Realtime Perception for Catching a Flying Ball with a Mobile Humanoid

Oliver Birbach, Udo Frese and Berthold Bäuml

***Abstract*— This paper presents a realtime perception system for catching flying balls with DLR's humanoid *Rollin' Justin*. We use a two-staged bottom up approach in which we first detect balls as circles and feed these measurements into a multiple hypothesis tracker (MHT). The novel circle detection scheme works in realistic scenes without tuning parameters or background assumptions. We extend the classical multi-hypothesis tracking with prior information about the expected trajectories, therefore limiting the number of hypotheses in the first place. Since the robot starts moving while the ball is still tracked, the cameras shake heavily. A 6-DOF inertial measurements unit (IMU) is integrated to compensate this motion. Using ground-truth from a marker based tracking system we evaluate the metrical accuracy of the motion compensation as well as the tracker's prediction accuracy while in motion.**

I. Introduction

Human society cherishes sports as a noble activity showing mastership in perception, body control, and in tactically pursuing a goal. The same is applicable to robots, where performing a sports activity is an excellent realtime benchmark for perception of dynamic scenes, for motion planning and control, and for action planning. Also, robots doing sports fascinate novices (and experts), because everyone understands what happens and can judge the robot's performance relative to a human. Hence, besides the benchmark view, robotic sport activities also illustrate the relation between humans and robots in a way accessible for everyone.

The activity reported here is DLR's wheeled humanoid robot *Rollin' Justin* [1] catching two tossed balls with his two arms and hands. This paper presents the perception system that tracks and predicts the flying ball(s) from cameras at the robots shaking head (Fig. 1). The subsystem of planning the motion for a single arm-hand unit according for this task was presented in [2] and adapted to *Rollin' Justin* [3].

A. Related Work

As with this work, the task of tracking and predicting balls was studied as part of robotic ball catching systems ([4], [5], [6], [7], [8]). All of these have a static setup using stereo cameras with rather wide baselines. Detecting the ball is done by pixel-wise segmentation, using color ([6], [7], [8]) or using the difference to a reference image [5]. Also, all these systems assume just one flying ball and only limited capability to handle false detections. Therefore, no data association has to be made and estimating the ball's position and velocity from the state is done using Extended Kalman Filters (EKF) ([5], [7]) or fitting a parabola to the measurements from both cameras ([4], [6], [8]).

This work was supported by DFG grant FR2620/1-1.

O. Birbach and U. Frese are with German Center for Artificial Intelligence (DFKI). 28359 Bremen, Germany oliver.birbach@dfki.de

B. Bäuml is with DLR Institute of Robotics and Mechatronics, Münchnerstr. 20, 82234 Wessling, Germany berthold.baeuml@dlr.de

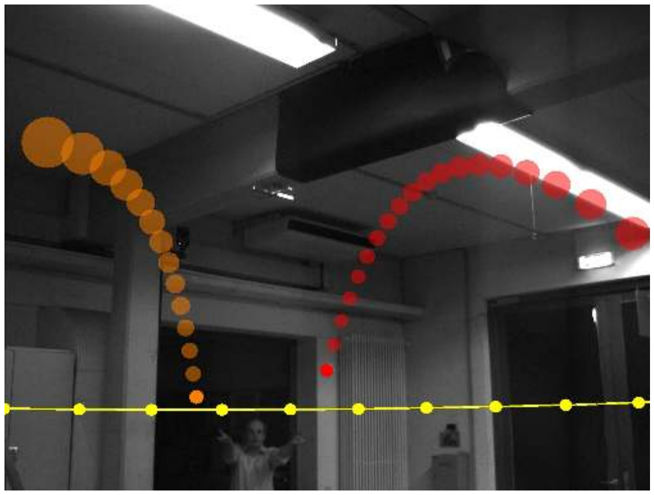
Fig. 1. The vision system tracks and predicts two thrown balls in realtime. Predicted trajectories of the two simultaneously thrown balls as well as an artificially horizon, depicting the camera's orientation, are shown.

Apart from robotic catching systems, Ribnick et al. [9] present a method for detecting arbitrary objects which were thrown. For this, regions of motion are extracted by computing the image's inter-frame difference. The centroids of the regions are then associated to a trajectory by a parabolic fit. Also, Ren et al. [10] present a system for fully automated 3D soccer ball tracking including the possibility to classify the ball's state as rolling, flying, in possession or out of play. Regarding the problem of data association, Yan et al. [11] provide a solution for trajectory generation as the optimal concatenation of tracklets containing true positive measurements and applied their method to tennis ball tracking.

Detecting and tracking balls is also used in broadcast television for augmenting the viewer's experience. Such systems are available for baseball [12], cricket and tennis [13].

B. Challenges and Contribution

In contrast to all aforementioned systems, our cameras are mounted at the humanoid's head. This gives rise to several challenges addressed in this paper: First and foremost, the cameras are not static but move when the robot moves and even shake from the reaction forces of a moving arm. This requires an IMU to compensate, i.e. to distinguish between camera and ball motion and also to measure gravity. It also precludes to use simple difference images for detecting the ball. Second, the baseline is much smaller (0.2m) than possible with stationary cameras. This increases the uncertainty

