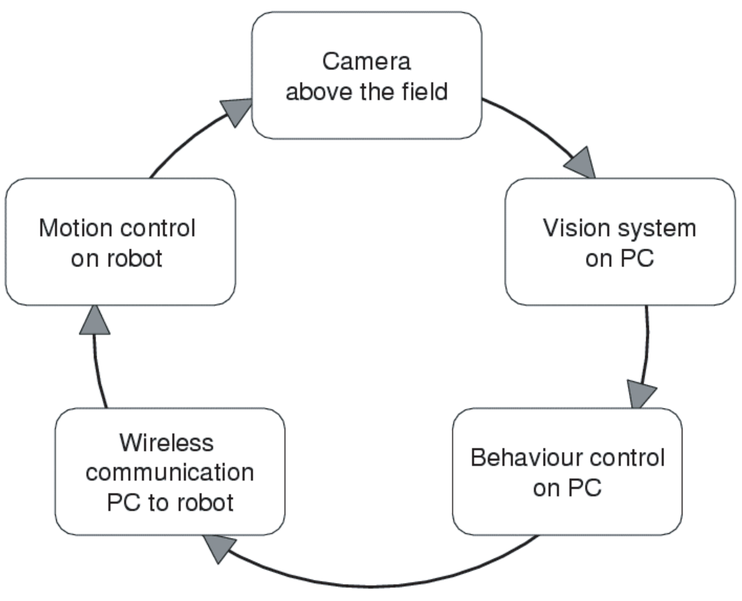
Predicting Away Robot Control Latency

Alexander Gloye, Mark Simon, Anna Egorova,
Sven Behnke, and Raúl Rojas

Freie Universität Berlin
Institut für Informatik
AG Künstliche Intelligenz
Takustr. 9,
14195 Berlin

The control delay problem

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. 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.

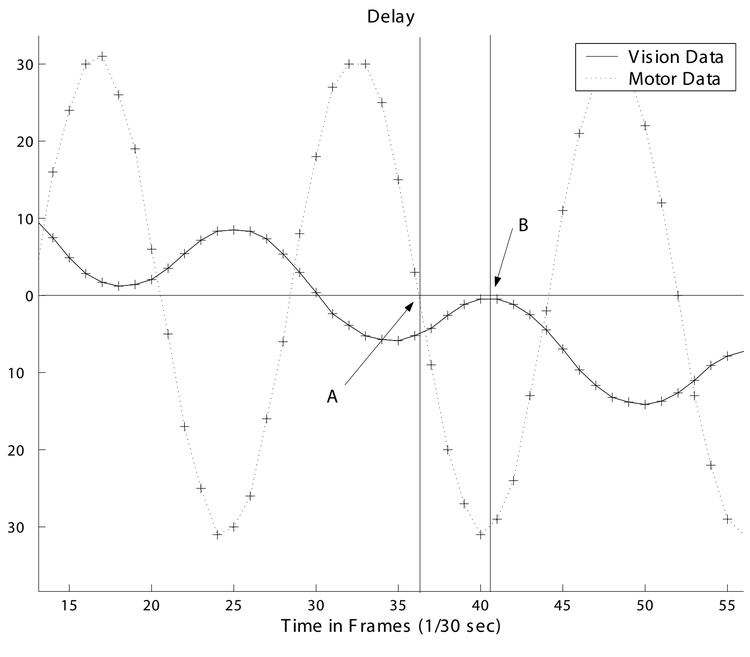


Sources of Delay

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.

