Real-time Implementation Of Dice Unloading Algorithm

Artem Vassilyev
Department of Pervasive Computing
Tampere University of Technology
Tampere, Finland
Email: artem.vassilyev@tut.fi

Jussi Parviainen
Department of Pervasive Computing
Tampere University of Technology
Tampere, Finland
Email: jussi.parvianen@tut.fi

Jussi Collin
Department of Pervasive Computing
Tampere University of Technology
Tampere, Finland
Email: jussi.collin@tut.fi

Jarmo Takala
Department of Pervasive Computing
Tampere University of Technology
Tampere, Finland
Email: jarmo.takala@tut.fi

***Abstract*—This paper shows the potential of using a physical model of gambling die as an input device for multimedia applications. We present a real-rime method for simulating fair 6-sided die on the base of loaded model of foam-rubber die equipped with inertial measurement unit. The method allows avoiding the negative influence that biased die produces on the final game result. The technique is based on filtering out duplicated rolls and building a fair result by accumulating 3 single, unique rolls. The measurements were done using inertial measurement unit that transmits data to the mobile device with Android application. The application processes the data and shows final result on the screen in real time. Both theoretical and practical sides of the implementation, including the set of experiments, are shown in this paper. A full technical description of method and technology that was used during the implementation is also presented.**

I. Introduction

Today game industry produces many different kinds of multimedia controllers that allow player to manage game process and naturally interact with media: joysticks, keyboards, remote controllers, VR helmets etc. This paper aims to show the new possibilities of using analog random number generator, so-called gambling die used in non-gambling tabletop games. Nowadays, all the digital tabletop games use digital random generator to simulate the die. We suggest to use normal 6-side plastic die equipped with several sensors for state classification and game control. Using of real model will make the game process more interactive and also can decrease the eyestrain. It assumed to use different dice with different physical parameters for every player to provide full compatibility with different dice manufacturers including custom-made dice.

Dice recognition techniques have been researched before by several authors [1], [2], [3], most solutions are based on using the camera and pattern recognition algorithms and, according to author's knowledge, inertial sensors were not yet considered. From the practical point of view using the camera imposes restrictions on the minimum size of die, area lighting and requires additional action from user since the camera vision angle is quite limited and resolution properties are limited too, so playable area is restricted by camera properties. In [4] authors show that different colors of dice can also affect on accuracy of detection; during experiments they found that dice that are less contrast are less amenable to recognition by 2% comparing with more contrast dice. From recognition point of view, method described in this article is more beneficial since it guarantees 100% recognition rate on flat surface.

Our initial idea consist of taking the attitude measurements using accelerometers and gyroscopes and send the result via existing transmission media to a game device(mobile phone, TV, etc.). The size of modern MEMS(Microelectromechanical systems) sensors allows to place it in the die of any shape and size. In this paper we propose to use IMU(Inertial Measurement Unit) that includes both types of sensor and able to communicate with other devices. The most important problem in this case is presented by bias of the die that will affect on game result during the long-term period. By default, every die can be considered as loaded die due to its physical imperfection. Only the ideal physical model of die with equal result probability can be considered to be totally unbiased, in another case any die has some physical issues that provides higher probability for one of the sides. Moreover, using of intentionally loaded die, which can be considered as cheating, is also possible during the game process. To prevent cheating and equalize the results in general we propose to use the algorithm that allows to generate fair results out of pseudo-randomly generated sequence of numbers, meaning to turn loaded die into unloaded.

