A Ping-Pong Ball Catching and Juggling Robot: a Real-Time Framework for Vision Guided Acting of an Industrial Robot Arm

Holger H. Rapp
Institute of Measurement and Control Theory
Karlsruhe Institute of Technology – KIT
Engler-Bunte-Ring 21
76131 Karlsruhe, Germany
holger.rapp@kit.edu

Abstract—**This paper investigates the possibilities of real-time control of standard industrial robot arms by the means of a ping-pong ball juggling system that is able to catch a ball thrown by a human and to keep the ball airborne for more than 30 min. The sensors are two industrial cameras operating at 60 Hz. We discuss the image processing and the controlling algorithms used. Emphasis is placed on comparing the widespread naive linear model and a novel physically correct model of the flight trajectory of the ball. While both models are sufficient for the juggling task, it is shown that only the non-linear model is able to predict the impact parameters sufficiently early and precise enough to solve the initial catching task.**

The performance of the complete systems is also documented in a supplementary video [8].

I. INTRODUCTION

Many of the juggling tasks that humans can perform have been investigated for machines as well. A comprehensive overview is given by [11].

The subtask we are interested in is best described as paddle juggling: the idea is that a ball is kept in the air by repeatedly hitting it upwards. If the starting conditions - the balls initial position and velocity - are precisely known this problem can be solved with an open loop controller and no sensors whatsoever [1], [9], [12]. Some uncertainties in the starting conditions, will requires some kind of sensory input and therefore a closed loop controller. The work of Rizzi et. al. is most influential in this area. Notably, [10] presents a paddle juggler that was remarkable for its time: a distributed system was used to interpret data from a stereo camera system to close the control loop of a custom built robot. The vision system and linear models notably constrain the starting conditions though, that means that the ball must be released in a very specific fashion, the machine is unable to catch a ball thrown by a human.

Recently, the use of stock industrial robots has become viable and those systems have therefore moved into the focus of research. [6] uses a 6 degrees-of-freedom (DOF) industrial robot comparable to ours, but keeps the constraints on the starting conditions of the ball for similar reasons as prior work.

This paper discusses our solution to the catching and paddle juggling tasks which we formulate as follows:

- The system must be able to catch a ball thrown by a human towards it. The ball should then be juggled and kept bouncing on the racket.

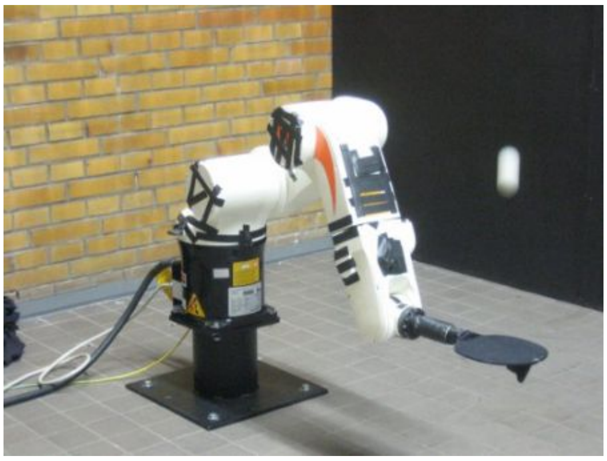
Fig. 1. The robot used in the experiments juggling a ball.

The novelty of this paper is therefore in the ability to catch the first ball when it is thrown by a human on the racket and smoothly transition to the juggling of the ball. We do away with the constraints on the starting conditions of the ball and allow for the uncertainty that comes with human involvement. As we will see, the naive linear model usually employed for the trajectory prediction of the ball is not sufficient to accomplish this task.

We therefore expand on the literature by investigating a physically exact flight model for the ball which allows for unconstrained starting conditions. This model is necessary for the initial catch of the ball where a linear model fails. We also provide solutions for engineering constraints commonly found in industrial robots which were not designed for real time control tasks. This includes coping with the big dead time before movements are started and the lack of trajectory control: our system only accepts incremental movement commands, i.e. no direct control over the movement speed or trajectory is possible.

