem is that optimization through gradient search becomes infeasible due to the increasing probability of the gradient estimate being approximately orthogonal to the true gradient. These and other related problems are collectively referred to as the Curse of Dimensionality [3].

Most high-dimensional natural data lies (approximately) on a low-dimensional manifold, which means that it is possible to represent it in fewer dimensions without (much) loss of information. Several techniques to find a transformation to such a lower-dimensional representation exist, the most well-known of which is PCA [11], which rotates the data around its mean such that the dimensions of the transformed data are linearly uncorrelated. PCA has been generalized to nonlinear transformations as Kernel PCA [25], where a kernel trick is employed to effectively nonlinearly transform the data before performing PCA. Finally, there are various manifold learning methods like Isomap [28] and LLE [24], which make assumptions about the geometry of the high-dimensional space.

Recently [10], advances have been made in the training of multi-layer feed-forward neural networks, which have the advantage that they can represent a broad class of functions [4]. Historically, such networks were trained from random initialization by backpropagation, which, being a local search, tends to get stuck in local optima or on plateaus [7]. In 2006, Hinton proposed [10] an unsupervised greedy layer-wise training procedure where each layer is trained as a Restricted Boltzmann Machine, causing the higher layers to model successively higher order correlations in the data. After the layers have been trained in this way, the weights are in a good region of parameter space (i.e., near good optima), and backpropagation can be used for supervised finetuning.

Hinton also showed [9] how deep networks can be used to reduce dimensionality by making the higher layers narrow and using the reconstruction error for finetuning. Since the purpose is not to reconstruct the high-dimensional data but to extract features from the state and action data that are useful for reinforcement learning, the learning signal that is used for finetuning is the mean squared error of the policy function approximator when fit to the data.

5.2 Approach

To gather training data, the Nao is programmed using the `ALMotion::moveTo` method to repeatedly walk to uniformly randomly distributed targets, and to record at each control timestep sensor readings, actual and desired joint positions, joint stiffnesses and the location of the target in a coordinate frame relative to the robot. The data so obtained lies in a very high-dimensional space, but the process quickly generates a large amount of data that is concentrated in a narrow region of the space, which helps to avoid the problem of data sparsity.

The data is subsequently transformed into state and action vectors for learning, which involves a dimensionality reduction step. High-dimensional state vectors $\tilde{s} \in \mathbb{R}^n$ consist of sensor readings, joint positions and target location. In order to capture motion information, the sensor readings and joint positions of several consecutive timesteps are used. The high-dimensional action vectors $\tilde{a} \in \mathbb{R}^m$ contain desired actuator positions. The

[1]As is the case with Q-learning, the choice of parameterization biases the algorithm.

Figure 10: How dimensionality reduction is applied.

$$\begin{pmatrix} \tilde{s}_1 \\ \vdots \\ \vdots \\ \tilde{s}_n \end{pmatrix} \xrightarrow{\Phi_{\mathcal{S}}} \begin{pmatrix} s_1 \\ \vdots \\ s_l \end{pmatrix} \xrightarrow{\pi} \begin{pmatrix} a_1 \\ \vdots \\ a_k \end{pmatrix} \xrightarrow{\Phi_{\mathcal{A}}^{-1}} \begin{pmatrix} \tilde{a}_1 \\ \vdots \\ \vdots \\ \tilde{a}_m \end{pmatrix}$$

immediate reward is taken to be the negative Euclidean distance to the target.

The high-dimensional state and action spaces are much too large to directly work in. However, due to correlations in the sensory data and joint positions, the intrinsic dimensionality is expected to be low. Dimensionality reduction is applied to obtain mappings $\Phi_{\mathcal{S}} : \mathbb{R}^n \to \mathbb{R}^l$ and $\Phi_{\mathcal{A}} : \mathbb{R}^m \to \mathbb{R}^k$, with $k < m$ and $l < n$. The former is used to map a high-dimensional state vector $\tilde{s}$ to a low-dimensional representation s. The latter is used inversely, to translate the low-dimensional representation a of an action back into the high-dimensional space of desired joint angles.

This scheme is illustrated in Figure 10.

In order to find suitable mappings $\Phi_{\mathcal{S}}$ and $\Phi_{\mathcal{A}}$, Deep Auto-Encoders are pretrained to represent the data distribution and subsequently finetuned for fitting an initial policy π_0 from the low-dimensional states s to the low-dimensional actions a. The initial policy found in this way corresponds to the stock motion controller implemented by the `ALMotion::moveTo`. It may be improved by using the training data for reinforcement learning. This would result in an improved policy which would then be used to gather more training data, to which the Deep Auto-Encoders would be trained again. Repetition of this process is expected to gradually improve the policy. Due to time constraints, however, this experiment is left for future research.