Originally, the idea of turning loaded die into unloaded one belongs to John von Neumann who is famous for research in statistics and game theory [5]. But in the current research we used the methods described by Ari Jules in [6] which is also based on the work of von Neumann. These methods allow to generate a fair result out of minimum three independent rolls. The only side effect of this method lies in the increasing number of rolls that takes more time for a player to get the fair result. The method doesn't take much computational power

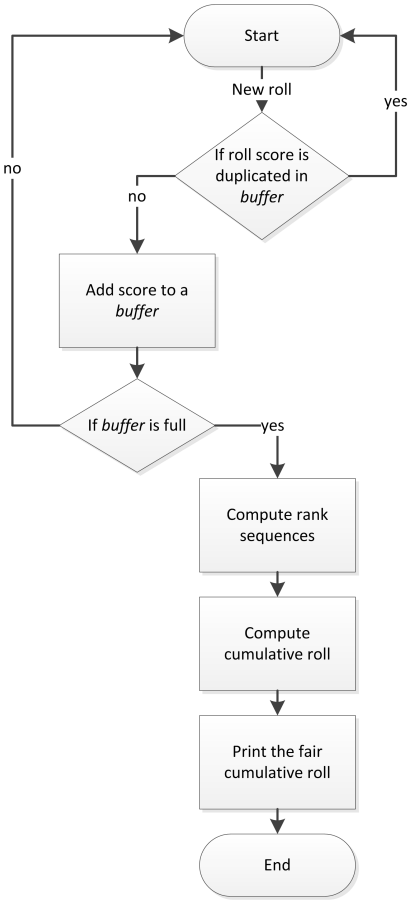

Fig. 1. Application flowchart

and can be implemented in any modern device with CPU and data memory.

Another aspect discussed in the article is an upper surface detection. It can be done in different ways but we present the method using roll and pitch values of the die. Roll and pitch values are calculated based on accelerometer and gyroscope measurements. There are several ways to implement described in [7], [8], [9]. One of the problems was to detect the moment when the die is finally stopped since there are situations possible when the die is balancing on one of the edges and flips in a second. During this second accelerometer shows that the die is static but surface detector cannot detect the face properly. For this purpose we included the countdown timer in our implementation that waits pre-defined amount of seconds before calculating the upper surface.

II. Algorithm

In this chapter we describe several key points of implementation from the algorithmic point of view. The implementation is represented as a simple dice game with main steps shown on Figure 1. The game starts immediately after user shacked the dice. The IMU continues transmitting data until die *finally* stops. Even if the die stopped moving for a moment it sends the event that can be detected as a *final* stop by an application. A good example of it is an edge balancing. To prevent this unambiguity, we made the IMU to continue sending data during the next 3 seconds after stop, the states are shown on Figure 3. After the *final* stop, application runs unbiasing algorithm and shows the result depending on the algorithm output.

One of the most important parts is to separate different stages of dice game. The whole game process can be divided onto several parts:

- Player takes the die and holds it in the hand being ready to roll.
- Player shakes the die in order to start the game. This functionality was implemented to prevent accidental start of the game.
- Player has thrown the die and the die is falling or rolling over the floor.
- The die has stopped rolling.

In our implementation every part is clearly identified by either IMU itself or an application and both are behave according to corresponding scenarios.

Next chapters of article describe algorithms that were implemented to provide game functionality, including: state classifier, upper surface detector, die unbiasing. These algorithms were combined in one solution but the description is presented in different subsections below.

A. Unbiased die simulation

In this subsection the process of extracting fair roll values from a biased die is described. The algorithm is based on a paper of Ari Juels [6]. For this particular case the algorithm was adapted to be used exactly with 6-sided die but can be easily modified for n-sided dice. During research, authors tried to presume the most typical scenario where biased dice make the biggest effect on the game result. Suppose there are two players throwing two different dice with different mass centers: one in the center of die(suppose to be an almost fair one), another with the mass displaced to the bottom. Obviously, the frequency of values for every player is different and unfair. The algorithm allows to generate a fair random output for both players by combining several biased rolls

$$\mathrm{R} = \{\mathrm{r}_1, \mathrm{r}_2\, \mathrm{r}_3 \ldots \mathrm{r}_\mathrm{n}\}, n \geq 3 \quad (1)$$

where R is a representation of a sequence of roll results needed to generate a single fair roll, *n* is the minimum possible amount of rolls. The output is equally random for both players and the only difference is the amount of rolls that every player should do to get fair result. Detailed description of the algorithm including rank sequence generating and cumulative roll computation are given in [6]. The algorithm was implemented as a real-time application described in Section III. The key steps are shown on the Figure 1.