II. EXPERIMENTAL SETUP

The experimental setup is depicted in Fig. 2. The main calculations are done by a standard PC (2.2 GHz, 2 GB) running Linux. Attached to this computer are two cameras which act as sensors for the application. The computer processes the input from the cameras and sends the movement commands to the robot via the industrial PC that contains the proprietary control software for the joints. The movement of the robot acts back

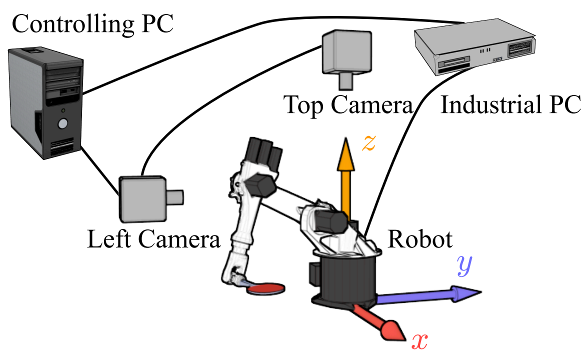

Fig. 2. Experimental setup and origin of work space.

on the flying ball closing the control loop. The software of the framework is written in Python and all calculations are done in the robot's right-hand world coordinate system which has its origin in its foot. The throw direction of the ball is along the negative x-axis. The Robot used in this system is a Kuka KR5 sixx R850 with a reach of 850 mm, 6 degrees of freedom and a theoretical maximum movement speed of 7.5 m/s. It is delivered with an industrial PC running a real time operating system and an embedded version of Microsoft Windows as frontend to the user. The canonical way of getting these robots to move is by off-line teaching them control points and connecting path segments which relate the points to each other. The robot will then follow a path as soon as it is triggered to move. The path cannot be changed while the robot is moving.

For our system, real-time control was mandatory. Here, we used the software KUKA.Ethernet RSI XML 1.1 by the manufacturer of the robot. The software allows to add Cartesian corrections to off-line taught trajectories by sending them in an XML format via UDP over an Ethernet network to the industrial PC [2]. With some modifications, the framework also accepts degenerate off-line trajectories of no movement at all or deltas of arbitrary size. This is equivalent of being able to give the robot a position set-point which it tries to reach as fast as possible. But this does not offer direct control over timing and speed of the robot. Especially repeatability is not guaranteed and there is a certain variation in the dead time and the moved trajectory.

The sensory part of the setup comprises of two IEEE 1394b grayscale cameras (Point Grey Flea2). The cameras are synchronized and acquire images with 640x480 pixels at 8 bit color depth with 60 Hz. The Python library to read images from the cameras has been specially developed for this task with an emphasis on low overhead and high acquisition performance. We released it as an open source project [7].

III. Algorithm

The complete task can be broken down into the following individual problems:

- Measuring the current position of the ball $\boldsymbol{b}$ in three-dimensional space (see Sec. III-A)
- Determining the impact parameters for a desired height over ground z_i (see Sec. IV).
- Determine the ideal movement of the robot for keeping the ball bouncing:
  - Determining the ideal orientation of the racket for reflecting the ball straight upwards (see Sec. III-B1)
  - Determining the ideal movement command to hit the ball with maximum movement speed (see Sec. III-B2).
  - Make sure that deviations in the controlling are corrected over time (see Sec. III-B3). This ensures the stability of juggling.

The general flow of information is depicted in Fig. 3. As soon as a new image pair is acquired from the cameras, the individual images are processed and the ball is searched in each of them. If the ball is detected in each image, the two coordinates are used to triangulate the current ball position $\boldsymbol{b}$ in space (see Sec. III-A). This ball position is then fed into a predictor as a new measurement. The predictor uses a model (see Sec. IV-A) to predict the impact position and impact time of the ball. This information is fed to two distinct controllers – max impulse and horizontal reflection – which output the set-point for the robot movement: a Cartesian position $\boldsymbol{r}_{\text{setpoint}}$ and two angles defining the normal of the racket's surface $(\varphi_{\text{setpoint}} = (b, c))$ (see Sec. III-B). The sample-and-hold element waits until its inputs have converged. It will then issue one movement command to the robot which acts back on the ball by hitting it, effectively closing the control loop.

A PID controller corrects the output angles of the vertical reflection controller and ensures the stability of juggling.

A. *Image Processing*

The image processing is designed to be fast and effective. The experimental setup is stationary and under constant illumination. This allowed us to finely tune the parameters. As a result practically no misdetections of the ball have occurred in our experiments.