5.3 Expected difficulties

One of the difficulties with this process is that the training of deep belief networks is a long and tedious process. It cannot be fully automated as it requires constant attention to make sure the network is not overfitting to the training data, the weights and biases are not diverging, and that hyperparameters such as the learning rate are set correctly.

Another problem with deep belief networks is that one has little control over the kinds of features learned. Since the pretraining procedure is unsupervised, there is no incentive for the network to learn features that are useful for the task at hand. Moreover, the supervised finetuning comes too late in the process to affect the set of features chosen [6]. This may be remedied by introducing the supervised criterion earlier, however, when and how to do so has to our knowledge not been investigated. A better solution may be to simply learn a large set of features, and then select a subset of them to maximize performance on the supervised task.

Due to the inverse dimensionality reduction of the actions selected by the policy, the stability properties of policy gradient search no longer hold. While small changes in the policy still result in small changes in the low-dimensional actions a, they do not necessarily result in small changes in the high-dimensional actions $\tilde{a}$. This is because the mapping $\Phi_{\mathcal{A}}$ that relates them can be a highly complex function.

Among the information that is gathered in the data gathering phase is the target location from the perspective of the robot. The `ALMotion::getRobotPosition` is used to compute this location, however, it is unclear how this is implemented and whether it can be used when movement is not done through the `ALMotion` API but through direct control of the joints. If it cannot be used, then computing a suitable reward function will be a complex task involving much domain knowledge.

6 Goalkeeping

Actively stopping shots is usually achieved by one of two actions [23]. If the intersection of the ball with the goal-line is close by, the goalkeeper assumes a crouching stance in order to cover as much space as possible. When the intersection is far away, the goalkeeper dives to save the ball. Since diving may damage the Nao however, a different approach must be taken.
Nao goalkeeping is still very primitive and involves moving as little as possible. Current strategy usually involves only leaving the goal when the ball has come to a stop close-by [23]. As with shooting, the Nao goalie is often committed to an action once it has initiated it [21].
In order

7 Kicking

Currently the most common approaches of implementing the shooting motion are keyframe-based techniques [22]. The major disadvantage of this approach is its inflexibility. During the execution of a kicking motion the Nao is committed to the action even though the position of the ball might change. Therefore a different, more dynamic approach was researched. Most SPL-teams use a large array of static kicks for different situations that may come up during a match. This inevitably leads to a rigid shooting behaviour. A better way of dealing with different scenarios may be to create a small set of parameterized kicks [2]. A kick engine selects the appropriate parameters and then executes the sequence of actions for a given kick. In order to implement a parameterized

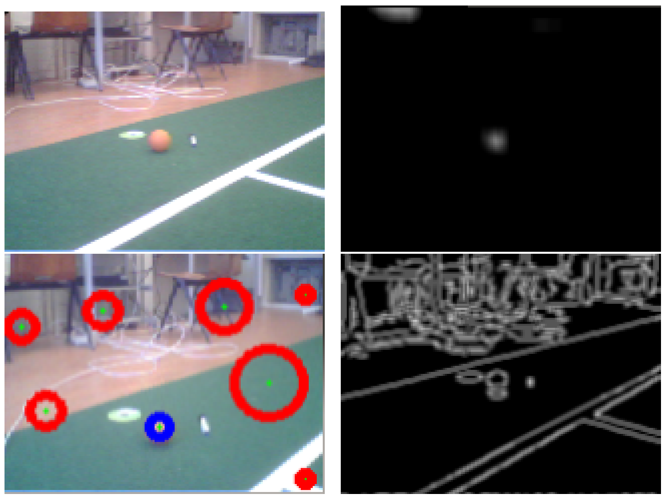


Figure 11: Noise is ignored by using circle detection.

kick, a keyframe-based kick was investigated, shooting the ball at an angle. While most available kicks are stable at every frame, it was postulated that fixing balance in subsequent frames might result in a faster kick.

8 Integration of functionality

In order to accurately detect the ball, both color and shape detection were utilised in series. The centres of all candidate circles are compared to the coordinates returned by the color detection. In case the color detection coordinates lie within the radius of the closest circle, a ball is returned as found. As Figure 11 shows, this results in a very accurate model of the ball. To subsequently walk towards the ball, since no learning has yet taken place, the standard walk present in the NaoQi is used. This is done by giving a set walking speed towards the target, which is stopped as soon as the ball is lost. Finally, the Nao uses a simple goal detection based on the color detection described above. this allows the Nao to rotate around the ball until it stands directly in front of the goal. The kick executed at this moment is designed to be at an angle so as to avoid hitting the opponent's goalkeeper.