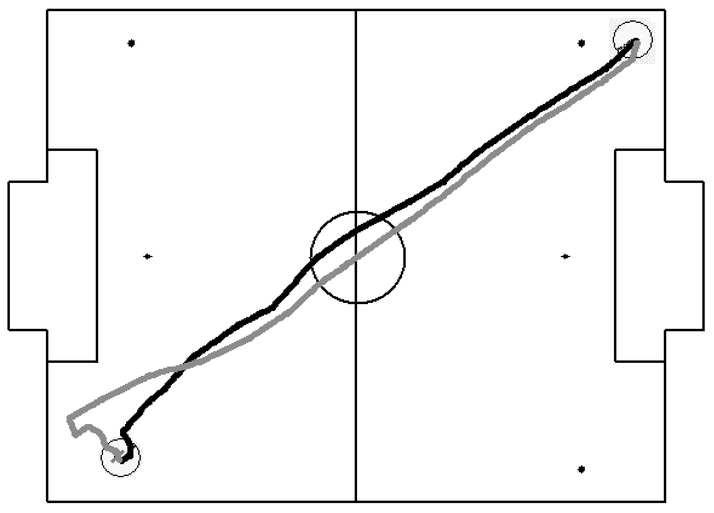
The delay sources and their impact are the following:

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.



Measuring the delay

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 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



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.

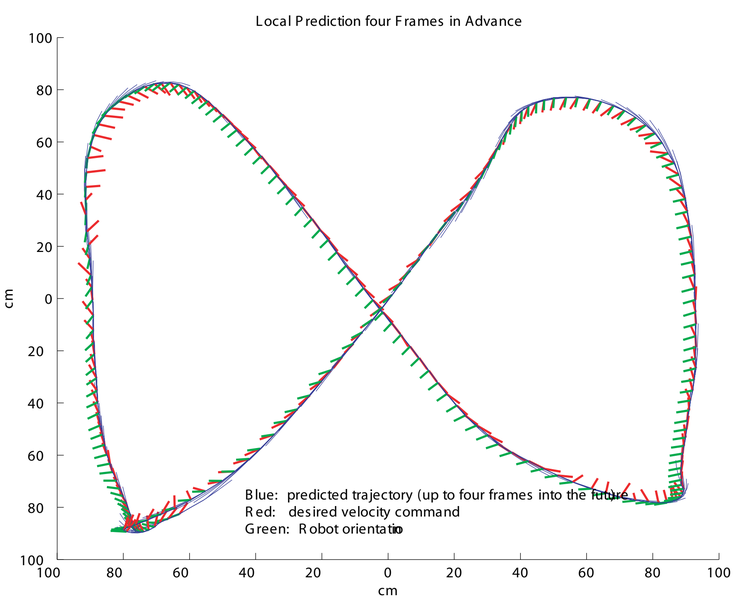
Predictor Design

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.

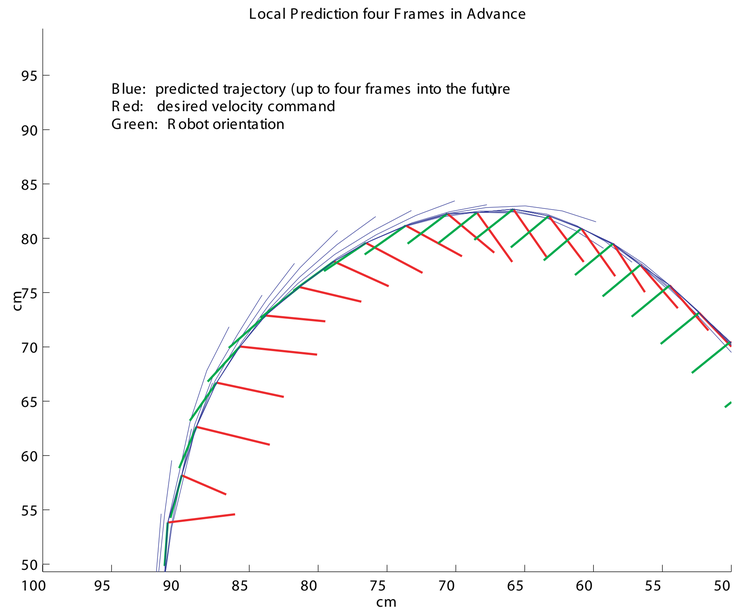
The position data consists of six vectors -- the difference vectors between the current frame and the other six frames in the past, given as (x,y)-coordinates. The orientation data consists of six angles, given as difference of the robot's orientation between the current frame and the other six ones 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 -180° and 180° 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.
.



