Supplementary Information: 3-D rotation tracking from 2-D images of spherical colloids with textured surfaces

Vincent Niggel, Maximilian R. Bailey, Carolina van Baalen, Nino Zosso, and Lucio Isa*

Laboratory for Soft Materials and Interfaces, Department of Materials, ETH Zurich, CH-8093, Zurich, Switzerland

E-mail: lucio.isa@mat.ethz.ch

Particle characterization and imaging

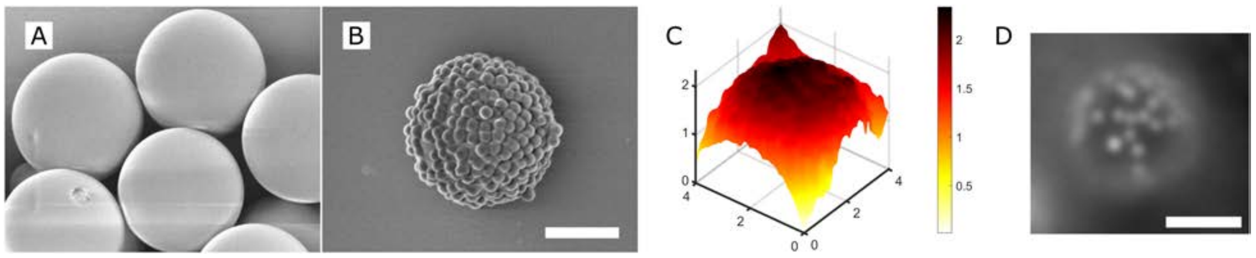


Figure SI1: (A) SEM image of the core silica particles (diameter = 3.5 μm). (B) SEM image of a fluorescent raspberry particle (diameter = 4 μm). Scale bar: 2 μm - both (A) and (B). (C) AFM topography image of the surface a raspberry particle, used to measure its roughness (units in μm). (D) Fluorescence image of a raspberry particle. The random distribution of fluorescent asperities forms a unique pattern that allows tracking rotation. Scale bar: 2 μm.

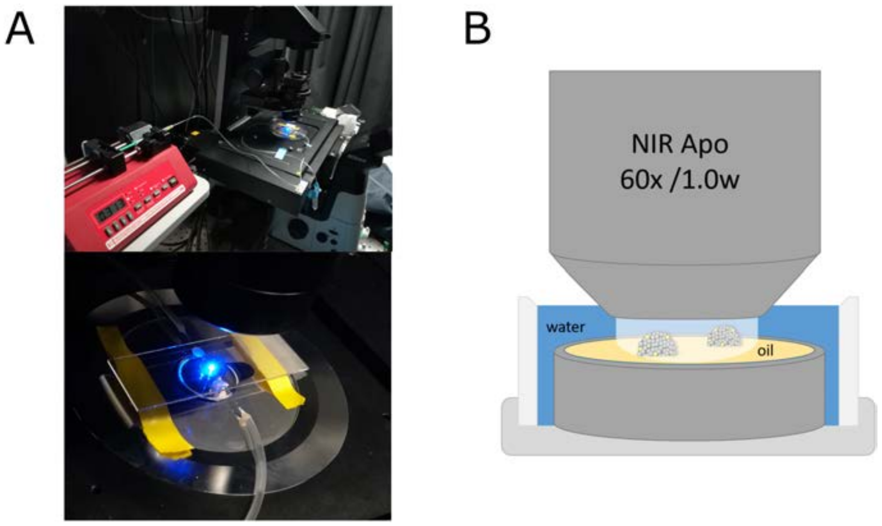


Figure SI2: (A) Images of the setup for the particles flowing in a capillary channel. (B) Sketch of the experimental setup used for the particles imaged at a water/oil interface.

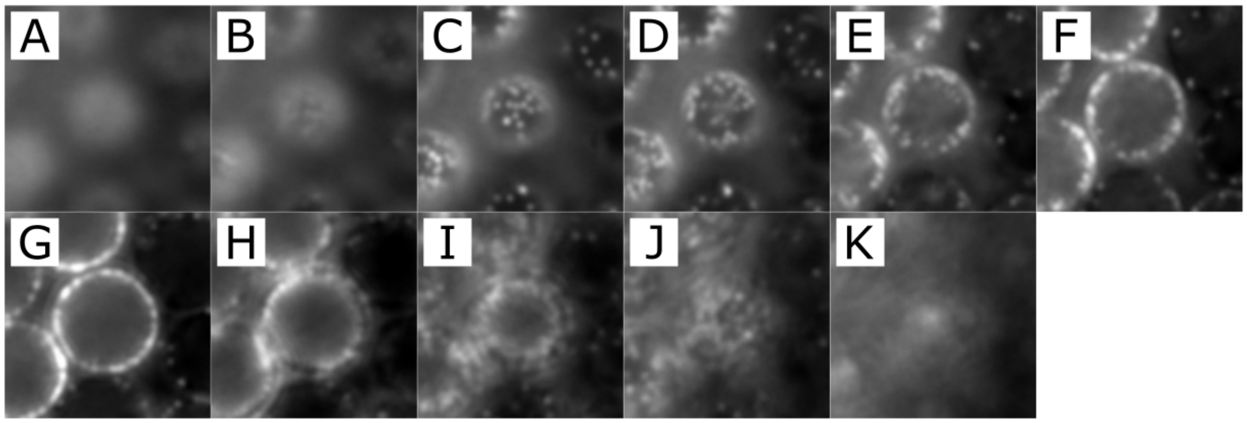


Figure SI3: Fluorescence z-stacks of raspberry particles taken at different heights in the sample (A: -2.99 μm, B: -2.53μm, C: -1.93 μm, D: -1.60 μm, E: -1.07 μm, F: -0.60 μm, G: 0.06 μm, H: 0.99 μm, I: 1.92 μm, J: 2.65 μm, K: 3.65 μm). The best imaging conditions to track rotation are in image C.

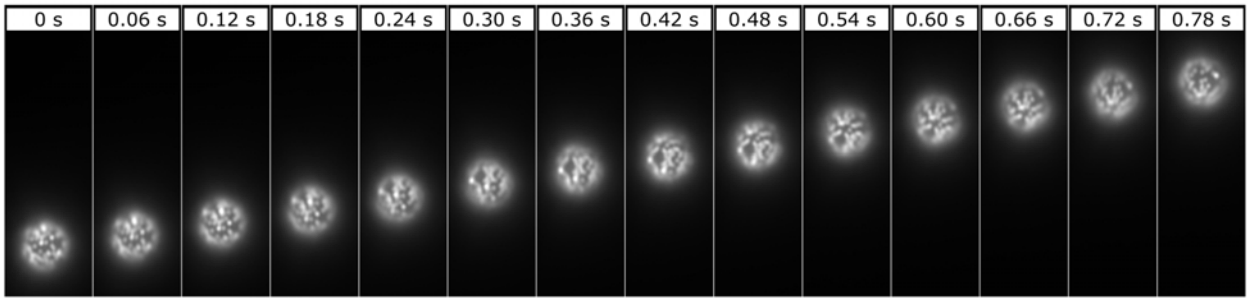


Figure SI4: Time series of fluorescence micrographs of a particle rolling in a capillary channel under flow. The rotation of the particle causes a change of randomly distributed intensity signal generated by the fluorescent berries attached on the particle surface.

Simulation and rotation of model raspberry particles

MATLAB code to simulate fluorescent raspberry particles

The workflow illustrated in Figure SI5 shows the general idea used to create the projection images of a textured spherical surface.

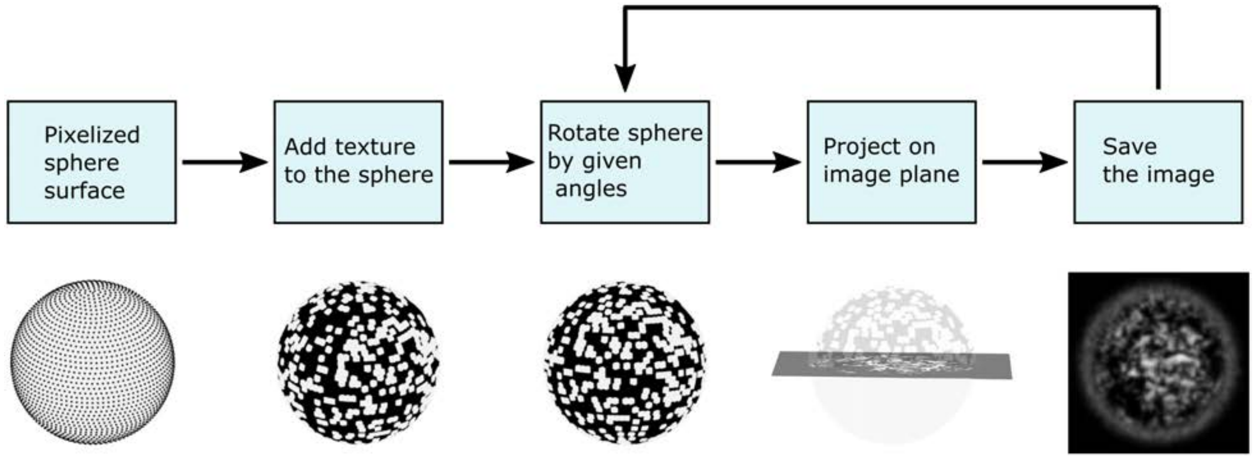


Figure SI5: We first discretize the sphere's surface before adding a defined texture to it: each point on our discretized surface is associated to a pixel intensity. The sphere surface is than rotated by a given angle before creating the projection of its upper part on the image plane. The image is finally saved before repeating the process for a different rotation, until all desired rotated images are saved.

We outline below the code used to generate simulated raspberry particles to test the 3-D rotation registration algorithm against ground truth. Briefly, berries are represented as bright spots on a dark background, in analogy with the experimental fluorescence images of the particles. We first generate a mesh of equidistributed points on a sphere. Berries are then randomly added in an iterative process by rotating the mesh by discrete angles to a specified grid point onto which the berry is placed. This process is repeated until the desired amount of berries is present on the surface of the sphere. A partial overlap of the berries is allowed to mimic the experimental situation where, even if the fluorescent asperities cannot interpenetrate, their fluorescent signals can overlap. After creating the 3-D raspberry particle, images representative of the experimental micrographs can be obtained by projecting the surface of the simulated particle onto the "focal" plane by interpolation.

By subsequently rotating the simulated raspberry particle in 3-D space and projecting the corresponding surface, representative images for different rotations are generated.

The first section of the code defines different parameters used in the simulation.

- imsize = final size of the simulated image (see Figure SI6)
- angle_view (see Figure SI6)
- Radius_view (see Figure SI6)
- N_berries = number of berries attached to the spherical core particle
- R_part = radius of the spherical core particle. Usually R_part = Radius_view / sin(angle_view) (see Figure SI6).
- R_berries = radius of the berries. Since the berries are simulated as ellipses, it is only an indicator of the size. (see Figure SI6)
- std_berries = standard deviation of the size of the berries
- Nb_image = number of images in the final simulation
- Max_ring = max intensity value of the ring's pixels defined by the Gaussian function *fx1* before adding more noise with *fx2* .

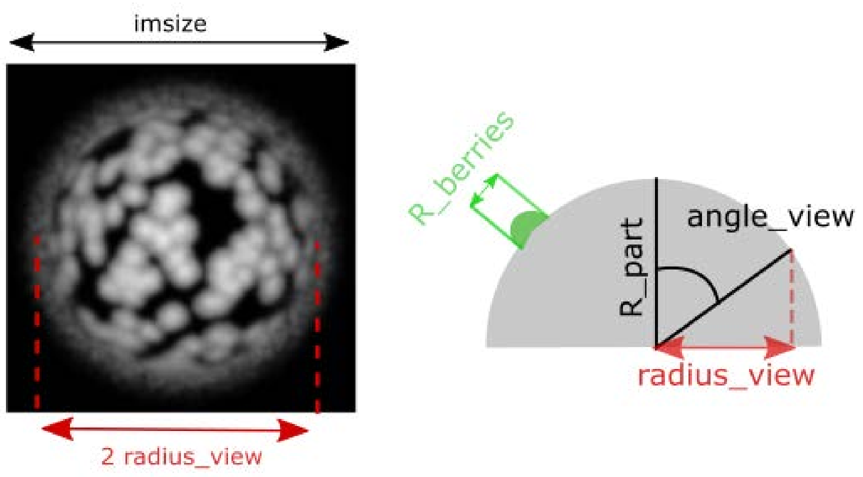


Figure SI6: Example of the projected surface of a simulated raspberry particle. The definitions of the parameters: *imsize*, *radius_view*, *angle_view*, *R_part* and *R_berries* are represented on the image and the scheme.

To better approximate the experimental data with our simulated images, we also include a fluorescent ring around the particles, as observed in the micrographs. This is a purely aesthetic choice, as the ring is not included in the cropped images later used for the analysis, however it is a desirable feature in experiments, where it helps with centre finding. The ring is simulated via two Gaussian functions *fx1* and *fx2*, which define the mean value and standard deviation of the pixel intensity distribution along the ring, respectively. The parameters for these functions were determined by analyzing a set of experimental images (see Figure SI7).

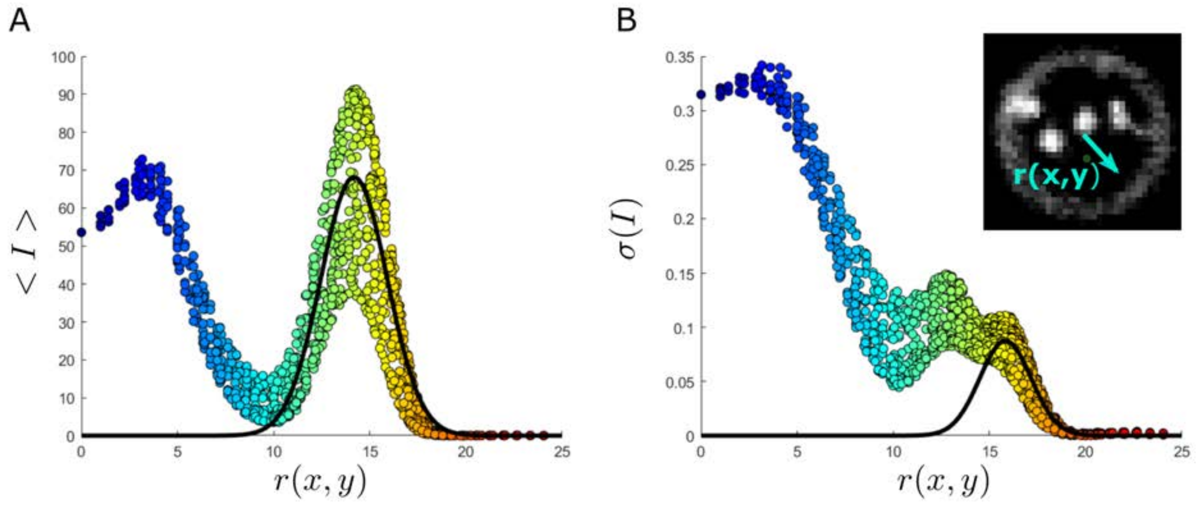
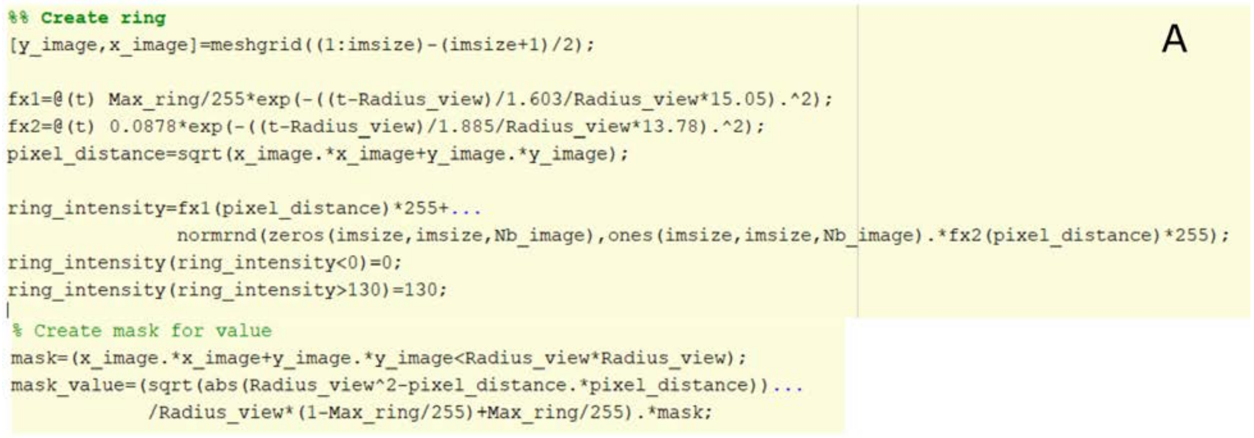


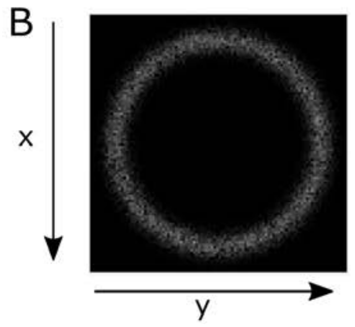
Figure SI7: (A) Mean pixel intensity $< I >$ as a function of radial position $r(x, y)$ obtained by averaging over 3600 experimental images. The region with $r(x, y)$ roughly between 13 and 16 pixels corresponds to the outer ring, and we extract parameters using the Gaussian fit function $f(r) = a_1 e^{-\left(\frac{r-a_2}{a3}\right)^2}$ (black line) to define *fx1*. (B) Standard deviation of the pixel intensity $\sigma(I)$ as a function of radial position $r(x, y)$ obtained from over 3600 images. The region of the curve corresponding to the ring can also be fitted by a Gaussian function (black line), whose values are used to define the function *fx2* in the simulation. The variance is normalized by the maximum intensity value for an 8-bit image (255). Colours are used for visualisation, and indicate distance from the particle centre