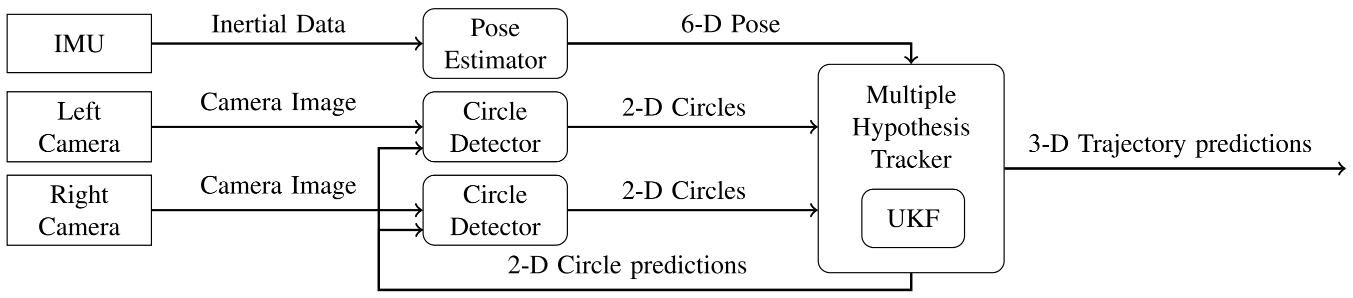

Fig. 2. Data flow of the perception system: Circles are independently detected in both images and passed as measurements to an Multiple Hypothesis Tracker (MHT). The MHT also receives the integrated head pose from the IMU which is needed in the camera measurement model. In the MHT a list of Unscented Kalman Filters (UKF) tracks different hypotheses for different balls, one for each hypothesis of each ball. The UKFs of the balls of the most likely hypothesis are predicted and passed to the motion planner.

in depth direction. We compensate by using high resolution (1600×1200@25Hz) cameras, however making circle detection computationally demanding. Third, the cameras see the ball in front of an undefined background, namely the person throwing it and other spectators. This precludes simple color segmentation. It further requires a sophisticated multi-hypothesis tracker that can discriminate false-alarms, such as someone's head, from the real ball by observing whether it moves like a flying ball over time.

This paper continues our previous work [14], where we showed tracking and prediction of up to three balls which were tossed around by four people outdoors. The novel contributions are: A method for converting the circle detector responses to likelihoods resulting in fewer false-positives (Sec. II), a prior on expected trajectories in the MHT (Sec. III), use of an IMU to compensate for shaking cameras (Sec. IV), integration into the real robot *Rollin' Justin*, and a metric evaluation of the prediction accuracy with a commercial marker based tracking system (Sec. VI). Fig. 2 shows a data-flow overview of the system.

II. Circle Detection

The most popular way to detect a ball in an image is the circle Hough-transform ([15], [16], [17]). Conceptually, it counts the number of pixels along every hypothetical circle where the image gradient is radial (Fig. 3a). This criterion involves two hard thresholds. One on the gradient norm for classifying edge pixels (usually hand-tuned) and one on the angle δ between the gradient and the radial direction (often equivalent to 1 pixel at the center). Many methods threshold the gradient, operating on a binarized edge image [18].

Hard thresholds often impair robustness and in particular the gradient norm threshold depends highly on scene contrast and illumination requiring frequent adjustment. Hence, we propose a circle detection criterion that is invariant under linear illumination changes and avoids hard thresholds and tuning parameters. By passing the best $N = 25$ local maxima to the MHT we also avoid thresholding the response itself.

A. Contrast Normalized Sobel Filter (CNS)

Gradient filters (e.g. the Sobel) ignore additive intensity changes, but scale linearly with intensity. They could be normalized for illumination invariance by dividing by the gradient norm, i.e. using the direction only. However, this approach gives random results in (almost) uniform image areas. Instead, we adopt the normalization in template matching and divide by the square root of the local image variance.

$$C = \frac{\sqrt{2}\left(\left(\begin{smallmatrix}+1 & 0 & -1\\ +2 & 0 & -2\\ +1 & 0 & -1\end{smallmatrix}\right) * I,\ \left(\begin{smallmatrix}+1 & +2 & +1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{smallmatrix}\right) * I\right)^T}{\sqrt{16\left(\begin{smallmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{smallmatrix}\right) * I^2 - \left(\left(\begin{smallmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{smallmatrix}\right) * I\right)^2 + \varepsilon^2}} \tag{1}$$

The image is I and $*$ denotes convolution. We use the Sobel filter in the nominator and compute the local image variance by $V(X) = E(X^2) - E(X)^2$ with a weighting mask in the denominator. $\varepsilon = 1$ is the discretization unit of pixel intensity preventing $0/0$ in constant image areas. Sobel filters combine $\left(\begin{smallmatrix}1 & 2 & 1\end{smallmatrix}\right)$ low-pass and $\left(\begin{smallmatrix}+1 & 0 & -1\end{smallmatrix}\right)$ differentiation. So the weighting mask is the corresponding combination $\left(\begin{smallmatrix}1 & 2 & 1\end{smallmatrix}\right) * \left(\begin{smallmatrix}1 & 2 & 1\end{smallmatrix}\right)^T$ of low-pass filters. The factors $\sqrt{2}$ and 16 normalize the vector length to $[0 \dots 1]$. Intuitively, the norm of the CNS filter indicates *gradient purity* rather than *gradient intensity* being 1 for a pure linear gradient and gradually lower when there are other components in the local image (Fig. 3c).

B. Circle Response

We use the squared scalar product of the CNS with the radial direction as an indicator of how well the local image at a point along the circle looks like a circle (Fig. 3d)

$$R(x,y,\alpha) = \left(\begin{pmatrix}\cos\alpha\\ \sin\alpha\end{pmatrix} \cdot C(x,y)\right)^2 = |C(x,y)|^2 \cdot \cos^2\delta \tag{2}$$

The scheme (Fig. 3a) is similar to Hough-transform. However, it replaces the hard gradient intensity threshold with a soft illumination invariant indicator of gradient purity $|C(x,y)|^2$. The hard threshold on the angle δ between the gradient and the radial direction is replaced with a soft penalty factor $\cos^2(\delta)$ (Fig. 4).