To detect a ball in an image, the image is first binarized with a fixed threshold, then contours are detected. All detected contours are then checked for various shape parameters (e.g. area, x/y ratio, position, skewness) which can all be easily calculated from the contour's moments [4]. If more than one candidate contour remains, the one closest to the point used in the last frame is favored.

To get a proper view-ray in our Cartesian world space, we have to account for the lens distortions of the cameras. We calibrated our cameras according to an established lens model [3]. Since we are only interested in the undistortion of our sub-pixel exact ball position, we numerically invert the model only for this one point in each step.

The position of the ball in the pixel coordinate frame of the images corresponds to view rays which can be used to triangulate the position of the ball in space. The positions of the cameras with respect to each other and to the robot's coordinate frame are calibrated.

B. *Controllers*

The predictors evaluated for this setup are discussed in the next section (see Sec. IV) in more detail. All predictors output an estimate of the three impact parameters: the impact position

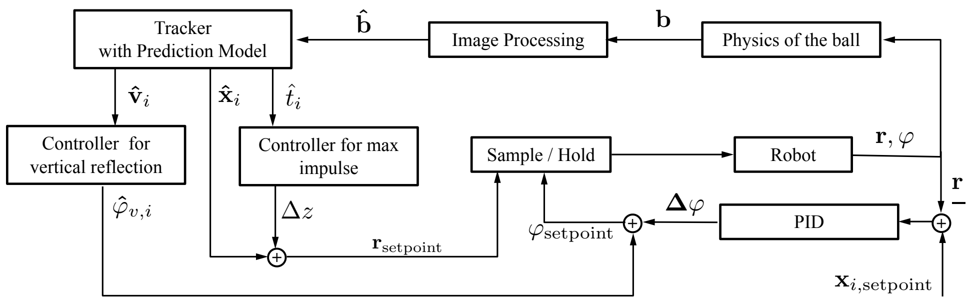

Fig. 3. Control diagram

$\boldsymbol{x}_i$, the impact speed $\boldsymbol{v}_i$ and the impact time t_i. $\boldsymbol{x}_i$ is used directly to position the racket, the other two parameters are used to determine the normal vector for the racket expressed through the two Euler angles $\boldsymbol{\varphi} = (b,c)$ and the point in time when the robot should start to move upwards to give a maximum impulse in z direction onto the ball.

1) Controller for vertical reflection: It is assumed that the ball makes a ideal elastic collision with the racket and is therefore reflected. The ball arrives with a speed $\boldsymbol{v}_i$ and we want it to be reflected straight upwards: its direction after reflection should be $\boldsymbol{d} = \boldsymbol{e}_z$. We can use the reflection law

$$\boldsymbol{d} = \boldsymbol{v}_i - 2(\boldsymbol{v}_i^T \boldsymbol{n})\boldsymbol{n} \tag{1}$$

to derive the required normal vector $\boldsymbol{n} = (n_x, n_y, n_z)^T$ of the racket. For this, we multiply (1) from the left with $\boldsymbol{v}_i^T$ and solve for $\boldsymbol{v}_i^T \boldsymbol{n}$, choosing the sign of the solution so that $\boldsymbol{n} = \boldsymbol{e}_z$. Re-substituting this in (1) and solving for $\boldsymbol{n}$ results in

$$\boldsymbol{n} = \frac{\boldsymbol{d} - \boldsymbol{v}_i}{\sqrt{2(||\boldsymbol{v}_i|| - \boldsymbol{v}_i^T \boldsymbol{d})}}. \tag{2}$$

The robot expects the set-point orientation in Euler angles, therefore the controller must output the two corresponding angles that lead to this normal vector. They can be calculated from $\boldsymbol{n}$ via $b = \arctan n_x/n_z$ and $c = \arctan n_y/n_z$.