B. Classification

In the current research authors successfully reused applied states of motorcyclist on the states of gambling die. The motion model of die is quite similar with the motorcycle motion

model described in [10] and [11]. "Motion" and "static" states of vehicle are fully identical with the corresponding states of die in the game. And "motorcycle crash event" is assumed to mean the initial shake of the die. State classification has been implemented as a described in [12]. The following states are recognised by the algorithm implemented in the IMU:

- **Static** The die is not moving.
- **Move** The die is in the air or rolling.
- **Shake** The die was shaken.

For the classification task, we use a supervised Bayesian maximum a posteriori classifier (MAP). A supervised classification process starts by collecting a training data set with known states. This set is used to obtain

$$z_j \sim N(\boldsymbol{\mu}_j, \Sigma_j), \tag{2}$$

the distribution of the observed $q-by-1$-vector z given that the observation comes from the class j. At this stage, the mean vector $\boldsymbol{\mu}_j(q \times 1)$ and the covariance matrix $\Sigma_j(q \times q)$ are here assumed to be perfectly known (learned from a training set). We also assume that for all classes $\Sigma_j > 0$ (i.e. Σ_j is positive definite). If (2) holds, the density function of z_j is

$$f_{z_j}(z; \boldsymbol{\mu}_j, \Sigma_j) = \frac{1}{(2\pi)^{q/2}\sqrt{|\Sigma_j|}} \exp[-\frac{(z-\boldsymbol{\mu}_j)^T \Sigma_j^{-1}(z-\boldsymbol{\mu}_j)}{2}]. \tag{3}$$

A new observed $z = z_x$ can then be classified by maximizing (3) over all the classes $j = 1...p$. To assign a probability to the classification result, we need to use unconditional prior probabilities $P(C = j)$, and assume that all the possible classes are included. Then the probabilities for all the classes are obtained from the Bayes' rule:

$$P(C = j|z_x) = \frac{f_{z_j}(z_x)P(C = j)}{f_z(z_x)}, \tag{4}$$

where f_z is the unconditional density function for the observation z. Prior probability can be used to control the false alarms or false negatives.

C. Die upper surface detection

Upper surface detection plays a key role as a representation of a roll result. User can visually recognize the amount of pips on the upper side of die but for the IMU the only sources of information are acceleration and angle speed from sensors. Using sensor data IMU calculates and sends the attitude data in radians, roll and pitch to user device. Next, using the corresponding information about roll and pitch, the virtual model of die is being built on the screen and the face detection algorithm starts. Upper surface detection algorithm is implemented on the user-side(currently on Android-based smartphone). It allows to move additional computations from IMU and decrease power consumption of die. Current software implementation is done in Unity [13] framework as the most user-friendly and easy-to-deploy IDE(Integrated development environment). Another reason of using Unity was to create universal software that can be used with dice of any amount of edges (for example, there are two types of 6-side dice exist: Asian and Western, that differs by mark-up of two sides only). In the current version user can easily change the skin and switch between different colors and shapes of die. By using Unity framework it is significantly easier to change the skin and change the markup than update the firmware for IMU.

The main principle of upper surface recognition is based on calculating of dot product between the unit vectors matching with x,y and z axis in the local frame and a unit vector pointing up

$$r_n = v_n \cdot u, \tag{5}$$

where r_n is an amount of rotation above each of the axis x,y and z in the interval of [-1;1], v_n are the unit vectors matching with axis x,y,z and u = {0,1,0} is a vector pointing up in a local frame. r_n = 1 means that the corresponding axis is pointing up in global frame, and down if r_n = -1 respectively.

III. Implementation

Real-time implementation is a fully-functional prototype of game application. Currently it allows to simulate the die on any Android device and represents the result of unloading algorithm. Next versions will allow to use several dice in the same game and support multiple users. Application consists of the following parts:

- Hardware implementation. Two foam-rubber dice loaded with an IMU on different distances from the center of die.
- Software implementation. Android phone with the application for the visualization of algorithm data and automatic detection of states.

