Handling robot collisions in the state estimator module of the MI20 robotsoccer system

Sido Grond
Department of Computer Science
University of Twente
Enschede, Netherlands
Email: s.grond@tiscali.nl; robotsoccer@ewi.utwente.nl

Albert L. Schoute
Department of Computer Science
University of Twente
Enschede, Netherlands
Email: a.l.schoute@utwente.nl

***Abstract*— This paper describes a method to correct the effect of collisions in the state estimator of the MI20 robotsoccer system. The existing state estimator tracks all robots on the field by predicting new states and matching them with data from the vision system to obtain new positions and velocities of all the objects on the field. In general this method works very accurate, unless collisions happen. In that case, robots are projected on each other, causing identification mismatches. This thesis describes an approach to overcome that problem.**
First, collisions are detected using the *Separating Axis Theorem*. After a collision is detected, it is corrected. This happens in two phases: in the first phase the robots are set back to a position where they only touch each other. In the second phase a collision response model is applied, which yields the new velocities of the robots.
Evaluation of the method gives a good result. When testing, almost no identification mismatches occurred and an analysis of the distance errors shows a significant decrease.

I. Introduction

In 2002 the MI20[1] robotsoccer project was started at the University of Twente. Since then, a system has been developed that competes at the MiRoSoT European Championships every year and occasionally on the World Cup events.

The MI20 system is designed as a distributed system. It means that the system is divided in a number of modules which all handle a specific task and communicate with each other over a TCP/IP connection. However, in practice the system is run at one PC most of the time, discarding the necessity of the network connection.
The separation of the modules is also a good way to be able to work with a team on different aspects simultaneously. In figure 1 an overview is given of the interdependencies between the modules.

All robots of the MI20 team have a unique identifier(ID). This ID is used in the RF communication of the wheel velocities, so every robot can extract his velocities from the broadcasted package.
To be able to send the right velocities to the adequate robot, the decision system also has to know these IDs, so it knows which robot is positioned where. Because all robots look identical we cannot distinguish them from each other when looking at one camera image. Therefor, the tracking of robots is done in the state estimator.
When the software system is started, all team robots are assigned the accompanying ID in the user interface. For every frame the camera sends, the state estimator predicts the new states of the objects on the field. This prediction is matched with the images acquired from the vision system, so the IDs can be tracked when the robots move.
The main problem of this method is the swapping of robot IDs. This swapping occurs because predictions of the new robot states don't account for any physical contact with other objects like robots, walls or the ball. This can make it possible for robots to be predicted on the same position or to partly overlap.
This paper describes a method to handle those false predictions, by detecting them and making a correction. The ultimate goal is to have no swapping of IDs anymore when a game is played.

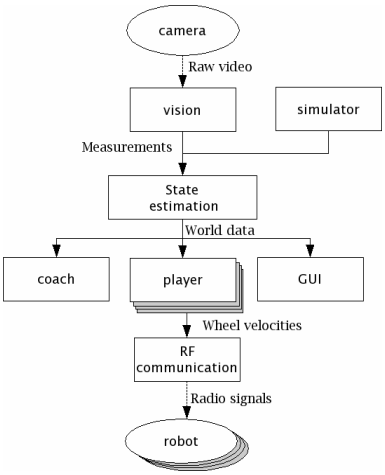


Fig. 1. An overview of the MI20 system. The oval parts represent hardware and the square parts software.

The remainder of this paper is outlined as follows. The concept of a state estimator is discussed in section II.

[1]Mission Impossible Twente

Section III discusses the role of collisions in robotsoccer. In section IV the used collision detection algorithm is described. In section V a method for determining the contact point of the robots is presented and in section VI a physical model is shown to reflect the occurences on the field as naturally as possible. The given method will be evaluated in section VII. Finally, a conclusion and some recommendations are stated in section VIII.

II. State Estimation

State estimation is the process of predicting the trajectories of the ball and the robots. These predictions are used by other parts of the system to plan and execute robot actions. One of the main goals of state estimation is to keep track of the team robots. These robots can not be distinguished from each other by only taking a look at one image from the camera module. Instead, the state estimator assigns an identifier to every team robot first and then tracks the robots by comparing successive images.