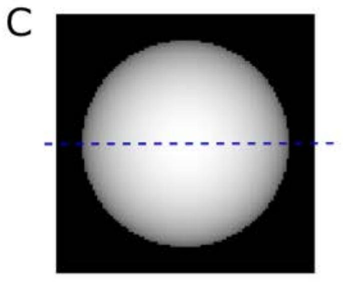
From the extracted parameters, we define a ring where each pixel intensity away by a distance $r(x, y)$ from the particle center is determined by the value of the function $fx1(r(x, y))$ plus a noise term with zero mean and a standard deviation defined by $fx2(r(x, y))$. For each simulated projection of the raspberry particle, a new ring is generated. An example of a simulated ring can be found in Figure SI8. Here, we also define the variable *mask_value*. This mask is a hemisphere of radius *Radius_view*, with value 1 in the center, *Max_ring*/255 at *Radius_view*, where $0 <$ *Max_ring* < 255, and 0 for pixels whose distance to the center is greater than *Radius_view*. This mask is applied to the projection of the particle surface, to create a gradient of intensity depending on the position of the pixel from the particle/image center.



```
%% Create ring
[y_image,x_image]=meshgrid((1:imsize)-(imsize+1)/2);

fx1=@(t) Max_ring/255*exp(-((t-Radius_view)/1.603/Radius_view*15.05).^2);
fx2=@(t) 0.0878*exp(-((t-Radius_view)/1.885/Radius_view*13.78).^2);
pixel_distance=sqrt(x_image.*x_image+y_image.*y_image);

ring_intensity=fx1(pixel_distance)*255+...
               normrnd(zeros(imsize,imsize,Nb_image),ones(imsize,imsize,Nb_image).*fx2(pixel_distance)*255);
ring_intensity(ring_intensity<0)=0;
ring_intensity(ring_intensity>130)=130;

% Create mask for value
mask=(x_image.*x_image+y_image.*y_image<Radius_view*Radius_view);
mask_value=(sqrt(abs(Radius_view^2-pixel_distance.*pixel_distance))...
           /Radius_view*(1-Max_ring/255)+Max_ring/255).*mask;
```

$$\text{if imsize=100, y\_image} = \begin{pmatrix} -49.5 & \cdots & 0 & \cdots & 49.5 \\ | & & | & & | \\ -49.5 & \cdots & 0 & \cdots & 49.5 \\ | & & | & & | \\ -49.5 & \cdots & 0 & \cdots & 49.5 \end{pmatrix} \quad \text{x\_image} = \begin{pmatrix} -49.5 & — & -49.5 & — & -49.5 \\ \vdots & & \vdots & & \vdots \\ 0 & — & 0 & — & 0 \\ \vdots & & \vdots & & \vdots \\ 49.5 & — & 49.5 & — & 49.5 \end{pmatrix} \quad \text{and pixel\_distance} = \begin{pmatrix} 70 & \cdots & 49.5 & \cdots & 70 \\ \vdots & & \vdots & & \vdots \\ 49.5 & \cdots & 0 & \cdots & 49.5 \\ \vdots & & \vdots & & \vdots \\ 70 & \cdots & 49.5 & \cdots & 70 \end{pmatrix}$$





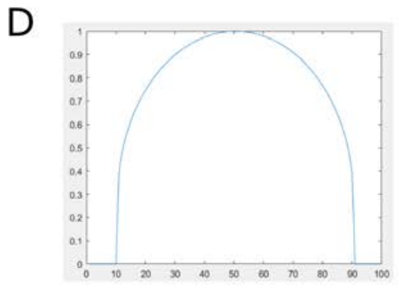
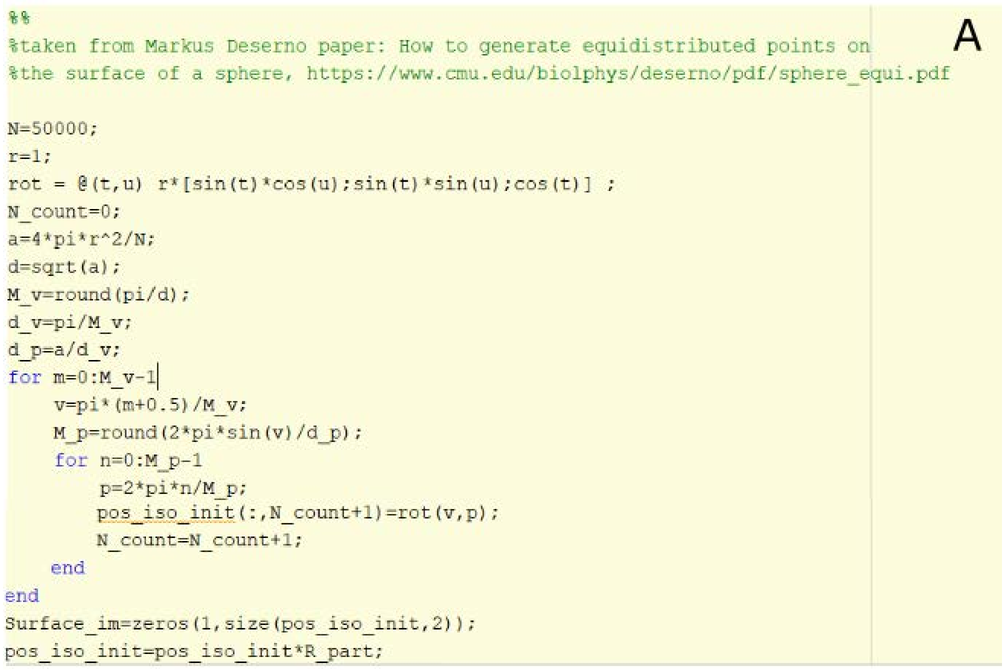


Figure SI8: (A) Snippet of the relevant code. (B) Example of a simulated ring using the functions *fx1* and *fx2* to define each intensity value. For each simulated image, a new ring is generated, which is saved in *ring_intensity*. (C) Example of *mask_value*. (D) Values of the mask corresponding to the blue line in the middle image.

A mesh of equally distributed points is then generated on a spherical surface, with radius *R_part*, following the method outlined by Markus Deserno[1]. The positions are saved in *pos_iso_init* while the value associated to each point is saved in *Surface_im*. An example of the distribution of point positions with $N = 100$ can be found in Figure SI9.



```
%%
%taken from Markus Deserno paper: How to generate equidistributed points on
%the surface of a sphere, https://www.cmu.edu/biolphys/deserno/pdf/sphere_equi.pdf

N=50000;
r=1;
rot = @(t,u) r*[sin(t)*cos(u);sin(t)*sin(u);cos(t)] ;
N_count=0;
a=4*pi*r^2/N;
d=sqrt(a);
M_v=round(pi/d);
d_v=pi/M_v;
d_p=a/d_v;
for m=0:M_v-1
    v=pi*(m+0.5)/M_v;
    M_p=round(2*pi*sin(v)/d_p);
    for n=0:M_p-1
        p=2*pi*n/M_p;
        pos_iso_init(:,N_count+1)=rot(v,p);
        N_count=N_count+1;
    end
end
Surface_im=zeros(1,size(pos_iso_init,2));
pos_iso_init=pos_iso_init*R_part;
```

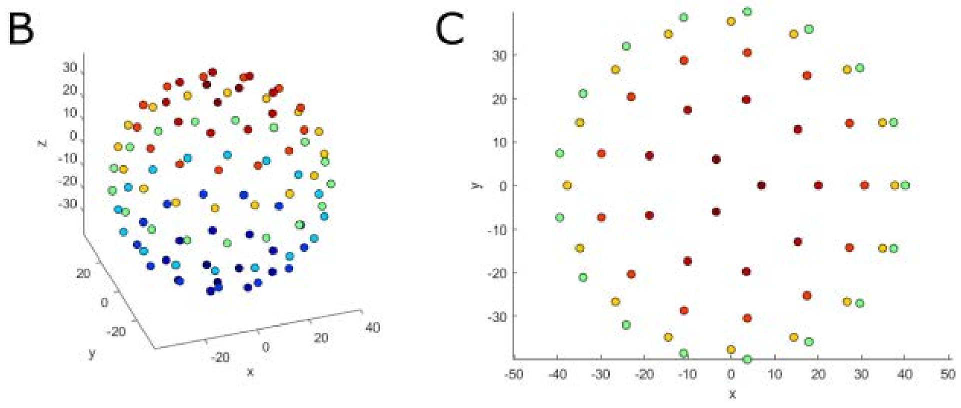
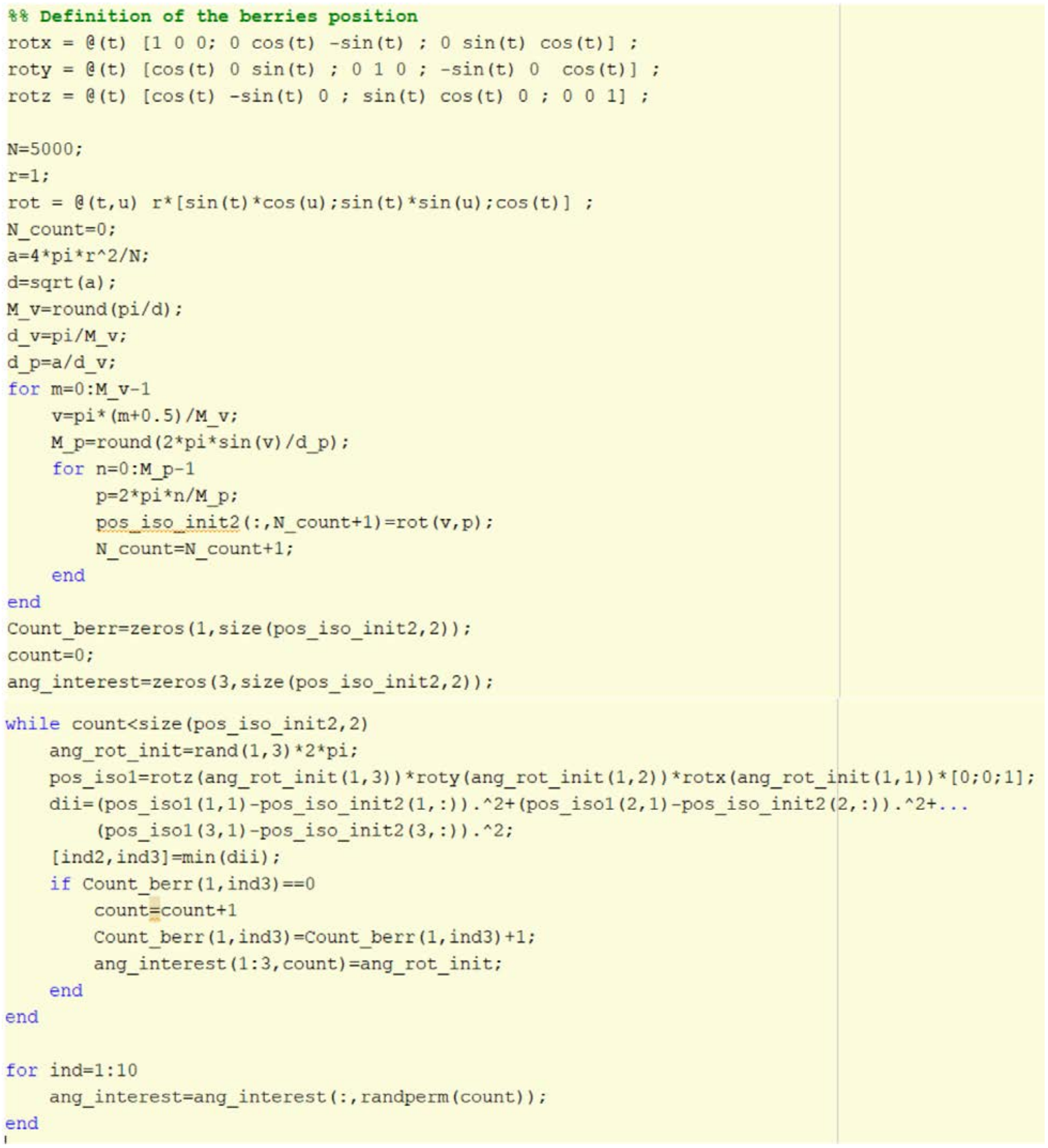


Figure SI9: (A) Snippet of the relevant code. (B) Side view of a 3D plot of a distribution of N = 100 equidistant points on a spherical surface obtained by an adaptation of a code from[1]. (C) Top view of the same distribution. The color indicates the value of the z coordinate.

We then define the locations of the simulated fluorescent berries. Our 3-D surface model will be created by rotating the spherical mesh and placing berries at the subsequent coordinate on the grid. However, simply generating random numbers for θ_x, θ_y, and θ_z will not create an uniform distribution of berries on the surface of the sphere. The problem is similar to the case where the positions are picked by using random uniformly distributed spherical coordinates, creating a non-uniform distribution of points on the sphere[2]. Therefore, we use a while loop to find N angles $(\theta_x, \theta_y, \theta_z)$, where each of them is associated to one predefined point *pos_iso_init2* on the surface of the sphere. The definition of *pos_iso_init2* is the same as in section 3 of the code. We further permute the distribution by randomly shuffling the matrix storing the angles *ang_interest* 10 times with the in-built MATLAB function *randperm*.



```
%% Definition of the berries position
rotx = @(t) [1 0 0; 0 cos(t) -sin(t) ; 0 sin(t) cos(t)] ;
roty = @(t) [cos(t) 0 sin(t) ; 0 1 0 ; -sin(t) 0  cos(t)] ;
rotz = @(t) [cos(t) -sin(t) 0 ; sin(t) cos(t) 0 ; 0 0 1] ;

N=5000;
r=1;
rot = @(t,u) r*[sin(t)*cos(u);sin(t)*sin(u);cos(t)] ;
N_count=0;
a=4*pi*r^2/N;
d=sqrt(a);
M_v=round(pi/d);
d_v=pi/M_v;
d_p=a/d_v;
for m=0:M_v-1
    v=pi*(m+0.5)/M_v;
    M_p=round(2*pi*sin(v)/d_p);
    for n=0:M_p-1
        p=2*pi*n/M_p;
        pos_iso_init2(:,N_count+1)=rot(v,p);
        N_count=N_count+1;
    end
end
Count_berr=zeros(1,size(pos_iso_init2,2));
count=0;
ang_interest=zeros(3,size(pos_iso_init2,2));

while count<size(pos_iso_init2,2)
    ang_rot_init=rand(1,3)*2*pi;
    pos_iso1=rotz(ang_rot_init(1,3))*roty(ang_rot_init(1,2))*rotx(ang_rot_init(1,1))*[0;0;1];
    dii=(pos_iso1(1,1)-pos_iso_init2(1,:)).^2+(pos_iso1(2,1)-pos_iso_init2(2,:)).^2+...
        (pos_iso1(3,1)-pos_iso_init2(3,:)).^2;
    [ind2,ind3]=min(dii);
    if Count_berr(1,ind3)==0
        count=count+1
        Count_berr(1,ind3)=Count_berr(1,ind3)+1;
        ang_interest(1:3,count)=ang_rot_init;
    end
end

for ind=1:10
    ang_interest=ang_interest(:,randperm(count));
end
```

Figure SI10: Snippet of the relevant code.

The next section of the code generates the berries on the surface of the simulated particle. For each berry, we rotate our sphere from its initial position by a randomised set of angles taken from *ang_interest* (defined above). We then add the new berry to the top of our pixelized surface *Surface_im* before rotating the sphere to its initial orientation (Figure SI11).

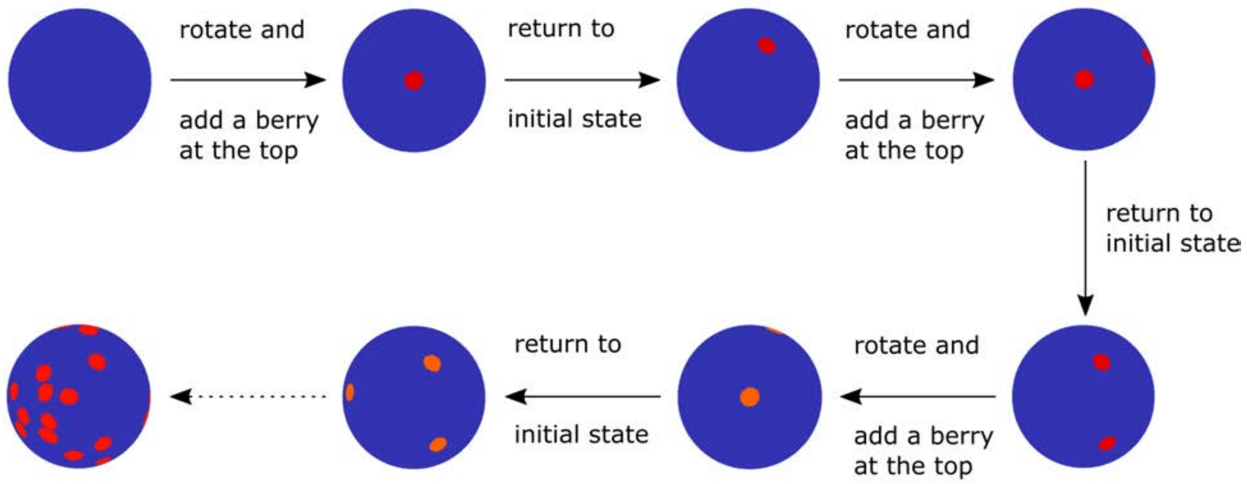


Figure SI11: The sphere is rotated from its initial state, and a berry is then added to the top of the rotated surface. The modified surface is then rotated back to its initial state. This process is iterated until the desired number of berries are added to the surface of the sphere.

To better mimic the observed experimental micrographs, the simulated berries are ellipsoidal, and the values of the short and long axes are saved in *radiusX* and *radiusY*. The values are generated using a mean value term *R_berries*, to which a random number from a normal distribution with mean 0 and standard deviation *std_berries* is added. The orientation of the ellipse on the sphere is defined by *berr_ang*. The intensity values of each berry, *Iso_berr_value*, is determined by two contributing terms; *fx3* to create a gradient of intensity depending on the pixel distance to the ellipse center, and a randomised value to simulate noise on the pixel intensity values. The overall intensity profile of the rough sphere *Surface_im* is generated by recursively adding the values of *Iso_berr_value* to the previous *Surface_im* for each new berry. If the intensity profiles of two berries overlap, for each overlapping pixel we retain the highest intensity value (see Figure SI12).