2) Controller for maximum impulse: Vertical reflection is not enough: the ball loses kinetic energy due to friction and deformation each time it collides with the racket. It is therefore mandatory to put energy into the ball to keep it airborne. Since we can not control the speed of the robot's movements directly, we must give it a movement command at a precise time so that its movement speed is at its maximal value when the ball collides with the racket. The delay T_0 between movement command and achieving maximum speed was determined in off-line experiments beforehand. Knowing T_0, we can give the robot an upwards movement command by an amount of Δz at time $t_i - T_0$, the robot will then hit the ball with its maximum speed. After a constant delay ΔT, the command to move down again is issued to make sure that the robot is ready for the next bounce. The output of the controller for maximum impulse can therefore be written as

$$\Delta z(t) = \Big[\sigma\big(t - (t_i - T_0)\big) - \sigma\big(t - (t_i - T_0 + \Delta T)\big)\Big]\delta z, \tag{3}$$

with σ being the Heavyside step-function and δz being the constant amount that the racket should move upwards, in our case $\delta z = 0.05\,\text{m}$.

3) PID controller: There are some sources of deviations: the racket might be installed with a slight offset in the angles compared to the robot coordinate system, the dead time of the robot varies slightly from movement command to movement command, the controller for vertical reflection neglects angular velocity and the deformability of the ball in its calculations and all inputs to the controllers are estimations. This leads to instability: the robot would loose the ball after a few bounces when it would leave its control space. We want the robot to always juggle the ball where it can reach it easily and where its movements are as fast as possible.

To achieve this, a PID controller is used that corrects the angles outputted by the controller for vertical reflection. Its input is the difference between the desired impact position $\boldsymbol{r}_{i,\text{setpoint}}$ in the middle of the working area of the robot and the estimated impact position $\boldsymbol{x}_i$. The integral part of the controller guarantees that the real impact position gets closer and closer to the set-point with each consecutive bounce.

IV. PREDICTION OF THE TRAJECTORY

Given N measurements and a desired impact height z_i, we want to predict the impact position $\boldsymbol{x}_i$, impact speed $\boldsymbol{v}_i$, and impact time t_i of the ball.

We first have to choose a model that describes the physical conditions of the throw (see Sec. IV-A), then we have to decide on a mechanism to predict the impact state of the model given some observations (see Sec. IV-B). The combination of model and predictor should make the predictions accurately with N being as small as possible.

A. Models

We investigated two distinct models. The next section introduces the naive linear model which is commonly used in the literature. It is analytically tractable but is unable to capture the physics.

The following section discusses a physically correct model that incorporates all of the dominant effects on the movement of the ball.

1) Naive Model: A point mass m is assumed for the ballistic motion. The only force that acts on the mass is the gravitational force $\boldsymbol{F}_\text{g} = m(0,0,-g)^T = -mg\boldsymbol{e}_z$. For slow speeds and high masses m, air drag and other non-linearities can be ignored.

This model results in the well know flight parabola with the initial position $\boldsymbol{x}_0 = (x_0, y_0, z_0)^T$ and initial speed $\boldsymbol{v}_0 =$

$(v_{0_x}, v_{0_y}, v_{0_z})^T$:

$$\boldsymbol{x}(t) = \boldsymbol{x}_0 + \boldsymbol{v}_0 t - \frac{1}{2} g t^2 \boldsymbol{e}_z \quad \text{and} \quad \boldsymbol{v}(t) = \boldsymbol{v}_0 - m g \boldsymbol{e}_z t \tag{4}$$

Solving $x_z(t_i) = z_i$ – with z_i being the desired impact height – for t_i gives either one or two solutions in the future. If there are two solutions, we are always interested in the later time because this is the one where the ball will come down on the racket:

$$t_i = \frac{1}{g}\left(v_{0_z} + \sqrt{v_{0_z}^2 + 2g(z_0 - z_i)}\right). \tag{5}$$

Substituting this into (4) directly yields the desired information for the impact parameters.

The naive model has the advantage of being linear and therefore of being analytically tractable. However for a flying ping-pong ball, the assumption of no interaction with the surrounding air is incorrect: friction is acting on the very light ping-pong ball in non neglectable ways.

2) Physically Correct Model: We therefore present a novel model which takes the effect of friction into account. The dominant effects are the decelerating force $\boldsymbol{F}_\mathrm{d}$ due to air drag and the Magnus effect $\boldsymbol{F}_\mathrm{M}$, which changes the flight direction of the ball due to its rotation. We will discuss both effects in this order.

a) The Air Drag: The Reynolds number for our case is in the magnitude of

$$\mathrm{Re} = \frac{\rho v d}{\eta} \approx 9200, \tag{6}$$

with $\rho \approx 1.29\,\mathrm{kg/m^2}$ being the density of air which depends on the temperature of the surroundings, $v \approx 3\,\mathrm{m/s}$ being an estimate of the flying speed of the ball, $d = 0.04\,\mathrm{m}$ being the flow diameter of the ball and $\eta = 17.1\,\mu\mathrm{Pa\cdot s}$ being the viscosity of air. The Reynolds number is high, we can safely assume the border case of turbulent flow in all cases.

