TWO APPROACHES TO MOBILE ROBOTS SIMULATOR DESIGN

Gregor Klančar[1], Mišel Brezak[2], Ivan Petrović[2], Drago Matko[1]

[1]University of Ljubljana, Faculty of Electrical Engineering,
1000 Ljubljana, Tržaška 25, Slovenia
[2]University of Zagreb, Faculty of Electrical Engineering and Computing,
Unska 3, Zagreb, Croatia

gregor.klancar@fe.uni-lj.si (Gregor Klančar)

Abstract

This paper presents the design of a mobile robots simulator which is applicable for robot soccer or for general use. The simulator consists of a group of mobile robots, the ball and includes knowledge of their dynamic behavior modeling, collisions modeling and visualization. Two different approaches to the design of this simulator are given and compared. By the first approach the simulator physics background was completely developed by our team, which enabled us to get a better insight into the problem domain and gave us the possibility to efficiently solve some simulator specifics. First the model of ball and robot motion was derived and then complex approximate collisions models, where the real robot shape is taken into consideration. Some new ideas of collision formulation, realization and real robot shape inclusion are used. By the second approach the simulator was developed using freely available physics engine ODE – Open Dynamics Engine. This engine already includes physical background for rigid bodies dynamic; it is up to the user to define mechanical and physical parameters of the objects to simulate e.g. dimensions, masses, friction, joints and the like. The implementation of both simulators are described and a comparisons of their reality description, computational efficiency, effort and knowledge needed to built the simulator, advantages and disadvantages are given.

Keywords: Robot Simulator, Modeling, Collision Detection, Physics Engine.

Presenting Author's biography

Gregor Klančar. He received his B.Sc. and Ph.D. degrees in 1999 and 2003 from the Faculty of Electrical Engineering of the University of Ljubljana, Slovenia, where he is currently employed as a researcher. His research interests are in the area of fault diagnosis methods, multiple vehicle coordinated control and mobile robotics.

1 Introduction

In this paper two approaches to the design of simulator for multi-agent group of robots are described and compared: (i) classic approach, where the simulation is based on a derived model of dynamic behavior and collisions, and (ii) use of freely available physics engine ODE - Open Dynamics Engine [1]. Generally, the classic approach has an advantage of better insight into the problematic and gives the possibility for various customizations if required, while physics engine based approach eases simulator development procedure, but offers less flexibility.

The main purpose of the simulator design procedure is to obtain a realistic simulator which would be used as a tool in the process of strategy and control algorithms design for real world robot soccer as well as for other mobile-robotics related topics. To assure transferability to the real system the obtained strategy algorithms have to be designed on a realistic simulator.

The main motivation for robot soccer simulator development was to design and study multi-agent control and strategy algorithms in FIRA Middle or Large League MiroSot category (5 against 5 or 11 against 11 robots). However, on FIRA's (Federation of International Robot Soccer Association) official website (www.fira.net) there exists a simulator for SimuroSot league, which could only be used in Middle League MiroSot (5 against 5 robots).

A similar simulator was built by [2] where robot motion is simulated by dynamic model, collisions remaining oversimplified. There also exist a number of other simulator applications [3] but not many papers are available. Other problem is specificity of each simulator which implies that many available simulators are not usable in our robot soccer domain and vice versa. An important part of every realistic robot soccer simulator is collision modeling and simulation. Good mathematical background in rigid body collisions modeling and simulation could be found in [4]. Another useful contribution in the field of robotic simulator is [5] where collisions are treated by spring-dumper approach rather than by impulse force only.

For the first simulator design approach, some vital parts of the simulator are explained and modeled in more detail, beginning with the kinematics and dynamic motion modeling considering kinematics constraints and, further on dealing with different collisions modeling. The stress is given to the motion modeling where the assumptions of pure rolling conditions are made and dynamic properties are included. The results of this part are motion models of the ball and the robot with differential drive. Some new ideas of collision formulation and realization (taking into account the real robot shape) are used as well. Collisions are simply solved by mathematically correct discontinuous change of velocities (states of the velocity integrators), which is more convenient for realization than simulating collisions by applying impulse force [4, 5]. However, collisions are only described by approximate models, which are sufficient enough for realistic behavior of the obtained simulator. Precise collision modeling is usually very demanding because of many factors, which should be considered during collision. When simulating a realistic game a precise collision modeling is less important than motion modeling. This is because the game strategy is designed to play a good game where different collisions are undesired and we want to avoid them. Nevertheless collisions still happen and have to be handled. The problems of collision detection and the method of finding the exact time of the collision are exposed too. For the latter the zero crossing algorithms from Matlab Simulink are used. The simulator code of motion and collision models is implemented in C++ programming environment.