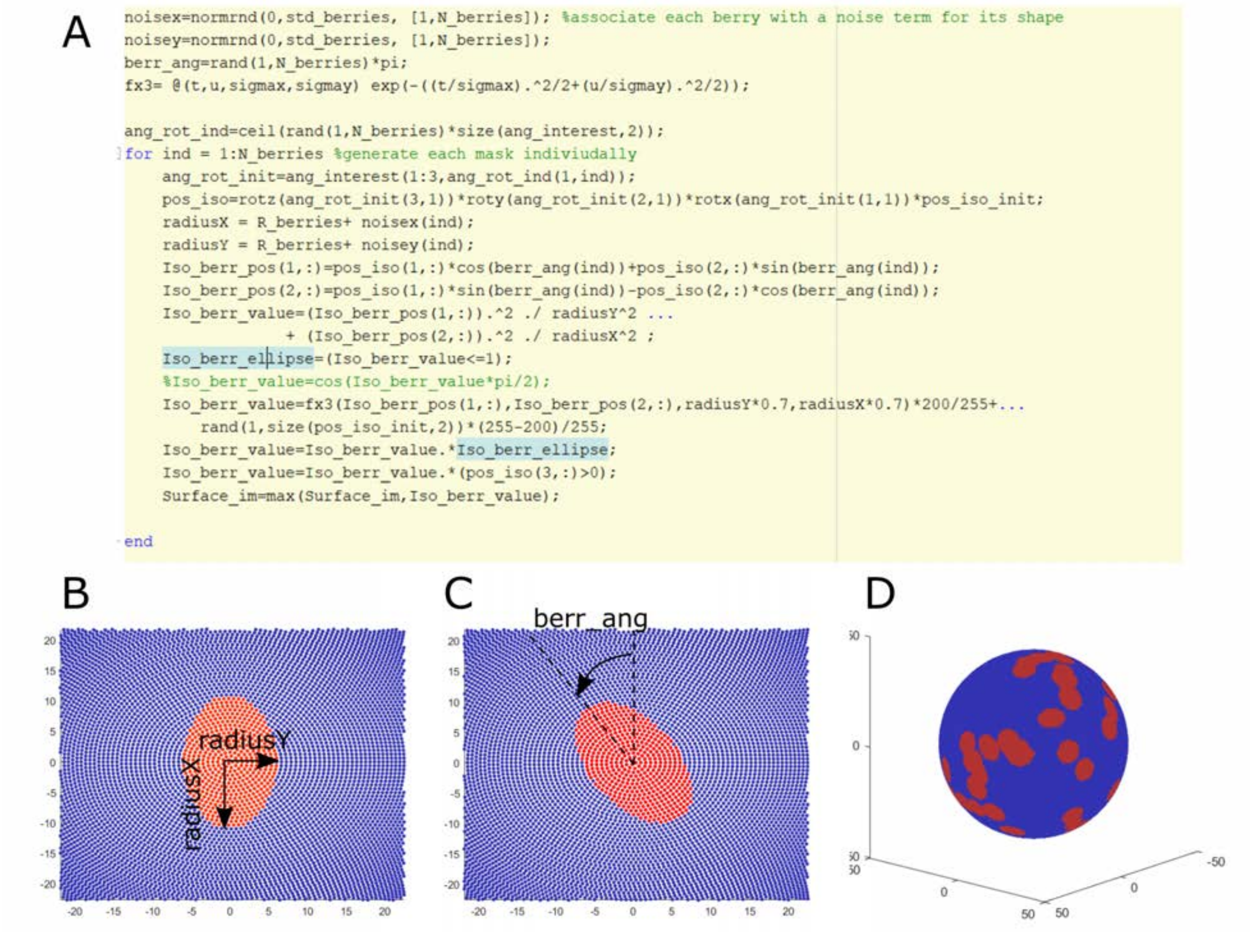
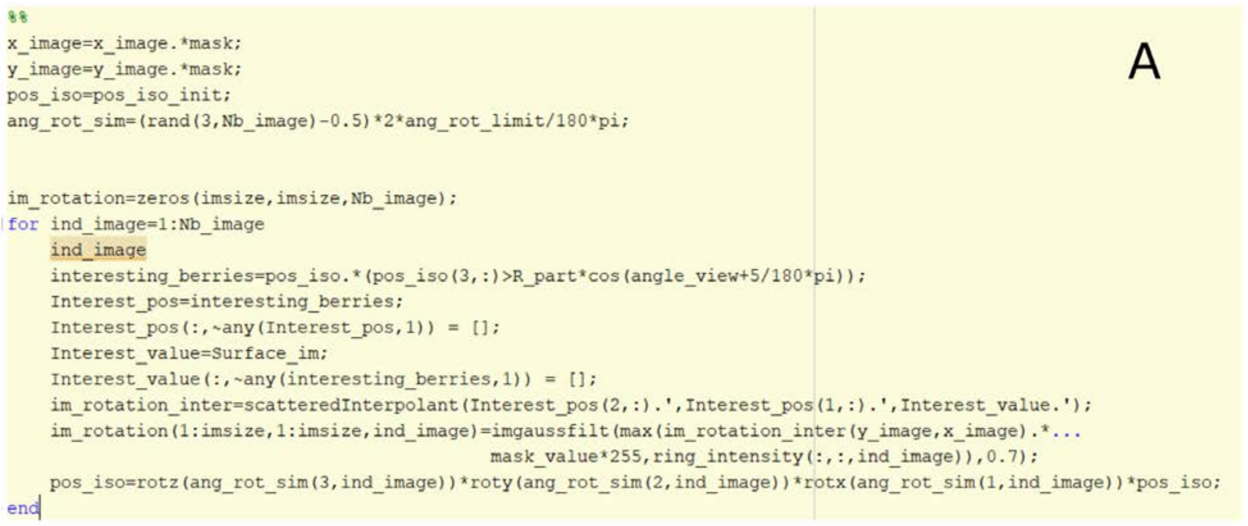
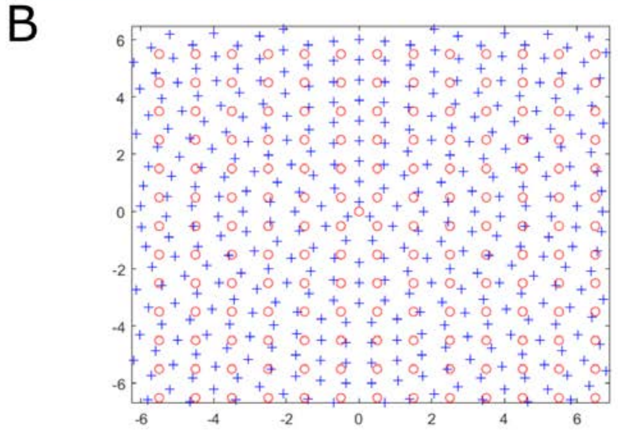

```
noisex=normrnd(0,std_berries, [1,N_berries]); %associate each berry with a noise term for its shape
noisey=normrnd(0,std_berries, [1,N_berries]);
berr_ang=rand(1,N_berries)*pi;
fx3= @(t,u,sigmax,sigmay) exp(-((t/sigmax).^2/2+(u/sigmay).^2/2));

ang_rot_ind=ceil(rand(1,N_berries)*size(ang_interest,2));
for ind = 1:N_berries %generate each mask indiviudally
    ang_rot_init=ang_interest(1:3,ang_rot_ind(1,ind));
    pos_iso=rotz(ang_rot_init(3,1))*roty(ang_rot_init(2,1))*rotx(ang_rot_init(1,1))*pos_iso_init;
    radiusX = R_berries+ noisex(ind);
    radiusY = R_berries+ noisey(ind);
    Iso_berr_pos(1,:)=pos_iso(1,:)*cos(berr_ang(ind))+pos_iso(2,:)*sin(berr_ang(ind));
    Iso_berr_pos(2,:)=pos_iso(1,:)*sin(berr_ang(ind))-pos_iso(2,:)*cos(berr_ang(ind));
    Iso_berr_value=(Iso_berr_pos(1,:)).^2 ./ radiusY^2 ...
                + (Iso_berr_pos(2,:)).^2 ./ radiusX^2 ;
    Iso_berr_ellipse=(Iso_berr_value<=1);
    %Iso_berr_value=cos(Iso_berr_value*pi/2);
    Iso_berr_value=fx3(Iso_berr_pos(1,:),Iso_berr_pos(2,:),radiusY*0.7,radiusX*0.7)*200/255+...
        rand(1,size(pos_iso_init,2))*(255-200)/255;
    Iso_berr_value=Iso_berr_value.*Iso_berr_ellipse;
    Iso_berr_value=Iso_berr_value.*(pos_iso(3,:)>0);
    Surface_im=max(Surface_im,Iso_berr_value);

end
```

Figure SI12: (A) Snippet of the relevant code. (B) Top view of the pixelated surface of a sphere with the position occupied by the computed berry (red). (C) Top view of the pixelated surface of a sphere with the position occupied by the computed berry which is now rotated by *berr_ang* = 30° (red). (D) 3-D view of the position of different berries (red) on the sphere (blue).

Having generated the surface of our raspberry particles, we obtain the corresponding 2-D images by projecting the upper half of the particle surface onto our simulated "focal" plane. The pixel intensity values of the focal plane, $\{x\_image, y\_image\}$, are interpolated from the pixels of *Surface_im* which lie above the height of the focal plane. This threshold value above the z-axis is defined by the opening angle *angle_view*. The particle mask and ring are finally applied to these points (see Figure SI13). To then generate the different rotated images of the raspberry particle, *Surface_im* is rotated by a set of angles *ang_rot_sim*, and the previous operation is repeated until the desired number of images *Nb_image* are computed.



```
%%
x_image=x_image.*mask;
y_image=y_image.*mask;
pos_iso=pos_iso_init;
ang_rot_sim=(rand(3,Nb_image)-0.5)*2*ang_rot_limit/180*pi;


im_rotation=zeros(imsize,imsize,Nb_image);
for ind_image=1:Nb_image
    ind_image
    interesting_berries=pos_iso.*(pos_iso(3,:)>R_part*cos(angle_view+5/180*pi));
    Interest_pos=interesting_berries;
    Interest_pos(:,~any(Interest_pos,1)) = [];
    Interest_value=Surface_im;
    Interest_value(:,~any(interesting_berries,1)) = [];
    im_rotation_inter=scatteredInterpolant(Interest_pos(2,:).',Interest_pos(1,:).',Interest_value.');
    im_rotation(1:imsize,1:imsize,ind_image)=imgaussfilt(max(im_rotation_inter(y_image,x_image).*...
                                            mask_value*255,ring_intensity(:,:,ind_image)),0.7);
    pos_iso=rotz(ang_rot_sim(3,ind_image))*roty(ang_rot_sim(2,ind_image))*rotx(ang_rot_sim(1,ind_image))*pos_iso;
end
```



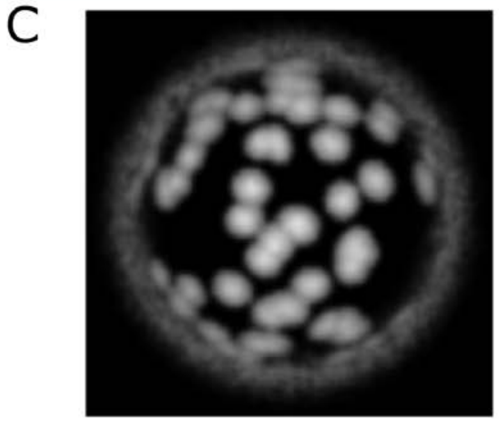


Figure SI13: (A) Snippet of the relevant code. (B) Relative positions of the pixels on the surface of the simulated sphere (blue crosses), and their respective projection onto the ”focal” plane by interpolation (red circles). (C) Example projection of a simulated raspberry particle onto the focal plane.

3-D rotation tracking: approach, code and applicability

Approach

As described in the main text, the measurement of the true rotation is obtained from the identification of those angular displacements that maximize the correlation between a first reference image and a second one corresponding to a prospective rotation of the subsequent frame. Because the images are 2-D projections from a 3-D spherical surface, before applying the rotation, the 2-D images need to be processed and projected back onto a sphere, as shown below in Figure SI14.

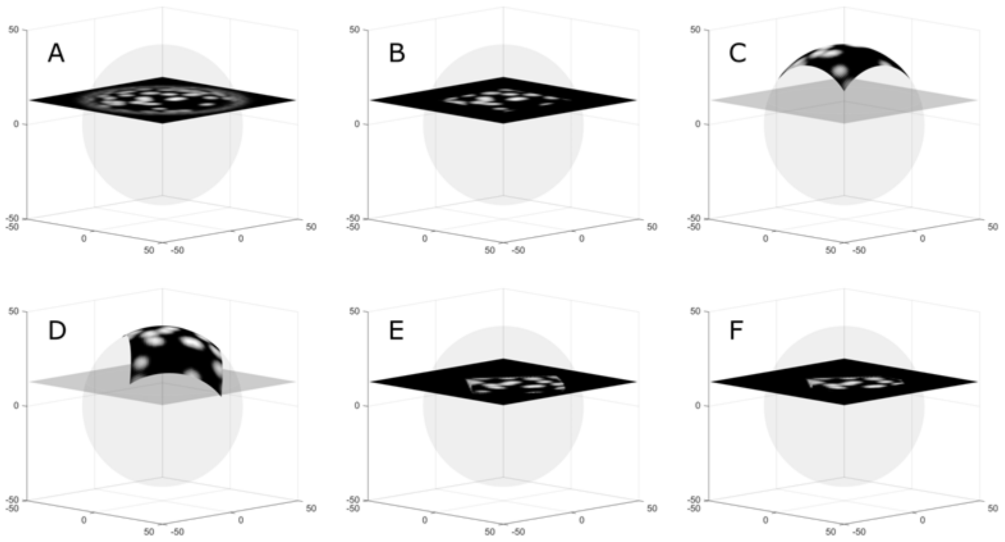


Figure SI14: Procedure for 3-D image rotation: We assume that images are projections of the surface of a sphere onto the focal plane of the microscope (A). The image is cropped to the central portion of the particle (B). We then project the cropped region of the image onto a sphere (C), before applying the desired rotation (D). The image of the rotated surface is projected back onto the focal plane (E), and finally masked (F) before calculating the correlation.

In order to ensure that the correlation is carried out only on relevant portions of the images, it is important that an appropriate mask is applied to both the reference and the rotated images, as exemplified below in Figure SI15.

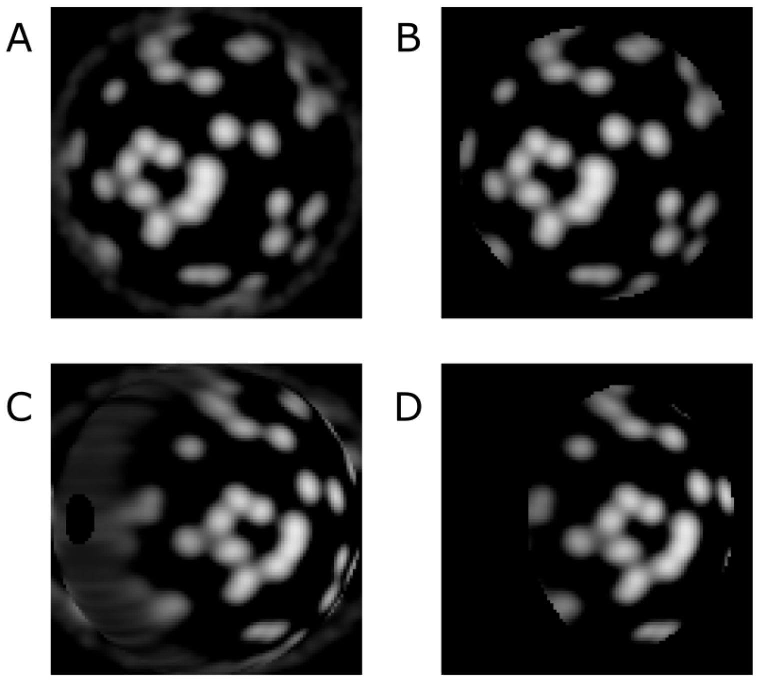


Figure SI15: Example of the application of a mask to an image. (A) Initial (simulated) image. (B) Initial image after application of the mask. (C) Image A rotated by 35° around the x-axis. (D): Image C after applying the same mask applied in B, but this time rotated by 35° around the x-axis. We emphasize that rotating a mask will change the dimensions of the final, masked image.Therefore when comparing the images, the same rotated mask need to be applied to both reference and rotated images. The x-axis runs vertically in the plane of the image.

In particular, the application of a mask to retain only a limited region around the particle centre is motivated by the fact that the information density of a 3-D surface projected onto a 2-D plane is a function of position, as illustrated in the scheme in Figure SI16. Specifically, the length of the arc $s = R(\theta_2 - \theta_1)$ and the length of its projection $l = R(\sin\theta_2 - \sin\theta_1)$, can be calculated as a function of the angles θ_1 and θ_2. For small angles, the s/l ratio is close to 1, but it rapidly increases as the position approaches the edge of the hemisphere. Consequently, as one moves towards the edge of the hemisphere, the projected arc length l vanishes, implying that the information contained in the 3-D image cannot be properly mapped in 2-D. Therefore, information on rotations coming from berries at the periphery of the particle is discarded by cropping the image around the center and applying a mask, avoiding the contribution of berries ”disappearing” from the field of view. Effectively, for our analysis, we limit our range of angles $-50° < \theta < 50°$.

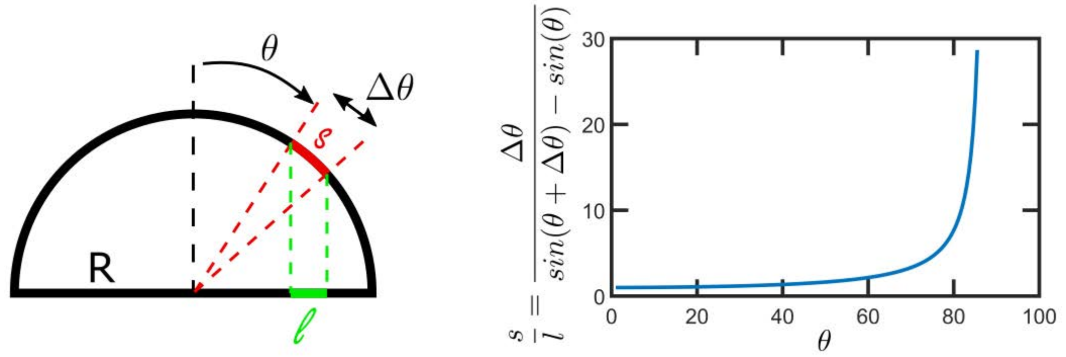


Figure SI16: Left: Scheme of the projection of an arc onto the diameter of a circle. Right: The ratio s/l as a function of θ (in degrees) with a fixed $\Delta\theta = \frac{5°}{180°}\pi$.

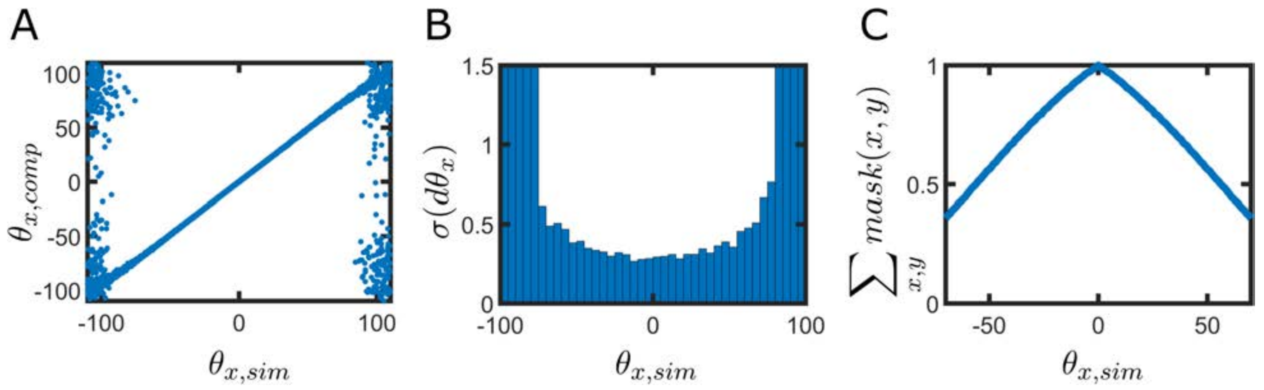


Figure SI17: Our angle registration method is applied to images which are a restricted projection of the particle surface onto a 2-D plane, and therefore it is limited to a smaller range of rotations in the x- and y- axes. To probe the valid angles which can be detected by our method, we test the approach on 5000 simulated images. Details of the particle simulation can be found in the Simulation Code section. We test the accuracy of our method for rotations in 1-D by rotating our particle about the x-axis with θ_x^{sim} in the range $[-110^\circ\ 110^\circ]$, in the presence of rotations about the y-axis with θ_y^{sim} between $[-10^\circ\ 10^\circ]$. No rotations about the z-axis were applied. (A) Computed angle θ_x^{comp} with our 3D registration method relative to the ground-truth θ_x^{sim} input into the simulation. Angles in the range $[-130^\circ\ 130^\circ]$ in increments of 1° were screened (beyond the initial range of simulated, true values), and accompanying rotations about the y-axis in the range $[-16^\circ\ 16^\circ]$ with increments of 1° for θ_y^{comp} were screened (beyond the initial range of simulated, true values). (B) Evolution of the standard deviation of the error in computed angles about the x-axis; $\mathrm{d}\theta_x = \theta_x^{comp} - \theta_x^{sim}$. For smaller rotations, the standard deviation of the error remains relatively stable, however, it begins to deviate for $|\theta_x^{sim}| \geq 30^\circ$. (C) Evolution of the normalised sum of the Boolean mask for different rotation angles θ_x about the x-axis. As $|\theta_x|$ increases, the applicable mask size and thus the sum of the boolean mask decreases. Until $|\theta_x^{com}| = 30^\circ$, approximately three quarters of the information contained in the original mask (with no rotation) is retained. From A., it initially appears that $\theta_{x,comp}$ and $\theta_{x,sim}$ are in good agreement for $|\theta_x, sim| < 90^\circ$. However, from B., we note that $\sigma(d\theta_x)$ already deviates for $|\theta_{x,sim}| > 50^\circ$ with a significant divergence for angles greater than 85°. From this analysis, we conclude that we can recover rotations about the x- and y- axes with relative confidence up to angles of 30°. Beyond this range, for the explored mask sizes, the effect of the mask size or limited surface information from the obtained images may lead to unreliable results.

