Predicting Away Robot Control Latency

Sven Behnke[2], Anna Egorova[1], Alexander Gloye[1],
Raúl Rojas[1], and Mark Simon[1]

[1] Freie Universität Berlin, Institute for Computer Science
Takustraße 9, 14195 Berlin, Germany
[2] International Computer Science Institute
1947 Center St., Berkeley, CA, 94704, USA
`http://www.fu-fighters.de`

Abstract. This paper describes a method to reduce the effects of the system immanent control delay for the RoboCup small size league. It explains how we solved the task by predicting the movement of our robots using a neural network. Recently sensed robot positions and orientations as well as the most recent motion commands sent to the robot are used as input for the prediction. The neural network is trained with data recorded from real robots.
We have successfully field-tested the system at several RoboCup competitions with our *FU-Fighters* team. The predictions improve speed and accuracy of play.

1 Introduction

The time elapsed between making an action decision and perceiving the consequences of that action in the environment is called the control delay. All physical feedback control loops have a certain delay, depending on the system itself, on the input and output speed and, of course, on the speed at which the system processes information.

In the RoboCup small size league a global vision module is used to determine the positions of all robots on the field. In order to control the behavior of the robots in a way that is appropriate for the situation on the field we need the exact positions of them at every moment. Because of the delay inherent in the control loop, however, the behavior control system actually reacts to the environment four frames ago (about 132 ms). When moving faster – our robots drive at a speed of up to 2 m/s – the delay becomes more significant as the error between the real position and the position used for control grows up to 20 cm.

In order to correct this immanent error we have developed a neural network which processes the positions, orientations, and the action commands sent to the robots during the last six frames. It predicts the actual positions of the robots. These predictions are used as a basis for control. We use real recorded preprocessed data of moving robots to train the network.

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 predicts 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 to our system architecture. Then we explain how the delay is measured and we present some other approaches to eliminate the dead time. Section 4 describes architecture and training of the neural network used as a predictor. Finally, we present some experimental results and some plans for future research.

2 System Architecture

The small size league is the fastest physical robot league in the RoboCup competition relative to the field size. Top robot speeds exceed 2m/s and acceleration is limited by the traction of the wheels only, hence a robot can cross the entire field in about 1.5 sec. This is possible, since the sensing is done mainly by a camera overlooking the field and behavior control is done mainly by an off-the-field computer. Action commands are sent via a wireless link to the robots that contain only minimal local intelligence. Thus, the robot designer can focus in this league on speed, maneuverability, and ball handling.

Our control system is illustrated in Fig. 1. The only physical sensor we use for behavior control is one S-Video camera[1]. It looks at the field from above and produces an output video stream, which is forwarded to the central PC. Images are captured by a frame grabber and given to the vision module.

The global computer vision module analyzes 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. It is described in detail in [3].

Based on the gathered information, the behavior control module then produces the commands for the robots: desired rotational velocity, driving speed and direction, as well as the 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 pulse generators which are integrated in each motor.

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

[1] There are various other sensors on the robots and in the system, but they aren't used for behavior control. For example, the encoders on the robots are only used for their motion control.

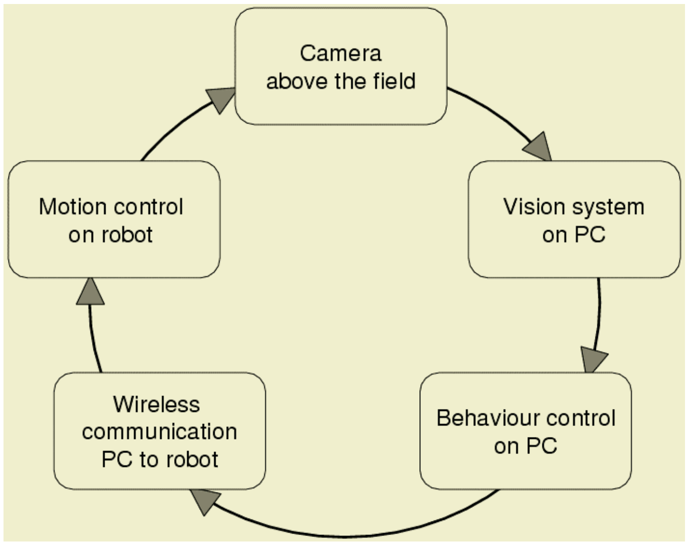

Fig. 1. The feedback control system. All stages have a different delay. But only the overall delay is essential for the prediction.

stages of the control loop contribute to the control delay that is also known as dead time. Although image capture, computer vision, and behavior control run at a rate of 30 Hz, motion control runs at 80 Hz, and the entire system is optimized to minimize delay, all the small delays caused by the individual stages add up to about 100 to 150 ms. For the purposes of control, all delays can be aggregated to a single dead time. This is illustrated in Fig. 2(a). If a robot moves at 2 m/s and the dead time is 100 ms, the robot will travel a distance of 20 cm before we perceive the consequences of an action decision. This causes problems when robots move fast, producing overshooting or oscillations in movement control.

In order to measure the system delay, we use the following simple technique. We let the robot drive along the field with a speed that is determined by a sinusoidal function. This means, the robot moves back and forth with maximum speed in the middle of the path, slowing down and changing direction at 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.