With a rapid progress of computer graphics used in computer games, animated movies and other purposes a number of physics engines have appeared which can realistically simulate rigid body dynamics considering variables such as mass, inertia, velocity, friction, etc. Some of available physics engines are ODE – Open Dynamics Engine, Ageia physX, AERO, Karma in Unreal Engine and many others. Their usage enables computer simulations, animations and games such as racing games to appear more realistic. Depending on their usage there exist two types of physics engines, namely real-time and high precision. When dealing with interactive computing (e. g. video games), the physics engines are simplified in order to perform in real-time. On the other hand high precision physics engines require more processing power to be able to calculate very precise physics and are usually used by scientists and computer animated movies. Some of physics engines are free and open source. As such they can also be used to simulate physics in different research oriented experiments. These packages are usually comprehensive and therefore quite difficult to manage, use and modify. To find out what advantages can be obtained by using such physical engines, we have also conducted the second approach to simulator design by using ODE physics engine.

The paper is organized as follows. In section 2 a brief system overview is revealed, followed by the mathematical model derivation of basic agents (robots and ball) in section 3. Then some new ideas of collisions modeling considering complex robot shape are presented in more detail in section 4. In section 5 simulator development using ODE physics engine is described, and in section 6 both simulator implementations are compared. The paper ends with conclusions and some ideas for future work.

2 System overview

The robot soccer set-up (see Fig. 1) consists of ten Middle League MiroSot category robots (generating two teams) of size 7.5*cm* cubed, orange golf ball, rectangular playground of size 2.2×1.8*m*, color camera and personal computer. Color camera is mounted above playground (each team has its own) and is used as a global motion sensor. The objects are identified from their color information; orange ball and color dresses of robots. The agent-based control part of the program calculates commands for each agent (robot) and sends them to the robot by a radio connection. The robots are then driven by two powerful DC motors; one for each wheel.

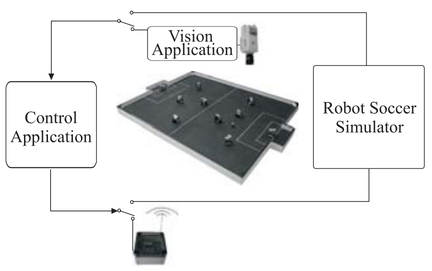


Fig. 1 Robot soccer system overview

The role of the simulator developed in the paper is to replace the real playground, camera, robots and ball, which are expensive and need a large place to be set up. Therefore the simulator must include mathematical models of motion as well as collisions which happen on the playground.

3 Mathematical modeling

To simulate robot soccer game first mathematic motion equations of the moving objects should be derived. The playground activities consist of two kinds of moving objects: robot and ball. Therefore their motion modeling [6] is presented in the sequel.

3.1 Robot Model

The kinematics and dynamic motion equations for mobile robot in Fig. 2 are derived.

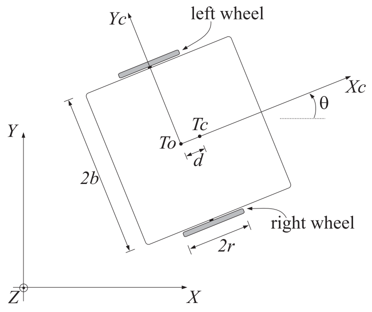


Fig. 2 Mobile robot symbol description

Where $T_o=(x_o,\ y_o)$ is robot geometric centre and $T_c=(x_c,\ y_c)$ is its mass centre. Supposing pure rolling conditions of the wheels the following kinematic constraints can be written:

$$\begin{aligned} \dot{y}_c\cos\theta-\dot{x}_c\sin\theta-\dot{\theta}d&=0\\ \dot{x}_c\cos\theta+\dot{y}_c\sin\theta+b\dot{\theta}&=r\dot{\phi}_r\\ \dot{x}_c\cos\theta+\dot{y}_c\sin\theta-b\dot{\theta}&=r\dot{\phi}_l \end{aligned} \tag{1}$$

Where θ is robot orientation, ϕ_r and ϕ_l are wheels angles and d is distance between mass centre and geometric centre. According to the first constraint in equation (1) robot cannot slide in the sideways while the second and the third constraints describe pure rolling of the wheels. The null space of kinematic constraints (1) defines the robot kinematics motion equation given as

$$\begin{bmatrix}\dot{x}_c\\ \dot{y}_c\\ \dot{\theta}\\ \dot{\phi}_r\\ \dot{\phi}_l\end{bmatrix}=\begin{bmatrix}\frac{r}{2b}\left(b\cos(\theta)-d\sin(\theta)\right) & \frac{r}{2b}\left(b\cos(\theta)+d\sin(\theta)\right)\\ \frac{r}{2b}\left(b\sin(\theta)+d\cos(\theta)\right) & \frac{r}{2b}\left(b\sin(\theta)-d\cos(\theta)\right)\\ \frac{r}{2b} & -\frac{r}{2b}\\ 1 & 0\\ 0 & 1\end{bmatrix}\cdot\begin{bmatrix}\dot{\phi}_r\\ \dot{\phi}_l\end{bmatrix} \tag{2}$$

