Predicting away robot control latency

Alexander Gloye,[1] Mark Simon,[1] Anna Egorova,[1]
Fabian Wiesel,[1] Oliver Tenchio,[1] Michael Schreiber,[1]
Sven Behnke,[2] and Raúl Rojas[1]
Technical Report B-08-03

[1] Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany
[2] International Computer Science Institute, Berkeley, CA, 94704, USA
http://www.fu-fighters.de

Abstract. This paper describes a method to reduce the effects of the system immanent delay when tracking and controlling fast moving robots using a fixed video camera as sensor. The robots are driven by a computer with access to the video signal. The paper explains how we cope with system latency by predicting the movement of our robots using linear models and neural networks. We use past positions and orientations of the robot for the prediction, as well as the most recent commands sent. The setting used for our experiments is the same used in the small-size league of the RoboCup competition. We have successfully field-tested our predictors at several RoboCup events with our *FU-Fighters* team. Our results show that path prediction can significantly improve speed and accuracy of robotic play.

1 Introduction

The time elapsed between deciding to take an action and perceiving its consequences in a given environment is called the control latency or delay. All physical feedback control loops exhibit a certain delay, depending on the system inertia, on the input and output speed and, of course, on the speed at which the system processes information.

In the RoboCup small size league we use data from two video cameras fixed above the field to determine the positions of all robots on the field. The robots can be thought as circular, with a maximum diameter of 18 cm. The field is 2.4 by 2.8 meters long. An off-the-field computer sends commands to the robots, which are very fast, sometimes reaching peak speeds of several meters per second.

In order to react as precisely as possible to a given situation and to calculate the behavior of the robots we need to know their exact positions at every moment (i.e. at every video frame). However, due to the system delay, the system can actually react to commands only after some time. When moving faster the delay becomes very important since the error between the real position of the robot and the position used for control may grow up to 20 cm. The last frame captured by the cameras reflects a *past* position of the robot, and we need to send commands so that they are consumed by the robot in a *future* frame. Predicting the present position of the robot is therefore not enough: we need to predict its future position, at the time when the new commands will arrive and will be consumed.

In order to correct the immanent error associated with the system's latency we apply neural networks and linear models to process the positions, the orientations, and the action commands sent to the robots during the last six frames. The models predict the positions of the robots four frames after the last available data (that is, four frames from the past into the future). These predictions are used as a basis for control. We use real recorded pre-processed data of moving robots to train the models and teach the system to predict their positions four frames in advance.

The concept of motor prediction was first introduced by Helmholtz when trying to understand how humans localize visual objects (see [6]). His suggestion was that the brain predicted the gaze position of the eye, rather than sensing it. In his model, the predictions are based on a copy of the motor commands acting on the eye muscles. In effect, the gaze position of the eye is made available before sensory signals become available.

The paper is organized as follows. The next section gives a brief description of our system architecture. Then we explain how the delay is measured and we present some other approaches to eliminate latency. Section 5 describes architecture and training of the linear models and neural network used as predictors. Finally, we present some experimental results and some plans about future development.

2 System Architecture

The small size league is the fastest physical robot league relative to field size in the RoboCup competition. Top robot speeds exceed 2m/s and acceleration is limited only by the traction of the wheels, so a robot may cross the entire field in much less than two seconds. Action commands are sent via a wireless link to the robots which contain only minimal local intelligence on a microcontroller. Thus, the robot design in this league focuses on speed, maneuravability, and ball handling.

Our control system is illustrated in Figure 1. The only physical sensors we use for behavior control are two S-Video cameras.[3] The cameras capture a view of the field from above and produce two output video streams, which are forwarded to the central PC. Images are captured by frame grabbers and passed to the vision module.

The global computer vision module processes the images, finds and tracks the robots and the ball and produces as output the positions and orientations of the robots, as well as the position of the ball. The vision system is described in detail in [3].

[3] There are various other sensors on the robots and in the system, but they are not used for behavior control. For example, the shaft encoders on the robots are only used for motion control.

Based on the information collected, the behavior control module then produces the commands for the robots: desired rotational velocity, driving speed and direction, as well as the possible activation of the kicking device. The central PC then sends these commands via a wireless communication link to the robots. The hierarchical reactive behavior control system of the FU-Fighters team is described in [2].

Each robot contains a microcontroller for omnidirectional motion control. It receives the commands and controls the movement of the robot using PID controllers (see [4]). Feedback about the speed of the wheels is provided by the impulse generators in each motor.

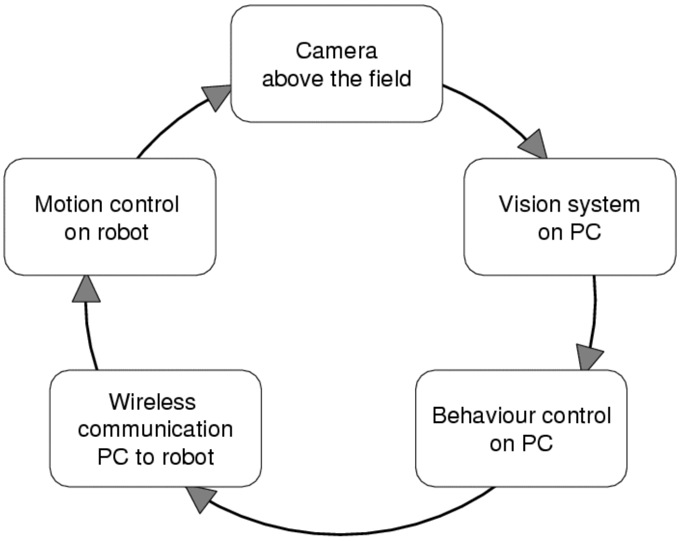

Fig. 1. The feedback control system. All stages have a different delay. But only the overall delay is essential for the prediction.