The state estimation module receives images (called frames) from the vision module at a certain rate, in the MI20 system at 30 frames per second. This comes down to a period of 0.033ms between successive frames. The vision module extracts all the objects from the frame, but doesn't give information about assigned identification numbers. It is a task of the state estimation module to keep track of this. This state estimate becomes available as so called *world data* to the rest of the system.
The design and improvement of the state estimator currently used is described extensively in [1] and [2]. This section is meant as a short overview of the state estimator concept.

Obviously, to estimate a state in the system, one first has to define the term *state*. The state $\boldsymbol{x_k}$ of a team robot in period k is defined by

$$\boldsymbol{x_k} = (x, y, \theta, \dot{x}, \dot{y}, \omega)^T \tag{1}$$

Here, x and y form its position, θ its orientation, $\dot{x}$ and $\dot{y}$ its linear velocity components and ω its angular velocity.
For team robots, a control vector $\boldsymbol{u_k}$ can be defined

$$\boldsymbol{u_k} = (c_v, c_\omega)^T \tag{2}$$

which contains all the possible control signals, being the linear and angular velocity. In the MI20 system these control signals change the wheel velocities of a robot.

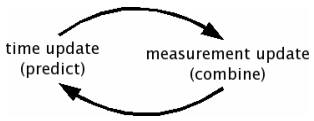


Fig. 2. Phases in the state estimator

The estimation of a new state can be calculated with various methods, but in the MI20 system the 'Extended Kalman Filter' is chosen. See [2] if you want to know why. The estimation is done in two phases(figure 2). First, in the *time update phase*, a prediction $\hat{\boldsymbol{x}}_k^-$ is made for the object state at moment k, when the next measurement will come in.

$$\hat{\boldsymbol{x}}_k^- = \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_{k-1}) \tag{3}$$

This prediction is based on the last estimated state $\hat{\boldsymbol{x}}_{k-1}$ and the control vector $\boldsymbol{u}_{k-1}$ (if available: so only for team robots). After this prediction is made, a new measurement is received from the vision module in the *measurement update phase*. A measurement $\boldsymbol{z}_k$ can be modeled as shown in (4).

$$\boldsymbol{z} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{\theta} \end{pmatrix} \tag{4}$$

Now, the next state $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$ is calculated from the prediction $\hat{\boldsymbol{x}}_{\boldsymbol{k}}^-$ and the measurement $\boldsymbol{z}_k$ as in (5).

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{h}(\hat{\boldsymbol{x}}_k^-)) \tag{5}$$

Here, $\boldsymbol{K}_k$ is the *Kalman Gain* and $\boldsymbol{h}(\cdot)$ the *measurement function*. The Kalman Gain determines the influence of the measurement and the predicted state.
After the measurement update phase has finished, the previous states are associated with the new ones. This process is called *gating* ([1]). Gating normally works quite simple. It takes an estimated state and matches it with the closest of the previous states, as long as it obeys a certain threshold. When no distance can be found that is shorter than the threshold, the threshold is enlarged for the next frame. The gating procedure will also fail when according to the prediction, two robots have driven through each other. Then, a predicted state can be closer to the previous state of another robot, causing a mismatch in the identifier assignment.

III. Collision modeling

The Kalman filter is based purely on the state and control vectors. Therefor, it is difficult to constrain the prediction with physical boundaries like other robots or the wall surrounding the field. Because of this, the predictions should be corrected for these impossible situations before the next measurement comes in. The new filter loop then looks like figure 3.

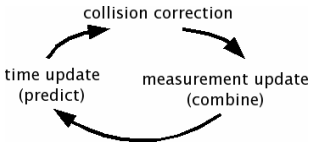


Fig. 3. The new filter loop for the Kalman filter

The only impossible situation discussed in this paper is the projection of two robots on the same spot. In [3] and [2], solutions for other types of collisions are presented.