There are many possibilities to counter the adverse effects of the control delay. The easiest way would be to move slower, but this is frequently not desirable, since fast movement is a decisive advantage in play. Another possibility would be to reduce precision requirements, but this would lead to unnecessary collisions with other players and uncontrolled ball handling.

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 con-

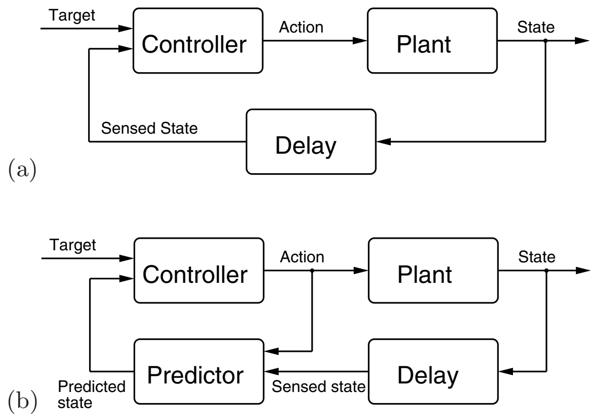

Fig. 2. Control with dead time: (a) control is difficult if the consequences of the controller actions are sensed with significant delay; (b) a predictor that is trained to output the state of the plant can reduce the delay of the sensed signal and simplify control.

sequences 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.

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. 2(b), where a predictor module approximates a negative delay. It has access to the delayed plant state as well as to the undelayed controller actions and is trained to output the undelayed 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 a Extended Kalman-Bucy Filter [9] to predict non-linear systems. But this approach requires a good model of the plant. We propose to use a neural network as a predictor for the robot motion, because this approach doesn't 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 from the visual feedback.

4 Neural Network Design

Since we have no precise physical model of the robot, we train a three layer feed-forward network to predict the robot motion. The network has 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]. A great advantage of the neural network is that it can be easily trained again if something in the system changes, for example if a PID controller on board the robot 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.

4.1 Input Vector

The input presented to the neural network is based on position and orientation of the robot as perceived by the vision module during the last six frames, as well as the last few motion commands sent to the robot. Some preprocessing is needed to simplify the prediction task. The preprocessing assumes translational and rotational invariance. This means that the robot's reaction to motion commands does not depend on its position or orientation 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 the training. The action commands are also 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.

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

4.2 Target Vector

The target vector we are using for training the network consists of two components: the difference vector between the current position and the position four frames forward in the future and the difference between the current orientation and the orientation four frames ahead. They are encoded in the same format as the input data.

4.3 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 specialized behaviors developed for this purpose.

To cover all regions of the input space, the robot must encounter all situations that could happen during game play. They include changing speed over a wide range, 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 neural network has no information about obstacles and hence can not be trained to handle them. If we would include such cases in the training set, the network 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 including additional input features for the neural network, e.g. a sensor for obstacles, and thus handle also this situation, but this would complicate network design and would require more training data to estimate additional parameters.

5 Results

We have extensively tested the neural network for the prediction of the position and orientation of the robots since its first integration into the FU-Fighters' system. It performs very well and we have nearly eliminated the influence of the delay on the system.

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. We compare the performance of the neural predictor to a linear predictor that has been trained on the same data.

The histograms in Fig. 3 show that the neural network prediction has much more samples with small errors than the linear prediction. The average position error for the linear prediction is 5.0471 cm and 2.8030 cm for the neural network prediction. The average orientation error for the linear prediction is 0.1704 rad (9.76°) and 0.0969 rad (5.55°) for the neural network prediction.

6 Conclusion and Future Work

We have successfully developed, implemented, and tested a small neural network for predicting the motion of our robots. The prediction compensates for the system delay and thus allows more precise motion control, ball handling, and obstacle avoidance. To make the perception of the world consistent, predictions

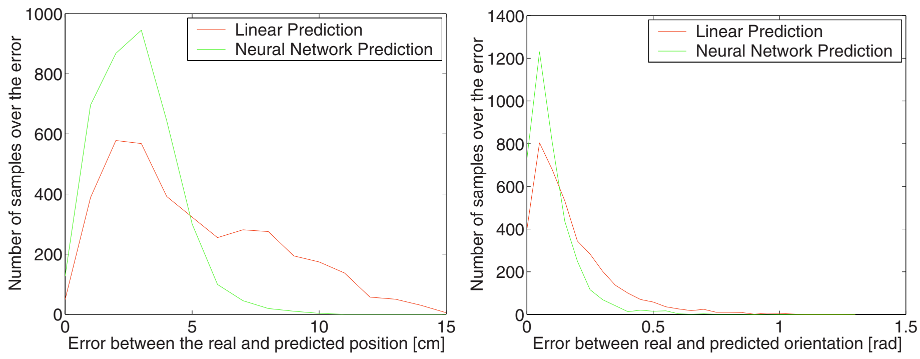

Fig. 3. Comparison between the histograms of linear and neural network predicted robot position (left) and orientation (right) error. Both histograms have about 3000 samples.

are not only used for own robots, but also for robots of the opponent team and the ball. However, here the action commands are not known and hence simpler predictors are used. We employ Kalman filters to model linear effects.

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 robot 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.

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.