The overall response for a circle x_c, y_c, r is obtained by

$$CR(x_c,y_c,r) = \frac{1}{2\pi}\int_{\alpha=0}^{2\pi} R(x_c + r\cos\alpha, y_c + r\sin\alpha, \alpha)\,d\alpha \tag{3}$$

integrating (2) along the circle (Fig. 3e). The range of R is limited with ≈ 0.12 for a random pixel in our images and 1 for a perfect edge. When instead using gradient

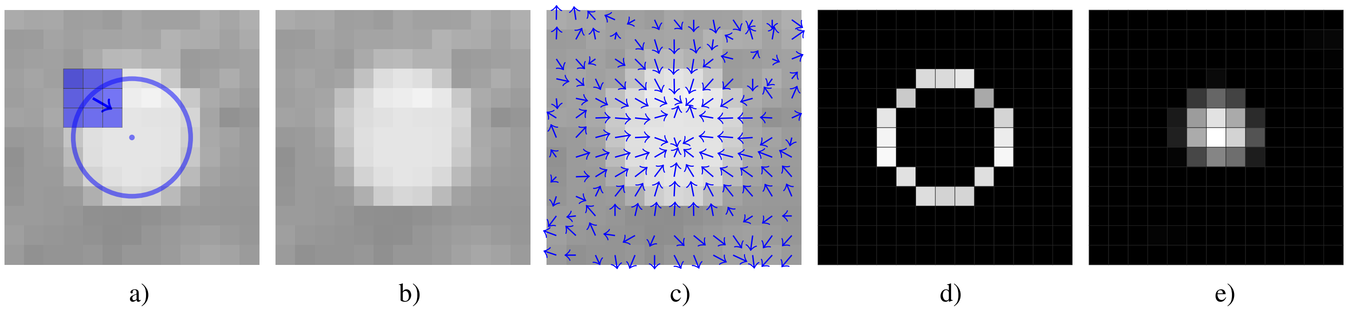

Fig. 3. The proposed circle detector: **a)** overall idea of how well the gradient vector at a circle pixel has radial direction, **b)** input image, **c)** CNS (1) vector image, **d)** response (2) at every circle pixel for a fixed circle, **e)** response (3) for $r = 3$ and different center x_c, y_c.

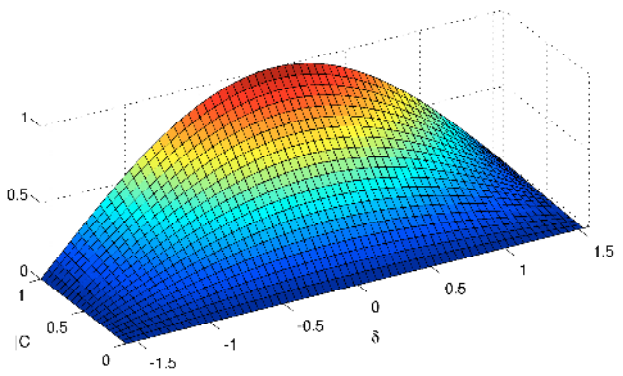

Fig. 4. Plot of the response (2) as a function of CNS norm $|C|$ and angular mismatch δ between the image gradient and the radial direction. The corresponding plot for the classical Hough-transform is 1 inside a hand-tuned rectangle and 0 outside.

intensity to weight the responses, we experienced that a small high contrast edge dominates the integral creating false responses, e.g. along window edges. The same happens when normalizing the contrast not per pixel but on the whole circle, e.g. by the average gradient intensity.

A mathematical derivation [14], omitted here for lack of space, further motivates the filter choice in (1) and the square in (2). It shows that $R(x,y,\alpha)$ is the fraction of image contrast around (x,y) that is a linear gradient in direction α.

Finally, smoothing the image with the above mentioned filter prior to (1) improves gradient precision and responses.

C. *Multiscale Approach and Modifications*

While the Hough-transform's hard thresholds impair robustness, they facilitate efficiency: For every pixel above the gradient threshold, only the circle centers along the line in gradient direction need to be considered.

Evaluation of (3) for our image format and circle sizes requires to compute (2) $\pi r_{\max}^2 wh = 34.7 \cdot 10^9$ times, where w and h are the image's width and height respectively. From the formula one can see, that by downscaling the image to half the resolution, computation time reduces by a factor of 16. Hence we use a multiscale pyramid, detecting every circle at the coarsest level where this is possible and refine progressively at finer levels. This needs only $32.3 \cdot 10^6$ evaluations.

TABLE I

DIFFERENT LEVELS OF THE MULTISCALE CIRCLE DETECTOR. THE LAST COLUMN ASSUMES SEARCHING FOR $N = 25$ CIRCLES. OVERALL $32.3 \cdot 10^6$ EVALUATIONS ARE NEEDED INSTEAD OF $34.7 \cdot 10^9$ FOR SINGLE SCALE.

scale	size	r	equiv. r on 1:1	# evaluations (global search)	# evaluations (refinement)
1:1	1600×1200				4.17 M
1:2	800× 600				2.07 M
1:4	400× 300	3…7	12…31	19.80 M	0.98 M
1:8	200× 150	4…7	32…63	4.43 M	0.50 M
1:16	100× 75	4…4	64…79	0.27 M	0.10 M
total			12…79	24.50 M	7.81 M

On the downside, the multiscale detector is slightly less robust, as it is harder to detect a small circle on a coarse level than the corresponding large circle on the fine level. Hence, the MHT provides predictions for flying balls which are additionally refined on the finest level. Table I shows the policy which radius is detected where.

D. *Conversion to Likelihoods*

The detector described so far has a tendency to find small false circles, because it is more likely that the average response of few pixels is high than of many. To compare responses probabilistically, we compute the likelihood that the response is coincidence. For that, the distribution of responses R for a radius r is assumed to be a Gaussian $\mathcal{N}(\mu, (\sigma r^\gamma)^2)$ and μ, σ, and γ are learned from training data.

$$L = \frac{1}{\sqrt{2\pi}(\sigma r^\gamma)} \exp\left(-\frac{(R-\mu)^2}{2(\sigma r^\gamma)^2}\right) \tag{4}$$

$$-\ln L = \Bigg(\underbrace{\frac{1}{2(\sigma r^\gamma)^2}}_{a} R - \underbrace{\frac{1}{2(\sigma r^\gamma)^2}\mu}_{b}\Bigg)^2 + \underbrace{\ln\left(\sqrt{2\pi}\sigma r^\gamma\right)}_{c} \tag{5}$$

For efficiency reasons we use negative log-likelihoods (NLL), so the computation in (5) reduces to evaluating $(aR-b)^2 + c$, with a,b,c precomputed for each r.