Dynamics motion equations are derived using Lagrange formulation

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_k}\right)-\frac{\partial L}{\partial q_k}+\frac{\partial P}{\partial \dot{q}_k}=f_k-\sum_{j=1}^{m}\lambda_j a_{jk} \tag{3}$$

where L is Lagrangian, P is power (dissipation) function, q_k are generalized coordinates, $f(t)$ is external force and λ_j are Lagrange multiplicator associated with j-th (j=1…3) constraint equation and a_{jk} is k-th (k=1…5) coefficient of j-th constraint equation. Lagrangian is defined by

$$\begin{aligned} L=\frac{m}{2}\left(\dot{x}_c^{\ 2}+\dot{y}_c^{\ 2}\right)+\frac{J}{2}\dot{\theta}^2+\frac{J_k}{2}\dot{\phi}_r^{\ 2}+\frac{J_k}{2}\dot{\phi}_l^{\ 2}+\\ +2m_k d\dot{\theta}\left(\dot{x}_c\sin\theta-\dot{y}_c\cos\theta\right) \end{aligned} \tag{4}$$

were $m=m_c+2m_k$, $J=J_c+2J_m+2m_k(d^2+b^2)$, m_c is body mass, m_k is wheel mass and J_c, J_k, J_m are moments of inertia for robot body around axis Z, for wheel around its axle and wheel around axis Z, respectively.

According to (3) the dynamic model is written as

$$\begin{aligned} m\ddot{x}_c+2m_k d\left(\ddot{\theta}\sin\theta+\dot{\theta}^2\cos\theta\right)-\lambda_1\sin\theta+\left(\lambda_2+\lambda_3\right)\cos\theta&=0\\ m\ddot{y}_c-2m_k d\left(\ddot{\theta}\cos\theta-\dot{\theta}^2\sin\theta\right)+\lambda_1\cos\theta+\left(\lambda_2+\lambda_3\right)\sin\theta&=0\\ J\ddot{\theta}+2m_k d\left(\ddot{x}_c\sin\theta-\ddot{y}_c\cos\theta\right)-\lambda_1 d+\left(\lambda_2-\lambda_3\right)b&=0\\ J_k\ddot{\phi}_r+\mu\dot{\phi}_r-\lambda_2 r&=\tau_r\\ J_k\ddot{\phi}_l+\mu\dot{\phi}_l-\lambda_3 r&=\tau_l \end{aligned} \tag{5}$$

where λ_1, λ_2, λ_3 are Lagrange multiplicators which can effectively be eliminated by the procedure given in [7, 8]. The dynamics of electric part (the motors) can usually be neglected, as electrical time constant are usually significantly smaller than mechanical time constants.

3.2 Ball Model

General motion of the ball on a plane can be described by five generalized coordinates as shown in Fig. 3.

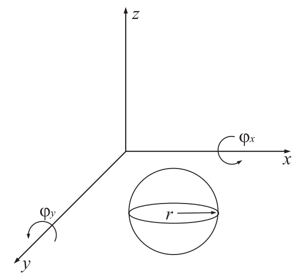


Fig. 3 The ball rolling on the plane

Dynamics motion equation can be derived using Lagrange formulation (3) where Lagrangian is defined as

$$L = \frac{1}{2}m(\dot{x}^2 + \dot{y}^2) + \frac{1}{2}J(\dot{\varphi}_x{}^2 + \dot{\varphi}_y{}^2 + \dot{\varphi}_z{}^2) \qquad (6)$$

where m is the ball mass and J is moment of inertia. Supposing pure rolling condition of the ball, the following relations $\dot{x} + r\dot{\varphi}_y = 0$, $\dot{y} - r\dot{\varphi}_x = 0$ hold, where r is ball radius. These relations are holonomic (integrable) constraints and are used to eliminate two generalized coordinates in (6). Further on by neglecting rotation around z axis $\omega_z = 0$, Lagrangian is rewritten as

$$L = \frac{m + \frac{J}{r^2}}{2}\left(\dot{x}^2 + \dot{y}^2\right) \qquad (7)$$

The power function is

$$P = \frac{1}{2}f_D\dot{x}^2 + \frac{1}{2}f_D\dot{y}^2 \qquad (8)$$

where f_D is dumping coefficient. Final motion equation of the ball are as follows

$$\ddot{x} = \frac{F(t) - \dot{x}\cdot f_D}{m + J/r^2}$$
$$\ddot{y} = \frac{F(t) - \dot{y}\cdot f_D}{m + J/r^2} \qquad (9)$$

4 Collision modeling

During the motion of the objects on the playground several collisions between them are possible. However in the sequel only a brief introduction is given a more detailed description of collision modeling can be found in [9]. The latter are given as submodels and describe the collision between moving objects: the robot-ball collision model, the robot-boundary collision model, the ball-boundary collision model and the collision between robots model.

4.1 Ball-Boundary Collision

In the collision between ball and boundary elastic collision is supposed where the tangential velocity component to the boundary remains the same while the normal velocity component changes sign and is multiplied by a factor less than one representing energy loss.