The collisions between robots could be detected by checking if for a predicted state the two robots have moved through

each other or are overlapping. To check if the robots moved through each other, the previous state of the robots is needed. Then, it will be simple to check if the paths have crossed.
To check if the predicted states of two robots are overlapping we use the Separating Axis Theorem as discussed in the next section. After a collision has been detected, the next step is to respond to this collision in a realistic way. This collision response has been divided into two phases. In phase 1, the robots are set back to positions at which they touch each other just before the collision would take place. This process is described in section V. In phase 2, the robots will respond to the collision in a realistic way, using a physical model. This phase is discussed in section VI.

IV. Collision detection

A common collision detection algorithm is the OBBTree algorithm [4]. OBB stands for Oriented Bounding Box and the idea is that all objects can be composed by a tree of bounding boxes. Of course, a robot of the MI20 system can be modeled by one oriented bounding box.
To test the overlap of two OBB's, the algorithm uses a method called *separating-axis theorem*. This theorem states that if two objects are not overlapping, a separating axis can be found on which the projections of these objects do not overlap. An example of this projection is given in figure 4.

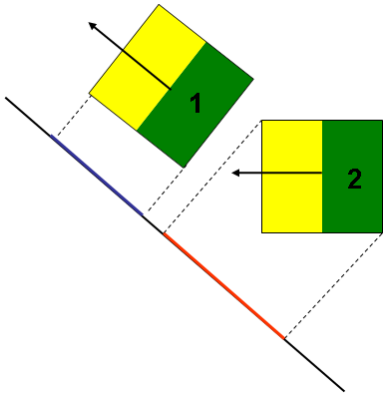
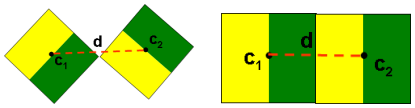


Fig. 4. Separating-Axis Example

The next problem is how to find an appropriate axis. The theory states that for each convex polygon, the normal of every edge and face would be a possible separating axis. So for the $2D$ case of robot soccer, the amount of axes to check for overlap between two robots, reduces to two axes per robot (because the normals of the top and bottom edges and of the left and right edges are aligned on the same axis). Now, if the two robots are projected to one axis there are two possibilities: the projections are overlapping on the axis and examination of the other axes is needed, or the projections are not overlapping and a separating axis has been found.
If for all four axes the projections overlap, the robots will also overlap and a collision has been detected.

V. Pre-Collision correction

When an overlap of two robots has been detected, they have to be pulled away from each other to positions where they only touch each other but do not overlap, being the moment just before they collide. This state will be called the *pre-collision point*.
For our robots, the distance is dependent on the orientation of the robots as can be seen in figure 5.



(a) Robots touching each other diagonally

(b) Robots touching each other horizontally

Fig. 5. The distance between the centers is larger on the left side, than on the right side

An accurate approach to determine the pre-collision point has been drawn from the area of traffic control. This method can calculate the positions exactly. It is based on the analysis of *mutual collision areas* [5]. A mutual collision area is the area on a field or road, which is crossed by both vehicles when they will follow their planned paths, see figure 6. This area is

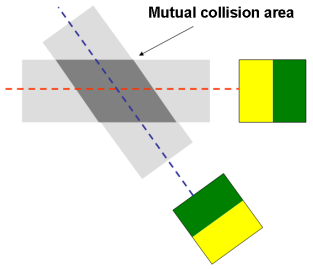


Fig. 6. Two vehicles crossing each other's paths.

the only area where a collision can occur. So, in our case the area is formed by combining the paths from the old positions, where no collision had occurred, to the new positions, where the robots are overlapping. However, the form of this area is dependent on the form of the vehicles.
For example, consider a robot R_i traveling a path P_i. The pose of the robot is defined by $(x(s_i), y(s_i), \theta(s_i))$, where s_i is the current path position. Now (s_i, s_j) is a collision configuration if the shapes of R_i and R_j overlap.
This concept is visualized in figure 7. Here, a configuration diagram of two rectangular robots is shown. The surrounding rectangle is called the *mutual critical area*. The size of this