The conversion to likelihoods works well and is theoretically elegant as the result could be used as the false alarm likelihood in the tracker. This gives the MHT the information that low response circles are more likely false alarms than high response circles. However, we also tried another alternative: The number of circles accepted as measurements by the MHT is counted. For this, circle measurements from a set of trajectories are extracted. The circle detector is then applied on the source images of the trajectories and the number of correct circle detections is then maximized with regard to μ, σ, and γ by evaluating all reasonable combinations in an offline fashion. This approach works even better and is hence use it by us. Unfortunately, it has no probabilistic interpretation anymore.

E. *Efficient Implementation*

A major challenge was processing 2×2MPixel images in $<$25ms leaving $>$15ms for the MHT. We therefore optimized the implementation using OpenMP multi-core and Single Instruction Multiple Data (SIMD, Intel SSE) parallelization.

The SIMD implementation of (1) processes 8 pixel at a time. All filters are factored into X and Y. First, in one line

$$\left(1\ 2\ 1\right) * I, \qquad \left(+1\ 0\ -1\right) * I, \qquad \left(1\ 2\ 1\right) * I^2 \tag{6}$$

are computed as 16, 16, and 32 bit integers. The result is stored for two lines and then filtered vertically, resulting in

$$\left(+1\ 0\ -1\right)^T * \left(1\ 2\ 1\right) * I, \qquad \left(1\ 2\ 1\right)^T * \left(1\ 2\ 1\right) * I\ , \tag{7}$$

$$\left(1\ 2\ 1\right)^T * \left(+1\ 0\ -1\right) * I, \qquad \left(1\ 2\ 1\right)^T * \left(1\ 2\ 1\right) * I^2. \tag{8}$$

Then (1) can be evaluated as a floating point number and converted back to signed 8 bit integer to save memory bandwidth. The downsampled image needed for the next coarser scale is obtained by averaging the low-pass filter over 2×2 pixel. It is important to do all these computations in one pass to save memory bandwidth. All this runs parallel with each core computing a horizontal stripe in the image.

Most performance critical is the SIMD implementation of (3) resp. (2) being evaluated $32.3 \cdot 10^6$ times (Tab. I). The values of sine and cosine and the address of the circle pixel relative to the center are precomputed. The memory access pattern is irregular with respect to α (around the circle) and r (different circles). But it is regular with respect to x_c, y_c, so we compute the response for a block of 16×16 (refinement: 8×8) circle centers and a fixed radius at a time. The next outer loop is iterating over different radii. Since there is a large overlap in the accessed pixel, these are usually cached. The outermost loop iterates the circle center blockwise over the image.

Intel's SSE3 includes an instruction `pmaddubsw` that multiplies 2×16 bytes adding adjacent products (16 bit). This computes $\cos \alpha C_x + \sin \alpha C_y$. However, one operand is unsigned, so we shift the CNS image by 1, i.e. 128, and correct the product by subtracting $\cos \alpha + \sin \alpha$. The result is squared (2) and accumulated (3). Overall, 6 instructions perform 8 evaluations of (2) in 2.6 cycles, 0.12ns/eval.

Again, the computation is executed in parallel on horizontal stripes and refinement is also parallelized. Figure 5 shows the computation time over image size and number of cores. As we hoped, it scales well up to four cores.

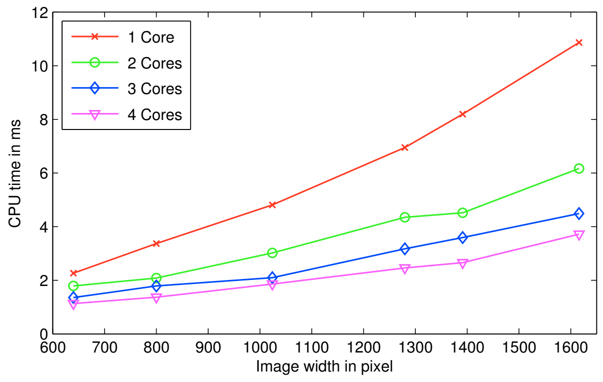

Fig. 5. Computation time of the circle detector on 4:3 images of different size with 1 to 4 cores on a desktop Intel Xeon^{TM} W3520 @2.67GHz.

F. *Discussion*

The presented circle detector is illumination invariant and contains neither hard thresholds nor parameters that need to be tuned to the current scene or lighting situation. This statement holds with two restrictions:

First, of course at some point the system takes a hard decision by letting the robot start moving, However, in passing as many detected circles as possible to the MHT, we delay this decision. This is better, because the MHT has much more context, namely the whole image sequence, to decide, whether something is a ball or not.

Second, the parameters in (5) and the multiscale policy in Table I needed somehow to be chosen. However, the first were learned from training data and we once tuned the second for the available computation time. Neither was changed when moving from the lab to other settings. So, we believe they are not hand-tuned parameters.

Finally, on a desktop PC the detector was even faster than needed. This allowed moving to an 2.5 times slower embedded PC inside Rollin' Justin using 2 cores per camera.

III. Multi-Hypothesis Tracker

We use Cox' [19] extension of Reid's [20] Multi-Hypothesis Tracker (MHT) with an Unscented Kalman Filter (UKF) [21, §3.4] for every hypothesis as the core algorithm for predicting one or more trajectories from the set of detected circles. We give an intuitive description here, details are found in [14] and [19].

A. *Algorithm Overview*

The MHT is a probabilistic algorithm for estimating target states (here ball position and velocity) from uncertain measurements. It is related to a Kalman Filter (KF), however the MHT considers not only measurement noise, but also probabilities for a measurement not to be detected and for a spurious measurement to occur. It models target dynamics (here ball flight) as a KF, but also randomly appearing and disappearing targets, i.e. the number of targets is estimated.