9 Conclusion

Creating a fully functional Nao soccer team is a difficult and time consuming endeavour. At the onset of the project, the initial planning was somewhat too ambitious. Due to time constraints, several aspects of Nao soccer have been left unimplemented. The most important of these for a competitive SPL-Team is the implementation of the state-controller in order to receive instructions. In order to play soccer intelligently, Naos should also be able to locate themselves on the field by recognizing the field lines. Parameterizing a kick to shoot the ball in different angles depending on where the goal is would also be a great improvement. In order to stop far away balls without jeopardizing the vulnerable Nao fingers, an alternative for diving should be researched. Further research could also focus on playing as a team. For example, assigning different roles, such as defender or attacker to the field players.

References

[1] Aldebaran (2013). Nao software 1.14 documentation.

[2] Barrett, S., Genter, K., Hester, T., Quinlan, M., and Stone, P. (2010). Controlled kicking under uncertainty. *The Fifth Workshop on Humanoid Soccer Robots at Humanoids 2010.*

[3] Bellman, Richard (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.

[4] Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, Vol. 2, No. 1, pp. 13–20.

[5] Duda, R.O. and Hart, P.E. (1972). Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, Vol. 15, No. 1, pp. 11–15.

[6] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, Vol. 11, pp. 625–660.

[7] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics.*

[8] Gouaillier, D., Hugel, V., Blazevic, P., Kilner, C., Monceaux, J., Lafourcade, P., Marnier, B., Serre, J., and Maisonnier, B. (2008). The nao humanoid: a combination of performance and affordability. *CoRR, vol. abs/0807.3223.*

[9] Hinton, G.E. and Salakhutdinov, R.R. (2006). Reducing the dimensionality of data with neural networks. *Science*, Vol. 313, No. 5786, pp. 504–507.

[10] Hinton, G.E., Osindero, S., and Teh, Y.W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, Vol. 18, No. 7, pp. 1527–1554.

[11] Jolliffe, I. (2005). *Principal component analysis*. Wiley Online Library.

[12] Kitano, H. (1998). *RoboCup-97: robot soccer world cup I*, Vol. 1395. Springer.

[13] Manjunath, BS, Ohm, J.R., Vasudevan, V.V., and Yamada, A. (2001). Color and texture descriptors. *IEEE Transactions on circuits and systems for video technology*, Vol. 11, No. 6, p. 703.

[14] McDonnell, MJ (1981). Box-filtering techniques. *Computer Graphics and Image Processing*, Vol. 17, No. 1, pp. 65–70.

[15] Olivares, P.C., Benavent, P.M., Herrero-Pérez, D., Noguera, J.F.B., and Martínez-Barberá, H.Robust and efficient embedded vision for nao in robocup.

[16] (2012). Opencv dev zone.

[17] Peters, J. and Schaal, S. (2006). Policy gradient methods for robotics. *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 2219–2225, IEEE.

[18] Pot, E., Monceaux, J., Gelin, R., and Maisonnier, B. (2009). Choregraphe: a graphical tool for humanoid robot programming. *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*, pp. 46–51, IEEE.

[19] Puterman, Martin L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

[20] Robocup Technical Committee (2012). Robocup standard platform league (nao) rule book.

[21] Rodriguez, V., Rodriguez, FJ, and Matellán, V. (2011). Localization issues in the design of a humanoid goalkeeper for the robocup spl using bica. *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pp. 1152–1157, IEEE.

[22] Röfer, Thomas, Laue, Tim, Bösche, Oliver, Sieverdingbeck, Ingo, Wiedemeyer, Thiemo, and Worch, Jan-Hendrik (2009). B-Human Team Description for RoboCup 2009. *RoboCup 2009: Robot Soccer World Cup XII Preproceedings* (eds. Jacky Baltes, Michail G. Lagoudakis, Tadashi Naruse, and Saeed Shiry), RoboCup Federation.

[23] Röfer, Thomas, Laue, Tim, Müller, Judith, Graf, Colin, Böckmann, Arne, and Münder, Thomas (2012). B-Human Team Description for RoboCup 2012. *RoboCup 2012: Robot Soccer World Cup XV Preproceedings* (eds. Xiaoping Chen, Peter Stone, Luis Enrique Sucar, and Tijn Van der Zant), RoboCup Federation.

[24] Roweis, S.T. and Saul, L.K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, Vol. 290, No. 5500, pp. 2323–2326.

[25] Schölkopf, B., Smola, A., and Müller, K.R. (1997). Kernel principal component analysis. *Artificial Neural NetworksICANN'97*, pp. 583–588.

[26] Sezgin, M. and Sankur, B. (2004). Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, Vol. 13, No. 1, pp. 146–165.

[27] Sutton, R.S., McAllester, D., Singh, S., Mansour, Y., et al. (2000). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, Vol. 12, No. 22.

[28] Tenenbaum, J.B., De Silva, V., and Langford, J.C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, Vol. 290, No. 5500, pp. 2319–2323.

[29] Watkins, Christopher J. C. H. and Dayan, Peter (1992). Q-learning. *Machine Learning*, Vol. 8, No. 3-4, pp. 279–292.