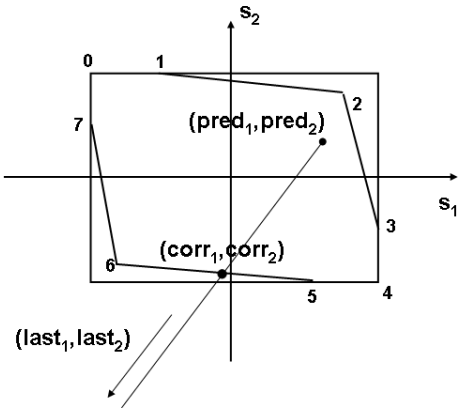


Fig. 7. Configuration diagram of squared vehicles crossing each other under an angle γ where $\gamma > 90°$

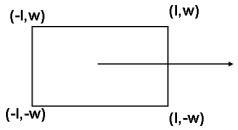


Fig. 8. Corners of a rectangular robot

area depends on the width and height of the two robots and the angle γ at which they cross. The octangular form inside the rectangle represents the *collision states* of the two vehicles, being all the possible configurations where the robots overlap. If we define the points on a robot as in figure 8, the marked points can be calculated as follows:

$$\begin{array}{ll} 0\text{: } (s_i, s_j) = (-r_i - l_i, -r_j - l_j) & 4\text{: } (s_i, s_j) = (\ r_i + l_i,\ \ r_j + l_j) \\ 1\text{: } (s_i, s_j) = (-r_i + l_i, -r_j - l_j) & 5\text{: } (s_i, s_j) = (\ r_i - l_i,\ \ r_j + l_j) \\ 2\text{: } (s_i, s_j) = (\ q_i + l_i, -q_j - l_j) & 6\text{: } (s_i, s_j) = (-q_i - l_i,\ \ q_j + l_j) \\ 3\text{: } (s_i, s_j) = (\ r_i + l_i,\ \ r_j - l_j) & 7\text{: } (s_i, s_j) = (-r_i - l_i, -r_j + l_j) \end{array}$$

$$\text{where} \quad r_i = \frac{w_j + w_i cos\gamma}{sin\gamma} \qquad r_j = \frac{w_i + w_j cos\gamma}{sin\gamma}$$
$$q_i = \frac{w_j - w_i cos\gamma}{sin\gamma} \qquad q_j = \frac{w_i - w_j cos\gamma}{sin\gamma}$$

These formulas can be used to calculate the exact positions of two robots in which they only contact each other and don't overlap. This is done with the knowledge of figure 9. In this figure, $last_1$ and $last_2$ are the previous known positions of the robots, $pred_1$ and $pred_2$ the predicted positions as calculated by the state estimator, w the half width of a robot , l the half length of a robot and finally γ the angle between the two traveled paths. These last three variables can be substituted directly into the given formulas to get figure 7.

Because the two robots are overlapping, the predicted configuration (s_1, s_2) has to be somewhere in the collision area of this figure. But how can we determine the pre-collision configuration? It is calculated with the help of the positions of $last_1$, $last_2$, $pred_1$ and $pred_2$. First, the intersection of

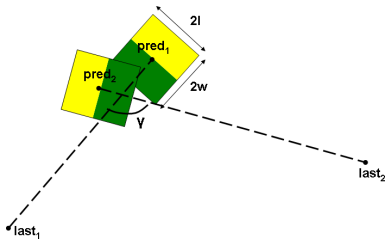


Fig. 9. The collision of two robots

the two paths $P_1 = [last_1, pred_1]$ and $P_2 = [last_2, pred_2]$ is defined as the center point of the octangular form. Now, s_i is the position relative to the intersection for robot i on the path P_i. So, a configuration (s_1, s_2) is a combination of two positions on the paths P_1 and P_2.

Next, we can draw two states in the configuration diagram: $(last_1, last_2)$ and $(pred_1, pred_2)$, see figure 7 again.

Note that this joint trajectory line through both states is not necessarily going through the origin, because both vehicles generally are not at the crossing at the same time. Furthermore, $Robot_2$ has traveled a longer distance[2] than $Robot_1$ so the slope of the line is not $45°$, but a bit more steeper.