Conceptually, the MHT maintains a mixture distribution, where every hypothesis is a fixed assignment of measurements to targets. It is represented by a probability and a list of Gaussians. The probability defines how likely it is that this assignment is true and the list of independent Gaussians gives the distribution of the target state conditional on the assignment. With a fixed assignment, every Gaussian can be updated by a classical filter, in our case a UKF where the concrete formulas of ball-flight and of mapping a world ball to an image circles are implemented. The MHT dynamic step simply consists of executing a dynamic step on each UKF. The measurement step in principle updates every Gaussian in every hypothesis with every measurement systematically going through all possible assignments and creating many new hypotheses. The probability of an hypothesis is multiplied by probabilities for "everything that happened", i.e. constants for missed observations (P_S), spurious measurements (λ_F), appearing targets (λ_N) and disappearing target (P_χ). In the case of assigning a measurement to a target, the UKF measurement update returns the so-called Mahalanobis-distance, a measure of consistency. It is converted into a probability and multiplied into the probability of the hypothesis.

This is the main mechanism of the MHT: Measurements that are consistent with the state, i.e. over time with the dynamic model, lead to hypotheses with high probability. Inconsistent measurements generate low probability hypotheses, lower than the hypothesis that they were spurious.

B. *Implementation*

The description above is conceptual, the actual implementation [19] is optimized with clever data-structures, an efficient assignment algorithm, and pruning of hypotheses. In our UKF the measurement model is the usual pin-hole camera with radial distortion, with measurement uncertainty $\sigma_{x,y}$ for the center and σ_r for the radius of the circle. The dynamic model [14], [22] is

$$\ddot{b} = g - \alpha \cdot \|\dot{b}\| \cdot \dot{b}, \tag{9}$$

with b ball position, g gravity and air-drag coefficient α. Dynamic noise is considered as σ_Q.

The MHT is executed twice per stereo-frame, first on the left then on the right image. This way it finds both the stereo correspondences and the correspondences over time.

C. *Prior*

There are many false-alarms from (roughly) circular looking objects. Most of them are discarded but some by coincidence resemble a ball trajectory and become tracks in the MHT. To rule them out, we include the prior information, that the ball is thrown from some typical position and towards the robot. The information is learned from training data (currently simulated). For that, a 6-D Gaussian is fitted to a sample of initial states (position and velocity) from several tracked balls. This Gaussian defines with uncertainty from where a ball is typically thrown (position), in roughly what trajectory (velocity) and that it is thrown towards the robot (correlation between both). Theoretically, it could be used to initialize the UKF. Practically, there are two problems:

First, the prior Gaussian is large causing linearization errors in the UKF for the first visual measurement. Instead,

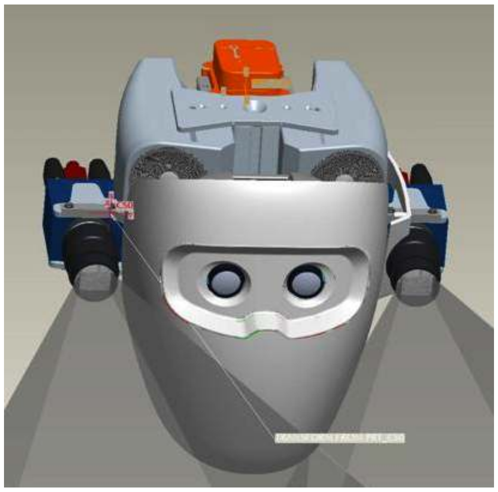

Fig. 6. The head as the camera-IMU-rig where the IMU is the orange box on the top. The angle of view for each camera is 49°. The delay between physical event and arrival at the PC are 40ms for the cameras and 5ms for the IMU.

we map the first measurement (circle center and radius) to a position. The covariance of that position is obtained by sigma-point propagation through that mapping, i.e. linearizing at the position derived from the first measurement, not at the prior's mean. The resulting Gaussian is then fused with the prior using the Kalman filter update equation, since once the measurement is converted to a position, the measurement function is linear. The result is the state of the UKF which processes all following measurements as usual.

Second, we want the system to act independently from which direction the ball is thrown. This is enforced by rotating the initial states to a normalized position for learning the Gaussian prior and rotating the Gaussian back to the actual position before fusing it.

IV. Inertial Pose Estimation

The state $(b, \dot{b})$ must be expressed in a static coordinate-system, because only there (9) holds. A *static* camera could be calibrated relative to a world frame with known gravity vector, e.g. the robot base. For Rollin' Justin this is more difficult, because while the kinematic chain between head and arms is precise, the torso has elasticity and hysteresis and the wheels have dampers and slip on the floor. Even, when the arms move the reaction forces jiggle the head.

Our solution is to view the head-arm system as a self-contained "catching device" that is *somehow* moved by the rest of the robot and this motion is exclusively obtained from a head-mounted IMU (Fig. 6). This view makes the perception module independent, allowing to test it with a manually moved camera-IMU-rig. As a matter of fact, this was essential, as Rollin' Justin's experimental time is scarce and the perception system could be developed independently.

A. *IMU Integration*

So, our world frame is defined by the IMU pose at start up rotated such that Z points against measured gravity. The head

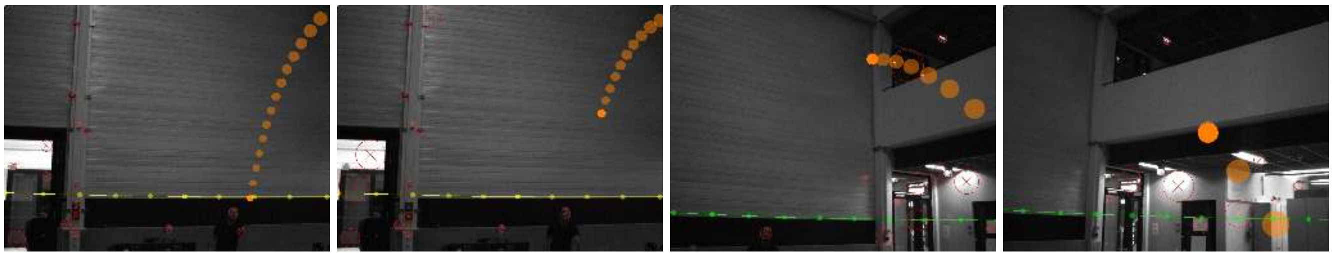

Fig. 7. Four images from an image sequence recorded during a ball catch showing detected circles, the detected track, its predicted trajectory and an artificial horizon. A yellow horizon indicates that only orientation whereas a green horizon indicates that the full 6-D pose is estimated, see (Sec. IV).