4.2 Robot-Ball Collision

Similar procedure as in ball-boundary collision is followed in robot-ball collision. Also actual robot shape is considered which is demonstrated in Fig. 4.

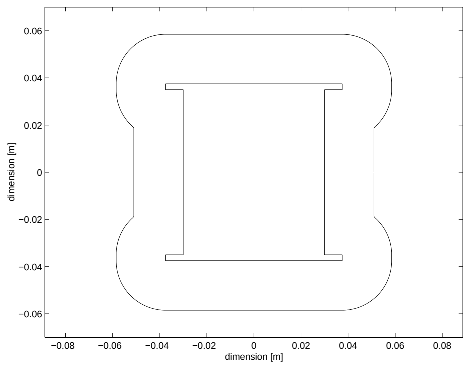


Fig. 4 Robot shape and its outer shape

Outer shape of the robot is obtained by recording ball centre while it rounds the robot. The collision is then treated as point collision where robot tangential and normal velocities on outer robot shape in point of collision are determined by $v_{x_1} = v - \omega r(\varphi)\sin\varphi$, $v_{y_1} = \omega r(\varphi)\cos\varphi$. Where v is linear velocity and ω is angular velocity of robot centre, $r(\varphi)$ is the distance from the robot centre to the collision point and φ is the angle from the local robot axis x to the line connecting the robot centre and the collision point.

The playground coordinate system is rotated so that axis x is in tangential direction of the outer shape of the robot (in the point of collision).

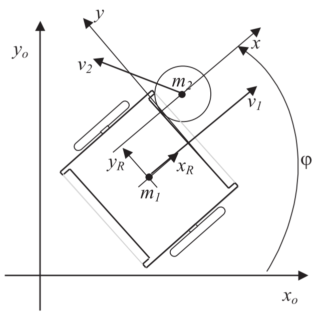


Fig. 5. Collision of two spheres

Mathematically the collision model is based on kinetic energy and momentum balance equations for two spheres as follows

$$
\begin{aligned}
w_{x_1} &= \frac{-m_2 v_{x_1} + m_1 v_{x_1} + 2m_2 v_{x_2}}{m_1 + m_2} \\
w_{x_2} &= \frac{2m_1 v_{x_1} + m_2 v_{x_2} - m_1 v_{x_2}}{m_1 + m_2} \\
w_{y_1} &= v_{y_1} \\
w_{y_2} &= v_{y_2}
\end{aligned}
\tag{10}
$$

where indexes 1 and 2 stand for the robot and ball, v represents the velocities before and w the velocities after the collision, while m_1 is robot and m_2 ball mass respectively.

4.3 Robot-Boundary Collision

During the game such collisions are undesired therefore a motion algorithm includes also obstacle avoidance capabilities. If in the simulated game such collisions happen they are treated by a very simple model for realization in the simulator code which describes collision only approximately. It is therefore not intention to capture exact collision but just to have a simple solution to handle such collision situations.

If robot hit the boundary with two or more corners its motion is stopped as long as the robot tries to advance in the boundary. At one corner collision with the boundary the robot starts to rotate around that corner if the approaching angle between the robot and boundary is big (more than 15°) otherwise it slides along the boundary.

4.4 Collisions Between Robots

Among mentioned collisions in robot soccer game the most challenging one is collision between robots. Robots velocities after collision are obtained from known force impulse which happened in the collision. The detailed procedure to estimate velocities of two rigid bodies after collision is described in [4, 9]. The idea is to calculate relative velocities in the point of collision before and after collision in normal direction. It is always true that absolute value of relative velocity in normal direction after collision remains the same comparing to absolute value of relative velocity in normal direction before collision in that point. From that property the amplitude of force impulse can be calculated. Once having estimated impulse linear and angular velocity after collision can be calculated. Linear velocity $\vec{v}^+$ and angular velocity $\vec{\omega}^+$ for robot mass centre after collision can be calculated by using relations:

$$
\begin{aligned}
\vec{v}^+(t_0) &= \vec{v}^-(t_0) + \frac{\vec{J}(t_0)}{M} \\
\vec{\omega}^+(t_0) &= \vec{\omega}^-(t_0) + I^{-1}\left(\vec{r} \times \vec{J}(t_0)\right)
\end{aligned}
\tag{11}
$$

where t_0 is time of the collision, M is mass of the robot, I is corresponding moment of inertia, J is force impulse because of collision and $\vec{r}$ is a displacement vector between mass centre $\vec{x}$ and point of collision $\vec{p}$ (see Fig. 10), while the sign in the superscript denotes time instant of the collision (- before and + after collision).

5 Use of Open Dynamics Engine

In the second approach freely available physics engine for rigid body dynamics simulation - ODE (Open Dynamics Engine) was used. We have chosen ODE because it is free, platform independent, with an easy to use C/C++ API, it has integrated collision detection, supports simulation of friction, and lots of documentation is available.