Now, the line through these two points will also cut the boundary of the collision area at some configuration point $(corr_1, corr_2)$. This configuration point is the point we are seeking: the robots will only touch each other at that point, as can be seen in figure 10.

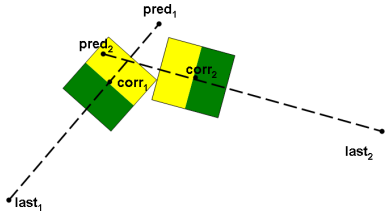


Fig. 10. The corrected version of two colliding robots. This scheme represents the moment just before the collision is about to take place.

As a conclusion it can be said that this method is a good way to calculate the pre-collision point positions. However, there are a few problems with this solution especially when the paths of the robots are nearly parallel, but these cases can be corrected quite simple.

VI. Collision response

As mentioned before, the physical response model is the second phase of the correction of a robot collision. The main goal of the correction should be to determine the new positions and linear velocities of the robots after the collision, so the gating procedure has as accurate inputs as possible. The secondary goal is the determination of the new orientations and angular velocities of the objects.

[2]so traveled faster

The base of the physical model was extracted from the four articles on physics in game programming by Chris Hecker [6]. In this article it is assumed that a collision always occurs between an edge and a vertex (figure 11).
The starting point at which the collision model will apply is the moment when the two robots just touch each other, being the moment just before the actual collision which we determined in the previous section.

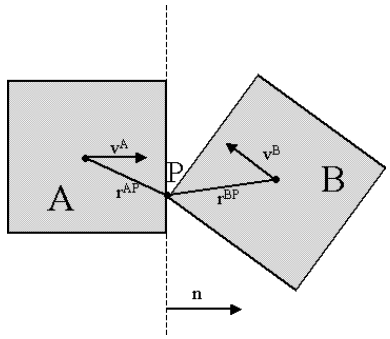


Fig. 11. Components involved in the collision between two robots.

When determining the new velocities of the robots, one has to keep in mind that the velocity of a rotating object can be split up into two factors: the linear velocity $\mathbf{v}$ at which the *center of mass* of the object moves and the angular velocity ω of the points around the center of mass. Now the total velocity $\mathbf{v}^{AP}$ of a point P on object A is defined as

$$\mathbf{v}^{AP} = \mathbf{v}^{A} + \omega^{A}\mathbf{r}_{\perp}^{AP} \tag{6}$$

Here, $\mathbf{r}_{\perp}^{AP}$ is a perpendicular vector of $\mathbf{r}^{AP}$, which is the vector pointing from the center of mass of the robot to the point P, see figure 11 again.

The first step in the collision response model is to determine the collision point P. Of course, this is the point at which the edge of one robot, touches a vertex of the other robot. We have calculated this point in the previous section. Next the *collision normal* $\mathbf{n}$ has to be found, this vector is simply the outward pointing normal of the edge that contains the collision point.
To calculate the new linear velocities, a relative velocity $\mathbf{v}^{AB}$ is introduced. This velocity is simply the difference between $\mathbf{v}^{AP}$ and $\mathbf{v}^{BP}$:

$$\mathbf{v}^{AB} = \mathbf{v}^{AP} - \mathbf{v}^{BP} \tag{7}$$

The most important variable in a collision is the impulse factor. This factor is calculated with 'Newton's Law of Restitution'. An impulse can be seen as a very large (or even infinite) force applied in a very small period of time, giving the finite impulse. This causes an object to change its speed immediately as opposed to a non-instantaneous force.
The impulse factor depends on the velocities of the colliding objects, the collision normal, the masses of the objects and an elasticity factor e. This factor ranges from $e = 1$, a perfectly elastic collision where the objects bounce immediately, to $e = 0$, a perfectly inelastic collision where the objects will stick together. For a collision between two robots, e close to 0 seems to be a reasonable choice.
From 7 we can derive the *relative normal velocity*, which is the component of $\mathbf{v}^{AB}$ parallel to the collision normal $\mathbf{n}$:

$$\mathbf{v}^{AB} \cdot \mathbf{n} = (\mathbf{v}^{AP} - \mathbf{v}^{BP}) \cdot \mathbf{n} \tag{8}$$