The air drag of a body in turbulent flow is described by

$$\boldsymbol{F}_\mathrm{d} = \frac{1}{2}\rho A C_D |\boldsymbol{v}|^2 \frac{\boldsymbol{v}}{|\boldsymbol{v}|}, \tag{7}$$

with $A = \pi r^2$ being the cross section of the ball that is hit by the flow. $\boldsymbol{v}$ is the velocity of the ball, C_D is the drag coefficient which depends on the geometry of the flying object and is between 0.1 and 0.5 for a spherical object in surroundings of high Reynolds numbers.

b) The Magnus Effect: If a sphere with radius r rotating with angular velocity $\boldsymbol{\omega}$ is thrown through a medium with speed $\boldsymbol{v}$, the flow speed of the medium is different for its top and bottom part. This corresponds to a difference in static pressure at the two sides of the sphere and leads to a buoyant force of

$$\boldsymbol{F}_\mathrm{M} \approx 2\rho C_M \pi r^2 (\boldsymbol{\omega} \times \boldsymbol{v}) \tag{8}$$

given that $|\boldsymbol{\omega}| r \ll |\boldsymbol{v}|$ which is the case here. This effect is called Magnus force. C_M is a dimensionless number called Magnus factor that depends on the Reynolds number and on the geometry and surface roughness of the object in the flow. For our experiment with a reasonable smooth sphere, we expect this to be < 0.5.

c) Combined Model: This leads to the complete model, which is the combination of gravitational force $\boldsymbol{F}_\mathrm{g}$, Magnus force $\boldsymbol{F}_\mathrm{M}$ and air drag $\boldsymbol{F}_\mathrm{d}$:

$$\begin{aligned} \boldsymbol{F} &= m\ddot{\boldsymbol{x}} = \boldsymbol{F}_\mathrm{g} + \boldsymbol{F}_\mathrm{d} + \boldsymbol{F}_\mathrm{M} & (9)\\ &= -g\boldsymbol{e}_z - \frac{1}{2}\rho A C_D |\boldsymbol{v}|^2 \frac{\boldsymbol{v}}{|\boldsymbol{v}|} - 2\rho C_M \pi r^2 (\boldsymbol{\omega} \times \boldsymbol{v}) & (10) \end{aligned}$$

This model describes the flight curve of our ping-pong ball physically correctly and exhaustively, but it has the disadvantage of high complexity: $\boldsymbol{\omega}$ is changing significantly from throw to throw but can't be measured and must therefore be estimated from the measurements of the position of the ball. C_D and C_M can be assumed to be constant in our case, but can't be directly measured either.

B. Kalman Predictors

Each of these models need to be fitted to measurements and predicted forward in time to get the impact parameters. A good predictor must be able to predict sequentially (or on-line) when new measurements arrive, should be able to incorporate prior knowledge - like the probability density function (PDF) of the starting conditions - and should offer confidence information about the impact parameters. For example if we knew the impact position up to a precision of a few centimeters, we can be sure that the ball would hit the racket because of the racket's size.

All of these features are provided by the Kalman filter. It is widely used in tracking tasks in robotics and computer vision. It estimates sequentially and is therefore very fast, it can incorporate prior knowledge in form of a Gaussian prior distribution over the model's parameters and its state predictions also provide variance information, which can be directly interpreted as confidence information.

The naive flight curve model can be used directly as model for the Kalman filter while adding a system variance that accommodates for the non-linearities in the flight curve of the ball. This should enhance the prediction performance by allowing the state to change slightly in a non-linear way. Another advantage is that the initial state and variances can be learned from training data; this effectively incorporates prior knowledge about the throwing speeds.

The physically correct models uses a non-linear filter known as the unscented Kalman filter [5] (UKF): the basic idea is to interpret state and covariances of the filter as a Gaussian approximation to the correct PDF and to translate this through the non-linear model. This is done via the unscented transform: one chooses enough points from the PDF to fully describe a Gaussian distribution and translates those through the model. The translated points describe a Gaussian approximation to the a-posteriori probability.