A. Inertial measurement unit

Two dice of size 75 mm x 75 mm x 75 mm were slit in different levels to simulate the loaded dice with different center of mass as it shown on Figure 2. IMUs were placed inside of the dice and fastened in order to transmit the actual information about the attitude, dice state and the upper face. We used a unit based on ARM Cortex M3 microprocessor with the maximum frequency of 72MHz, 512kB of program memory and 64kB of data memory. The device is also equipped with Bluetooth transmitter and memory card module for data transfer and logging. The size of board is equal to 45 mm x 45 mm x 10 mm.

State classifier and attitude determination algorithm were implemented as a FreeRTOS tasks. Reading, calibrating of the measurements from sensors and adding them to a buffer were implemented using two another tasks. One more task is responsible for inter-task communication and outer communications. Handling of measurement data and inter-task messaging is done using thread-safe queues.

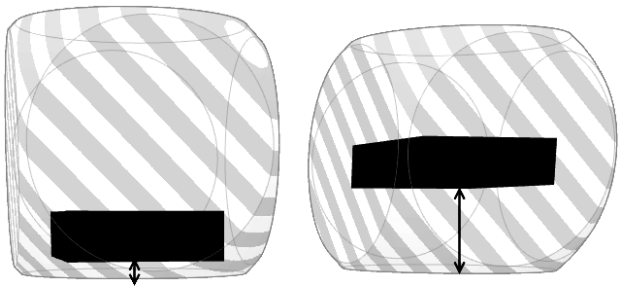

Fig. 2. Position of mass in the dice

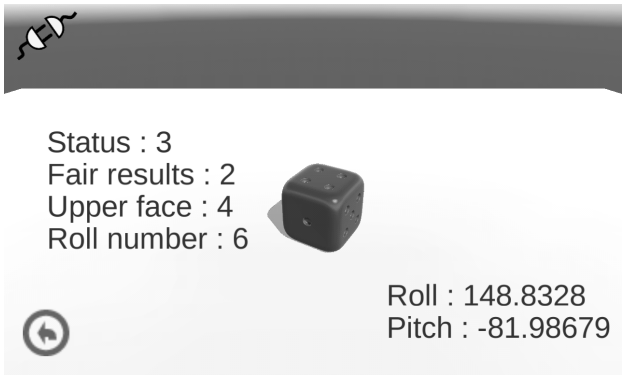

Fig. 4. Application interface

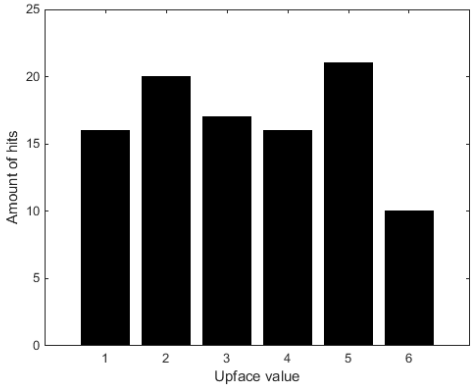

Fig. 5. Results of rolling the die with mass in the center, 100 times

B. Android Application

Android application consists of two parts: Unity application as a front-end and Java Android application for passing the messages to Unity engine and determining the states of dice as a back-end. Java application is packed to a JAR file and pushed to Unity. Such architecture was chosen since Unity is not able to call the native Android functions and handle the threads by itself thus the bridge between Unity and Android is needed.

Initially, Android application opens Bluetooth connection to the measurement unit and sends a *Start measurements* message. After getting the message unit starts analysing accelerometer and gyroscope messages and sends roll, pitch and the current state of IMU back to the phone(the yaw is not significant for the current research and more complicated to compute). Java application receives and handles the messages, makes a decision about the current state of an application and waits until the Unity call. Unity engine calls update functions implemented in a bridge object after every frame update.