Code

Here, we describe the key elements of the code used to measure angular displacements.

The section *"images folder's path"* allows the user to define the path to the folder where the images are stored. This folder should only contain the images and no other documents.

The next section *"Definition of the parameters"* allows the user to define most of the parameters for the code.

- Image_number = Number of images to analyze
- image_size = Size of the resized images (see Figure SI18)
- radius = window radius applied to the initial image to crop (see Figure SI18)
- R_part= Radius of the particle in the size_init scale (see Figure SI18)
- mask_r = proportion of the image with mask
- gfilt = size of the Gaussian filter from the MATLAB function imgaussfilt
- contrastp=contrast parameter from the MATLAB function imadjust;
- angle_scanx = angles that will be scanned for the rotation about the x-axis. The x-axis is defined as a vector of size (1,:).
- angle_scany = angles that will be scanned for the rotation about the y-axis. The y-axis is defined as a vector of size (1,:).
- angle_scanz = angles that will be scanned for the rotation about the z-axis. The z-axis is defined as a vector of size (1,:).

Once the parameters are defined, the code resizes the images to a new size *image_size* from a cropped zone twice the size of *radius* of the original image A, while applying a Gaussian filter, adjusting the contrast and applying the mask. In the first image that appears, the user verifies that the chosen parameters are appropriate.

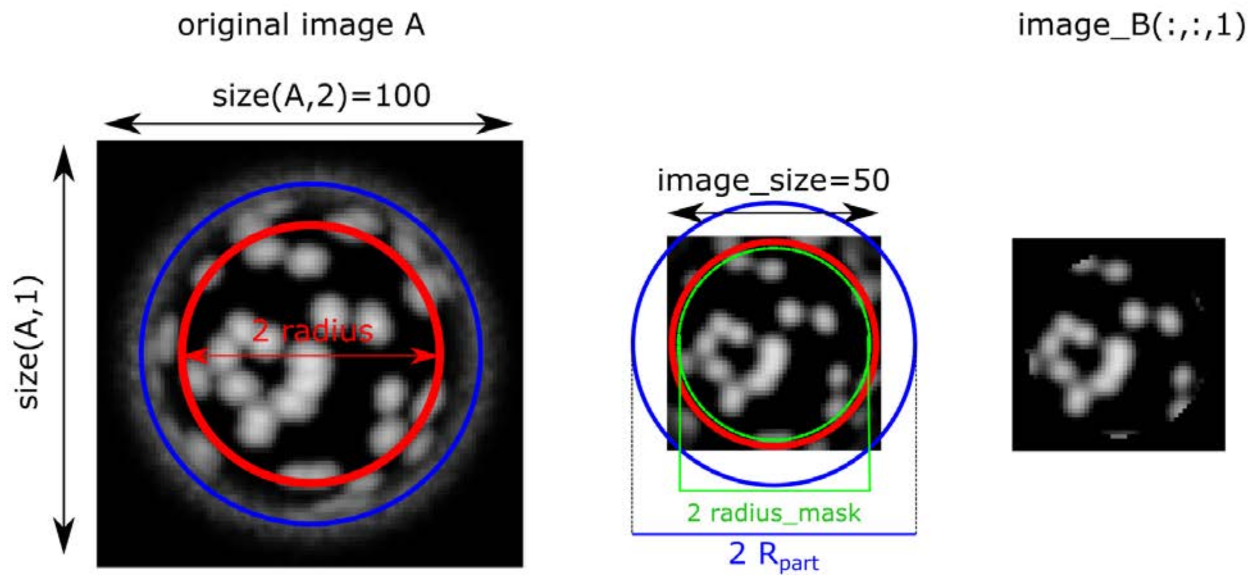


Figure SI18: Schematic of the different parameters used in our code.

In the next section *"Initialize all wished rotations"*, the code will first apply the image transformations previously defined by the user to all images and store the resulting *ind*th images in *image_B(:,:,ind)*, where *ind* is the frame number. After defining the rotation matrices *rotx*, *roty*, *rotz*, i.e. the rotations around the axis **x**, **y** and **z**, we store the positions of each pixel relative to the origin corresponding to the center of the image in (*x_pos_init*,*y_pos_init*). We then define our initial mask image *interesting_pixel_mask* but also the position *z_pos_init*, whose value is given by $\sqrt{R^2 - x^2 - y^2}$ if $x^2 + y^2 < R^2$ and $z = 0$ everywhere else. To accelerate the code, the position matrices are vectorised, so the rotation can be later directly applied to all the vectors. The positions are now stored in *pos_init*. After initializing the different matrices, we compute, through different "for" loops, all possible rotation combinations of the values in *angle_scanx*, *angle_scany* and *angle_scanz*. For each iteration *ind_progress*, we compute the new position *pos_rot* of the pixels after the rotation, then reshape the position in the same matrix size as the image (*x_rot* and *y_rot*) and save the value for each instance in *x_rot1(:,:,ind_progress)* and *y_rot1(:,:,ind_progress)*. We also compute the shape of the associated mask using *x_rot* and *y_rot*, and store it in *mask(:,:,ind_progress)*. Finally, we store the angles scanned in *ang_progress(:,:,ind_progress)*.

The ordering of the rotation matrices is important when applying a transformation to the position vector. As we call the MATLAB in-built interp2 function to interpolate an image at t_1 and compare it to the "true" image at t_0, the rotations are defined as $pos\_rot = rotz(\theta_z)roty(\theta_y)rotx(\theta_x)pos\_init$. In the opposite case, if an interpolated image at t_0 is compared to the "true" image at t_1, the rotations are defined as $pos\_rot = rotx(-\theta_x)roty(-\theta_y)rotz(-\theta_z)pos\_init$ and the resulting rotation angles are $\{-\theta_x, -\theta_y, -\theta_z\}$ (see Figure SI19).

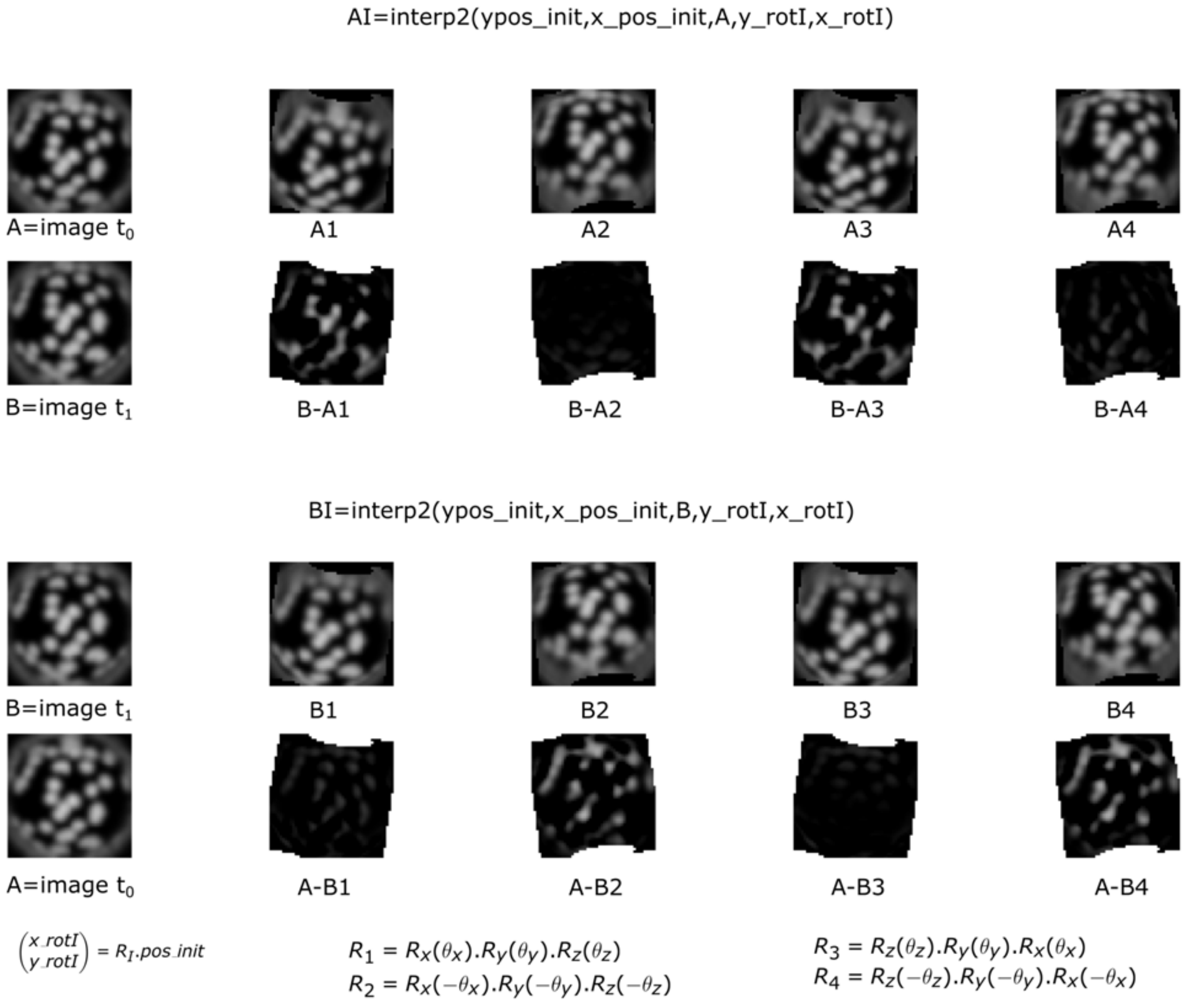

AI=interp2(ypos_init,x_pos_init,A,y_rotI,x_rotI)
A=image t_0
A1
A2
A3
A4
B=image t_1
B-A1
B-A2
B-A3
B-A4
BI=interp2(ypos_init,x_pos_init,B,y_rotI,x_rotI)
B=image t_1
B1
B2
B3
B4
A=image t_0
A-B1
A-B2
A-B3
A-B4
$\begin{pmatrix} x\_rotI \\ y\_rotI \end{pmatrix} = R_I.pos\_init$
$R_1 = R_x(\theta_x).R_y(\theta_y).R_z(\theta_z)$
$R_2 = R_x(-\theta_x).R_y(-\theta_y).R_z(-\theta_z)$
$R_3 = R_z(\theta_z).R_y(\theta_y).R_x(\theta_x)$
$R_4 = R_z(-\theta_z).R_y(-\theta_y).R_x(-\theta_x)$

Figure SI19: Original image A and interpolated images A1, A2, A3 and A4 after application of the rotation matrices R1, R2, R3 and R4, as defined at the bottom of the figure, respectively. If an image at t_0 is compared to the next at t_1 (i.e. rotating image A to match image B), it is necessary to use the rotation matrix $R = R_xR_yR_z$, and note that the computed angles are opposite to the ones used in the simulation case. Conversely, if image B is rotated to match image A, then the relevant rotation matrix is $R = R_zR_yR_x$, and the computed rotation angles correspond to the the simulated ones. The image differences in the bottom row of each panel highlight the effect of the order of the application of the the rotation matrices and the sign of the rotation angles.

The next section of code defines which image indices to compare through *imstep*. It is important that the images separated by *imstep* are sufficiently distinguishable for the correlation-based analysis. A figure will pop up so the user can visually inspect the two separate images, and thus evaluate if the value of *imstep* is appropriate or needs to be refined. Additionally, the code can be initially run on a few images (around 50) and the user can examine the distribution of computed angles and assess if a bigger or smaller step between images indices (*imstep*) is necessary or if the angle limit needs to be adjusted.

The last section of the code computes all the theoretical, rotated images corresponding to the previously defined rotation angles, using the in-built MATLAB function interp2. The generated, rotated images with coordinates (*x_rot1*, *y_rot1*) are obtained from the original data *image_B(:,:,ind_image+imstep)* with coordinates (*x_pos_init*, *y_pos_init*), and the data is stored in *new_image*. For each theoretical, rotated image, the *mask* corresponding to the rotation is applied to the rotated image and the reference image *image_initial*, and the correlation between the two masked images is then determined using the in-built MATLAB function corr2, and stored in *correlation_res(ind_progress,ind_image)*. The MATLAB function corr2 applied to the matrix A and B, computes the following quantity:

$$corr2(A,B) = \frac{\sum_m \sum_n (A_{mn} - \bar{A})(B_{mn} - \bar{B})}{\sqrt{\left(\sum_m \sum_n (A_{mn} - \bar{A})^2\right)\left(\sum_m \sum_n (B_{mn} - \bar{B})^2\right)}}$$

with $\bar{A}$ the mean value of A and $\bar{B}$ the mean value of B.

Finally, the maximum correlation for each image *ind_image* is found, and the corresponding index *ind_max* is used to extract the corresponding angles from *ang_progress*. These angles are saved in *res_ang*. The first row of this matrix corresponds to the angles of the rotation about the axis **x**, the second corresponds to the angles of the rotation about the axis **y**, while the last row corresponds to the angles of rotation around the axis **z**. Each column corresponds to the estimated angles between the i^{th} image and the $(i + imstep)^{th}$ one.

Method applicability and sensitivity

As alluded to above and in the main text, there are several parameters, which can be affect the applicability and the precision of our method, at the cost of computational speed. To evaluate the effect of the different parameters, we applied our image registration method to analyse 500 simulated images of a rotating raspberry particle.

Effect of angle step and image size.

In Figure SI20-A, we evaluate the true rotations in increments between 0.5° and 4° (*ang_step* = 0.5°, 1°, 2°, 3°, 4°) for a raspberry particle with an image size = 50 pixels (*image_size* = 50), and in Figure SI20-B, the effect of changing *image_size* between 15 and 75 pixels (*image_size* = 15, 20, 30, 50, 75) was evaluated for a fixed angular increment of 2° (*ang_step* = 2°). In all cases, the cropped image corresponds to 75% of the particle size, which is rescaled to an image size of 50 x 50 pixels and a mask radius of *R_mask* = 24 pixels image was applied for the analysis.

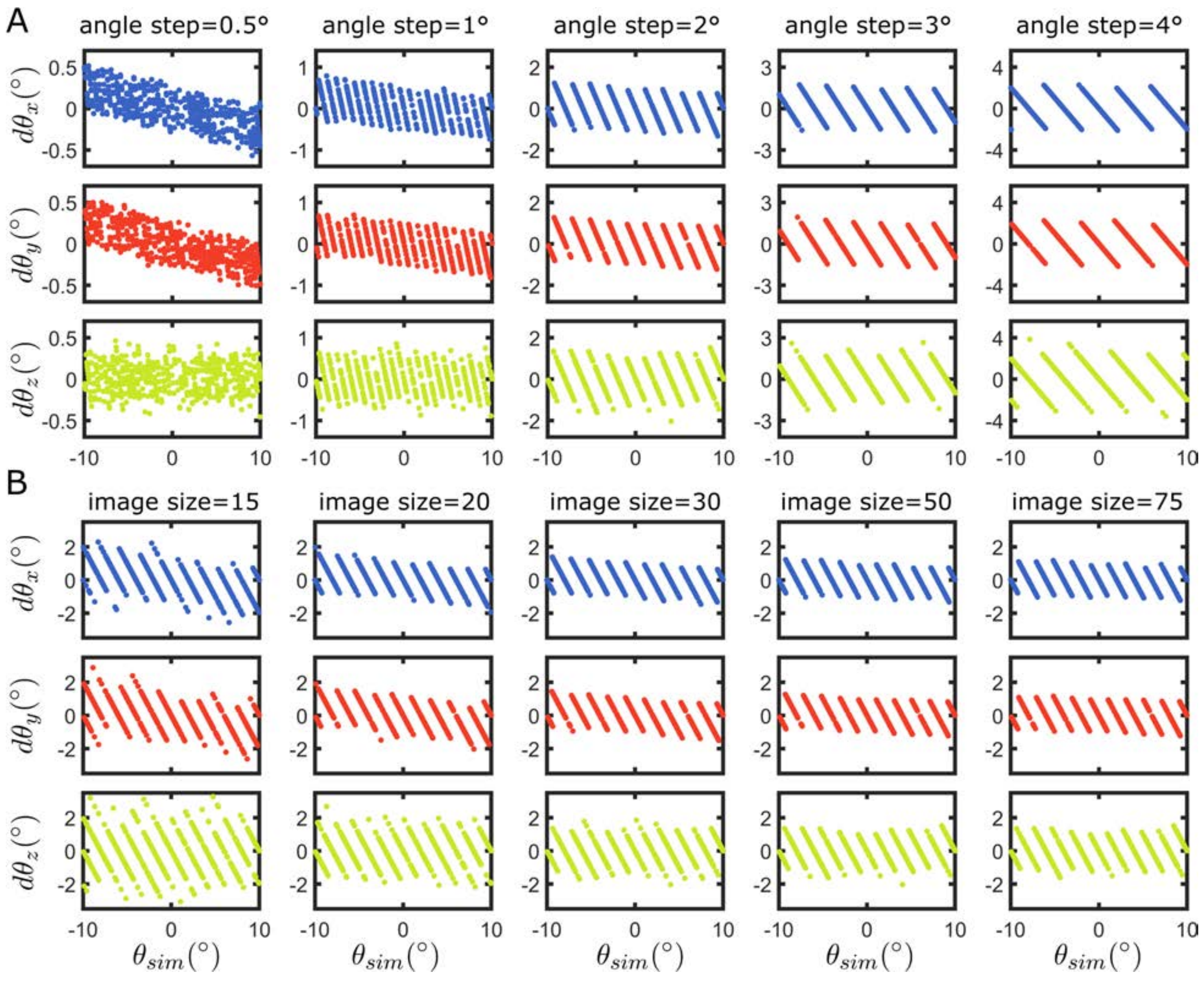


Figure SI20: Evolution of the difference $d\theta = \theta_{comp} - \theta_{sim}$ between the angles computed with our method θ_{comp} and the angles used in the simulation θ_{sim} as a function of θ_{sim} for different discretization steps *ang_step* (A) or for different image sizes *image_size* *ang_step*= 2° (B).

We find that increasing the resolution of the discretized space, controlled by *ang_step*, will accordingly improve the accuracy of the computed angles (see Figure SI20-A). When evaluating the analysis of the simulated data by adjusting *ang_step* from 2° to 4°, the error $d\theta$ between the retrieved angles and the true angles of the simulation is confined between -*ang_step*/2 and *ang_step*/2 for the rotations around the x- and y-axis, while the distribution of the error in the detected angular displacements around the z-axis is broader. This discrepancy can potentially be attributed to the properties of the discretization of the particle in an image. For a simulated image of 50 pixels, rotations around the x- and y-axis typically result in features translating by more than 1 pixel, if the angular displacement is greater than 2° (Figure SI21). For in-plane rotations (about the z-axis), similar results can be observed. However, in the latter case, the maximum feature displacements occurs at the periphery of the image, whereas for rotations around the x- and y-axis, the largest pixel changes from rotations occur at the centre of the image. As the particle centre possesses the highest information density, this might cause the observed lower precision of angle registration about the z-axis compared to the x- and y-axes (Figure SI22).