ODE can be used for simulation of wheeled robots, as well as legged robots, and is particularly suitable for real time simulation of moving objects in virtual environment. Simulation in ODE is based on rigid body dynamics, where each body can have arbitrary mass distribution and is described with its position and orientation in 3D space, linear and angular velocity, mass, position of the center of mass, and inertia matrix. Bodies are connected to each other through different kinds of joints, which include hinge, slider, ball, fixed, contact etc. ODE also has integrated collision detection engine, which based on given information about shape of each body, can identify the bodies that touch each other and passes the resulting contact points to the user which can then create contact joints between bodies that touch. Body's shape has no influence on body dynamics; it only affects the collision detection. ODE uses a highly stable integrator, so that simulation error cannot accumulate in the way that the simulation becomes unstable, but accuracy can be achieved only if step size is small, so that we use the minimum step size so that real time simulation is still possible.

Here we will give a short description of mobile robots simulation procedure based on ODE. Procedure consists of two stages: initialization and simulation loop. In initialization stage first an object that acts as container for rigid bodies and joints is created, which is called dynamics world. Then also a collision world is created which is container for geometric objects that represent shape of simulated bodies and thus enable collision detection. For each object in simulation, such as robot, ball or playground, the corresponding rigid bodies as well as geometric objects that represent simulated objects are created and added to the dynamics and collision world, respectively. For example a robot consists of three rigid bodies and corresponding geometry objects: robot's chassis, which is represented by box rigid body and two drive wheels, which are represented by cylinder rigid bodies and geometry objects. Here the impact of the castor wheels on robot dynamics can be neglected because of their small mass, and only the dynamics of contact between castor wheels and ground is important for simulation. Therefore the castor wheels are not modeled with corresponding dynamic bodies but only with cylinder geometric objects that have the role of detecting of contact with the ground so that contact dynamics can be simulated, which enables maintaining the robot balance. In this way, under normal conditions (e.g. the robot is not overturned), in each time instance both drive wheels and only one (or none) of the castor wheels have contact with the ground. To achieve realistic collision behavior, additional geometric objects are added to robot, which model the actual robot shape shown in Fig. 4. Then for

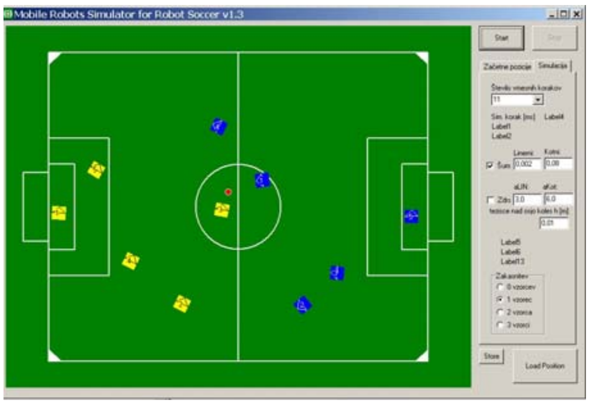


Fig. 6. Screen shot of the first simulator (based on classical approach)

each single body and corresponding geometric object, its parameters (dimensions, initial position and orientation, mass etc.) are set. Initialization phase ends with creation of joint objects in dynamics world. In our case, only hinge joints are required, where a hinge connects each drive wheel with robot chassis.

In simulation loop stage, in each simulation step forces are applied to the drive wheels, according to velocity commands sent from strategy module. Then collision detection engine is invoked, and for every detected collision point a contact joint is created. Then the parameters of the contact joint (such as friction coefficient, restitution parameter etc.) are set for each contact joint and joint is added to contact joint group. The friction coefficient for contact joint between drive wheel and the ground has high value because drive wheels are coated with high grip tires. On the opposite, the contact joint between castor wheel and the ground has low friction coefficient value because castor wheels are made out of smooth metal. Finally, simulation step is taken, where new states for each object are computed.

6 Comparison of Two Approaches

A comparison of two described simulator design approaches is given. The first simulator approach uses the classical way for deriving kinematics, motion and collision equations while the second approach uses physics engine ODE to obtain simulator core. Both simulators GUI's are shown in Fig. 6 and Fig. 7.

Collision simulation is more challenging than motion simulation without collision. To compare behavior of both simulators in such conditions, results of three simulation experiments are given in the sequel. Finally, to find out how close the simulators describe the real world situations, results of correspondent experiments with real robots and ball are presented as well. The values of all relevant variables in experiments are sampled using sample time of 33.3 ms, and results of all experiments are graphically presented using only every fifth sample (i.e. with 165 ms resolution). Additionally, the intern sample time that was used by ODE engine in second simulator was 1 ms.