Unfortunately, the whole system accumulates a significant delay on the way between capturing the environment and reacting to commands. In our system, the feedback (the result) of an action is typically perceived between 100 ms and 150 ms after the time when the last frame was captured (about four frames delay). This causes problems when robots move fast, producing overshooting or oscillatory movement. One possible solution is to drive slowly, but this is often not desirable.

Our approach to solve the latency problem is to predict the position and the orientation of the robot a few frames forward into the future and to use these predicted values for the behavior control rather than the values from the vision directly. We feed the last captured robot positions and orientations (relative to the current robot position and orientation) and action commands to a feed-forward neural network or a linear model that is trained to predict the robot position and orientation for a point in time 132 ms later. This approximately cancels the effects of the delay and allows fast and exact movement control.

A simple example can illustrate what can be gained from predicting the positions of the robots in future frames. Fig. 2 (the gray path) shows a robot driving without prediction. The robot drives first to the target position with a given speed, but because of the delay it does not stop right on it; the robot drives further against the wall and then, with low speed, again to the target position. This is shown also in the curve from Fig. 3 (the upper one), where the distance function from the target remains constant (the robot drives around the target) for a certain time.

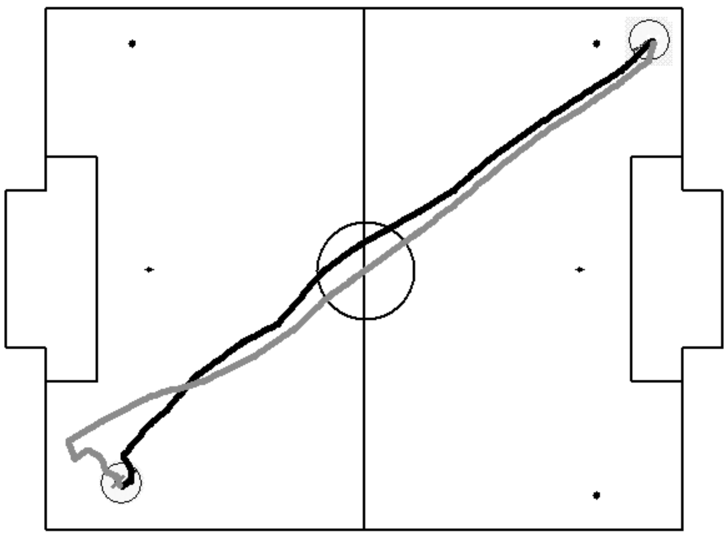

Fig. 2. Driving a robot to a given target with (black) and without (gray) using a prediction. The lines show the robot's driving path from start to target.

In contrast, in Fig. 2 (the black path) the behavior of the same robot is shown when prediction is used. The robot drives much more precisely to the target position, without overshooting. The corresponding driving functions are shown in Fig. 3 (lower curve).

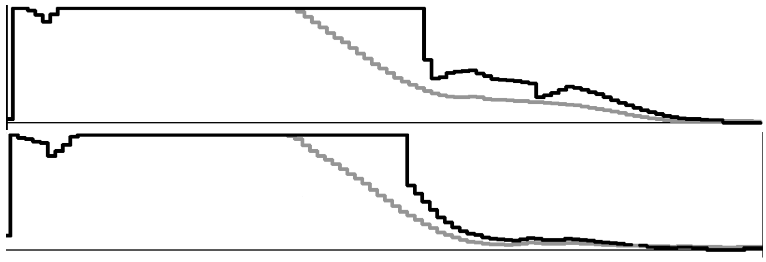

Fig. 3. The robot's speed functions (black) and the target distance functions (gray) are shown for a robot driving to a given target without (upper) and with (lower) prediction.

3 Delay: Measurement, Consequences, and Approaches

As with all control systems, there is some delay between making an action decision and perceiving the consequences of that action in the environment. All stages of the loop contribute to the control delay that is also known as dead time.

The delay sources and their impact are the following:

- Camera integration time. The CCD chip in each camera must integrate the image. The integration time is variable, but let us assume that it is equal to 10 ms.
- Transmission to the framegrabber. Two half-images are transmitted to the framegrabber at 30 fps, that is, it takes 33 ms until the two half-images have been captured.
- Transmission to main memory. The framegrabber transmits the data through the PCI bus of a PC to the main memory. At 640 by 480 color pixels, the picture size is 1.2 MB and the PCI bus sustained data rates are about 60 MB/s. We are using two cameras to capture the field, so sending the information to memory takes about 2.4/60 s = 40 ms.
- Computer vision. This is very fast in our system, it takes around 1 ms. Synchronizing the two camera inputs takes 3 additional milliseconds.
- Behavior control. A decision is taken in about 1 ms (plus 5 ms for displaying the data on the screen).
- Wireless communication. The commands are sent using the serial interface and a wireless module. The latency of both is about 17 ms, 7 ms due to buffering in the serial FIFO and another 10 ms for sending data to the last one in a set of robots.
- Command interpretation. The robot has to evaluate the commands, which are interpreted every 8 ms on the robot.
- Robot reaction. Finally the robot has to react to the commands.

Adding these sources of latency we get 10+33+40+4+6+17+8 = 118 ms. The robot reaction to commands is not contained in this value, but can be significant. A robot does not react immediately after a command has been sent and interpreted. The robot has a system inertia, which is very difficult to model analytically, since it depends on many variables. The system inertia adds up to the hardware latency.