A major problem for our case is that the Kalman filter can only track system states, i.e. the current parameters of the model. To predict the impact time t_i with the current system state we trace the model forward in time until we reach the impact height z_i. This can be done analytically for the naive model. For the physically correct model one has to use a

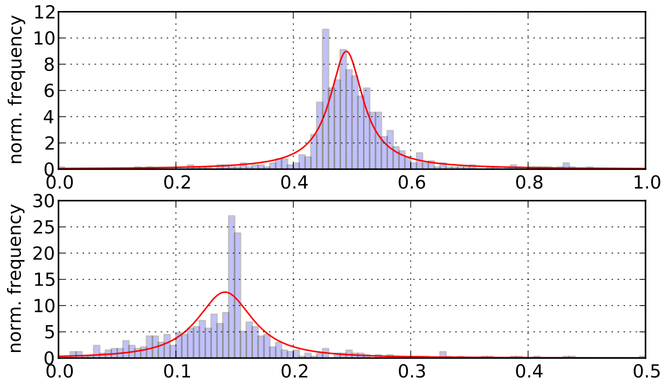

Fig. 4. Determination of constants C_F (upper) and C_M (lower).

ODE solver with event handling that can integrate the current state forward in time and report any zero crossings in the z component. The so found impact time t_i is then used to predict the system state at impact.

V. EXPERIMENTS

A. Model Experiments

1) Generation of the Ground Truth: We used 746 distinct throws for our experiments. All throws were conducted by humans and there were 14 different throwers. For the model evaluation, we used all throws, even those that were not caught by the robot. We investigate the measurements till the first impact on the racket happened, because all consecutive bounces are easier: the vertical speeds are very small if the horizontal reflection worked properly on the first bounce. The following prediction can therefore be initialized with a narrow prior distribution and converges much faster. All later bounces are similar to the second one and can be handled similarly. This is therefore a reoccurring system.

Since the ball was only measured at discrete time intervals, we cannot guarantee that we have really seen the ball at the point of impact. Therefore we have to make an estimate to where the ball really hit the racket. This is done by fitting the physically correct model (see Sec. IV-A2) to all measurements of the throw using a Levenberg-Marquardt algorithm. This fit can than be used to find the state at impact height z_i. This data was used as ground truth for the evaluations. The remaining problem is the estimation of the values C_D and C_M, which need to be determined before fitting individual throws can be properly done. For this, all throws were taken into account. A physically correct model was fitted to each individual throw while treating the initial ball position $\boldsymbol{x}_0$, speed $\boldsymbol{v}_0$, angular velocity $\boldsymbol{\omega}_0$, C_D and C_M as unknowns to be varied. This gave us samples from a distribution over C_D and C_M which can be seen in Fig. 4. The many outliers come from throws that do not fit the model: for example balls that were only rolling on the floor. To cope with the outliers, we fitted a students-t distribution to this data, which gives a sane first mode of $C_M = 0.1418$ and $C_D = 0.4903$. These values have been used to fix C_M and C_D. The ground truth was generated by refitting all throws without C_M and C_D in the parameter set and predicting the impact position. A sample throw with the ground truth can be seen in Fig. 5.

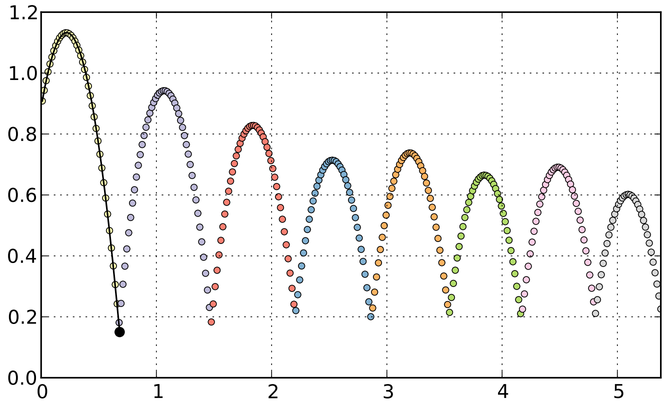

Fig. 5. $t[s] - z(t)[m]$. First 8 bounces of a sample throw. The colored dots define the measurements, each color represents one bounce. The black line is the fitted physically correct model for the first bounce and the black dot is the ground truth of the impact position.