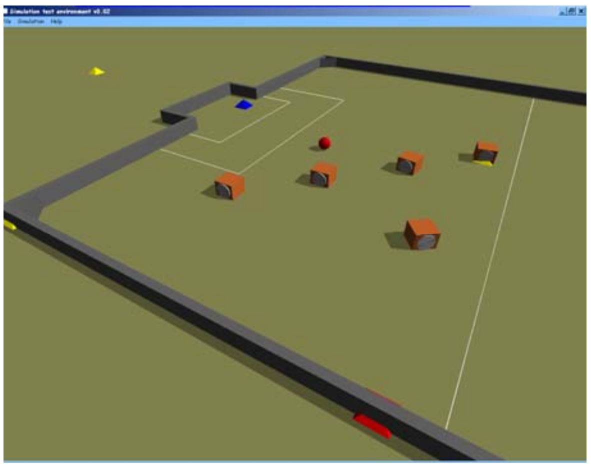
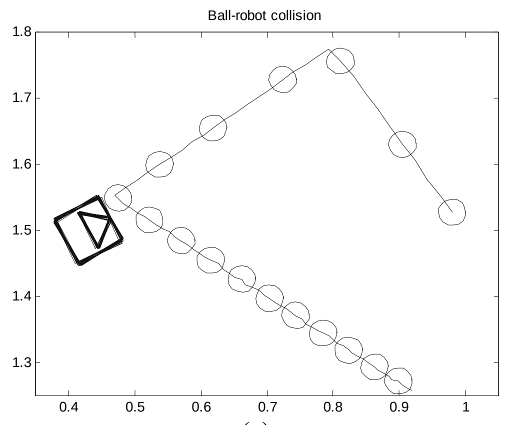
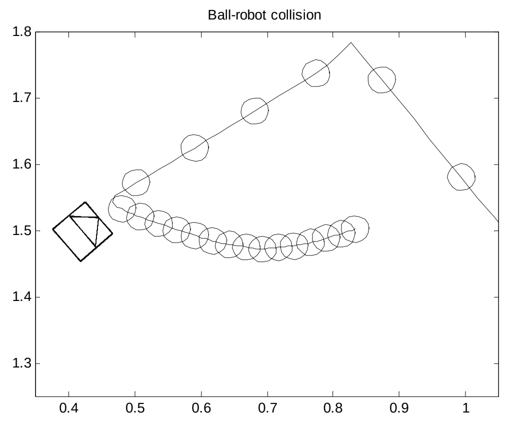


Fig. 7. Screen shot of the second (ODE-based) simulator

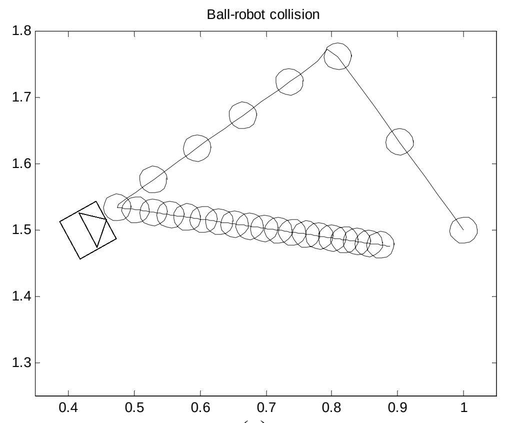
In the first experiment the ball starts to move towards the playground boundary with initial velocity *v*=1m/s, and then it collides with the boundary and finally with the non-moving robot. The behavior of both simulators in this experiment can be visually compared by observing different scenarios in Fig. 8. It can be seen that both simulators show similar results and consider the actual robot shape shown in Fig. 4, except that the second simulator produces somewhat lower ball velocity after collision and slightly different final ball direction. Besides that, if the experiment is observed in 3D viewer, the second simulator shows a small ball jump after collision with boundary or robot. This is more realistic, because in the real world, on higher velocities ball rotation causes the jump in the case of ball collision with the boundary, which is also verified by the real world experiment shown in Fig. 8 (c). In this experiment initial state of the ball was slightly different than the one used in simulations, because in real world it is difficult to reproduce exact ball initial state. It can be concluded that results of both simulators are close to real world behavior, except that in real world the ball no more has straight line trajectory on low velocities, which is caused by non ideally flat ground plane, but of course this would be difficult to model in simulator.



(a)



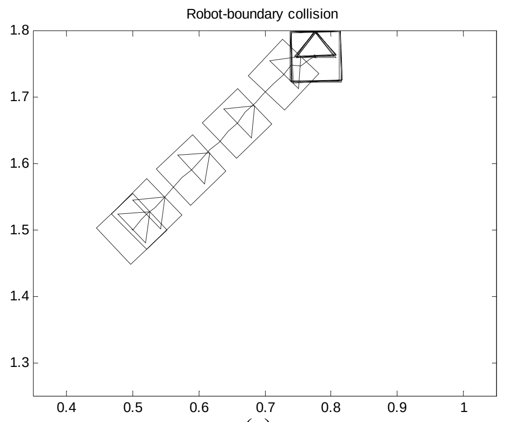
(b)