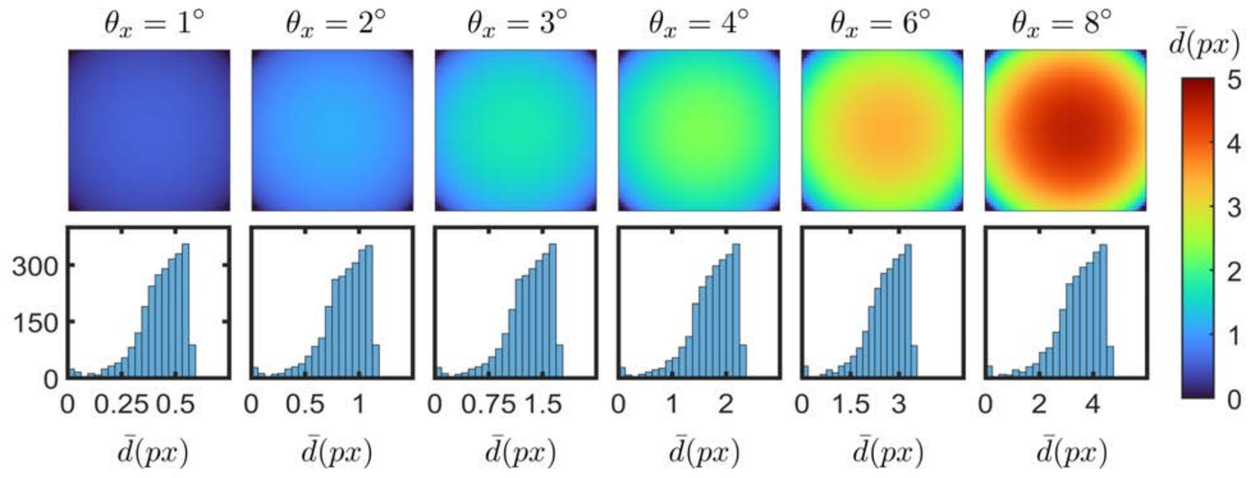


Figure SI21: Displacement value map $\bar{d} = \sqrt{(x_n - x)^2 + (y_n - y)^2}$ and $[x_n\ y_n\ z_n]' = \mathrm{R}_{\theta_x}$ $[x\ y\ \sqrt{R_{part}^2 - x^2 - y^2}]'$ for different rotation angles θ_x about the x-axis for a cropped 50 pixel particle image. The histograms in the bottom row show the distribution of $\bar{d}$ in the corresponding images above. The maximum, absolute displacements for rotations out-of-plane (about the x- and y-axes) occur at the centre of the image. As seen from the histograms, a significant number of points will experience a displacement $\bar{d}$ greater than 1 pixel when the rotation angle is greater or equal to 2°. Therefore, for an image of 50 pixels, an out-of-plane rotation of 2° can be detected by comparing the pixel shifts between two subsequent images before and after rotation.

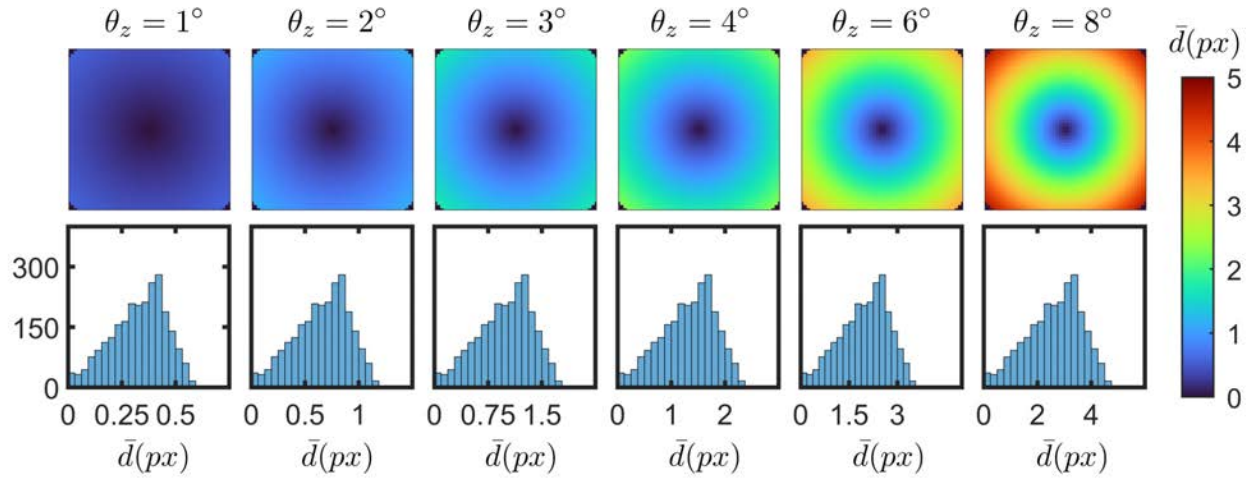
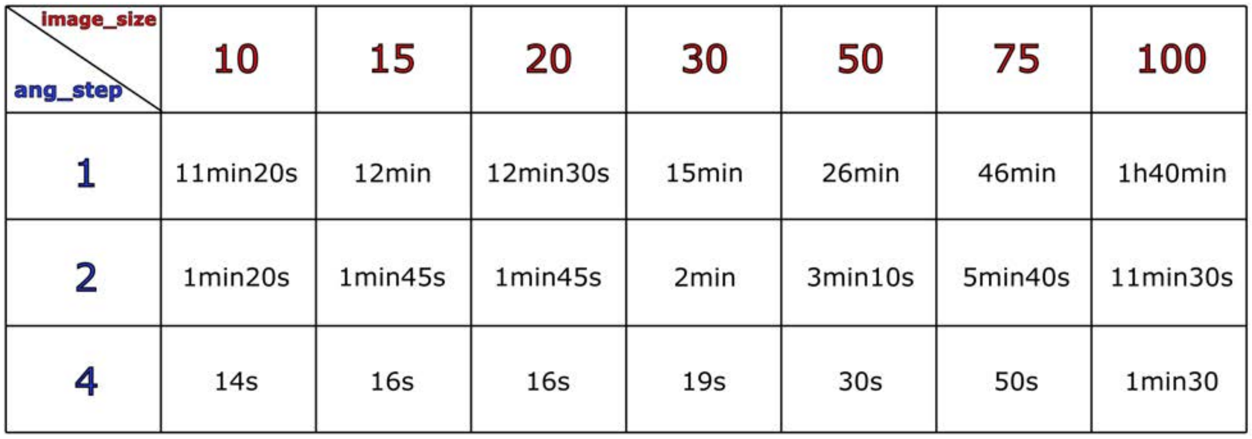


Figure SI22: Displacement value map $\bar{d} = \sqrt{(x_n - x)^2 + (y_n - y)^2}$ with $[x_n\ y_n\ z_n]' = \mathrm{R}_{\theta_x}\ [x\ y\ \sqrt{R^2_{part} - x^2 - y^2}]'$ for different rotation angles θ_z about the z-axis for a cropped 50 pixel particle image. The histograms in the bottom row show the distribution of $\bar{d}$ in the corresponding images above. For in-plane rotations in-plane (about the z-axis), the maximum absolute displacements occur at the borders of the image. As seen from the histograms, a significant number of points will experience a displacement $\bar{d}$ greater than 1 pixel when the rotation angle is greater or equal to 3°. Therefore, the threshold for detection of in-plane rotation is greater than that for out-of-plane rotations, as the central portion of the image with the highest information density also displays lower displacements for in-plane rotations.

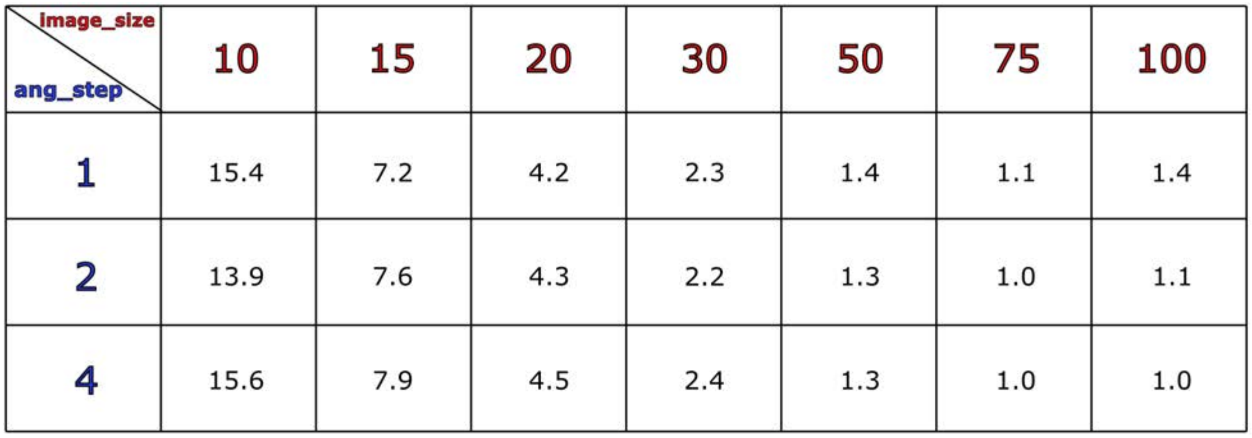
Returning to the analysis performed with *ang_step* = 0.5° and *ang_step* = 1°, the error distribution for the rotations around the z-axis does not degrade compared to that observed for larger *ang_step*, indicating an improvement in angle registration with decreasing *ang_step*. In contrast, with decreasing *ang_step*, the recorded angles around the x- and y-axis shows a tendency to underestimate the true angles, and this effect is more pronounced as the absolute value of the real angular displacements increases. However, there is a trade-off between the size of the discretisation step *ang_step* used and the computational time required for our method. For example, determining the angular displacements between 500 images, ranging between [-12° +12°], using *ang_step* = 4° for *image_size* = 50 takes 14s, which increases to approximately 3 min for *ang_step* = 2°, and 26 min for an angle step of 1° (see Figure SI23).

An alternative approach to reduce the computational time, without increasing the increments of *ang_step* used, is to reduce the size of the images (*image_size*) evaluated in the correlation process. Keeping *ang_step* fixed at 2°, approximately 11 min is required to process images with *image_size* = 100, which is reduced to around 6 min for *image_size* = 75, 3 min for *image_size* = 50, and 2 min for *image_size* = 30 (see Figure SI23). Nevertheless, decreasing the computational time by reducing the image size will also impact the precision of our method. From our simulated data, we find that with *ang_step* = 2°, the precision of the registered angles remains stable until an image size of 50 px is used. After this point, smaller images tends to decrease the precision of the rotation angles around the z-axis while it slightly biases the estimated angles around the around the x- and y-axis to be lower than the true values. This effect is increased as the image size is further reduced (see Fig SI SI20-B). Please note that some of the computation time does not simply scale with the size of matrices because of fixed times required to call functions, etc. (see Figure SI24)



image_size / ang_step	10	15	20	30	50	75	100
1	11min20s	12min	12min30s	15min	26min	46min	1h40min
2	1min20s	1min45s	1min45s	2min	3min10s	5min40s	11min30s
4	14s	16s	16s	19s	30s	50s	1min30

Figure SI23: Time required to run the code on a set of 500 images for different *image_size* and different *ang_step* with *ang_lim* = 16. As expected, increasing the angle increment size and thus reducing the number of angles analysed reduces the computational time required, while increasing the image size increases the computation time.



image_size / ang_step	10	15	20	30	50	75	100
1	15.4	7.2	4.2	2.3	1.4	1.1	1.4
2	13.9	7.6	4.3	2.2	1.3	1.0	1.1
4	15.6	7.9	4.5	2.4	1.3	1.0	1.0

Figure SI24: Ratio between the actual and the expected computational time required to compute the angles from a set of 500 images for different *image_size* and different *ang_step* with *ang_lim* = 16. The expected time is calculated with respect to the time required for the reference case *image_size* = 75 and *ang_step* = 2, which is multiplied by $\frac{image\_size^2}{75^2}$ (area) and the number of angles to compute, divided by the number of angles to compute when *ang_step* = 2. For *ang_step* = 1, the number of angles to compute is 33^3, while for *ang_step* = 2, the number is 17^3 and 9^3 for *ang_step* = 4. This ratio depends only weakly on *ang_step*, while it varies significantly if *image_size* changes. Although smaller images are processed faster, various computational steps are image-size independent, and therefore the decrease in computation time does not scale with *image_size*^{2} .

Effect of particle radius and center location

The predicted particle radius and the accurate identification of the particle centre are further parameters, which are critical to the precision of our method, even if they do not affect the computational time. We demonstrate the effect of errors in their definition in Figure SI25, by adjusting the particle radius (A) or shifting the image centre (B) from their true values, while keeping the parameters *image_size* and *ang_step* fixed at 50 pixels and 2°, respectively. An incorrect estimation of the particle radius will not noticeably affect the precision of the angles extracted for rotations around the z-axis. However, errors in the input particle radius will bias the results of both the x- and y-axis. Underestimating the particle radius leads to an overestimation of the angular displacement, which becomes more pronounced as the true rotations become larger. In contrast, overestimating the particle radius leads to underestimated rotation angles. In both cases, there appears to be a linear increase of the errors in the predicted angular displacements with the error in the estimated particle radius. We also note that underestimating the the particle radius seems to have a more drastic effect on the error of angle registration compared to its overestimation (see Figure SI25 - A).

Errors in the localisation of the particle centre can also have a significant impact on the accuracy of the estimated angles. When the tracked centre of $I_1(x, y)$ is shifted along the x-axis compared to the ground truth, whereas the reference image *I(x,y)* is perfectly centred, the errors in the estimated angular displacements are predominantly found around the y-axis with a systematic increase in the error as the accuracy of tracking **x** decreases (see Figure SI25 - B). In contrast, the distribution of errors in the recorded angles around the x- and z-axes gradually increases, but the extracted rotations are always centred around the true, simulated value. Similar trends are observed if the error in the particle centre estimated is instead along the y- or xy-axis. In the case that all images ($I_1(x, y)$, *I(x,y)*) are tracked with the same localisation error **dx** along the x-axis, the width of the distribution of errors in θ_x and θ_z increases, however, there is no shift in the measured value for θ_y (see Figure SI26).

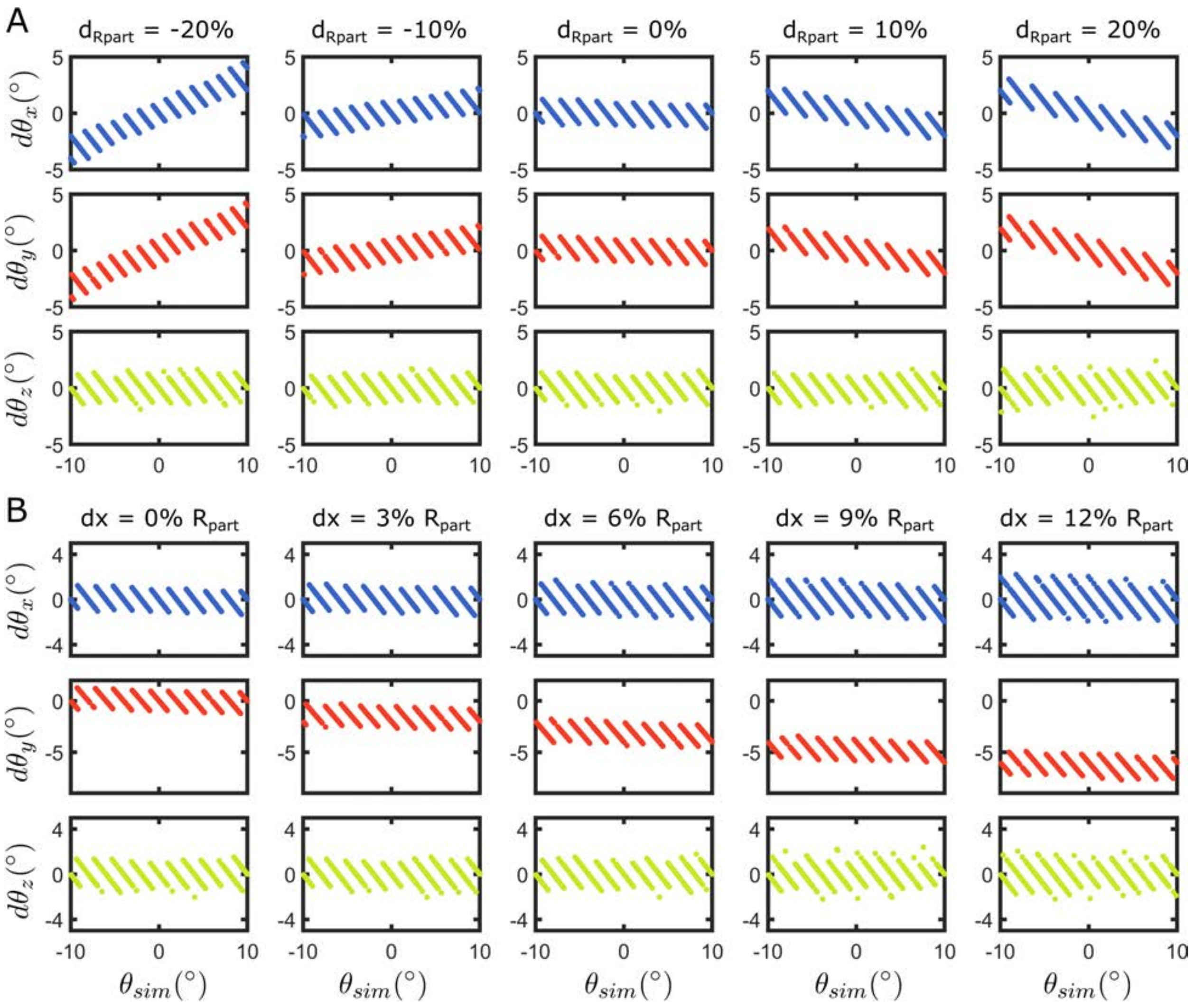


Figure SI25: Evolution of the difference $d\theta = \theta_{comp} - \theta_{sim}$ between the angles computed with our method θ_{comp} and the angles used in the simulation θ_{sim} as a function of θ_{sim} for different mismatch values of the particle radius R_{part} compared to its theoretical value $R_{part,theo}$ with $d_{R_{part}} = \frac{R_{part} - R_{part,theo}}{R_{part,theo}}$ (A) or for shift values dx of the particle center with respect to the image's center along the x-axis for one image, while, in the second image, the particle is centered (B).