For a collision it is needed that this relative velocity is negative, but we already know that a collision is going to happen from the collision detection method, so this check is not needed. This velocity combined with e is needed to determine the new outgoing velocity:

$$\mathbf{v}_2^{AB} \cdot \mathbf{n} = -e\mathbf{v}_1^{AB} \cdot \mathbf{n} \tag{9}$$

Note that the minus sign implies that the direction of the velocity vector will be reversed after the collision. The subscripted 1 and 2 denote the incoming and outgoing velocities respectively.
Because we make the simplifying assumption that there is no friction at the point of collision, the impulse felt by an object is entirely in the direction of the collision normal. With the help of Newton's Third Law, we can say that the impulse felt by object A equals $j\mathbf{n}$ and the impulse felt by object B is simply $-j\mathbf{n}$. Here, j is a scalar expressing the amount of the impulse in the direction of the collision normal $\mathbf{n}$.
The total outgoing velocity for point P on object A is (remember 6)

$$\mathbf{v}_2^{AP} = \mathbf{v}_2^{A} + \omega_2^{A}\mathbf{r}_{\perp}^{AP} \tag{10}$$

The two right hand terms of 10 can be written as :

$$\mathbf{v}_2^{A} = \mathbf{v}_1^{A} + \frac{j\mathbf{n}}{M_A} \tag{11}$$

$$\omega_2^{A} = \omega_1^{A} + \frac{\mathbf{r}_{\perp}^{AP} \cdot j\mathbf{n}}{\mathbf{I}^{A}} \tag{12}$$

For object B, the plus signs in both equations should be replaced with a minus. In equation 11 the term $j\mathbf{n}$ is a measure for the amount of momentum applied at the object. The division with its mass makes it a measure of the new velocity (because momentum can be written as $M\mathbf{v}$).
In equation 12, the division by the Moment of Inertia $\mathbf{I}^{A}$ is done to make the equation an angular velocity. For a rectangular object like our robot with length l and width w and assuming uniform mass distribution:

$$\mathbf{I}^{A} = \frac{1}{12}M_A(l^2 + w^2) \tag{13}$$

Of course, the main variable we are interested in is j. To solve j we first use equation 9. In this equation we substitute equation 8, obtaining:

$$(\mathbf{v}_2^{AP} - \mathbf{v}_2^{BP}) \cdot \mathbf{n} = -e\mathbf{v}_1^{AB} \cdot \mathbf{n} \tag{14}$$

Then we substitute equation 10 giving:

$$(\mathbf{v}_2^{A} + \omega_2^{A}\mathbf{r}_{\perp}^{AP} - \mathbf{v}_2^{B} + \omega_2^{B}\mathbf{r}_{\perp}^{BP}) \cdot \mathbf{n} = -e\mathbf{v}_1^{AB} \cdot \mathbf{n} \tag{15}$$

And finally with the substitution of equations 11 and 12 and some rewriting we get j as expressed in 16:

$$j = \frac{-(1+e)\mathbf{v}_1^{AB} \cdot \mathbf{n}}{\mathbf{n} \cdot \mathbf{n}(\frac{1}{M_A} + \frac{1}{M_B}) + \frac{(\mathbf{r}_\perp^{AP} \cdot \mathbf{n})^2}{\mathbf{I}^A} + \frac{(\mathbf{r}_\perp^{BP} \cdot \mathbf{n})^2}{\mathbf{I}^B}} \tag{16}$$

Note that all right hand variables are known at the time of the collision and with solving j all new velocities can be calculated.

VII. Evaluation

A. Overall performance

To resemble a real match, three scenarios were created. These scenarios are:

1) Three robots colliding near the center of a 5x5 field
2) Two robot colliding near the bottom of a 5x5 field
3) A collision on the side of a 7x7 field

Two types of fields are used to determine if the distance of a robot to the camera has any influence on the results. To be able to say something about the added value of the physical model and the overall correction method, all scenarios are executed three times: one with no collision correction, one with collision correction and one without the use of the physical model.

