

# 1 Robotics.m

```

1 %Bouncing on a discretized surface and reorientation parallel to the
2 %tangent of the boundary
3
4 clear variables; close all; tic;
5
6 %This final version of the code makes use of bitmaps created in subfolders
7 %located in the "Bitmaps.n" folders (from local path)
8 %All units are in so called "pixels"
9
10 %Notes:
11 %The spatial separation is 1 pixel
12 %The time separation dt should be as small as possible thus it is logical
13 %to make it equal to dl. However, this is not strictly speaking required
14 %here (the value for dt can be changed)
15
16 %Conventions used in namings:
17 %No capital letter: simple (1x1) variable or 1D vector
18 %First letter capitalized: 2D matrix
19 %All capitalized: 3D matrix
20
21 %Exceptions to the above are listed below:
22 %c_sol is (*,5) 2D matrix | notation is consistent with that of t_sol
23
24
25 %% Part one: Loading the necessary data into matrices, from the folders
26 %Select main folder:
27 Bitmaps_directories = dir('*Bitmaps*');
28 str = {Bitmaps_directories.name};
29 prompt = {'Select main folder to load surface, walls, and/or workspace from'};
30 [selection_main,OK] = listdlg('PromptString',prompt,...
31                               'SelectionMode','single',...
32                               'ListSize',[500 150],...
33                               'ListString',str);
34 if(~OK); return; end; %Make sure something was selected
35 Bitmaps_dir = str(selection_main);
36
37 f = fopen([char(Bitmap_dir) '/n.txt'],'r'); n = fscanf(f,'%i'); fclose(f); ...
38 %size of the workspace (in pixels)
39 f = fopen([char(Bitmap_dir) '/rho.txt'],'r'); r = fscanf(f,'%i'); fclose(f); ...
40 %radius of the sphere/robot (in pixels)
41 mainDirectory = dir(char(Bitmap_dir));
42 subDirectories = find(vertcat(mainDirectory.isdir));
43 str = {mainDirectory(subDirectories).name};
44 %Remove the '.', '..' and 'Sphere_robot directories
45 good_str_index = (~strcmp(str, '.') & ~strcmp(str, '..') & ...
46                  ~strcmp(str, 'Sphere_robot'));
47 str = str(good_str_index);
48 %Choose to enter the surface and two walls separately or all at once in a
49 %workspace:

```

```

47 choice = {'Enter surface, and walls separately', 'Enter full workspace'};
48 [selection,OK] = listdlg('PromptString','Make a selection',...
49     'SelectionMode','single',...
50     'ListSize',[500 100],...
51     'ListString',choice);
52 if(~OK); return; end; %Make sure something was selected
53 if(selection == 1)
54     % Getting the surface matrix:
55     [selection_S,OK] = listdlg('PromptString','Select a surface:',...
56     'SelectionMode','single',...
57     'ListSize',[500 500],...
58     'ListString',str);
59     if(~OK); return; end; %Make sure something was selected
60     surf_str = str(selection_S); %Store the name of the selected surface
61     SURF=false(n,n,n);
62     for i=1:n
63         filename=[char(Bitmaps_dir) '/' char(surf_str) '/' int2str(i) '.png'];
64         SURF(:, :, i)=imread(filename);
65     end
66     WORKSPACE = (SURF);
67     clear('SURF');
68     % Getting the 1st wall matrix:
69     [selection_W1,OK] = listdlg('PromptString','Select the first wall:',...
70     'SelectionMode','single',...
71     'ListSize',[500 500],...
72     'ListString',str);
73     if(~OK); return; end; %Make sure something was selected
74     wall1_str = str(selection_W1); %Store the name of the selected wall 1
75     WALL1=false(n,n,n);
76     for i=1:n
77         filename=[char(Bitmaps_dir) '/' char(wall1_str) '/' int2str(i) '.png'];
78         WALL1(:, :, i)=imread(filename);
79     end
80     WORKSPACE = (WORKSPACE | WALL1);
81     clear('WALL1');
82     % Getting the 2nd wall matrix:
83     [selection_W2,OK] = listdlg('PromptString','Select the second wall:',...
84     'SelectionMode','single',...
85     'ListSize',[500 500],...
86     'ListString',str);
87     if(~OK); return; end; %Make sure something was selected
88     wall2_str = str(selection_W2); %Store the name of the selected wall 2
89     WALL2=false(n,n,n);
90     for i=1:n
91         filename=[char(Bitmaps_dir) '/' char(wall2_str) '/' int2str(i) '.png'];
92         WALL2(:, :, i)=imread(filename);
93     end
94     % Creating the complete WORKSPACE matrix which includes both walls and the
95     % surface.
96     % Note: such a matrix could potentially have a corresponding image that
97     % would be loaded (similarly to the above), in which case the above 3
98     % matrices could be ignored
99     WORKSPACE = (WORKSPACE | WALL2);
100     clear('WALL2'); %Cleaning up for memory space and speed
101 else