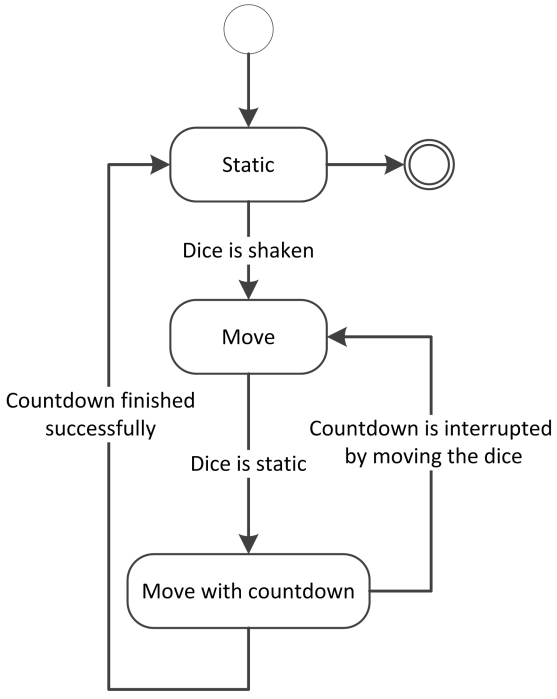

Fig. 3. IMU state flowchart

Unity is used due to represent the raw data in appropriate and understandable way. Application allows to track the roll and pitch of die, track final result, analyse it and generate unloaded result, save it to a file and show on the screen. The interface is shown on Figure 4. Current version uses 6-faced model of die but potentially can be used to represent die with N faces, where $N \leq \infty$.

IV. Results

The solution described above was tested both with real models of dice and in MatLab environment. Both dice have been rolled 100 times in the closed room with a flat floor, the upper surface of every roll was logged in file by the application, fair results were also logged and visualized. Figure 5 and Figure 6 show the actual results. The picture shows that the amount of successes for side number six is slightly lower than others. This could happen due to the position of mass: it was lying close to the surface six and could affect on rotation model of die.

Using the formula of binomial distribution the probability of getting this amount of successes for surface six can be

obtained as:

$$f(k,n,p) = \binom{n}{k} p^k (1-p)^{n-k}, \tag{6}$$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}, \tag{7}$$

, where $n \in N_0$ is a number of trials, $p \in [0,1]$ is a success probability in each trial, $k \in \{0,...,n\}$ is a number of successes. For 10 and 13 successes in a 100 tries the probability is equal to 0.021 and 0.07 respectively.

But it is hard to make any statistical assessment due to relatively small amount of experiments. 20000 rolls would be acceptable amount of rolls as it shown by experiments of Rudolph Wolf [14] but it takes significant amount of time. In the future research one could use dice-shaking machine to automate process.

After getting statistics from real rolls, a virtual simulation of loaded die was implemented using MatLab *rand* function. During research authors tried to create fully unfair physical die by placing IMU in different areas inside of die but couldn't get even a 30% probability of some selected face. This happened due to a soft material of dice that adds some extra "randomness" into physical motion. Thus, authors *simulated* a highly biased die that showed side with one dot with 50% probability and compared results with real rolls. The total amount of rolls that showed surface 1 is equal to 2503. By using the MatLab function of binomial probability density, authors checked a chance of getting the same result when using a fair die. Distribution is illustrated on Figure 8. The plot shows distribution of the most likely amount of showings for one single surface provided that probability of showing is equally likely for every face. Probability of getting 2503 showings of one single face is equal to

$$f(k,n,p) = \binom{n}{k} p^k (1-p)^{n-k} \leq 2.220446 * 10^{-16}, \tag{8}$$

which means that our MatLab simulation represents a totally unfair die.