Measuring the delay, in order to confirm the above estimate, can seem difficult, since it would require to synchronize clocks in the robot and in the controlling computer. However, we found a simple solution: In order to estimate the system delay, we use a special behavior. We let the robot drive on a straight line with a sinusoidal speed function. This means, the robot moves back and forth with maximum speed in the middle of the path, changing to zero towards both turning points. We then measure the time between sending the command to change the direction of motion and perceiving a direction change of the robot's movement. We obtain two curves displaced in time: one represents the velocities sent to the robot, the second the response of the robot. The shift between both curves is about 120 ms (see Fig. 4).

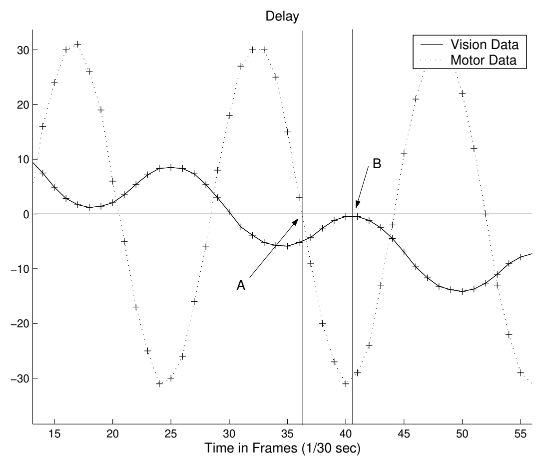

Fig. 4. The broken line shows the oscillations in motor speed, the continuous line shows the oscillations in one direction, as captured by the vision system. The motor changes rotation direction at A, the vision detects a change in movement direction at B. There are about four frames between A and B.

Control researchers have made many attempts to overcome the effects of delays. One classical approach is known as Smith Predictor [8]. It uses a forward-model of the plant, the controlled object, without delays to predict the consequences of actions. These predictions are used in an inner control loop to generate sequences of actions that steer the plant towards a target state. Since this cannot account for disturbances, the plant predictions are delayed by the estimated dead time and compared to the sensed plant state. The deviations reflect disturbances that are fed back into the controller via an outer loop. The fast internal loop is functionally equivalent to an inverse-dynamic model that controls a plant without feedback. The Smith Predictor can greatly improve control performance if the plant model is correct and the delay time matches the dead time. It has been suggested that the cerebellum operates as a Smith Predictor to cancel the significant feedback delays in the human sensory system [7]. However, if the delay exceeds the dead time or the process model is inaccurate, the Smith Predictor can become unstable.

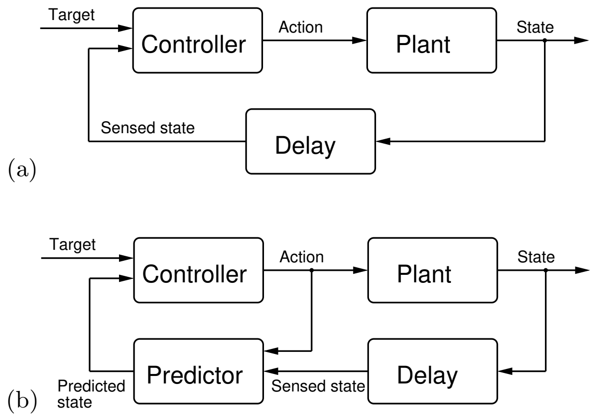

Fig. 5. Control with dead time: (a) control is difficult if the consequences of the controller actions are sensed with significant delay; (b) a predictor that is trained to output the current state of the plant can reduce the delay of the sensed signal and simplify control.

Ideally, one could cancel the effects of the dead time by inserting a negative delay of matching size into the feedback loop. This situation is illustrated in Fig. 5(b), where a predictor module approximates a negative delay. It has access to the delayed plant state as well as to the non-delayed controller actions and is trained to output the current plant state. The predictor contains a forward model of the plant and provides instantaneous feedback about the consequences of action commands to the controller. If the behavior of the plant is predictable, this strategy can simplify controller design and improve control performance.

A simple approach to implement the predictor would be to use a Kalman filter. This method is very effective to handle linear effects, for instance the motion of a free rolling ball [5]. It is however inappropriate for plants that contain significant non-linear effects, e.g. caused by the slippage of the robot wheels or by the behavior of its motion controller. For this reason, some teams use an Extended Kalman-Bucy Filter [9] to predict non-linear systems. But this approach requires a good model of the plant. We propose to use linear regression models and neural networks as predictors for the robot motion, because this approach does not require an explicit model and can easily use robot commands as additional input for the prediction. This allows predicting future movement changes before any of them could be detected.

4 Linear Models

4.1 Velocity and acceleration model

Tracking mobile robots with cameras fixed above the field corresponds to the problem of tracking a moving particle in two-dimensional space. We expect the trajectories of the particles to be smooth, since the robots have a considerable mass (one to two kilograms) and cannot change positions instantly. We will denote the coordinates of the robot's path by the time series

$$(x_1, y_1), (x_2, y_2), \ldots, (x_\ell, y_\ell),$$

where each point corresponds to one frame in the video stream. The time elapsed between successive frames is constant and can be set to $\Delta t = 1$, where the unit of measurement is 1/30 of a second.

A straightforward approach for the prediction of the robot's path, is to compute the velocity and acceleration of the robot with the help of the last three frames. The velocity v_x (x-direction) at the point x_t, y_t is approximated by

$$v_x(t) = x_t - x_{t-1}.$$

whereas the x-acceleration a_x can be approximated by

$$a_x(t) = v_x(t) - v_x(t-1) = (x_t - x_{t-1}) - (x_{t-1} - x_{t-2}) = x_t - 2x_{t-1} + x_{t-2}.$$

Similar approximations hold for the coordinate y. Now, x_{t+1} can be predicted as

$$x_{t+1} = x_t + v_x(t) + a_x(t) = x(t) + (x_t - x_{t-1}) + (x_t - 2x_{t-1} + x_{t-2}).$$

and symplifying this we obtain

$$x_{t+1} = 3x(t) - 3x_{t-1} + x_{t-2}.$$

This is a linear model for x_t based on the last three points. Most Kalman filters used for tracking moving particles are based on an empirical model of this form. The quality of the prediction is highly dependent on the quality of the estimated values of the velocity and acceleration.