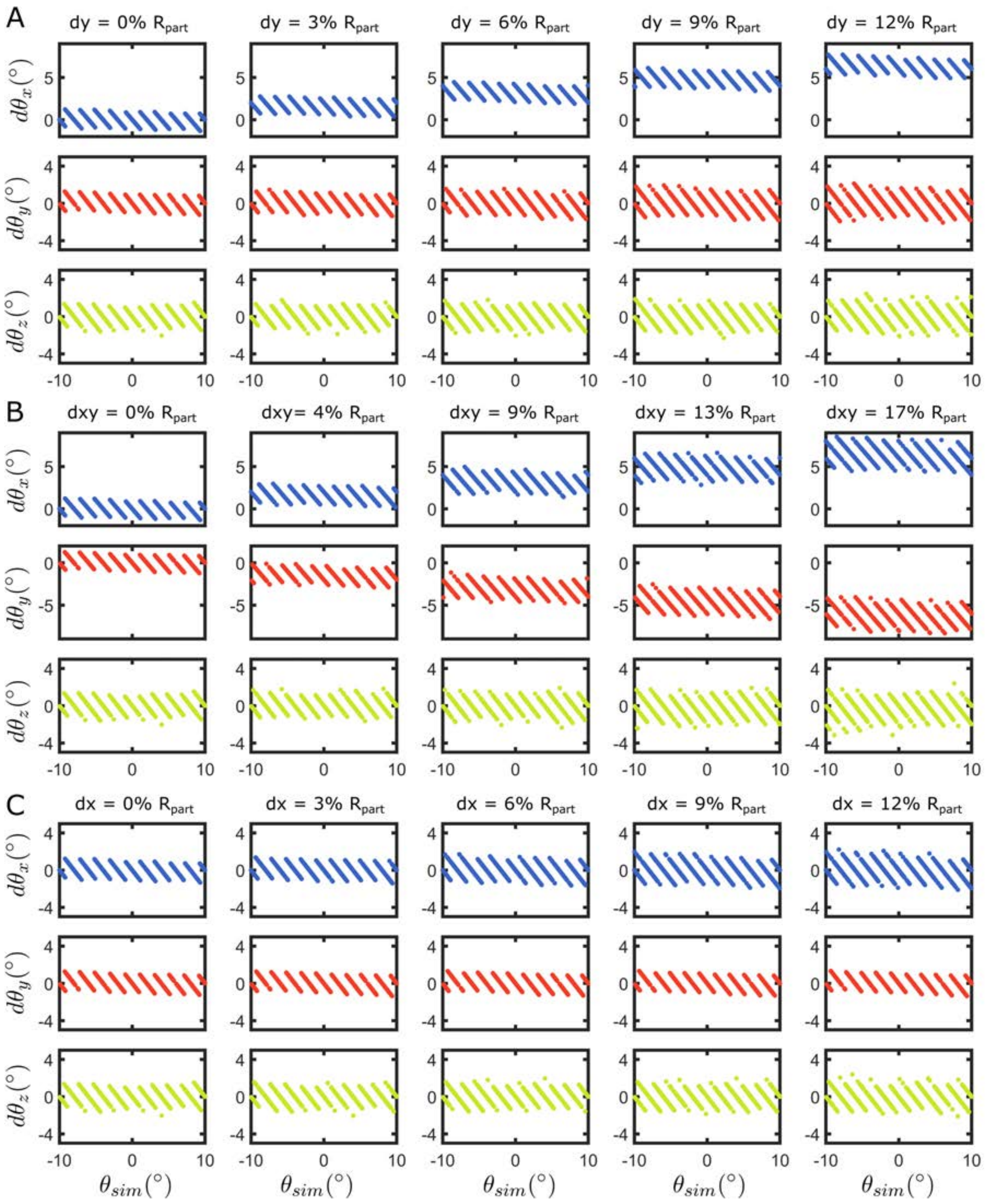
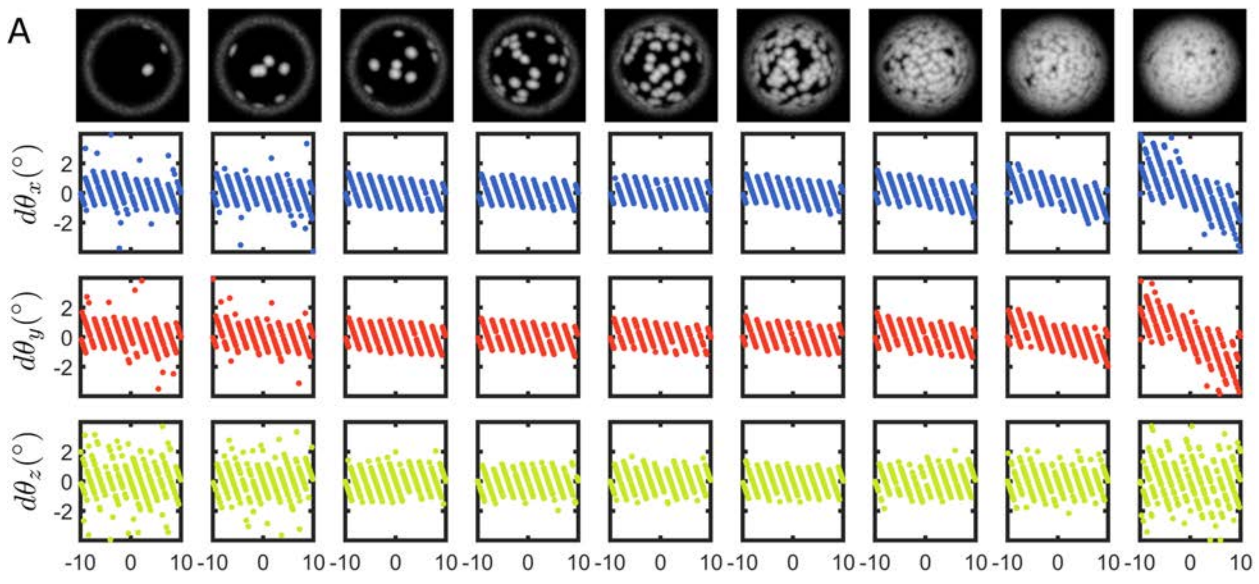


Figure SI26: Evolution of $d\theta = \theta_{comp} - \theta_{sim}$ between the angles computed with our method θ_{comp} and the angles used in the simulation θ_{sim} as a function of θ_{sim} for different scenarios where errors are introduced in particle centre finding. (A) Translations dy of the particle centre along the y-axis for one image, where the second image is properly centred. (B) Translations $dxy = \sqrt{dx^2 + dy^2}$ with dx = dy, of the particle center along the xy-axis for one image, where the second image is properly centred. (C) Translations dx of the particle centre along the x-axis for both images compared for rotation analysis

Effect of the coverage of fluorescent berries

Finally, we show that the surface density of fluorescent features (berries) on the particle surface also affects accuracy of the method. Essentially, the surface texture need to have sufficient distinguishable features to track the rotation and surfaces that are too uniform, i.e. with too few and too many berries, make the tracking difficult. However, as shown in Figure SI27, we observe that our method works for a broad range of berry coverages. As a practical note, thresholding and binarizing the images reduces the performance of the algorithm.

A



B

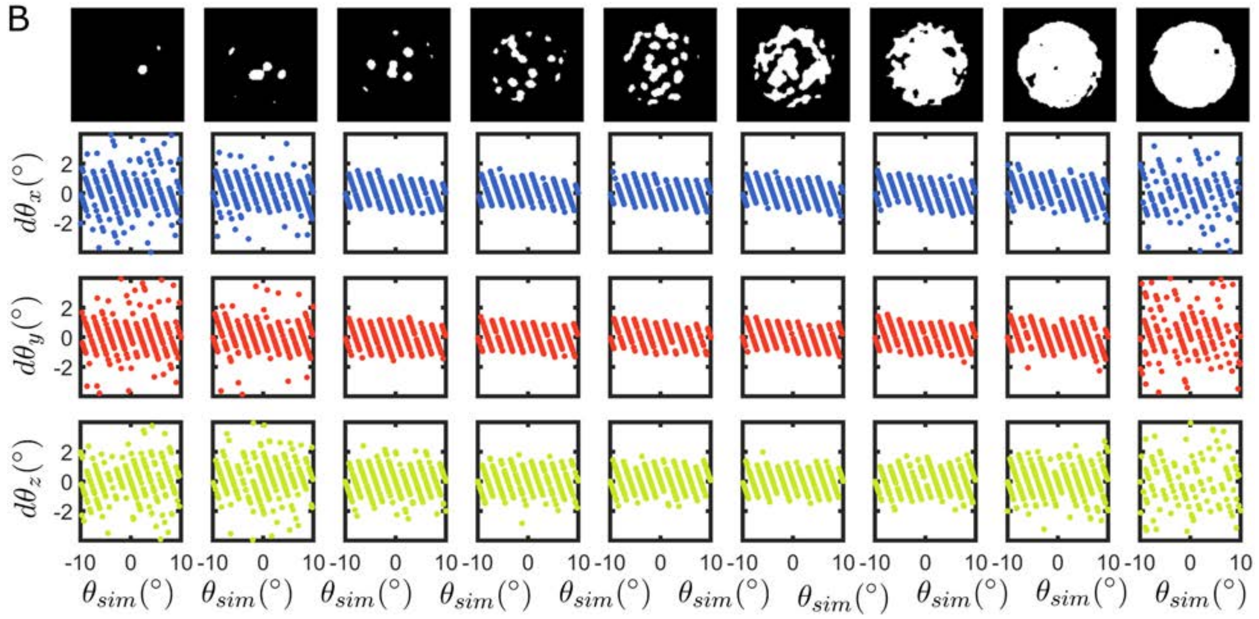


Figure SI27: Evolution of $d\theta = \theta_{comp} - \theta_{sim}$ between the estimated angles θ_{comp} with our method and the angles used in the simulation θ_{sim} between two images, as a function of the value of the simulation angle θ_{sim}, for different simulated raspberry particles and contrast. (A) Effect of increasing the berry coverage on the particles. (B) Same computations and in (A) but with binarized images.

Different approaches to track rotation

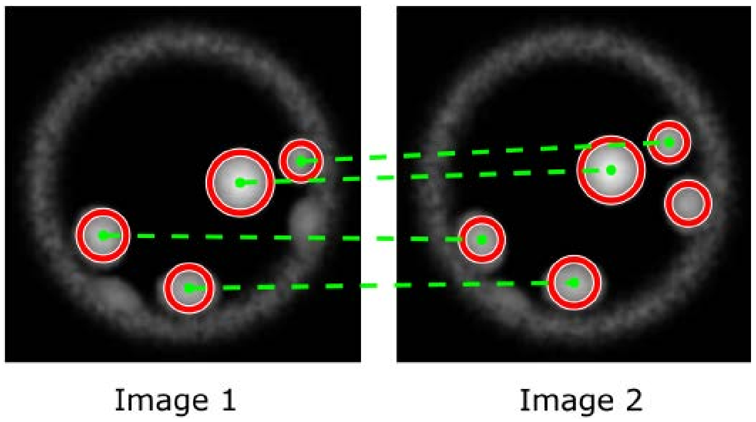


Figure SI28: Example of automated tracking and linking of berries between subsequent images. The positions of the berries are found with the function *imfindcircles* from MATLAB. An example of the tracking is shown with the red circles on both images *Image 1* and *Image 2*. The berries are matched then with the help of the code *trackmem* from Maria Kilfoil (Feb. 05) which is an MATLAB implementation of the particle tracking algorithms developed by Crocker and Grier[3]. The matched berries between *Image 1* and *Image 2* are linked by the dotted green lines.

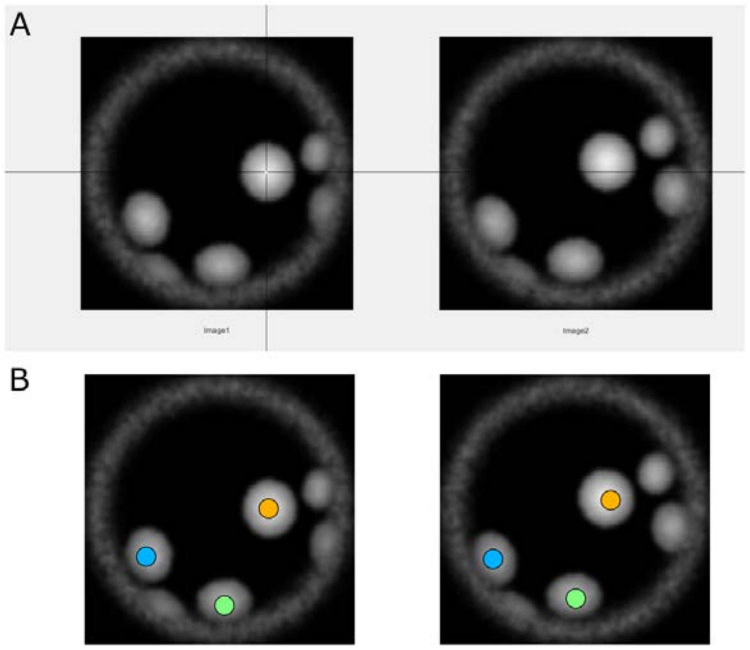


Figure SI29: Example of manual tracking and matching of the berries. (A) Screenshot of the window used in MATLAB to click on the particles' positions. The positions are recorded with the function *ginput*. Here, the berries between subsequent images are manually matched to eliminate potential tracking errors by automated algorithms. (B) Images of the particle in (A) with the berries identified by the colored dots.

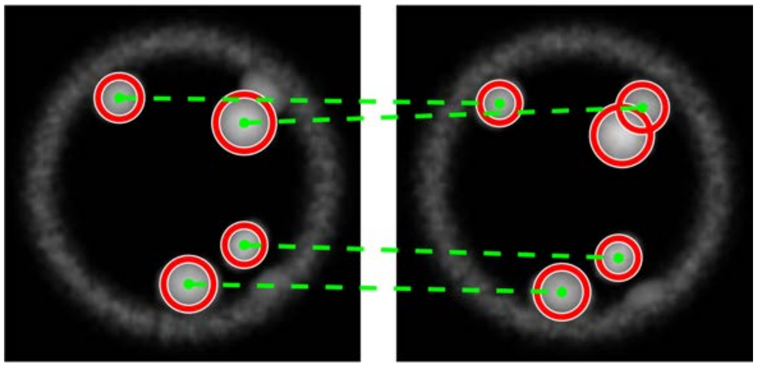


Figure SI30: Example of erroneous automated berries tracking and linking. The close proximity of two berries prevents their accurate localisation in the first image, leading to a mismatch during the linking step with the subsequent frame.

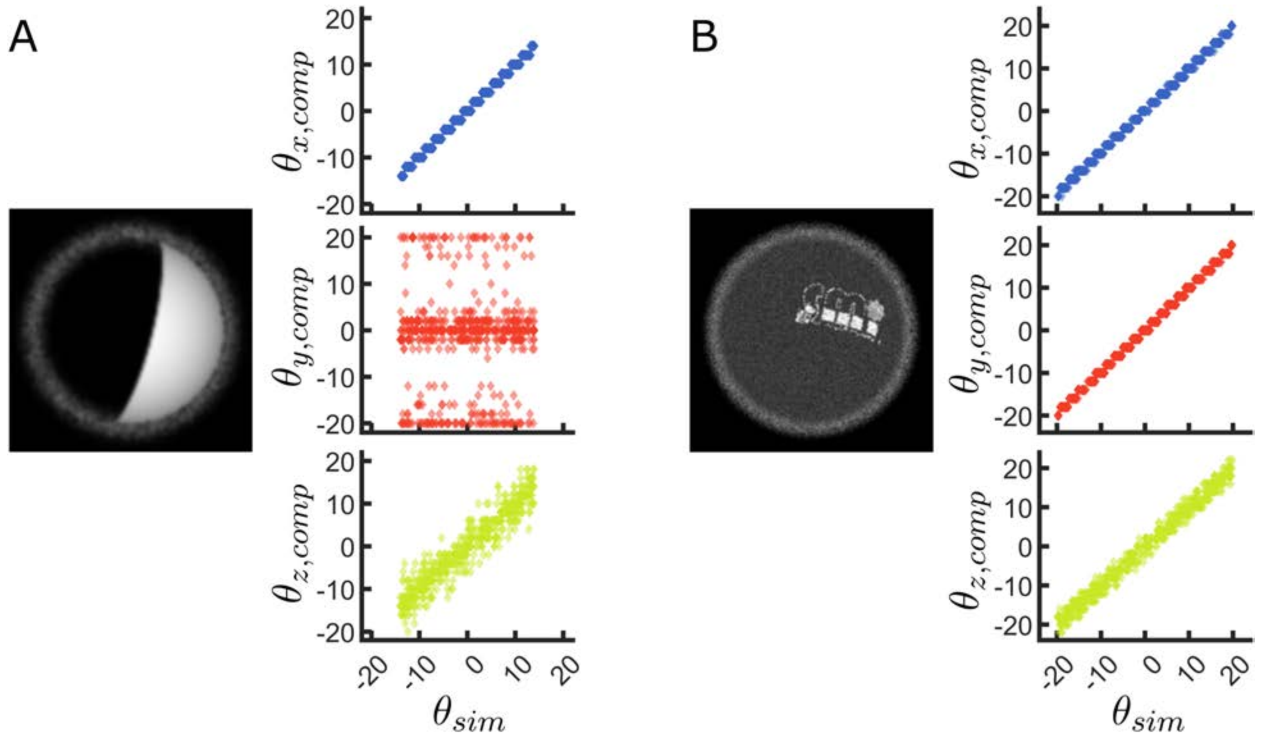


Figure SI31: Computed rotation angles around the x-axis $\theta_{x,comp}$, the y-axis $\theta_{y,comp}$ and the z-axis $\theta_{z,comp}$ compared to the corresponding angles from the simulation θ_{sim} for our 3-D rotation registration method with an angle limit of $\pm 20°$ and angle step of $2°$ for the case where the particle is a Janus particle (A) or a particle with our SMI logo on its surface (B). An image of the simulated particle used for the studies can be found at the left of the plots in each case. When generating the simulated data, the sphere is rotated in all 3 directions each time. We note that for the case in (B) where no rotational symmetry of the particle texture is present, our approach correctly tracks all three rotations, while in (A), where the surface of the Janus particle has an optical symmetry, the method's performance is significantly lower.

Flow of particles in a capillary channel

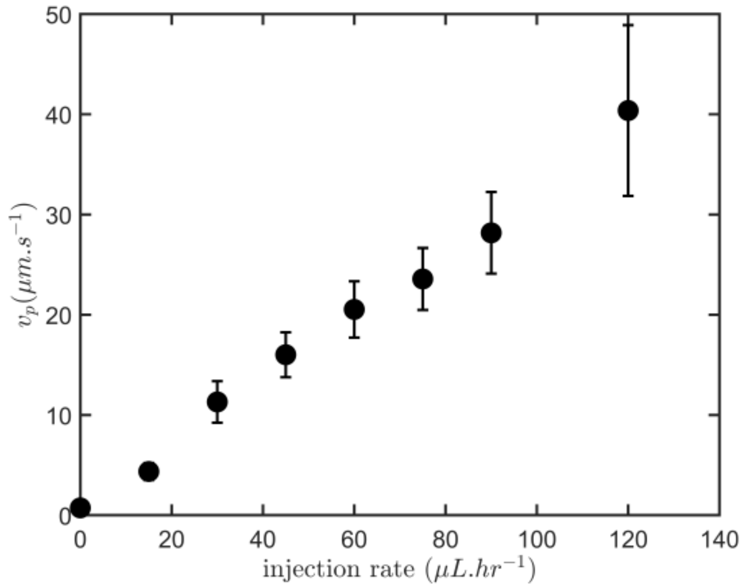
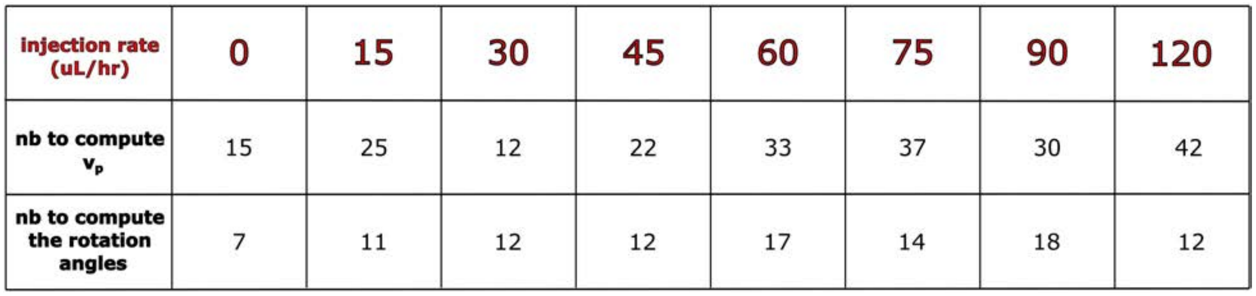


Figure SI32: Mean translation velocity v_p of the raspberries particles measured for different injection rates in the capillary channel. The error bars are the standard deviations of v_p for each injection rate. The number of particles used to obtain these data is reported in Figure SI33.



injection rate (uL/hr)	**0**	**15**	**30**	**45**	**60**	**75**	**90**	**120**
nb to compute v_p	15	25	12	22	33	37	30	42
nb to compute the rotation angles	7	11	12	12	17	14	18	12

Figure SI33: Number of particles (nb) used to compute the mean and standard deviation of translation velocity of the particles (v_p), as well as the number of particles used to determine their rotational motion, for the different injection rates.