There were totally 4 experiments done:

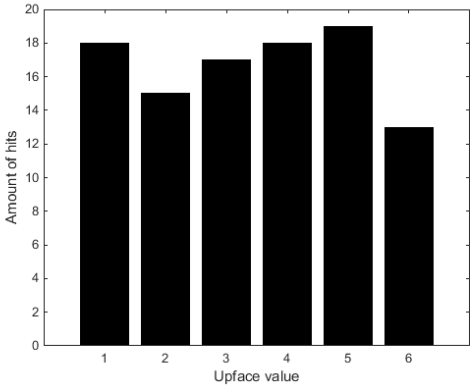

Fig. 6. Results of rolling the die with mass in the bottom, 100 times

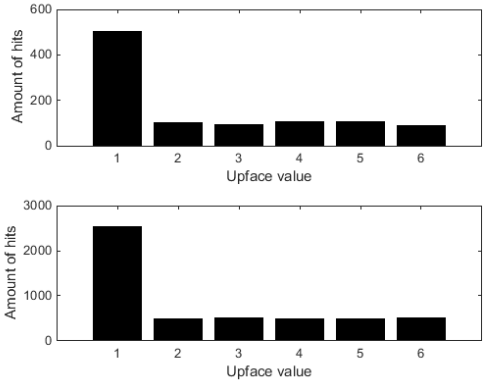

Fig. 7. Randomly generated sequence of 1000 numbers(top) and 5000 numbers(bottom)

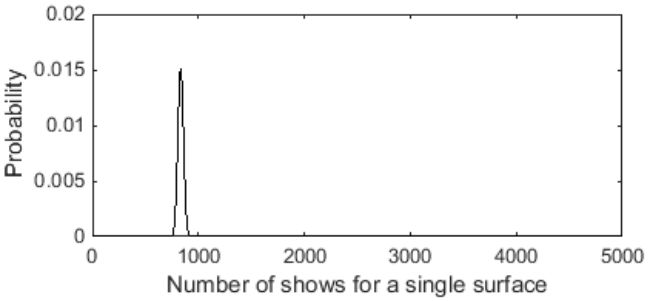

Fig. 8. Binomial distribution for 5000 rolls

- **Experiment 1** Real die. Mass in the bottom. Rolled 100 times.
- **Experiment 2** Real die. Mass in the middle. Rolled 100 times.
- **Experiment 3** Simulation. 1000 of random numbers were generated.
- **Experiment 4** Simulation. 5000 of random numbers were generated.

The goal of virtual tests is to show the average amount of rolls that every user should do to get a fair result if he uses biased and unbiased die. The results are represented in Table I. It shows that in the worst case every user should make the maximum of 6 rolls to get a fair result.

TABLE I
EXPERIMENTAL RESULTS

Exp.	Amnt. of unbiased rolls	Avg. amnt. of rolls per unbiased roll	Longest seq. of rolls
1	27	3,70	6
2	28	3,57	6
3	199	5,02	16
4	1062	4,70	19

V. Conclusion

In this paper authors presented a real-time application for unloading the loaded die. The outcome of the experiments shows that technology is quite reliable and can be implemented in different applications on different platforms. The application created during research can be used for both scientific and entertainment purposes: for collecting and processing roll statistics and representing game content. The method described in [6] is proved as working in real conditions and also can be used in other applications. The largest problem is an increased amount of rolls that player need to do before getting a fair result. As a possible solution we suggest to disable unloading algorithm in the application by the default and use it only if the drop rate of some specific side of die is significantly high. As an extra outcome we also show a new concept of using inertial measurement unit: as an input device for multimedia applications. The concept can also be used as a groundwork for a future research: inertial dice were never used before and got a huge potential in a field of statistics.

A good research topic would be a topic related to upper surface prediction i.e. prediction of upper face before die actually stops. It can be done by using simulation software but in case of finite faced die, physical model can be a subject to influence of many external conditions such as surface friction and air resistance. M. Kapitaniak in [15] shows that friction between table and die can affect significantly on the die motion. Moreover, in real-case scenario dice are rolling on non-perfect surface. In this case simulation will not be beneficial at all since surface information is not available for sensors and the only way we can use face prediction, will be prediction of die attitude in the air *before* it hits the ground. Similar research was done by Michael Small in [16] for the game roulette.

Another potential research could be done for the die with infinite amount of edges. The easiest way would be to build a spherical die model(infinite amount of edges can be considered as a sphere) and compute attitude of this ball during rolling. In this case surface friction and air resistance can be neglected. But another problem of infinite faced die could be stop detection: if the die is a well-balanced sphere, it can take significant amount of time to stop on flat surface. Here the problem of infinite faced die simulation smoothly turns into classical ball simulation.