It is then easy to see that we can obtain a better prediction using a general linear model in which x_{t+1} and y_{t+1} are of the form

$$x_{t+1} = a_0 x_t + a_1 x_{t-1} + \cdots a_m x_{t-m}$$

and

$$y_{t+1} = b_0 x_t + b_1 y_{t-1} + \cdots b_m y_{t-m}.$$

If there are correlations between the velocities in the x and y directions, we can even postulate a model of the form

$$x_{t+1} = a_0 x_t + a_1 x_{t-1} + \cdots a_m x_{t-m} + a'_0 y_t + a'_1 y_{t-1} + \cdots a'_m y_{t-m}$$

and

$$y_{t+1} = b'_0 x_t + b'_1 x_{t-1} + \cdots b'_m x_{t-m} + b_0 y_t + b_1 y_{t-1} + \cdots b_m y_{t-m}.$$

This may seem strange, but in the case of robots which have three wheels, the velocities in the x and y directions are correlated. In the model above, we are using the last $m+1$ positions of the robot to compute the position in the next frame. Therefore, we are using a moving window over the data of size $m+1$.

The model above is more general than the velocity and acceleration model, which it contains as a special case. Since it is a linear model, it can be solved with linear algebraic methods.

For a given time series of points in a path, let X denote the matrix

$$X = \begin{pmatrix} x_0 & x_1 & \ldots & x_m & y_0 & y_1 & \ldots & y_m \\ x_1 & x_2 & \ldots & x_{m+1} & y_1 & y_2 & \ldots & y_{m+1} \\ \vdots & & & & & & & \\ x_i & x_{i+1} & \ldots & x_{i+m} & y_i & y_{i+1} & \ldots & y_m \\ \vdots & & & & & & & \end{pmatrix}$$

of size $(n-m) \times (2m+2)$, where n is the number of data points we have in a trajectory that has been captured previously. Let x denote the vector $(x_{m+1}, x_{m+2}, \ldots, x_{i+m+1}, \ldots)^T$, which are the positions to be predicted for every prediction window. What we are looking for is a vector α such that

$$X\alpha = x$$

In the general case, there is no solution to this linear system of equations, but we can look for the vector α that minimizes the total quadratic error

$$E = (X\alpha - x)^T (X\alpha - x).$$

It is well-known that the general solution to the type of linear problem specified above is

$$\alpha = (X^T X) X^T x.$$

The same for the second coordinate

$$\beta = (X^T X) X^T y,$$

where y is the vector of targeted y-coordinates for the predictor.

5 Predictor Design

Since we have no precise physical model of the robot, we train linear models and a three layer feed-forward network to predict the robot motion.

The input data includes the vision data from the last six frames for the orientation and position of the robot, as well as the last few commands sent to it. Some preprocessing is needed in order to obtain good training results. Since we would like to simplify the problem as much as possible, we assume translational and rotational invariance. This means that the robot's reaction to motion commands does not depend on its position on the field. Hence, we can encode its perceived state history in a robot-centered coordinate system.

The position data consists of six vectors – the difference vectors between the current frame and the other six frames in the past, given as (x, y)-coordinates. The orientation data consists of six angles, given as difference of the robot's orientation between the current frame and the other six ones in the past. They are specified as their sine and cosine. This is important because of the required continuity and smoothness of the data. If we would encode the angle with a single number, a discontinuity between $-\pi$ and π would complicate training. The action commands are given in a robot-centered coordinate system. They consist of the driving direction and speed as well as the rotational velocity. The driving direction and velocity are given as one vector with (x, y)-coordinates, normalized by the velocity. They are given in the robot's local coordinate system.

Preprocessing produces seven float values per frame, which leads to a total of $7 * 6 = 42$ input values for the models.

The target vector we are using for training and testing the network consists of two components: the difference vector between the current position and the position four frames forward into the future and the difference between the current orientation and the orientation four frames ahead. They are given in the same format as the input data, without the robot commands.

The linear model consists therefore of 42 constants that have to be computed. We train a different linear model for the x coordinate and for the y coordinate. Remember that the data is given in the robot's reference frame – there is an asymmetry between the x and the y direction. In one direction the robot uses two wheels, in the other, three. The robot dynamics is different in each direction and therefore a different linear model is necessary.

Denoting the past positions of the robots in the robot reference frame by $(x_{-6}, y_{-6}), \ldots, (x_{-1}, y_{-1})$, the last orientations by $(cos_{-6}, sin_{-6}), \ldots, (cos_{-1}, sin_{-1})$, and the last commands by $(vx_{-6}, vy_{-6}, \theta_{-6}), \ldots, (vx_{-1}, vy_{-1}, \theta_{-1})$, the two linear models that we train have the form

$$x_4 = v^T \alpha \qquad\qquad y_4 = v^T \beta$$

where v is the input vector with the 42 parameters specified above and α and β are 42-dimensional vectors of linear weights.

On the other side, the neural network consists of 42 input units, 10 hidden units, and 4 output units. The hidden units have a sigmoidal transfer function while the transfer function of the output units is linear. We train the network with recorded data using the standard backpropagation algorithm [1]. Fig. 6 shows the general architecture of the network used.