pose is tracked in that frame by the usual IMU integration:

$$q_{t+\Delta t} = q_t \cdot \exp(\Delta t \cdot \omega_t) \quad (10)$$

$$v_{t+\Delta t} = v_t + \Delta t \cdot (q_t \cdot a_t \cdot \bar{q}_t - g) \quad (11)$$

$$x_{t+\Delta t} = x_t + \Delta t \cdot v_t \quad (12)$$

Here x is position, v velocity, q orientation as a quaternion, a and ω are acceleration and angular velocity from the IMU, and Δt is the time-step. The Rodriguez formula $\exp(v)$ rotates around v by $|v|$. Integration of low-cost IMUs suffers from drift and works for seconds at most (Sec. VI-A). Hence, during stand-still we switch to orientation tracking [23], [24], [25], fixing $v_t = 0$, but estimating gyro-bias and orientation.

Future work will be to replace this switching by an integrative approach. It could use the prior information that the robot's velocity is limited, just as orientation trackers do but still integrate full 6-DOF motion over a short time. This would allow the robot to start a catch while being in motion.

B. Structure of the Estimation Problem

To understand the consequences of IMU drift one has to consider the structure of ball prediction as an estimation problem. The problem is invariant to translation and rotation around gravity before the ball appears, because all observations and actions are relative to the robot's head. Except for air-drag, it is even invariant to linear motion due to the inertial-frame principle in physics. This means, the robot's velocity when observing the ball simply adds to the estimated velocity of the ball itself cancelling out in predictions relative to the robot. The air-drag term $-\alpha\|\dot{b}\|\dot{b}$ violates this invariance assuming static air in the world frame. Hence, an velocity error acts as a "false wind".

So, with limited velocity error, only the IMU error *during* the ball-flight actually affects the prediction accuracy. Even better, the error from the first phase of the ball-flight affects the final prediction only slightly. Consider the contribution of a measurement to the ball position relative to the robot in some moment. It is affected only by the IMU error accumulated since that measurement. So early measurements sustain larger errors than late ones, but they are less precise anyway due to the distance from the cameras. So, while we cannot quantify this, only a fraction of the IMU error accumulated during ball-flight actually affects the final prediction.

A further remark outlines the "relative" structure: It is easily derived that in stereo triangulation depth error relates to lateral error as depth to baseline, i.e. by a factor of ≈ 20 here. Hence, in calibration and detection, a relative error between both cameras is worse than a common-mode error. Similarly, after observing the ball for t_O and predicting it for t_P a relative error during the observation is amplified by a t_P/t_O (≈ 5 for early predictions), a common-mode error not.

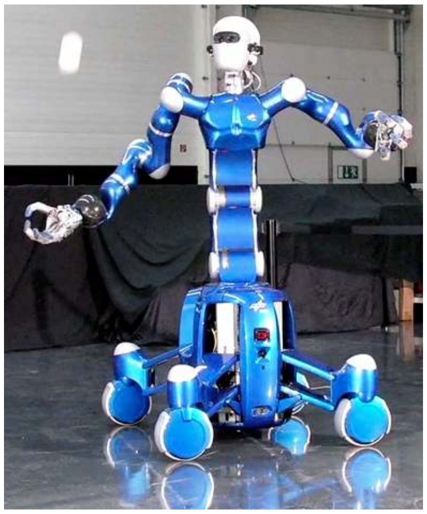

Fig. 8. DLR's *Rollin' Justin* catching a ball. For robust catching, the necessary accuracy is about 2cm in space and 5ms in time.

V. Calibration

Considerable effort has been spent in calibrating the system to meet the required accuracy. The intrinsic parameters as well as the relative pose of both cameras and the IMU was calibrated with a horizontally aligned checkerboard and using g as a vertical reference (RMS 0.33 px, 0.25°). This method only provides rotation, translation is measured manually.

All predictions are sent to the planner in the robot's head frame. Determining the relationship between this frame and the cameras was done by estimating the extrinsic parameters of the cameras with a single marker in both robot's hands and forward kinematics (RMS 1.5 px).

To provide ground truth, a rigid collection of markers was mounted on the robot's head. This allows an external tracking system [26] to measure the 6-D head pose. For that, the relationship between cameras and the marker collection was calibrated. Circles of thrown balls were extracted and fitted to trajectories using least-squares estimation. These and the corresponding ones obtained from the external tracker were