2) Results: To compare the linear model and the physically correct model, we used the following scheme: for each throw, we fed the predictor one measurement at a time and let it predict the impact parameters. We denoted the root squared-error (RSE) in the impact position RSE $\Delta\boldsymbol{x}_i$, the RSE in impact speed RSE $\Delta\boldsymbol{v}_i$, and the absolute error in impact time $|\Delta t_i|$ compared to the ground truth. Errors that were off by more then 5 units (m for position, m/s for speed and s for time) were discarded as outliers. In this way we got a mapping from the number of measurements used for a prediction N to a set of prediction errors $\{\text{RSE } \Delta\boldsymbol{x}_i, \text{ RSE } \Delta\boldsymbol{v}_i, |\Delta t_i|\}$.

The initial state and covariances for the Kalman filters were learned by partitioning the sets of throws into 4 sets of 149 throws and one set of 150 throws. Leave-one-out cross validation was used to learn the initial parameters and covariances and to test them for all throws.

The result of this testing scheme for N over RSE $\Delta\boldsymbol{x}_i$ and RSE $\Delta\boldsymbol{v}_i$ can be seen in Fig. 6. Depicted are mean (solid line), median (dashed line) and one standard deviation around the mean (dotted lines).

The complex model converges faster on the correct impact position which allows for earlier robot movement. The main improvement of the new model is in the impact velocity prediction though: the naive model's performance with predicting the impact speed is not sufficient to catch the initial throw by a human: the ball bounces out of the robots reach immediately. The convergence of the complex model is fast enough to make an excellent prediction in time.

For $N = 15$ the t-test was used to test the hypothesis that the RSE/absolute error of the naive model is smaller or equal to the one of the physically correct model. The test rejected the hypothesis for each impact parameter ($\boldsymbol{x}_i, \boldsymbol{v}_i$, and t_i) for a confidence level of 95%. We therefore have a significant confidence that the complex combination of model and predictor is better than the simple one.

The mean computation time for incorporating one new measurement and making an impact prediction was 0.16 ms for the naive model and 5.97 ms for the physically correct model on the PC controlling the experimental setup. The physically correct model is significantly slower because a differential

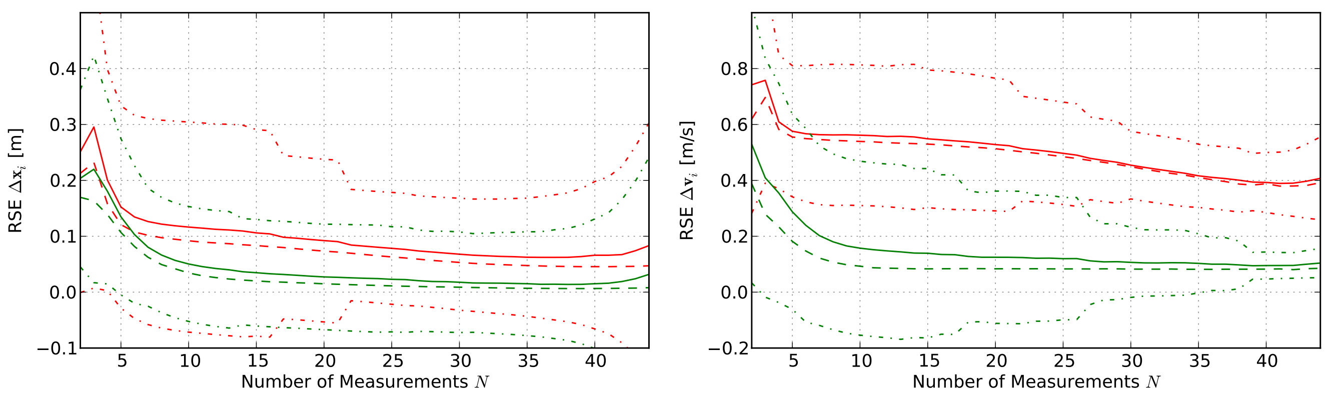

Fig. 6. Prediction performance comparison of impact position (left) and impact speed (right). Red: naive model with Kalman predictor. Green: Physically correct model with unscented Kalman predictor.

equation has to be integrated numerically in each step. Both methods are fast enough to be used in the real-time framework of this work.