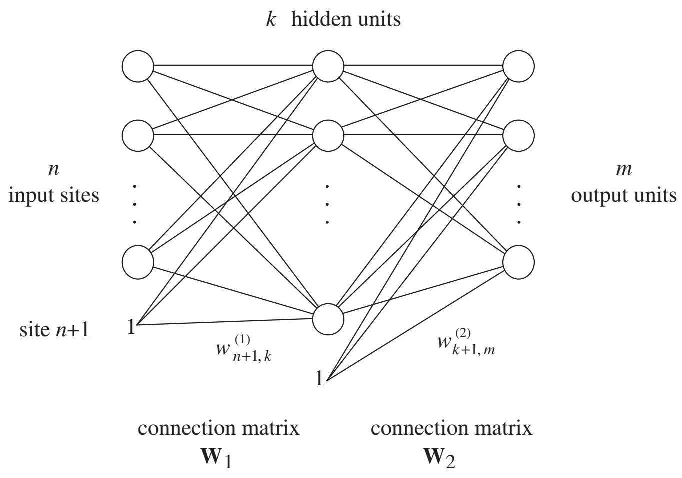

Fig. 6. Architecture of the multilayer neural network used for this report

A great advantage of the linear models and the neural network is that they can be easily trained again if something in the system changes, for example if a PID controller in the robot's electronics is modified. In this case, new data must be recorded. However, if the delay itself changes we only have to adjust the selection of the target data (see below) before retraining.

5.1 Data Collection and Constraints

Data for training the network is generated by moving a robot along the field. This can be done by manual control using a joystick or a mouse pointer, or by behaviors developed for this purpose. It is convenient to have a simple behavior that explores the space of possible directions and speeds, in order to generate enough training data.

To cover all regions of the input space, the robot must face all situations that could happen during game play. They include changing speed in a wide range of situations, rotating and stopping rapidly, and standing still. We also must make sure that the robot drives without collisions, e.g. by avoiding walls. This is necessary because the models have no information about obstacles and hence cannot be trained to handle them. If we would include such cases in the training set, the predictors would be confused by conflicting targets for the same input. For example driving freely along the field or driving against a wall produce the same input data with completely different target data.

We could solve this problem by mapping back to a global coordinate system and including additional input features, e.g. a sensor for obstacles, and thus handle also this situation, but this would complicate the predictor design and would require more training data to estimate additional parameters.

6 Prediction results

We have extensively tested both neural and linear predictors for position and orientation of the robots since its first integration into the FU-Fighters' system. The predictors perform very well and we have nearly eliminated the influence of the delay on the system.

6.1 Position Prediction

As one can see from Fig. 7 the two curves are nearly the same, just slightly shifted in the time. The black curve describes the prediction of the position of the robot, calculated by the neural network. The gray one displays the position of the same robot seen by the vision at that time and transferred to the system with delay. The middle line is the center of the field. The neural network predicts correctly the position of the robot seen by the vision 4 frames later. The vision displays, however, a smoother curve than the predictor. This results from the fast change of movement direction and decreases with proper and extensive training of the neural network.

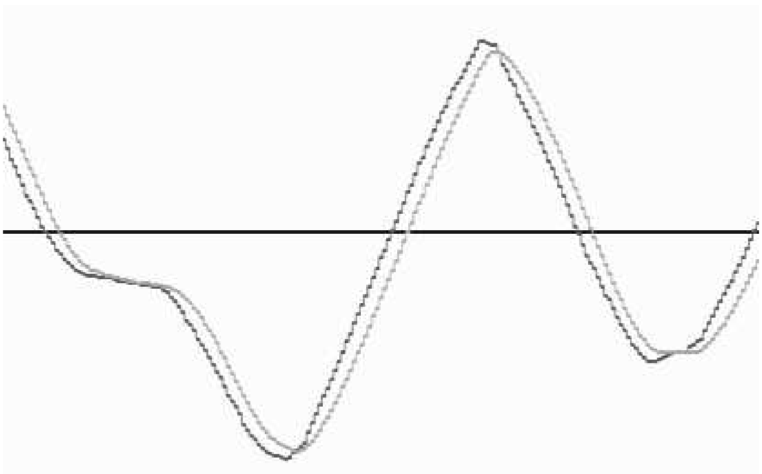

Fig. 7. Robot position from the vision (gray curve) and the predicted position from a neural network (black curve).

6.2 Orientation Prediction

Fig. 8 represents results from the orientation prediction via the neural network. The black curve is again the calculated orientation of the robot 4 frames forward in the future and the gray curve the actual orientation given by the vision. As one can see, the actual orientation curve is much more smoother than the predicted one. This is an effect from the very fast movement of the robot around its axis and the very fast change of direction and is much greater than by the position prediction.

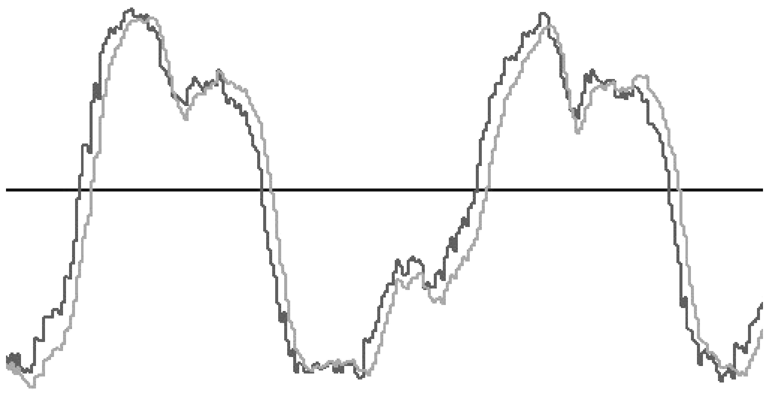

Fig. 8. Robot orientation from the vision (gray curve) and the predicted orientation (black curve).