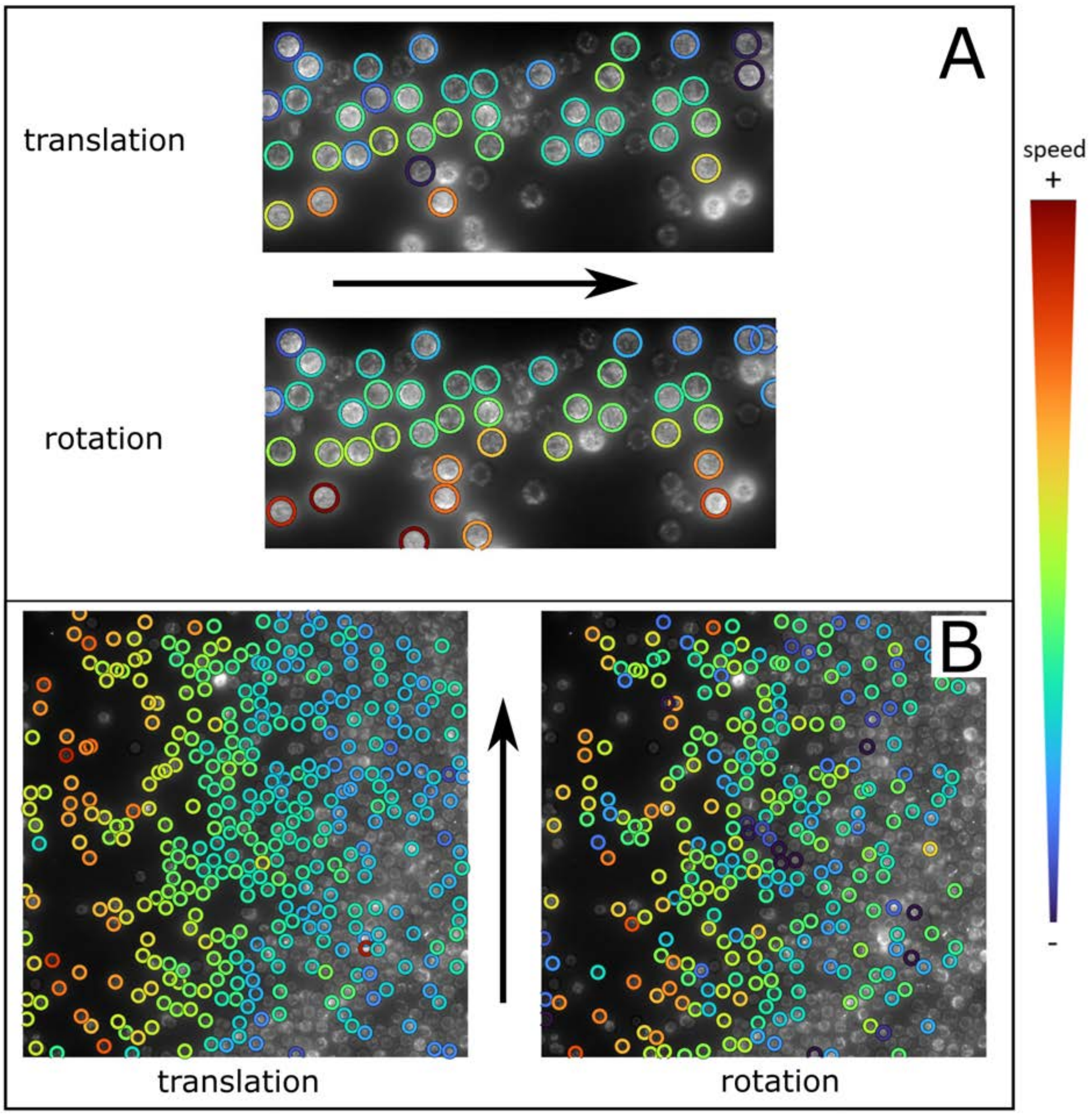


Figure SI34: Particle flown in a capillary channel (black arrows indicate direction of flow). A: Particles in close proximity to the capillary wall (top edge of the respective images) B: Particles flowing through the centre of the capillary, with a gradient of particle density.

Freely-rotating particles

As discussed in the main text, we can compare the measured rotational diffusion coefficients for freely moving particles next to a glass substrate.

We treat all the angular displacements along the three axes as from the same distribution, and fit the MSAD equation for one dimension $MSAD = 2D_R\tau$. Fitting up to $\tau = 1.2$ and averaging over the 3 particles shown in Figures 4 A-D, we obtain a rotational diffusion coefficient corresponding of $\hat{D_R} = 0.0141\ rad^2s^{-1}$, which corresponds to an average particle radius $\hat{R_p} = 2.28\ \mu$m, assuming Stokes-Einstein's relation $D_R = k_BT/(8\pi\eta R^3)$, with η being the fluid viscosity. This is good agreement with the measured particle radius of 2 μm obtained from electron micrographs. In contrast, by extracting an effective particle size from translational diffusivity and using the Stokes-Einstein's relation in bulk $D_T = k_BT/(6\pi\eta R)$, we initially obtain $\hat{R_p} = 5.11\ \mu$m. However, assuming that the true hydrodynamic radius of the particles' is the measured value of 2 μm, we apply Faxen's correction factor for a wall separation $h = 0.1R$ and find that the measured value is 92.3% of the expected value for this distance.

We observe that both the in-plane (z) and out-of-plane (average x-y rotation) show a linear dependence of the MSAD with lag time, as expecting for objects undergoing rotational diffusion. The fact that the slope (in a log-log plot) of the out-of-plane rotation is somewhat smaller indicates a greater degree of rotational drag along the z-axis.

As discussed in the main text, some drift is present at the fluid-fluid interface, as visible in the MSD data shown in in FigureSI35.

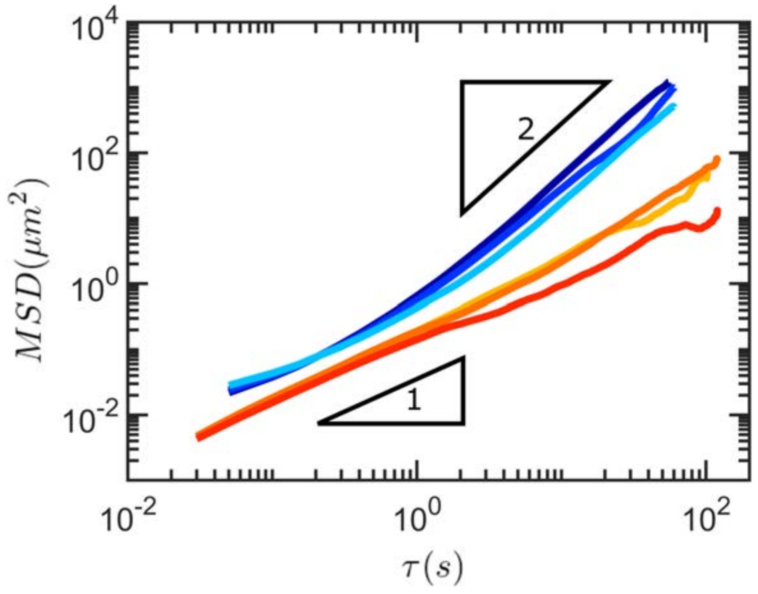


Figure SI35: Computed mean squared displacements of the 6 particles in Figure 5 of the main text, for increasing lag times τ. The presence of a ballistic regime at long lag times at the fluid interface (blue curves) a signature of convective drift, which is typically observed at fluid-fluid interfaces.

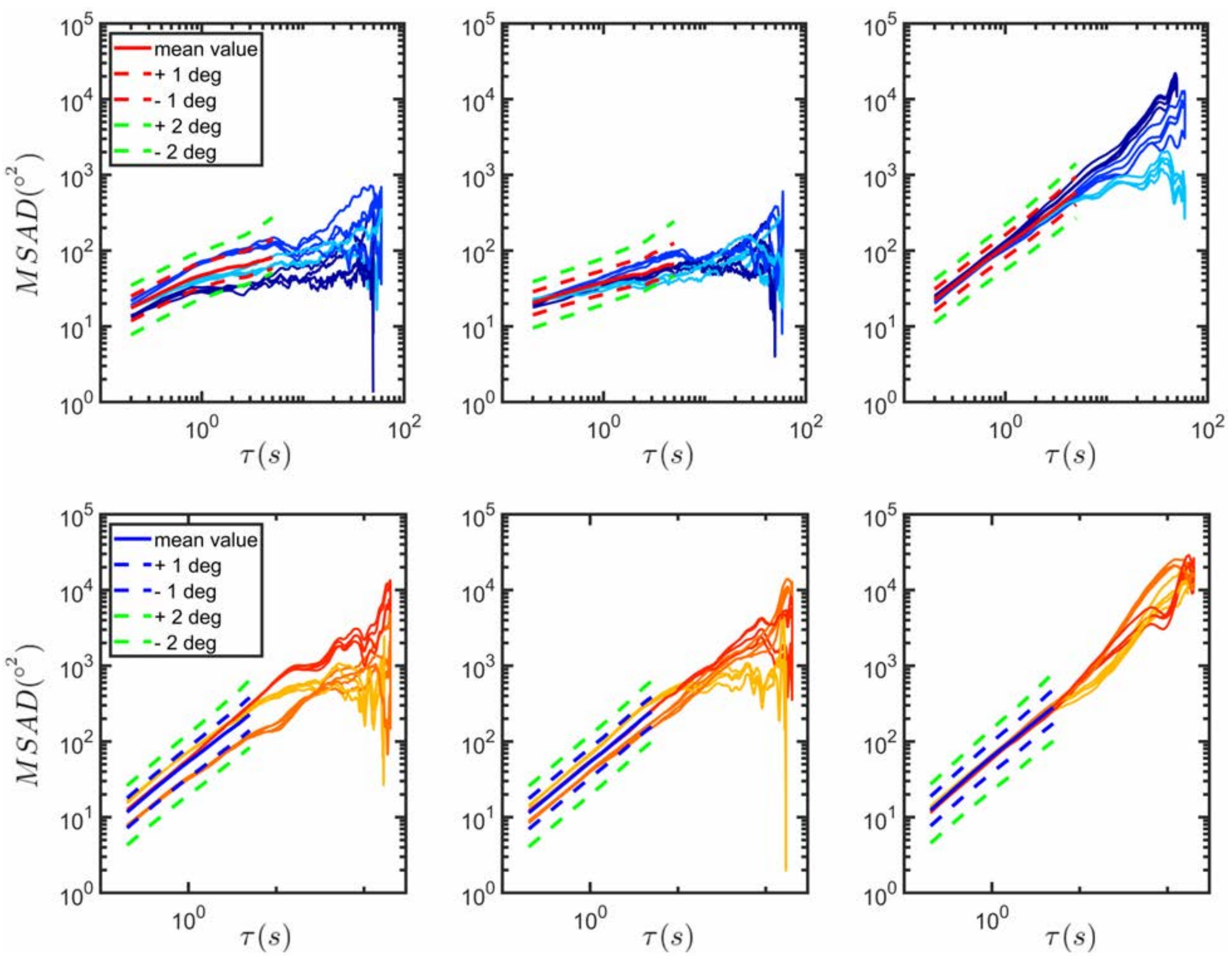


Figure SI36: Computed MSAD for the Brownian (passively diffusing) particles at the liquid-liquid (top row) and liquid-solid (bottom row) interface. Columns indicate the MSAD for the rotations θ_x, θ_y, and θ_z respectively. The dashed lines show the MSADs for a systematic over(under)estimation of the computed angles by 1 or 2 degrees. To compute the +1 degree line (resp. +2), to each measured angle, +1 (resp. +2) was added if the value was positive or null and -1 (resp. -2) was added if the value was negative. To compute the -1 degree line (resp. -2), for each measured angle θ, max(θ-1,0) (resp. max(θ-2,0)) is taken for positive values while for negative values min(θ+1,0) (resp. min(θ+2,0)) is taken and if $\theta = 0$, the taken value is also 0. This is to show the worst case scenarios where the estimated angles have a constant bias of 1° or 2° error.

Different considerations on the calculation of the MASD

Changing the order of rotations may affect the computed angular displacements due to discretization. In order to evaluate this effect, below we show the difference in the computed angles around the x-, y- and z-axis for different rotation orders (see Figure SI37). In this case, the difference in angles between the two rotation orders is typically within 2°.

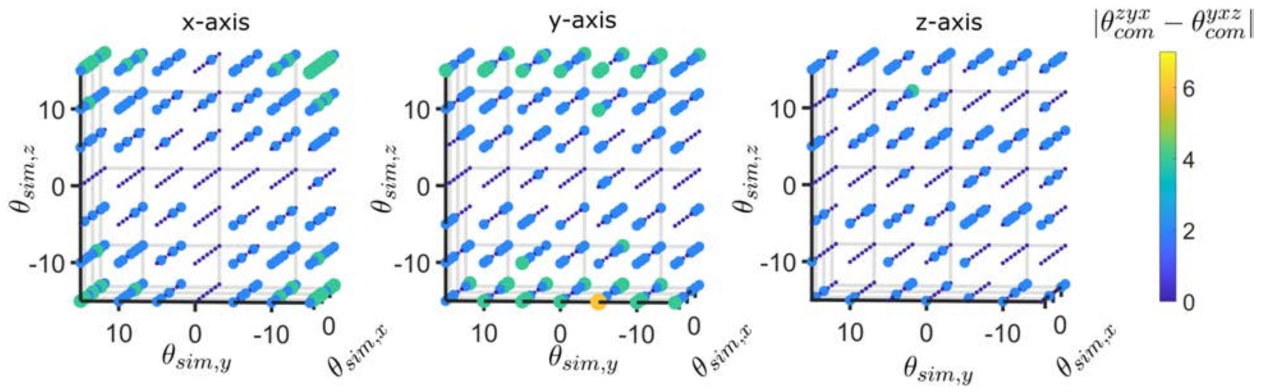


Figure SI37: Absolute difference between the computed rotation angles retrieve using a rotation matrix $R_{zyx}=R_z R_y R_x$ compared to the ones obtained regarding the rotation matrix $R_{yxz}=R_y R_x R_z$ for different simulated angles around the 3 axis. The plot on the left shows the computed angles around the x-axis, the one in the center around the y-axis while the one on the left shows the results obtained around the z-axis. .

Moreover, because angular displacements are bound to 2π in all directions, there is a difference between the true MASD and a cumulative MASD. In our computations, we always report the cumulative MASD, where we keep adding angles over time, but in general

$$R(\theta_z(t1)+\theta_z(t2), \theta_y(t1)+\theta_y(t2), \theta_x(t1)+\theta_x(t2)) \neq R(\theta_z(t2), \theta_y(t2), \theta_x(t2)).R(\theta_z(t1), \theta_y(t1), \theta_x(t1))$$

However, for small angular displacements, the two types of MASD are very close (see Figure SI38)

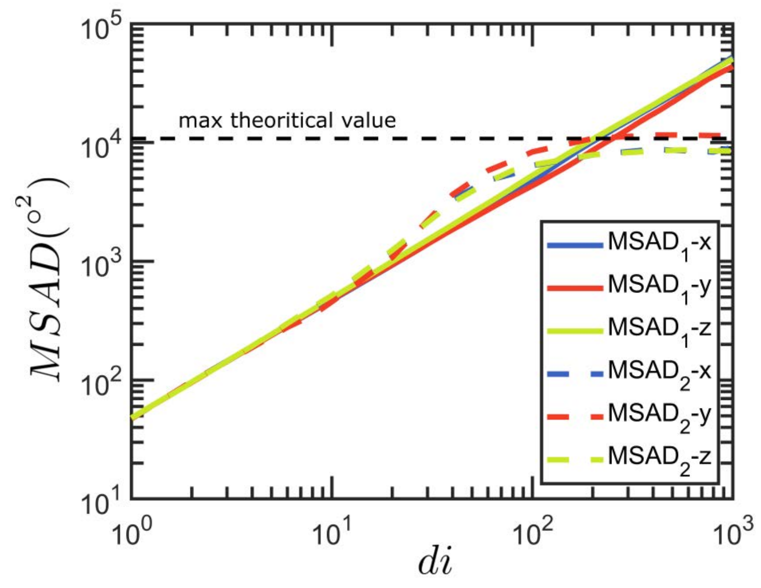


Figure SI38: Computed cumulative MSAD (MSAD1) and true MSAD (MSAD2) as a function of time lag between images (di) for a set of 40000 angles chosen randomly between $\pm 12°$. The cumulative MSAD is computed by always adding the angles while the actual MASD is computed by calculating the different rotation matrix associated to each rotation where the angular displacements are bound to $[-180°; 180°]$.

Moreover, in all the MSAD data shown in the main text, the rotations around one axis are typically greater than the ones around the other two, therefore the total rotation is practically independent of the rotation order (see Figure SI39)

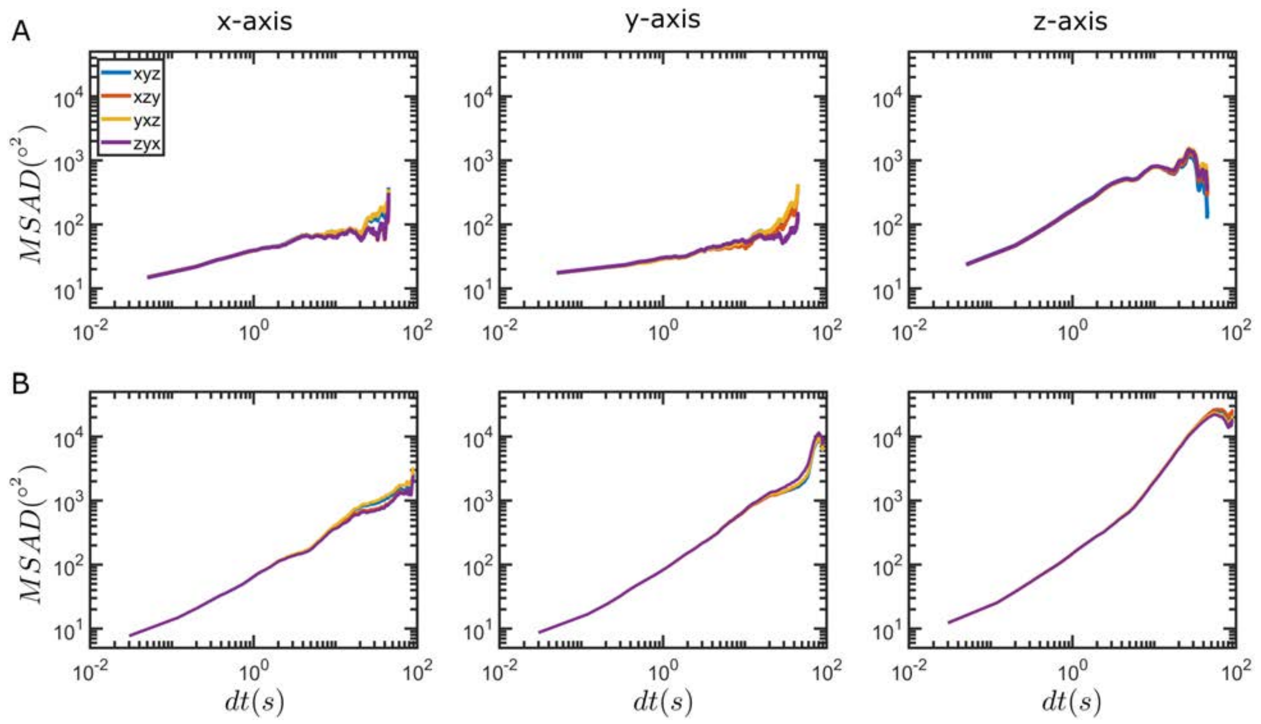


Figure SI39: Computed cumulative MSAD obtained with different rotation orders (xyz is first a rotation around the z-axis, then around the y-axis and finally the x-axis) to compute the rotation angles around the x-axis (first column), the y-axis (second column) or the z-axis (third column) for an experimental particle pinned at the water/oil interface (A) or an experimental particle freely moving in water above a glass slide (B). As it can be seen the order of rotations does not influence the calculated MASD, especially at short times.