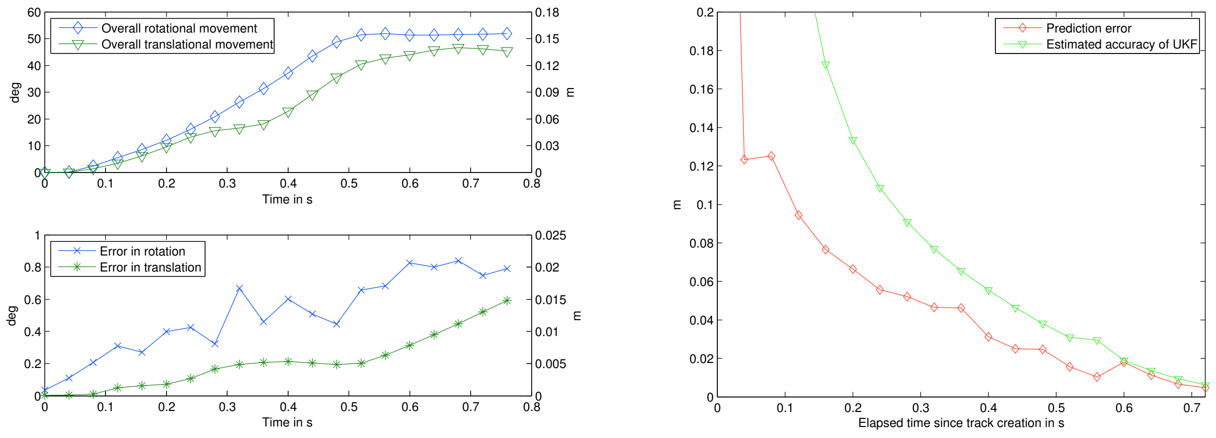

Fig. 9. **Left)** Overall rotational and translational movement (**top**) and the error (**bottom**) of the head pose during a catch made by the IMU during integration over time w.r.t. to ground-truth. **Right)** Prediction error over time as well as the estimated accuracy by UKF obtained by propagating the covariance.

then used to fit the desired transformation (RMS $0.01m$).

By using the marker based tracking system, the manual measurement of the translation between camera and IMU can be refined. By fitting the translation to the translational component of a head movement sensed by the IMU and the marker based tracking system, a more accurate representation of the frame is achieved (RMS $0.005m$).

VI. Experimental Evaluation

Beside the catch-rate (which is about 80%) as a practical indicator, separate experiments involving an external tracking system as ground-truth were conducted to assess the performance of the perception system. In all experiments a ball (8.5cm diameter, wrapped into retro-reflective foil), was thrown from about $5-7$m away towards the robot. See the sequence in (Fig. 7) for a view from the robot. The robot acted accordingly (Fig. 8), rotating its head to keep the ball in the camera image and moved the arm to the catch configuration as computed by the planning algorithm during the flight. The circle detector was instructed to detect the best $N = 25$ local maxima. Circles that lie within circles were excluded, as commonly observed at balls. This set of circles is then passed to the MHT which was configured according to (Tab. II). Here, two probabilities depend on the state of the track: If it is outside the image it is always a missing observation ($P_S = 1$). If it hits the ground the track ends ($P_\chi = 1$). Although only one ball was thrown, the results hold for multiple balls since multiple balls only pose a more difficult data association problem leaving the accuracy (from the UKF) unaffected.

A. IMU Integration

Metrical accuracy of the pose estimation using the IMU is presented in (Fig. 9) on the left. The plot shows the overall movement of the IMU (decomposed into rotational and translational displacement) since the robot indicated it will start to move soon as well as the error of the head pose estimation using the IMU. The movement lasts about 0.76s. For the rotation the total displacement is 51.9°. The

TABLE II
Parameters of MHT and UKF, confer Sections III-A, III-B

MHT				UKF	
λ_N	$1.08 \cdot 10^{-6}$	liklihood-ratio	0.01	$\sigma_{x,y}$	1.5px
λ_F	$6.15 \cdot 10^{-9}$	#hypotheses	10	σ_r	$0.15\%r$
P_χ	0 (1)	N-scan-back	10	σ_Q	0.1 m/s^2
P_S	0.05 (1)				

error in orientation increases over time reaching a maximum displacement of 0.8° at the end. Limiting the error at this stage is crucial, since due to wrong orientation estimation gravity and real acceleration caused by movement can not be distinguished. For the translation, the head travels about 14cm during the catch. The error increases over time (0.18cm at 0.25s and 0.53cm at 0.48s) with a final error of 1.48cm.

B. Tracking

Successful catches rely on accurate early predictions. To evaluate the prediction accuracy over time, we compared the predicted trajectory with the last 3D measurement of the externally tracked ball right before it hits the hand in (Fig. 9) on the right. Two curves are given: The overall error of the predicted position and the accuracy estimate of the UKF state, giving an intuition which overall accuracy can be expected.

The prediction is quite accurate right from the start (12cm after 0.04s), and becomes better in the following time steps, temporarily decreasing when the movement of the robot kicks in. From there, the accuracy gets better reaching a final accuracy of about 0.5cm. Unfortunately, this value is not of any particular value for the success of the catch, since the planning stage needs some time to react to these measurements. Usually, measurements obtained $0.16-0.2$s before the catch are used for the final catch position, which in this case have an accuracy of about 1.5cm.

C. Computing Time

As mentioned earlier, the perception system runs inside the mobile platform in an embedded system equipped with a Intel Core^{TM}2 Quad Q9000 @2.00GHz. This considerably changes the computation time. Processing the stereo images in parallel with two cores per image takes about 25ms. MHT takes about 5ms while idling and 10ms while tracking one or more balls. This leaves room for another 5ms which are reserved for difficult data association situations. The headpose is updated by integrating IMU measurements continuously and takes less than 2ms. Incorporating the latency between the physical event and the data's arrival, it takes about 75ms from capturing flying balls until the states are updated, predicted into the future and sent to the planning algorithm.

VII. Conclusion and Lessons Learned

We learned several lessons during this project which we believe are valuable for similar applications:

First, realtime is more than performance. Like many computer vision researchers we developed our software on Linux and not on a realtime OS, e.g. QNX, resulting in lost time-slices in the order of 50ms. This issue was hard to track down and required to deactivate several hardware components on the affected PC.

Second, for sensor fusion data must be timestamped w.r.t. a common clock and this is poorly supported by current hardware and drivers. Most sensors have a regular frequency and some even provide timestamps, but usually with respect to an internal clock. Clock synchronization, e.g. IEEE1588, would be an ideal solution but few sensors support a query-answer protocol needed for that. We resorted to timestamp all sensor data upon arrival on the PC and postprocessed them by a filter to find outliers and restore their periodicity. Overall, in our experience timestamping in a system built from components introduces many difficulties that could be avoided by better sensor protocols.