Figure 9 shows the result of training the linear model and predicting four frames after the last captured frame. The thin lines extending the path are the result of predicting the next four frames at each point. The orientation of the robot is shown with a small line segment, and the desired velocity vector for the robot is shown with another small segment. At sharp curves, the desired velocity vector is almost perpendicular to the trajectory. As can be seen, the predictions are very good for the linear model.

Figure 10 shows the same information, but magnified, in order to make it more visible.

To demonstrate the effect of the prediction on robot behavior, we have tested one particular behavior of the robot – drive in a loop around the free kick points – with linear and neural network prediction. The histograms in Fig. 11 compare three kinds of predictors: a neural network, a linear regression model, and a physical model (which approximates velocity). As can be seen, for the robot orientation, the neural networks has much more samples with smaller errors than a simple linear prediction with two frames (which essentially computes

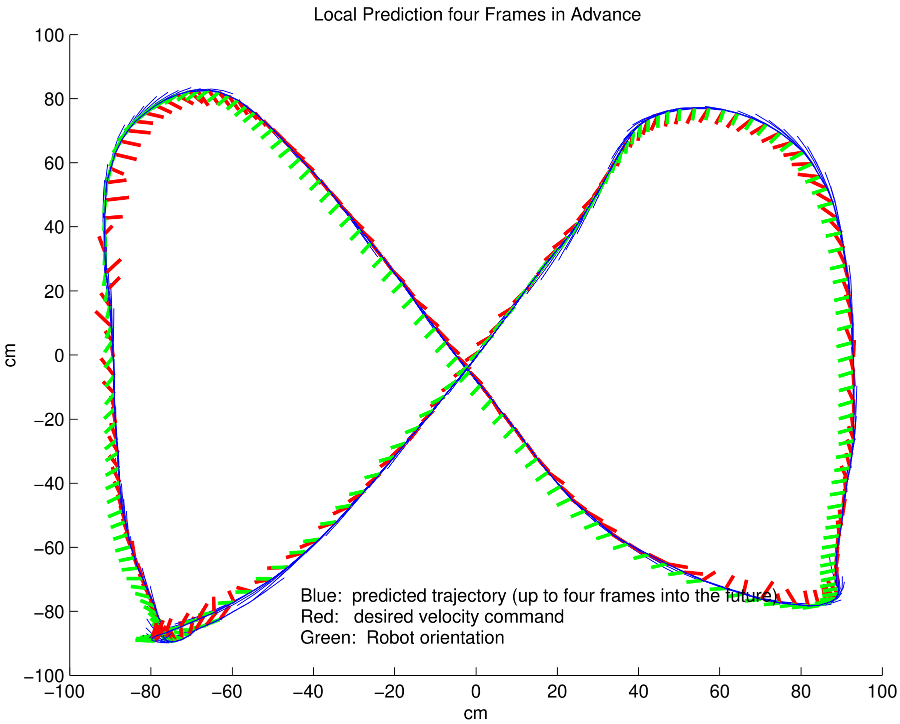

Fig. 9. A trajectory showing the predictions for four frames (thin lines) after each data point. The orientation of the robot is shown in green, and the desired velocity (the command sent) in red.

the current velocity), and is also better than the linear regression. The linear regression is slightly better for the position prediction than the neural network, and much better than the simple velocity model.

The average position error for the simple linear prediction is 3.48 cm, it is 2.65 cm for the neural network, and 2.13 cm for the linear regression. The average orientation error is 0.17 rad (9.76 degrees) for the simple linear prediction, 0.113 rad (6.47 degrees) for the linear regression and 0.08 rad (4.58 degrees) for the neural network. When independent predictors are combined to provide a weighted estimate, the prediction errors can partially cancel in some situations. The best predictor is therefore an average of the linear regressor with the neural network. In our system we can pick the prediction method from a menu for each robot type.

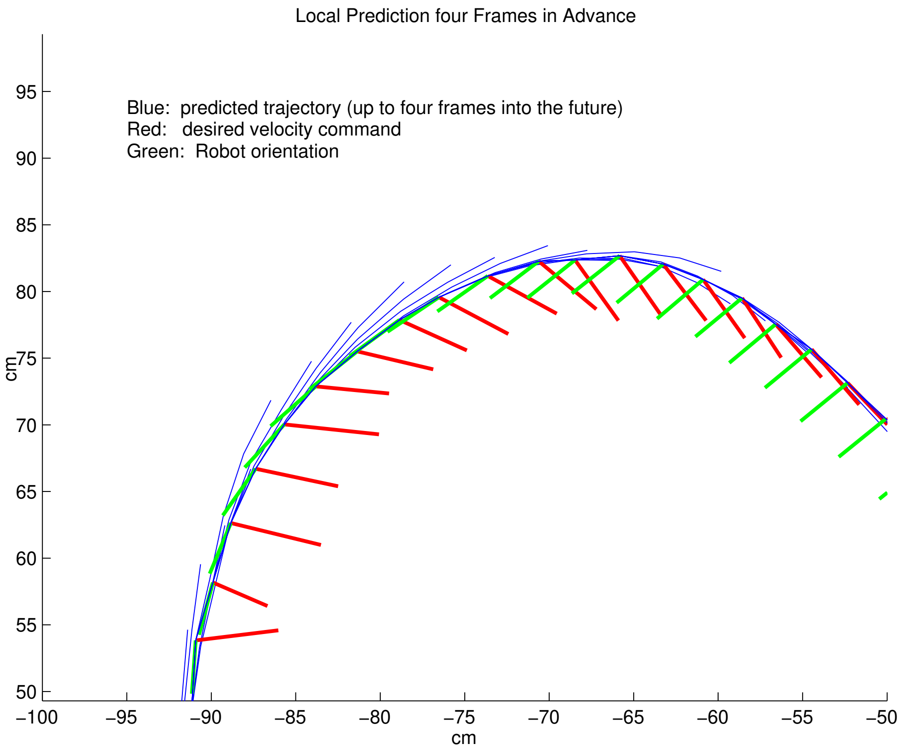

Fig. 10. A zoom of the predictions, orientation and desired velocity.