References

[1] W.-Y. Chen, P.-J. Lin, and D.-Y. Kuo, "Dice image recognition scheme using pattern comparison technique," in *in Proc. Int. Symp. Computer Consumer Control*, Taichung, Taiwan, June 4-6 2012, pp. 128–131.

[2] B. A. B. Correia, J. A. Silva, F. D. Carvalho, R. Guilherme, F. C. Rodrigues, and A. M. de Silva Ferreira, "Automated detection and classification of dice," in *Proc. SPIE 2423, Machine Vision Applications in Industrial Inspection III*, San Jose, CA, Mar. 27 1995, pp. 196–202. [Online]. Available: http://dx.doi.org/10.1117/12.205506

[3] I. Lapanja, M. Mraz, and N. Zimic, "Computer vision based reliability control for electromechanical dice gambling machine," in *Proc. IEEE Int. Conf. Industrial Tech.*, Goa, India, Jan 19-22 2000, pp. 325–328.

[4] G. S. Hsu, H. C. Peng, and S. M. Yeh, "Color and illumination invariant dice recognition," in *2012 IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC)*, Oct 2012, pp. 857–862.

[5] J. von Neumann, "Various Techniques Used in Connection with Random Digits," *J. Res. Nat. Bur. Stand.*, vol. 12, pp. 36–38, 1951.

[6] A. Juels, M. Jakobsson, E. Shriver, and B. Hillyer, "How to turn loaded dice into fair coins," *IEEE Trans. Inf. Theory*, vol. 46, no. 3, pp. 911–921, May 2000.

[7] Y. Hao, Z. Xiong, W. Gao, and L. Li, "Study of strapdown inertial navigation integration algorithms," in *Proc. Int. Conf. Intelligent Mechatronics Automation*, Chengdu, China, Aug 26-31 2004, pp. 751–754.

[8] M. Xu, N. Fan, and Z. Wang, "Study on extended kalman filtering for attitude estimation of micro flight vehicle," in *Proc. Int. Conf. Measuring Tech. Mechatronics Automation*, Shanghai, China, Jan 6-7 2011, pp. 457–460.

[9] H. Rehbinder and X. He, "Nonlinear pitch and roll estimation for walking robots," in *Proc. IEEE Int. Conf. Robotics Automation*, San Francisco, CA, Apr. 24-28 2000, pp. 2617–2622 vol.3.

[10] C. C.-Y., "On the detection of vehicular crashes-system characteristics and architecture," *Vehicular Technology*, vol. 51, no. 1, pp. 180–193, 2002.

[11] H. Tabata and K. Kushida, "Automatic accident reporting apparatus for two-wheeled vehicles," Apr. 29 2009, EP Patent 1,197,426.

[12] J. Parviainen, J. Collin, T. Pihlström, J. Takala, K. Hanski, and A. Lumiaho, "Automatic crash detection for motor cycles," in *Proc. Ann. Conf. IEEE Industrial Electronics Soc.*, Dallas, TX, Oct. 29 - Nov. 1 2014, pp. 3409–3413.

[13] Unity3d, what is unity? http://goo.gl/n3YtYX. [Online; accessed Dec. 2015].

[14] R. Rosenkrantz, "Concentration of distributions at entropy maxima (1979)," in *E. T. Jaynes: Papers on Probability, Statistics and Statistical Physics*, ser. Synthese Library, R. Rosenkrantz, Ed. Springer Netherlands, 1989, vol. 158, pp. 315–336. [Online]. Available: http://dx.doi.org/10.1007/978-94-009-6581-2_11

[15] M. Kapitaniak, J. Strzalko, J. Grabski, and T. Kapitaniak, "The three-dimensional dynamics of the die throw," *Chaos*, vol. 22, no. 4, 2012. [Online]. Available: http://goo.gl/Cga9iZ

[16] M. Small and C. K. Tse, "Feasible implementation of a prediction algorithm for the game of roulette," in *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, Nov 2008, pp. 1208–1211.