Trajectory of a robot showing the result of predicting the path four frames in advance

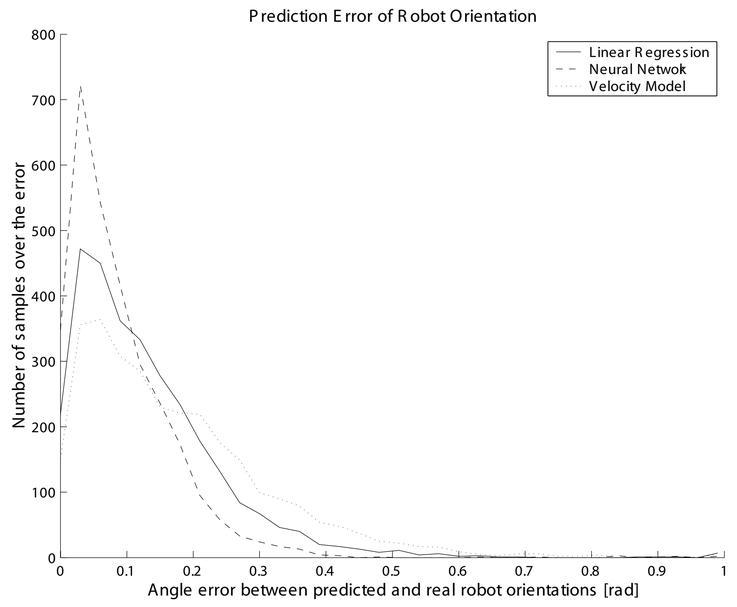
The linear model consists therefore of 42 constants that have to be computed. We train a different linear model for the x coordinate and for the y coordinate.
On the other side, the neural network consists of 42 input units, 10 hidden units, and 4 output units. The hidden units have a sigmoidal transfer function while the transfer function of the output units is linear. We train the network with recorded data using the standard backpropagation algorithm



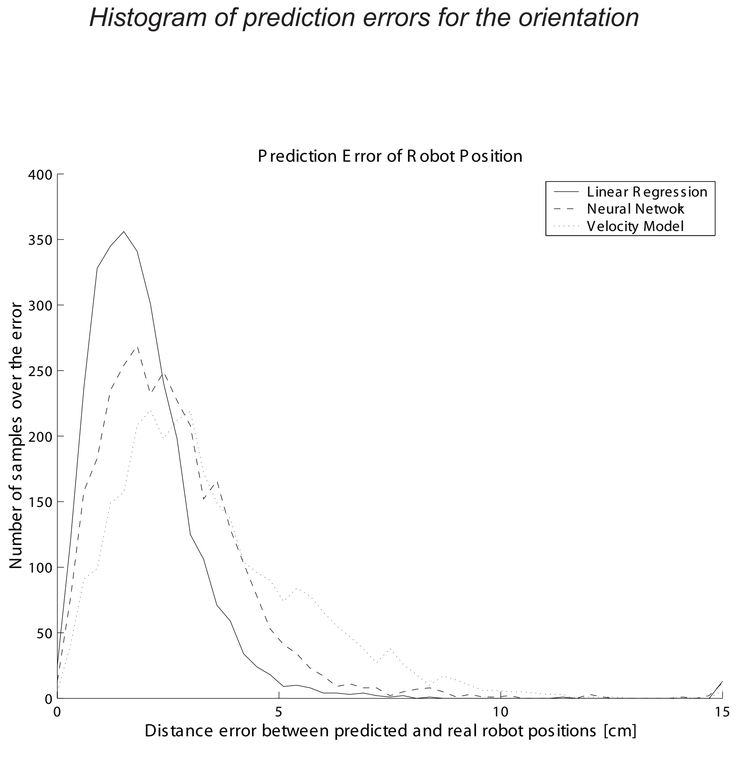
Detail of the predicted trajectories, four frames each time

Results

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 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 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 theneural 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.



Histogram of prediction errors for the orientation



Histogram of prediction errors for the position

This and other papers can be found at our website:

www.fu-fighters.de