7 Improving the predictors

A linear predictor with the same number of input variables as the neural network is competitive with it. We then tested a straightforward improvement for the linear predictor, which is also used for time series analysis with ARIMA models (autoregressive, moving average models). A linear predictor was trained to minimize the prediction error for the fourth frame after the last data, using a window of length five (the last five frames were used for the prediction). After four frames, the error between predicted position and actual position becomes available and can be used as a parameter for a new prediction. We used the x-direction and y-direction quadratic error for the last three points together with the window of five points and commands used for the prediction. The rationale behind this choice, is that the linear predictor can "sense" when the prediction is far-off and can compensate the error. This happens mostly when the robots makes sharp turns. The effect of considering the prediction error is equivalent to an artificial enlargement of the prediction window, without increasing excessively the number of parameters. When the last frame that arrived is frame i, we are predicting frame $i + 4$. For the prediction we use the frames $i - 4$ to

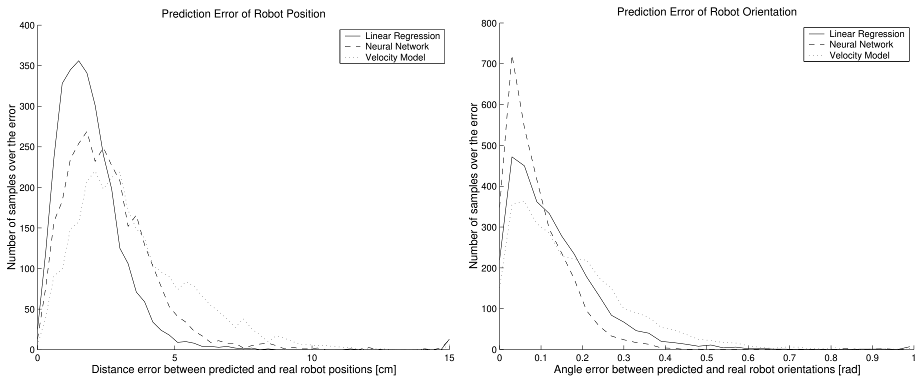

Fig. 11. Comparison between the histograms of linear and neural network predicted robot position (left) and orientation (right) error. Both histograms have about 3000 samples.

i and the prediction errors at $i-2$ to i. The prediction error for frame $i-2$ contains information from the frames $i-6$ to $i-10$. Therefore, we are using this information but in a condensed form, that does not make the number of free parameters explode.

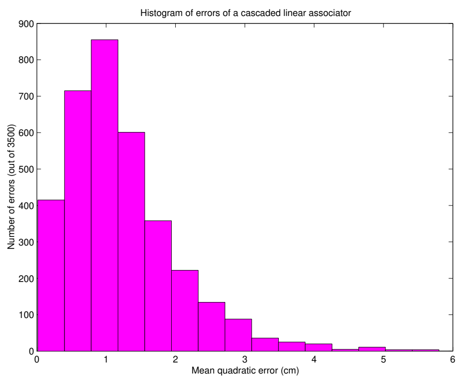

Fig. 12. The histogram of errors of a cascaded linear associator

A linear predictor with a mean squared error of 1.32 cm could be improved in this manner to a mean squared error of 1.23 cm for a certain data set (Fig. 12). This means that the method works, but since the accuracy gain is only marginal, we decided not to include this type of cascaded predictor in our control system. Much more promising seems to be to combine the output of a linear predictor with the output of a neural network, in order to increase the accuracy of the prediction. In the case of predictors with uncorrelated errors, such an approach should decrease the total mean error by a factor $\sqrt{2}$.

8 Measuring the vision noise

One important problem when tracking a robot using a video camera is finding out the magnitude of the noise introduced by the vision system. In the RoboCup environment, the absolute position error is not as important as the relative error. The absolute position of the robot is computed using a mapping from the image pixels to field coordinates. If there is a systematic error, and all absolute coordinates are shifted by a few millimeters, usually this will not impact the way the robots play, since the robots move relative to the other robots and the ball. We try hard to get a good map to absolute field positions, but the vision noise is for our purposes the more relevant problem.

We are able to measure the noise in the robot's coordinates by making the assumption that the real robot follows a smooth path: the difference between such a smooth path and the robot coordinates should be the vision noise.

To compute a smooth approximation to every point in the robot trajectory, we take three points before and three after the current point (in field coordinates). Using them, we train a linear model that predicts the position of the current point (both for the x and y coordinates). The average difference between the predicted (smoothed) position and the real position is the vision noise.

Using this approach we could determine that the noise introduced by our vision system is 3 mm. This means that the real position of the robot has a normal circular distribution with standard deviation of 3 mm (which corresponds to 2.1 mm noise in the x-direction and 2.1 mm noise in the y-direction), a surprisingly accurate value, considering that the robots are moving fast in the large field.

We tested this assumption, that the noise in the signal can be estimated by predicting intermediate frames, by generating the artificial path shown in Fig.13. We then added Gaussian noise to the path and tried to estimate the noise using a linear regression. The result was that we could estimate the noise correctly, up to a factor 1.2 when using three points before a frame and three points after a frame, to determine the position of the moving point in the selected frame. Therefore, our method overestimates the noise by 20 percent. In the case of our moving robots this means that 3mm is an upper bound on the real vision noise.