In table I the results of the overall performance test plan are presented. For each scenario a total of fifty runs is performed and the total number of swapped identifiers is counted and shown in the table. Of course, a running collision test is stopped if such a swap occurs. After the re-identification of the robots, the run is resumed.

	Case 1	Case 2	Case 3
Correction on	0	0	0
Physical model off	1	0	2
Correction off	8	2	14

TABLE I

Amounts of erroneous ID swapping. Each scenario is executed fifty times for each correction method.

The collision correction algorithm never swaps the robot IDs in this test, which is very nice! The influence of the additional physical model is small but certainly seems to help. We can state that the pre-collision correction accounts for the largest improvement.

B. Positional error

To give a more detailed evaluation of the algorithm, some other tests were performed where the distance of a prediction to its real position is compared with the distance of a corrected prediction to that position. We will call these distances the *positional errors* of the predictions. Because the real position is never known precisely, a position as observed by the vision system will be taken as an exact measurement. This idea is pictured in figure 12. In figure 13 the behaviour of a non-colliding robot is shown. The distance between the

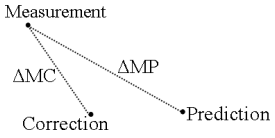


Fig. 12. Distances between a measurement, a prediction and a correction

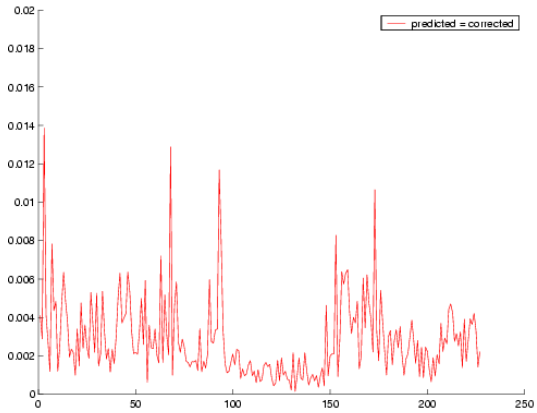


Fig. 13. Typical positional error of a moving non-colliding robot (frames $\times$ cm)

prediction and the measurement is quite large with peeks even above one centimeter. This is because an exact prediction is impossible due to estimated factors like the friction factor of the floor.

When looking at a robot involved in a collision, some time periods are especially interesting. The first period is some time before a collision will occur. At this moment, the positional error of the prediction will have some typical value for a moving robot. Also, because no collision occurs, the correction and prediction errors will be the same.
The second period is the time when a collision is detected and should be corrected. A good comparison can be made between the positional errors of the normal predictions and the corrected predictions. The corrected predictions should be significantly smaller than the uncorrected ones.
Furthermore, it is interesting to observe how a maximal error is responded to.
To test for the actual positional error, a total of fifty collision runs have been executed:

- 20 near the center of a 5x5 field
- 10 at the bottom of a 5x5 field
- 20 at the right side of a 7x7 field.

In table II the results are shown of twenty collisions near the center of the field. In this table, results are given for the average positional error ($mean$) and standard deviation(σ) of a prediction (ΔMP) and a correction (ΔMC).

The second and third rows show a relatively large positional error. It is clear that the positional error of the corrected

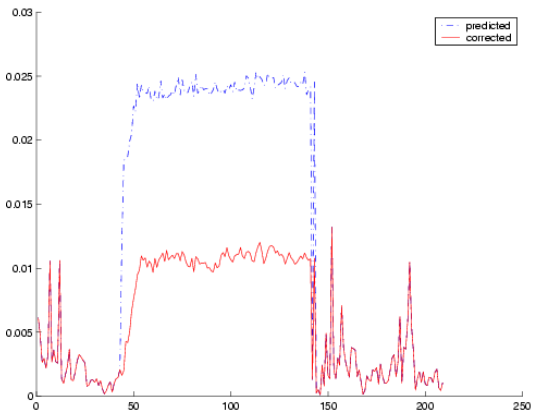


Fig. 14. Typical positional error of a colliding robot (frames × cm)

	ΔMP		ΔMC	
	Mean	σ	**Mean**	σ
Typical	0.1082	0.0634		
Peak in correction	2.2062	0.4492	1.0941	0.4259
Average correction	1.3107	0.8161	0.7228	0.4369

TABLE II

POSITIONAL ERRORS (IN CM) FOR 20 COLLISIONS AT THE CENTER OF THE 5X5 FIELD.

predictions is much smaller than the error of the uncorrected prediction. Also the peak in the prediction is significantly smaller as it is corrected. Because these peaks have the highest risk of causing an identifier swap, it is important that they are corrected well.