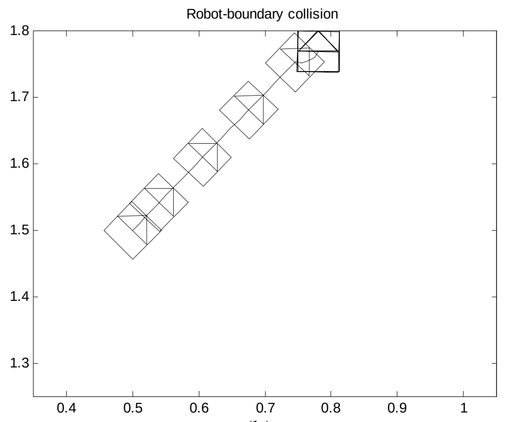
(c)

Fig. 8. Ball-robot collision experiment (a) first simulator (b) second simulator (c) real world

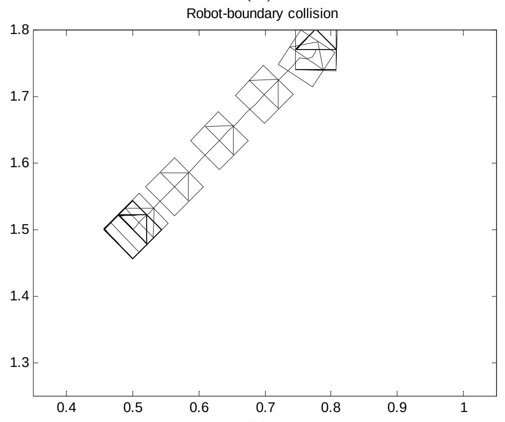
In the second experiment the robot starts to move towards the boundary at the 45° angle relative to the boundary while the motors of both wheels are controlled to produce the torque in order to maintain angular velocity of the wheels that is correspondent to robot linear velocity of 0.6 m/s. The results of this experiment can be seen in Fig. 9. It can be observed that both simulators, as well as real world robot, show almost identical behavior as in every case after the collision robot ends stuck in the boundary. But there is one very important difference between first and second simulator; in the case of the second simulator, by viewing the simulation in 3D viewer one can observe that after the robot finishes stuck on the boundary, the wheels of the robot are still rotating although the robot is not moving, which means that wheels are sliding. This is closer to reality, because in real world experiment the wheel sliding is also present. The simulation of this effect in the case of second simulator is possible because ODE engine is capable of simulating the friction effects.



(a)



(b)



(c)

Fig. 9. Robot-boundary collision experiment (a) first simulator (b) second simulator (c) real world

Finally, in the third experiment two robots with starting positions $R_1 = (0.5, 1)$ m and $R_2 = (0.8, 1.05)$ m start to move towards each other. During the whole experiment wheel motors of both robots are controlled to maintain linear velocity of 0.6 m/s. The results of this experiment are shown in Fig. 10, where the traces of the first and second robot are marked with solid and dashed lines, respectively.

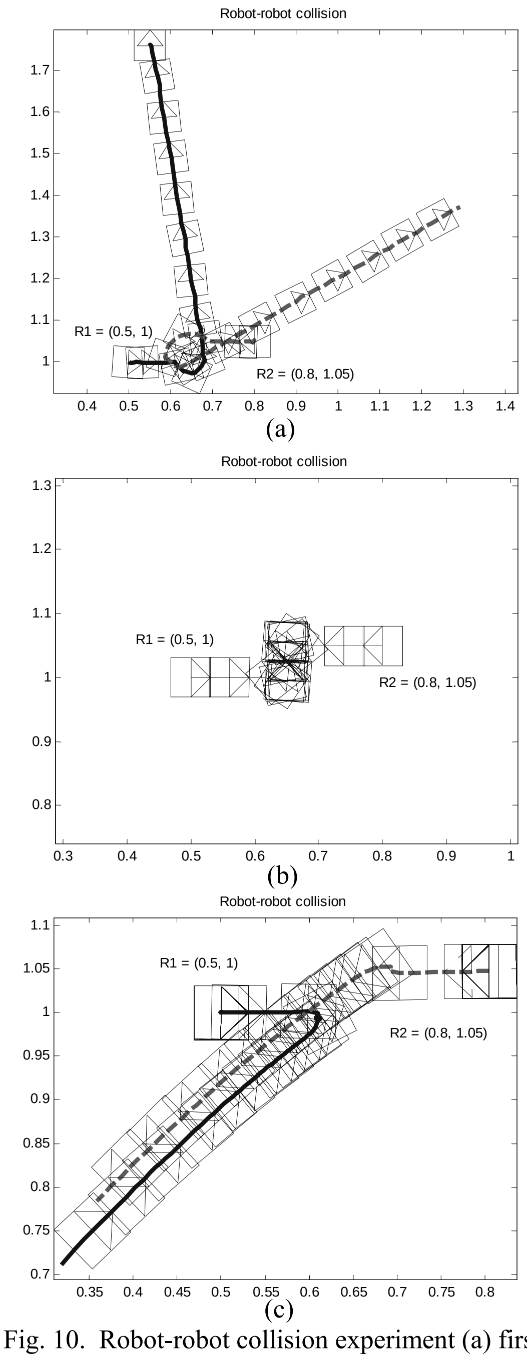


Fig. 10. Robot-robot collision experiment (a) first simulator (b) second simulator (c) real world