We then made another experiment: a linear model trained previously to predict the fourth frame was used with the original data. Let us call the data the vector $z_1, z_2, \ldots, z_m$ and the coeficients of the linear model $a_1, a_2, \ldots, a_m$. Then the output of the linear model is

$$z = a_1 z_1 + a_2 z_2 + \cdots + a_m z_m$$

We added normal distributed noise to the x and y coordinates, each with standard deviation of 2.1 mm and recomputed the output. The difference between

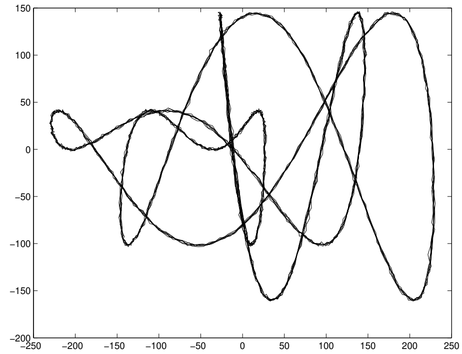

Fig. 13. An artificially generated path, with noise added.

the new result z' and a noisy version of z (that is z plus Gaussian noise) gives us an estimate of the error rate of the best possible predictor. The average deviation we obtained was 3.8 mm. This means that our linear predictors (with errors below 1.5 cm) are actually very accurate and very near to the minimum possible prediction error. No predictor can predict the position of a robot in the fourth frame with less than 3.8 mm error, because the vision noise prohibits it.

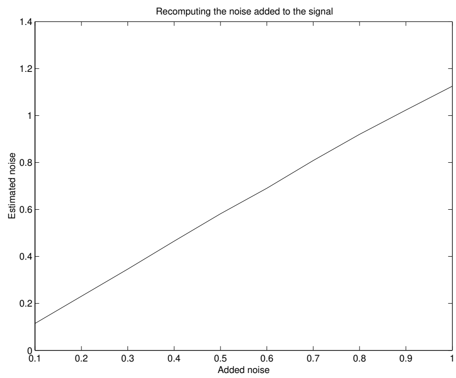

Fig. 14. Estimated noise using linear regression versus real noise (using four points before, and four points after a given point)

9 Effect of using commands for the prediction

Another interesting experiment consisted in testing how much of the quality of the predictor comes from the dynamics of the robot itself and how much from the knowledge of future commands. We trained a simple linear model and another one, which only used the position and orientation of the robots.

A linear predictor which was given access to the last six commands for a particular trajectory and a particular training set, had an error of 1.32 cm. The predictor which was blind to the previous commands to the robot had an error of 2.43 cm. This means that the knowledge of the previous commands helps to lower the prediction error by almost 1.1 cm, which is 45 percent, i.e. almost half, of the original error.

10 Conclusion and Future Work

We have successfully developed, implemented, and tested linear models and a small neural network for predicting the motion of our robots. The prediction compensates for system delay and thus allows more precise motion control, ball handling, and obstacle avoidance. To make the perception of the world consistent, prediction is not only used for our own robots, but also for the robots of the opponent team and the ball. However, here the action commands are not known and hence simpler predictors are used.

For advanced play, it would be beneficial to anticipate the actions of opponent robots, but this would require learning during a game. Such online learning is dangerous though, because it is hard to automatically filter out artifacts from the training data, caused e.g., by collisions or dead robots.

Another possible line of research would be to apply predictions not only to basic robot motion, but also to higher levels of our control hierarchy, where delays are even longer.

Finally, one could also integrate the neural predictor into a simulator as a replacement for a physical model. A simulator allows quick assessment of the consequences of actions without interacting with the external world.

If there are multiple action options during a game, this 'mental simulation' could be used to decide which action to take.

Another interesting application could be to use the average prediction error over time as an early predictor of possible robot malfunction or hardware problems. This would give an opportunity to change a robot before it stops working in the middle of a game.

References

1. Rojas, Raúl: Neural Networks – A Systematic Introduction. Springer Verlag, Heidelberg, 1996.
2. Behnke, Sven; Frötschl, Bernhard; Rojas, Raúl; Ackers, Peter; Lindstrot, Wolf; de Melo, Manuel; Schebesch, Andreas; Simon, Mark; Spengel, Martin; Tenchio, Oliver: Using Hierarchical Dynamical Systems to Control Reactive Behavior. Lecture Notes in Artificial Intelligence **1856** (2000) 186–195.
3. Simon, Mark; Behnke, Sven; Rojas, Raúl: Robust Real Time Color Tracking. Lecture Notes in Artificial Intelligence **2019** (2001) 239–248.

4. Rojas, Raúl; Behnke, Sven; Liers, Achim; Knipping, Lars: FU-Fighters 2001 (Global Vision). Lecture Notes in Artificial Intelligence **2377** (2002) 571–574.
5. Veloso, Manuela; Bowling, Michael; Achim, Sorin; Han, Kwun; Stone, Peter: CMUnited98: RoboCup98 SmallRobot World Champion Team. *RoboCup-98: Robot Soccer World Cup II*, pp. 61–76, Springer, 1999.
6. Wolpert, Daniel M.; Flanagan, J. Randall: Motor Prediction. *Current Biology Magazine*, vol. 11, no. 18.
7. Miall, R.C.; Weir, D.J.; Wolpert, D.M.; Stein, J.F.: Is the Cerebellum a Smith Predictor? *Journal of Motor Behavior*, vol. 25, no. 3, pp. 203–216, 1993.
8. Smith, O.J.M.: A controller to overcome dead-time. *Instrument Society of America Journal*, vol. 6, no. 2, pp. 28–33, 1959.
9. Browning, B.; Bowling, M.; Veloso, M.M.: Improbability Filtering for Rejecting False Positives. *Proceedings of ICRA-02, the 2002 IEEE International Conference on Robotics and Automation*, 2002.