Third, accuracy problems are difficult to track down. Many different sources can deteriorate prediction accuracy, ranging from sensor noise over calibration and timing problems to ordinary program errors. The 3D tracking system was essential in investigating these issues, since it allows to quantify errors in 3D as a function of time. Nevertheless, it must be mentioned, that the tracking system itself creates additional problems regarding timestamps and calibration.

To make a virtue out of necessity: One contribution of robotic sport activities is to operate robots in a regime that reveals problems which remain unnoticed in more forgiving applications. This is actually similar to sports training, where the goal is to come closer and closer to the level of performance the human body can do in principle.

References

[1] C. Borst, T. Wimböck, F. Schmidt, M. Fuchs, B. Brunner, F. Zacharias, P. R. Giordano, R. Konietschke, W. Sepp, S. Fuchs, C. Rink, A. Albu-Schäffer, and G. Hirzinger, "Rollin' justin: Mobile platform with variable base," in *Proc. of the IEEE Intern. Conf. on Robotics and Automation*, 2009, pp. 1597–1598.

[2] B. Bäuml, T. Wimböck, and G. Hirzinger, "Kinematically optimal catching a flying ball with a hand-arm-system," in *Proc. of the IEEE/RSJ Intern. Conf. on Intelligent Robots and Systems*, 2010, pp. 2592–2599.

[3] B. Bäuml, F. Schmidt, T. Wimböck, O. Birbach, A. Dietrich, M. Fuchs, W. Friedl, O. Eiberger, M. Grebenstein, C. Borst, U. Frese, and G. Hirzinger, "Catching flying balls and preparing coffee: Humanoid rollin justin performs dynamic and sensitive tasks," in *Proc. of the IEEE Intern. Conf. on Robotics and Automation*, 2011.

[4] B. Hove and J. Slotine, "Experiments in robotic catching," in *Proc. of the American Control Conf.*, 1991, pp. 380–385.

[5] U. Frese, B. Bäuml, S. Haidacher, G. Schreiber, I. Schaefer, M. Hähnle, and G. Hirzinger, "Off-the-shelf vision for a robotic ball catcher," in *Proc. of the IEEE/RSJ Intern. Conf. on Intelligent Robots and Systems*, 2001.

[6] M. Riley and C. G. Atkeson, "Robot catching: Towards engaging human-humanoid interaction," *Autonomous Robots*, vol. 12, pp. 119–128, 2002.

[7] C. Smith and H. I. Christensen, "Using COTS to construct a high performance robot arm," in *Proc. of the IEEE Intern. Conf. on Robotics and Automation*, 2007.

[8] G. Bätz, A. Yaqub, H. Wu, K. Kühnlenz, D. Wollherr, and M. Buss, "Dynamic manipulation: Nonprehensile ball catching," in *Proc. of the IEEE Mediterranean Conf. on Control and Automation*, 2010.

[9] E. Ribnick, S. Atev, O. Masoud, N. Papanikolopoulos, and R. Voyles, "Detection of thrown objects in indoor and outdoor scenes," in *Proc. of the IEEE/RSJ Intern. Conf. on Intelligent Robots and Systems*, 2007.

[10] J. Ren, J. Orwell, G. Jones, and M. Xu, "Real-time 3d soccer ball tracking from multiple cameras," in *British Machine Vision Conf.*, 2004.

[11] F. Yan, A. Kostin, W. Christmas, and J. Kittler, "A novel data association algorithm for object tracking in clutter with application to tennis video analysis," in *IEEE Intern. Conf. on Computer Vision and Pattern Recognition*, 2006.

[12] A. Guziec, "Tracking pitches for broadcast television," *Computer*, vol. 35, no. 3, pp. 38–43, March 2002.

[13] N. Owens, C. Harris, and C. Stennett, "Hawk-eye tennis system," in *Intern. Conf. on Visual Information Engineering*, 2003, pp. 182–185.

[14] O. Birbach and U. Frese, "A multiple hypothesis approach for a ball tracking system," in *Computer Vision Systems*, ser. LNCS, M. Fritz, B. Schiele, and J. Piater, Eds., 2009, vol. 5815, pp. 435–444.

[15] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Comm. of the ACM*, vol. 15, no. January, pp. 11–15, 1972.

[16] C. Kimme, D. Ballard, and J. Sklansky, "Finding circles by an array of accumulators," *Comm. of the ACM*, vol. 18, no. 2, pp. 120–122, 1975.

[17] H. Yuen, J. Princen, J. Dlingworth, and J. W. Kittler, "A comparative study of hough transform methods for circle finding," in *Proc. of the Alvey Vision Conf.*, 1989.

[18] D. Scaramuzza, S. Pagnottelli, and P. Valigi, "Ball detection and predictive ball following based on a stereoscopic vision system," in *Proc. of the IEEE Intern. Conf. on Robotics and Automation*, 2005, pp. 1573–1578.

[19] I. J. Cox and S. L. Hingorani, "An efficient implementation of Reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 18, no. 2, pp. 138–150, 1996.

[20] D. B. Reid, "An algorithm for tracking multiple targets," *IEEE Trans. on Automatic Control*, vol. AC-24, no. 6, pp. 843–854, 1979.

[21] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[22] N. de Mestre, *The Mathematics of Projectiles in Sport*. Cambridge University Press, 1990.

[23] E. Kraft, "A quaternion-based unscented kalman filter for orientation tracking," in *Proc. of the Intern. Conf. of Information Fusion*, 2003, pp. 47–54.

[24] J. L. Marins, X. Yun, E. R. Bachmann, R. B. McGhee, and M. J. Zyda, "An extended kalman filter for quaternion-based orientation estimation using marg sensors," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and System*, 2001, pp. 2003–2011.

[25] A. Kim and M. Golnaraghi, "A quaternion-based orientation estimation algorithm using an inertial measurement unit," in *Position Location and Navigation Symposium*, 2004, pp. 268–272.

[26] Advanced Realtime Tracking. GmbH, "ARTtrack2," 2010, `http://www.ar-tracking.de/`.