```

```

102 % Getting the full workspace matrix:
103 [selection.W,OK] = listdlg('PromptString','Select a full workspace:',...
104     'SelectionMode','single',...
105     'ListSize',[500 500],...
106     'ListString',str);
107 if(~OK); return; end; %Make sure something was selected
108 workspace_str = str(selection.W); %Store the name of the selected wall 2
109 WORKSPACE=false(n,n,n);
110 for i=1:n
111     filename=[char(Bitmaps_dir) '/' char(workspace_str) '/' int2str(i) '.png'];
112     WORKSPACE(:, :, i)=imread(filename);
113 end
114 end
115
116 % Getting the robot matrix!
117 ROBOT=false(n,n,n);
118 for i=1:n
119     filename=[char(Bitmaps_dir) '/' 'Sphere-robot/' int2str(i) '.png'];
120     ROBOT(:, :, i)=imread(filename);
121 end
122
123 % "Positionning" the robot on the surface
124 % This step is not necessary if the loaded ROBOT matrix is properly
125 % positioned. However, in our case we use a single sphere always located at
126 % the bottom center of the workspace, which can then easily be shifted around
127 % i.e. the default position of the bottom of the sphere with respect to the
128 % workspace is [n/2,n/2,0]
129 if(strcmp(surf_str,'Flat_surf'))
130     xshift = 0; yshift = 0; zshift = n/2;
131 elseif(strcmp(surf_str,'Flat_Surf'))
132     xshift = 0; yshift = 0; zshift = n/2;
133 elseif(strcmp(surf_str,'Tilted_x_Surf'));
134     alpha = pi/4; h = floor(r*((1/cos(alpha))-1));
135     xshift = 0; yshift = 0; zshift = n/2+h;
136 elseif(strcmp(surf_str,'Tilted_y_Surf'));
137     alpha = pi/4; h = floor(r*((1/cos(alpha))-1));
138     xshift = 0; yshift = 0; zshift = n/2+h;
139 elseif(strcmp(surf_str,'Tilted_xy_Surf'));
140     alpha = pi/4; h = floor(r*((1/cos(alpha))-1));
141     xshift = 0; yshift = 0; zshift = n/2+h;
142 elseif(strcmp(surf_str,'Sine_x_Surf'));
143     xshift = round(pi*r*2-n/2)-r/5; yshift = 0; zshift = 2*r;
144 elseif(strcmp(surf_str,'Sine_y_Surf'));
145     xshift = 0; yshift = round(pi*r*2-n/2); zshift = 2*r;
146 elseif(strcmp(surf_str,'Sine_xy_Surf'));
147     xshift = round(pi*r*2-n/2); yshift = round(pi*r*2-n/2); zshift = 2*r;
148 elseif(strcmp(surf_str,'Gaussian_dome_Surf'));
149     xshift = 0; yshift = 0; zshift = ...
150         floor(r^1.5)-1-ceil((r*(sqrt(r)-7.8)+abs(r*(sqrt(r)-7.8)))/2);
151 elseif(strcmp(surf_str,'PhotoShop_FlatSurf'));
152     xshift = 0; yshift = 0; zshift = 0;
153 else
154     fprintf('ERROR: The selection is not a known surface\n');
155     fprintf('The code might need to be updated\n');
156     return