	ΔMP		ΔMC	
	Mean	σ	**Mean**	σ
Typical	0.2442	0.1629		
Peak in correction	2.6717	0.4673	1.1953	0.5042
Average correction	1.0247	0.7093	0.5705	0.3990

TABLE III

POSITIONAL ERRORS (IN CM) FOR 10 COLLISIONS AT THE BOTTOM OF THE 5X5 FIELD.

At the bottom of the field (table III), the average prediction error during a collision is smaller. The correction shows a significant decrease of this error. The correction of the peak error is quite good again.

The collisions at the right side of the 7x7 field are shown in table IV. Surprisingly, the error averaged over all corrections is smaller than in the previous cases.
The peak error shows a relatively large error. Fortunately, the collision correction method can bring this down to about 1.36cm, which creates a much smaller risk on an ID swap.

	ΔMP		ΔMC	
	Mean	σ	**Mean**	σ
Typical	0.2225	0.2349		
Peak in correction	3.6712	1.9378	1.3560	1.1750
Average correction	1.2310	1.0144	0.4977	0.3819

TABLE IV

POSITIONAL ERRORS (IN CM) FOR 20 COLLISIONS AT THE RIGHT SIDE OF THE 7X7 FIELD.

VIII. CONCLUSION

Collisions between robots cause swapping of identifiers of the team robots. These identifiers are assigned by the state estimator who estimates new states for all objects on the field. Because all robots look identical the robots are identified by means of tracking. Tracking is done by determining corresponding predictions and measurements. These measurements are delivered by the vision system. The predictions are calculated with an Extended Kalman Filter. This filter has the previous state and the control signals of a robot as its input. External objects like other robots or walls are not considered in the calculation. Therefor, after the prediction has been made it has to be corrected for these external objects.

To detect a collision in the state estimator, the Separating Axis Theorem is used. This algorithm searches for a separating plane between two robots. If it is not found, a collision is detected.
The next step is to correct the impossible prediction. This is done in two steps. First, the robots are put back towards their original positions until they only touch each other. This is performed by an analysis of the mutual collision areas. With this method the exact positions of the robots just before the collision can be calculated.
Afterwards, a physical collision response model is used to determine the new linear and angular velocities, and consequently the new positions. It is based on finding the impulses for both objects by investigating the various forces applied in such a situation.

The evaluation of the correction for collisions between robots showed that almost no identifier swapping occurred anymore. The influence of the additional physical model was helpful, but the pre-collision correction accounts for the biggest difference. An examination of the positional errors shows a big difference between the corrected and uncorrected predictions. Furthermore, the peaks in the prediction errors where decreased significantly. This is important because only one large peak can cause an identifier swap.

REFERENCES

[1] N. Kooij, “The development of a vision system for robotic soccer,” Master’s thesis, University of Twente, 2003.
[2] E. M. Schepers, “Improving the vision of a robot soccer team,” Master’s thesis, University of Twente, 2004.

[3] S. Grond, “State estimation of colliding objects in a robotsoccer environment,” Master’s thesis, University of Twente, 2005.
[4] D. S. Gottschalk, M. C. Lin, “OBBTree: A Hierarchical Structure for Rapid Interference Detection,” 1999.
[5] A. L. Schoute and P. J. Bouwens, “Deadlock-free traffic control with geometrical critical sections,” SION Conference Computing Science. Stichting Mathematisch Centrum, 1999.
[6] C. Hecker, “Physics,” *Game Developer Magazine*, 1996-1997.