B. *Experiments for the complete system*

The physically correct model tracked by the unscented Kalman filter was used for all experiments below.

To test the performance of the complete system more throws were done. To avoid measuring the throwing skills of the human only the throws that the robot was able to catch and bounce five times were taken into account. The robot was then left to bounce the ball for at least 100 times ($\approx$ 2 min).

Three more throws were conducted where the robot was left to juggle for as long as he could.

1) Results: 98 of the 100 throws were ended manually after the 100 bounces, the remaining two throws ended by the ball lying on the racket because it had too little initial impulse to be kept bouncing by the robot (e.g. the parabola it was thrown in was too flat).

The three endurance experiments ended after 43 min, 46 min and 55 min because of a desynchronization of the cameras which were caused by a firewire bus reset triggered by the kernel on the controlling PC.

The performance of the complete systems was also presented in a supplementary video accompanying this paper. It can be found under [8].

VI. SUMMARY

This paper described an experimental setup for a ping-pong ball catching and juggling industrial robot arm. The hardware setup is all off-the-shelve comprised of two cameras, a PC and a small industrial robot. The proposed framework which consists on the image processing and controlling was described. The algorithms employed proved to be fast and stable enough for the task at hand. Two models for the flight of the ball have been discussed: the naive model usually employed in the literature and a novel physically correct model which captures the non linearity of the flight curve. Both models were tracked using corresponding linear and non linear variations of Kalman filters to predict the impact parameters. The naive model was unable to predict the impact speed good enough to allow a ball thrown by a human to be caught, the physically correct model performed good enough to solve the task at hand: It was shown that the robot is able to catch thrown balls and to keep them in the air in 98 of 100 cases and – if the patience of the experimenter allows it – keep juggling the ball for at least 30 min. A video was made to accompany this paper with sample throws and an explanation of the experimental setup [8].

It has been shown that even difficult real-time controlling tasks like tracking, catching and reacting to the physics of a fast moving ball are possible with today's industrial robots. However, the shortcomings of the robots must sometimes be equalized by improving models to capture reality more closely. To find widespread application in industry, the ease of use and flexibilities of the real-time components of today's industrial robots still need to be improved.

REFERENCES

[1] M. Buehler, "Robotic tasks with intermittent dynamics," PhD thesis, Yale University, New Haven, 1990
[2] Kuka Robot Group, "KUKA.Ethernet RSI XML 1.1," KUKA Roboter GmbH, 2007
[3] J. Heikkila and O. Silven, "A four-step camera calibration procedure with implicit image correction," in Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp 1106–1112., 1997.
[4] B. Jähne, "Digital image processing: concepts, algorithms and scientific applications,". Springer-Verlag London, UK, 1991.
[5] S. Julier, J. Uhlmann and H. Durrant-Whyte, "A new approach for filtering nonlinear systems," in Proc. of the American Control Conference, 3, pp 1628–1632. American Automatic Control Council, Evanston, IL, 1995.
[6] A. Nakashima, Y. Sugiyama and Y. Hayakawa, "Paddle juggling of one ball by robot manipulator with visual servo", Proc. IEEE ICARCV 2006.
[7] H. Rapp et al., "The pydc1394 camera library for Python," 2009 https://launchpad.net/pydc1394
[8] H. Rapp, Supplementary video to this paper, 2011 http://www.mrt.kit.edu/rappweb/pprobot_supplementary.mp4 (65 MB)
[9] P. Reist and R. D'Andrea, "Bouncing an unconstrained ball in three dimensions with a blind juggling robot", in Proc. ICRA 2009, pp. 1774–1781, IEEE 2009
[10] A. A. Rizzi, Whitcomb, "Distributed real-time control of a spatial robot juggler" in IEEE Computer, 25, pp. 1224, 1992.
[11] S. Schaal and C. G. Atkeson, "Open Loop Stable Control Strategies for Robot Juggling," in Proc. IEEE International Conference on Robotics and Automation (1993), 3, pp. 913-918
[12] T.L. Vincent, "Controlling a ball to bounce at a fixed height," in Proc. American Control Conference (1995), pp.842-846