Alternative methods for 2-D rotation tracking

As discussed in the main text, the fact that the displacements and the rotation of particles adsorbed at the interface are effectively confined in 2-D within the interface plane, allows us to compare our 3-D method with 2-D alternatives.

In Figure SI40, we illustrate the workflow for a 2-D rotation/registration method. First both images are cropped and resized to discard the data close to the particles' edge. Then, image 1 is rotated by an angle θ, before application of a mask (disk-shaped). The masked, rotated image is then compared to the second image i, to which the same mask was previously applied. Unlike the previous examples studying the 3-D rotation case, it is not necessary to adjust the mask shape based on the rotation angles studied. The registered angle is determined as the rotation angle which provides the best match (maximising the correlation coefficient) between the masked rotated image and the second image.

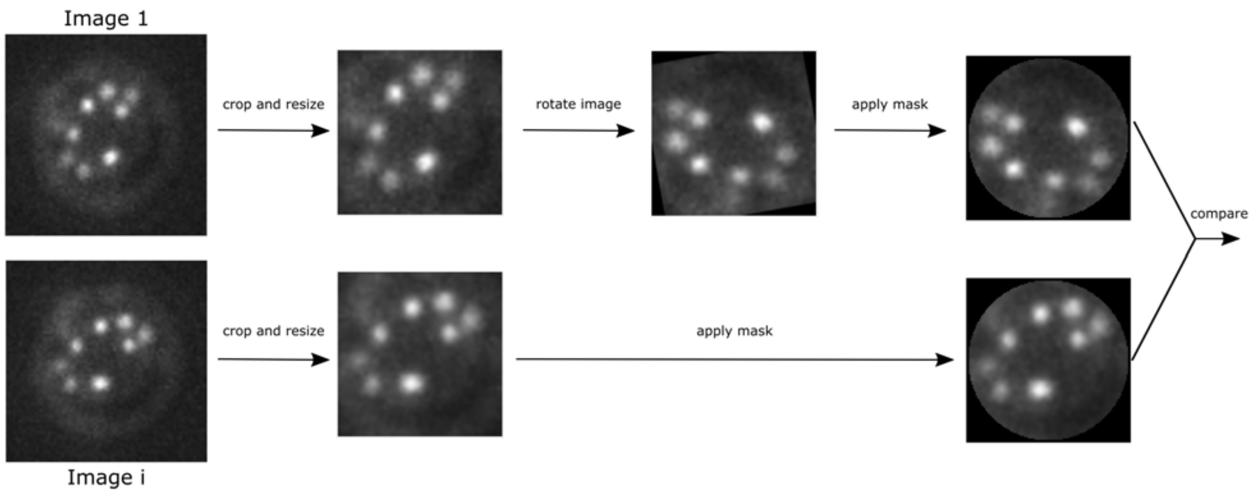


Figure SI40: Schematic of the image rotation registration method, applied to particles rotating in 2-D only at the interface.

In Figure SI41 we illustrate the work flow for the rotation/registration method based on the FFT of particle images. This method does not require a centered image of the particle, thus removing a potential source of error during tracking, however it only applies for a single particle in the field of view. The micrograph is first resized and added to a larger background image before computation of its 2-D FFT (using the in-built MATLAB function *fft2*), to increase detail in the resulting FFT image. The absolute value of this image is squared, then the image scale value is then logged before application of the MATLAB function *fftshift*. The image is then cropped to retain the useful information. The same processing is applied to the second image. Finally, the two FFT images are compared and rotation is measured using the method described in Figure SI40.

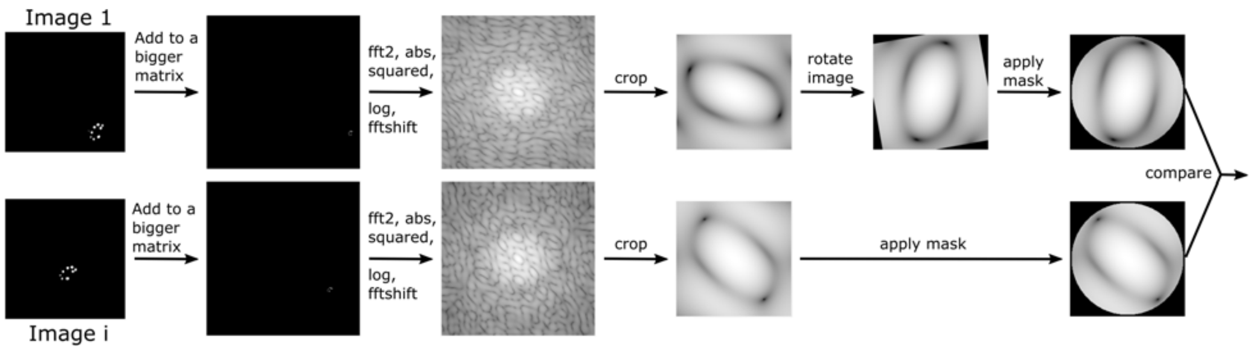


Figure SI41: Schematic of the rotation registration method based on the FFT of images with the experimental micrograph of the particle placed randomly in a bigger black frame.

Active particles

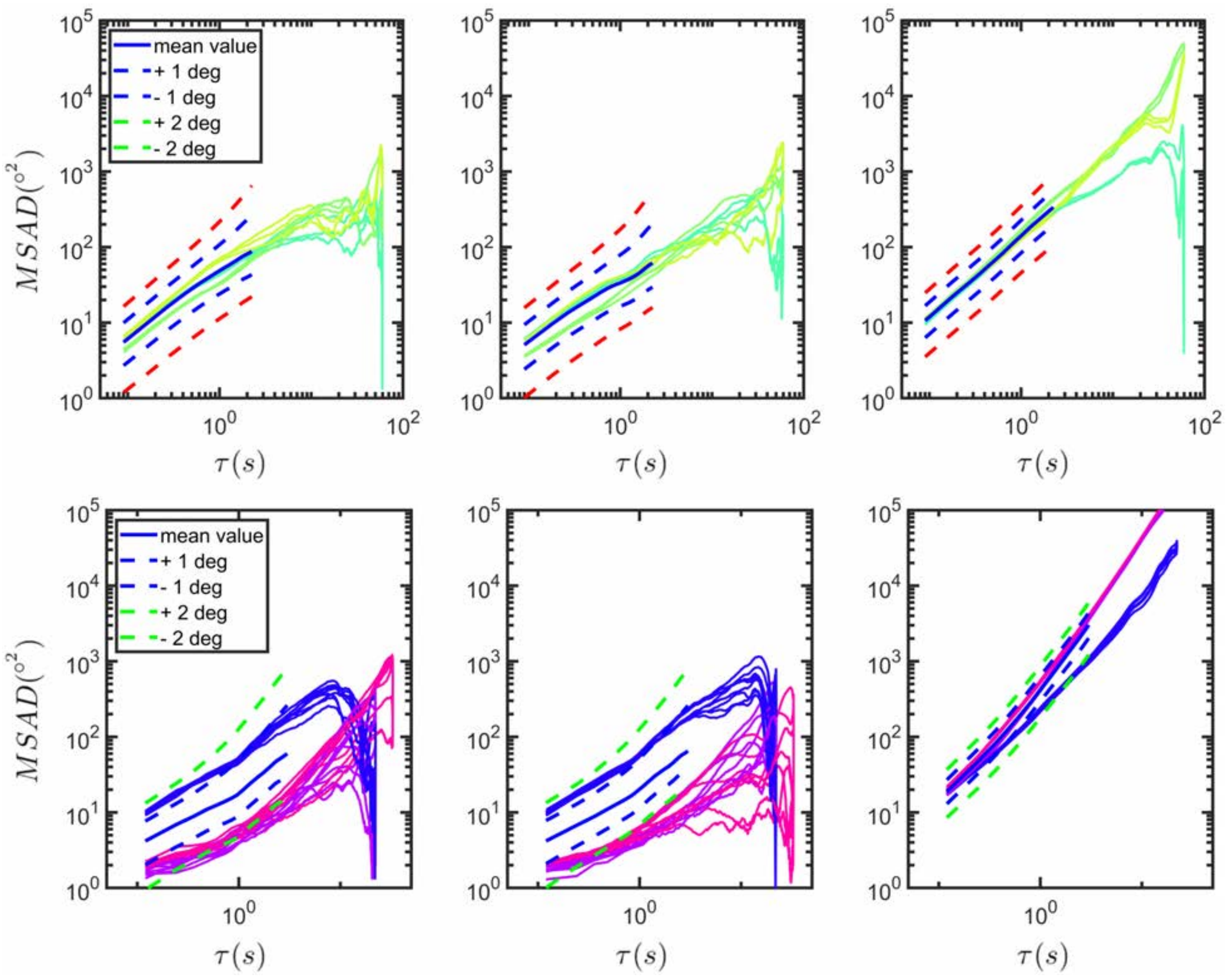


Figure SI42: Computed MSAD for the Janus raspberry particles (with Pt cap) in the absence of 3 v/v% H_2O_2 fuel (Brownian motion, top row) or in the presence of 3 v/v% H_2O_2 (active motion, bottom row). Columns indicate the MSAD for the rotations θ_x, θ_y, and θ_z respectively. The dashed lines show the MSADs for a systematic over(under)estimation of the computed angles by 1 or 2 degrees. To compute the +1 degree line (resp. +2), to each measured angle, +1 (resp. +2) was added if the value was positive or null and -1 (resp. -2) was added if the value was negative. To compute the -1 degree line (resp. -2), for each measured angle θ, max(θ-1,0) (resp. max(θ-2,0)) is taken for positive values while for negative values min(θ+1,0) (resp. min(θ+2,0)) is taken and if $\theta = 0$, the taken value is also 0. This is to show the worst case scenarios where the estimated angles have a constant bias of 1° or 2° error.

Here, we evaluate the error associated with tracking θ_z of active particles from their displacements, as it has been previously performed in the literature, e.g.[4,5]. Assuming that the orientation of an active particle coincides with its swimming direction, angular displacements can thus be derived from changes in the propulsion direction. However, this indirect method suffers from two main limitations: one associated to localisation errors during particle tracking and the other one deriving from the fact that, especially at short times, directed ballistic motion overlaps with random displacements from Brownian diffusion, such that displacement and orientation are actually decoupled. We investigate this errors by considering two equal displacements of a particle (black dots with fixed distance $\bar{d}$) from its starting point, with a true angle θ_{theo} between the subsequent displacements. We then introduce noise in the centre finding, defined by the green circle radius d_{noise} and define the error in centre finding by randomly sampling a coordinate (blue dots) located within the circle radius d_{noise}. From the blue dots, we now obtain a new measured angle θ_{noise} which accounts for the noise terms introduced by localisation error and the random displacements (see Figure SI43 - A). By computing 1000000 errors cases, the 0.05 quantile $\theta_{q(0.05)}$ and the 0.95 quantile $\theta_{q(0.95)}$ of the angle distribution θ_{noise} can be extracted (see Figure SI43 - B). The 'width' of the distribution $d_{\theta_q}^{15^\circ} = \theta_{q(0.95)}$-$\theta_{q(0.05)}$, which can be seen as the error one could obtain on the computed angle is decreasing as the position error noise d_{noise} get smaller compared to the theoretical distance between two points $\bar{d}$. In other words, as one could imagine, the measured angle is less prone to error if the error on the position is getting smaller compared to the travelled distance. For instance, with our simulation, one could expect an accuracy on the measured angle of less than one degree only if the distance between two points is more than 100 times the precision on the position localisation (see Figure SI43 - C). Moreover these results are almost independent of the theoretical angle of displacement θ_{theo} (see Figure SI43 - D).

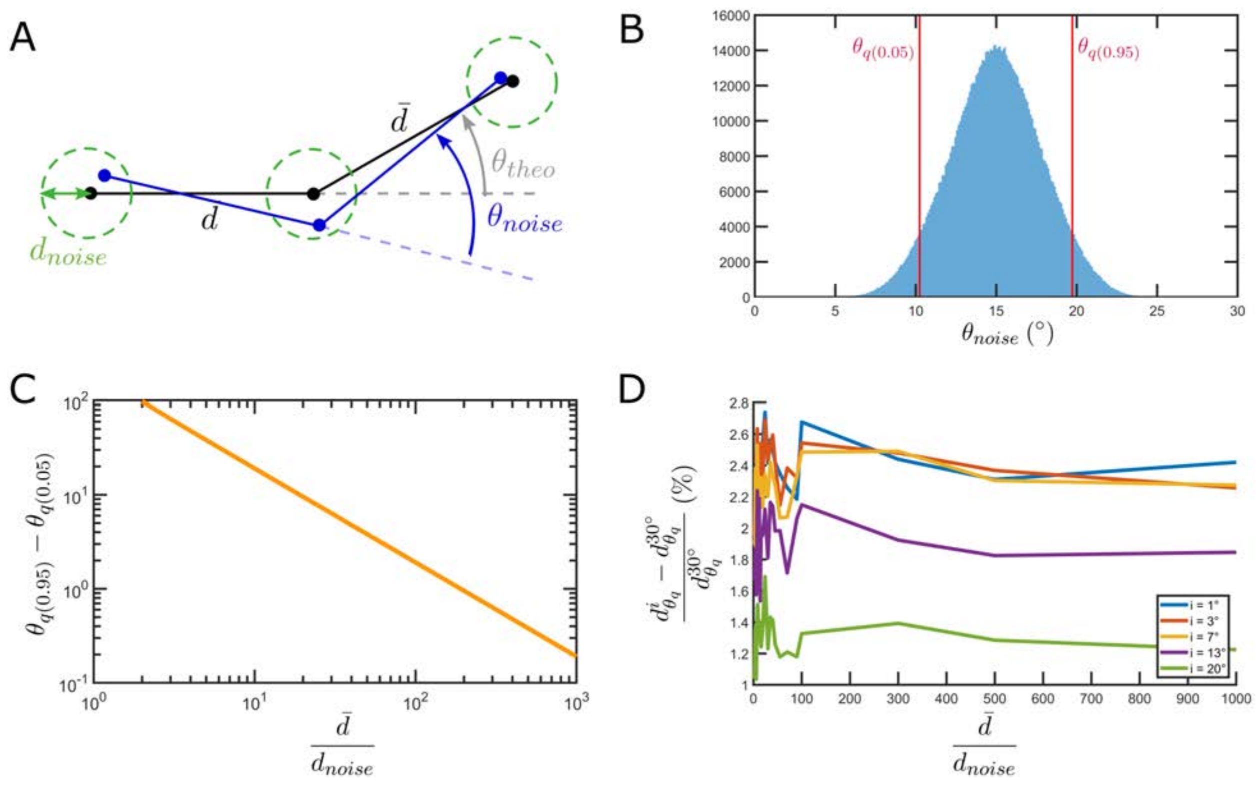


Figure SI43: (A) Scheme of our simulation to determine the influence of the position accuracy on the measured displacement angle. (B) Distribution of measured displacement angles θ_{noise} obtained by simulating 1000000 cases with $\theta_{theo} = 15°$ and the taking points with positions randomly picked with a maximum distance $d_{noise} = 0.1\ \bar{d}$ of the theoretical position. $\theta_{q(0.05)}$ (resp. $\theta_{q(0.95)}$ is the 0.05 (resp. 0.95) quantile of the distribution. (C) Evolution of the 'width' of the distribution $d^{15°}_{\theta_q} = \theta_{q(0.95)}$-$\theta_{q(0.05)}$ with $\theta_{theo} = 15°$ relative to the ratio between the theoretical displacement between two points $\bar{d}$ and the positional noise limit d_{noise}. (D) Evolution of the difference between the width of the distributions for the angle $\theta_{theo} = i°$ ($d^i_{\theta_q}$) and the one for the angle $\theta_{theo} = 30°$ ($d^{30°}_{\theta_q}$), normalised by the width of the distribution for the angle $\theta_{theo} = 30°$, for different ratios of $\bar{d}$ to d_{noise}.

Suppelmentary Movies

SM1: Recorded images of particles flowing in a glass capillary with an injection rate of 60 μL/hr. The white scale bar represents 8 μm. The right part shows a magnified view of each tracked particle in a comoving reference frame.

SM2: Recorded images of a particle freely diffusing in water close to a glass substrate. The scale bar represents 8μm. The right part shows a close-up of the tracked particle in a comoving reference frame.

SM3: Recorded images of a particle absorbed at a hexadecane - water interface. The scale bar represents 8μm.The right part shows a close-up of the tracked particle in a comoving reference frame.

SM4: Recorded images of a Janus particle with a 5 nm platinium cap freely diffusing in water close to a glass substrate. The scale bar represents 8μm. The right part shows a close-up of the tracked particle in a comoving reference frame.

SM5: Recorded images of Janus particles with a 5 nm platinum cap swimming in an 3 v/v% H_2O_2 aqueous solution. The scale bar represents 8μm. The right part shows a magnified view of each tracked particle in a comoving reference frame.

References

(1) Deserno, M. How to generate equidistributed points on the surface of a sphere. 2004; https://www.cmu.edu/biolphys/deserno/pdf/sphere_equi.pdf.

(2) Weisstein, E. W. Sphere Point Picking. From *MathWorld*–A Wolfram Web Resource. https://mathworld.wolfram.com/SpherePointPicking.html.

(3) Crocker, J. C.; Grier, D. G. Methods of Digital Video Microscopy for Colloidal Studies. *Journal of Colloid and Interface Science* **1996**, *179*, 298–310.

(4) Mestre, R.; Palacios, L. S.; Miguel-López, A.; Arqué, X.; Pagonabarraga, I.; Sánchez, S. Extraction of the propulsive speed of catalytic nano- and micro-motors under different motion dynamics. 2020; https://arxiv.org/abs/2007.15316.

(5) Dietrich, K.; Volpe, G.; Sulaiman, M. N.; Renggli, D.; Buttinoni, I.; Isa, L. Active Atoms and Interstitials in Two-Dimensional Colloidal Crystals. *Phys. Rev. Lett.* **2018**, *120*, 268004.