In this experiment the differences between two simulators are most notable, as in case of the first simulator after the collision robots split up and each robot continues to move in its own direction. But in case of second simulator, after the collision robots remain stuck on each other, while the wheels are sliding along the ground plane. The experiment with real robots shows that after the collision robots also remain stuck on each other while the wheels are sliding, but because the grip is different for each wheel, in most cases the pushing force of one robot takes over the other's push force and both robots together start to move slowly in an unpredictable direction. This shows that none of the simulators predicts precisely the behavior of the real robots. Nevertheless, the second simulator is here also closer to reality.

It can be concluded that both simulators give sufficient representation of reality, although the second simulator provides more realistic model because it can predict real world effects such as wheel sliding. In cases when detection of wheel sliding is important, as can be the case when designing and testing the robot soccer strategy, it could be more convenient to use the second simulator approach.

To summarize, we can emphasize advantages and disadvantages of each simulator. Advantages of ODE based physics engine based simulator are: already developed physical models of usually complex systems, stable integration methods, the user need less theoretical background, there is less possibilities for modeling mistakes, wrong assumptions or unjustified simplifications, it enables general usage, has verified operation and quite optimized performance. Also, the ODE greatly simplifies simulation of objects (e.g. robots) with complex shape, because all that has to be done is to approximate complex object shape with appropriate geometric objects, which would be very difficult to accomplish using classical approach.

On the other hand, the ODE package is comprehensive and therefore quite difficult to modify if required. In the case of very complex models, it could also be difficult to achieve sufficient performance. ODE uses various approximations to achieve better performance, which can result with insufficient precision for some simulation tasks. Therefore, in same cases it can be better or even mandatory to derive own physical models. This benefits with better insight to system inner characteristics and gives the possibility for various customizations and the simulation code can be made more efficient.

Besides robot soccer practical use of such simulators can be in non-prehensile object manipulations studies (without grasping), study of interaction of multiple agents and the like.

7 Conclusion

In this paper we describe design and implementation of mobile robots simulator designed for robot soccer or for general use. Two different approaches for simulator design are presented and compared. In the first approach the complete simulator physics background was developed by our team, and in the second approach we used freely available physics engine ODE.

It can be concluded that use of physics engines saves the cumbersome task of deriving mathematical model of complex systems, and depending on physics engine used, can also provide some additional features that would be otherwise hard to model. For example, ODE physics engine enables us to simulate friction, which means that unrealistic assumption of pure rolling condition can be avoided and situations when robot slip occur can be simulated. On the other hand, derivation of own model gives better insight to system inner characteristics and gives the possibility for various customizations if required.

Acknowledgement

This research was jointly supported by the Slovenian Research Agency of the Republic of Slovenia (grant No. BIHR 07-08-008) and the Ministry of Science, Education and Sports of the Republic of Croatia (grant No. 036-0363078-3018).

8 References

[1] R. Smith, Open Dynamics Engine, http://www.ode.org/

[2] T. C. Liang, J.S. Liu, A Distributed Mobile Robot Simulator and a Ball Passing Strategy, Technical Report TR-IIS-02-007, Institute of Information Science, Academia Sinica, Nankang, Taiwan, 2002.

[3] S. Moss, P. Davidsson, Multi-Agent-Based Simulation, Springer-Verlag, New York, 2002.

[4] D. Baraf, An Introduction to Physically Based Modeling: Rigid Body Simulation II – Nonpenetration Constraints, in: SIGGRAPH '97 Course notes, Carnegie Mellon University, 1997.

[5] E. Larsen, A Robot Soccer Simulator: A Case Study for Rigid Body Contact, Sony Computer Entertainment America R&D, March 2001.

[6] O. Egeland, J.T. Gravdahl, Modeling and Simulation for Automatic Control, Marine Cybernetics, Trondheim, Norway, 2002.

[7] G. Oriolo, A. Luca, M. Vandittelli, WMR Control Via Dynamic Feedback Linearization: Design, Implementation, and Experimental Validation, IEEE Transactions on Control Systems Technology, 10 (6) (2002) 835-852.

[8] N. Sarkar, X. Yun, V. Kumar, Control of mechanical systems with rolling constraints: Application to dynamic control of mobile robot, The International Journal of Robotic Research, 13 (1) (1994) 55-69.

[9] G. Klančar, M. Lepetič, R. Karba, B. Zupančič, Robot soccer collision modelling and validation in multi-agent simulator, Mathematical and computer modelling of dynamical systems, 9(2) (2003) 137-150.