```

```

156 end
157 ROBOT=circshift(ROBOT,[xshift,yshift,zshift]);
158 %So, we now have 2 matching (in size) 3D matrices: ROBOT and WORKSPACE
159
160 %% Part 2: Setting up the initial conditions:
161 %The ROBOT has it's own coordinate system which does not necessarily match
162 %that of the WORKSPACE. This allows us to keep the initial conditions for
163 %the robot fixed, and not have to worry about strange behaviors when
164 %rotating since those do occur especially when x0.robot is not close to
165 %-pi/2
166 x0_robot = -pi/2 + 1e-4;
167 y0_robot = 0;
168 psi0 = 0;
169 %Definition of the robot/sphere in it's own reference frame
170 robot = @(x,y) r*[cos(x).*cos(y); cos(x).*sin(y); sin(x)+1];
171
172 %The other initial conditions:
173 %Everything is pixelized (even time)
174 dl = 1; %Spatial separation (1 pixel)
175 dt = input('Enter dt (in units of 1/r: ');
176 dt = dt/r; %Time step: normalized by r so that shift=dl when rolling
177 t_start = 0; %Initial time for the robot to begin moving
178 t_final = input('Enter final time: ');
179 %Adjust t_final to make sure it is an integer value time dt:
180 delta = (t_final-t_start)*r-floor((t_final-t_start)*r);
181 t_final = t_final - delta/r;
182 current_time = t_start; %The current time
183 bounces = 0; %Number of bounces
184 break_val = 0; %Used to break out of loops
185
186 %The velocity vector
187 vx = 0; vy = 0; vz = 0; wz = 0;
188 fprintf('Enter initial rotation velocities:\n');
189 fprintf('Note that the velocities will be automatically normalized\n');
190 wx = input('Enter initial rotation velocity wx = ');
191 wy = input('Enter initial rotation velocity wy = ');
192 %Normalize the velocity vector
193 normalization = sqrt(wx*wx + wy*wy);
194 wx = wx/normalization; wy = wy/normalization;
195 V = [vx;vy;vz;wx;wy;wz];
196
197 %The progress bar:
198 handle = waitbar(0,'1','Name','Running...',...
199 'CreateCancelBtn',...
200 'setappdata(gcf,'canceling',1)');
201 setappdata(handle,'canceling',0);
202
203 t_sol = zeros(1,round((t_final-t_start)/dt+1));
204 t_sol(1) = current_time; %Initialize total time vector
205
206 %Initializing the matrix of contact points (with the surface)
207 Contact_pts = zeros(length(t_sol),3);
208 %Getting the first contact points:
209 %Before even starting there is not any previous contact point, and there
210 %should not be any hit point (otherwise an error will be caused)

```

```

211 prev_contact_pt = 'NaN';
212 guess = 'NaN';
213 if getappdata(handle,'canceling'); delete(handle); return; end
214 [contact_pt,hit_pt,error] = ...
215     Get_hit_cont_pts(ROBOT,WORKSPACE,prev_contact_pt,r,guess);
216 if(error); return; end %Checking for the error boolean value
217 Contact_pts(1,:) = contact_pt';
218
219 %The initial surface parameters:
220 x0_surf = contact_pt(1);
221 y0_surf = contact_pt(2);
222 z0_surf = contact_pt(3);
223
224 if getappdata(handle,'canceling'); delete(handle); return; end
225 %Getting the slopes z_x and z_y (del_z/del_x and del_z/del_y):
226 z_x_old = 0; z_y_old = 0; %Values to use if "inf" is reached
227 [z_x,z_y] = Get_slopes(WORKSPACE,contact_pt,r,z_x_old,z_y_old);
228 z_x_old = z_x; z_y_old = z_y; % update the "old" values.
229 %Initial contact coordinates all into the initial contacts vector:
230 contacts0 = [x0_robot, y0_robot, x0_surf, y0_surf, psi0];
231 c_sol = zeros(length(t_sol),5);
232 c_sol(1,:) = contacts0; %Initialize total contact coordinate vector
233 %% Part 3: Setting up the fixed parameters:
234 %Some parameters never need to be updated and should be taken care of now
235 %Locally, the surface is a plane:
236 K_surf = false(2,2); %Curvature tensor
237 T_surf = false(1,2); %Torsion form
238
239 %The robot is known to be a sphere:
240 M_robot = [r, 0; 0, cos(x0_robot)]; %Metric tensor
241 K_robot = [-1/r, 0; 0, -1/r]; %Curvature tensor
242 T_robot = [0, -1/r * tan(x0_robot)]; %Torsion form
243
244 %A few 'constants' that only need to be computed once:
245 %(Refer to the book for notations)
246 R = @(psi) [cos(psi), -sin(psi); -sin(psi), -cos(psi)];
247 K = false(2,2);
248 R_cocf = @(psi) [R(psi), [0;0]; [0,0],1];
249 p = [0;0;0];
250 p_hat = [ 0 -p(3) p(2); p(3) 0 -p(1); -p(2) p(1) 0 ];
251
252 fprintf('\n\t The program is about to loop around. \n')
253 fprintf('All parameters have been initialized\n');
254 fprintf('And the current time is: %g \n', current_time);
255
256 if getappdata(handle,'canceling'); delete(handle); return; end
257 toc
258 tic
259 %%-----%%
260 %% The above are only about setting things up %%
261 %% Below is the part where things move %%
262 %%-----%%
263
264 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Now we are looping around%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
265 %**Loop #1: Loop until final allowed time:

```

```

266 index1 = 1; %Index for the main general loop (the "master" index)
267     %Note: this index is used throughout the loops to update the
268     %contacts and time matrices
269 while current.time < t_final
270     %1
271     %First, the robot is sent rolling straight into the x-direction
272     %The robot only ever rolls about the wy-axis unless it is spinning to
273     %reorientate, in which case it is rotating about the wz-axis
274     %The orientation is only specified at the end of this comming loop by
275     %the bouncing (after loop #2)
276
277     %%
278     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
279     %%% Part four: Move-on and rotate until the orientation needs to      %%%
280     %%%                                     be reset                      %%%
281     % I need to loop around: after each small step the sphere coordinates
282     % need to be reset, but not those of the surface
283
284     hit = false; %The sphere has not yet hit the boundary
285     index2 = index1; %Index of loop 2
286
287     while(~hit) %Loop #2: Loop until a boundary is hit
288         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
289         %%% Part five: Getting bigZ and bigC                               %%%
290         % To avoid passing heavy functions to the function it is easier to
291         % directly compute p.hat here and pass it into the function
292         % Again, R_psi is computed here and passed to the function
293         % Same thing for R_cocf
294         R_psi = R(psi0);
295         R_cocf_psi = R_cocf(psi0);
296         [bigZ, bigC] = Get_bigZ_bigC(z_x, z_y, ...
297             x0_robot, y0_robot, p_hat, M_robot, K_robot, ...
298             T_robot, R_psi, R_cocf_psi, V);
299         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
300         %%% Part six: Now that we have bigZ and bigC we can roll:          %%%
301         % The time and contacts needs to be updated
302         t_sol(index2+1) = current.time + dt;
303         c_sol(index2+1,:) = ((bigZ^(-1))*dt*bigC + contacts0');
304         contacts0(3) = c_sol(index2+1,3);
305         contacts0(4) = c_sol(index2+1,4);
306         % Do not update (so reset) the values for x0_robot, y0_robot, and
307         % psi0:
308         c_sol(index2+1,1) = contacts0(1);
309         c_sol(index2+1,2) = contacts0(2);
310         c_sol(index2+1,5) = contacts0(5);
311         %Shift the robot to it's new position:
312         prev_contact_pt = Contact_pts(index2,:);
313         shift_x = round(c_sol(index2+1,3)) - round(prev_contact_pt(1));
314         shift_y = round(c_sol(index2+1,4)) - round(prev_contact_pt(2));
315         %Locally, everything is a plane. So we can use the equation of a
316         %plane:  $z = z_x*x + z_y*y + c \Rightarrow dz = z_x*dx + z_y*dy$ 
317         shift_z = round(z_x*shift_x + z_y*shift_y);
318         ROBOT = circshift(ROBOT,[shift_x, shift_y, shift_z]);
319         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
320         %%% Part seven: check whether a boundary was hit:                  %%%

```

```

321     guess = ...
322         round([c_sol(index2+1,3),c_sol(index2+1,4),prev_contact_pt(3)+shift_z]);
323     if getappdata(handle,'canceling'); delete(handle); return; end
324     percentage = round(current_time/t_final*1e4)/100;
325     waitbar((current_time-t_start)/t_final,handle,sprintf('%1.2f ...
326         '%',percentage));
327     [contact_pt,hit_pt,error] = ...
328         Get_hit_cont_pts(ROBOT,WORKSPACE,prev_contact_pt,r,guess);
329     if(error); return; end %Checking for the error boolean value
330     Contact_pts(index2+1,:) = contact_pt';
331     %Getting the slopes z_x and z_y (del_z/del_x and del_z/del_y):
332     [z_x,z_y] = Get_slopes(WORKSPACE,contact_pt,r,z_x_old,z_y_old);
333     z_x_old = z_x; z_y_old = z_y; % update the "old" values.
334     if(shift_x>r/10 || shift_y>r/10 || shift_z>r/10)
335         fprintf('shift values:\t %i \t %i \t %i\n', shift_x, shift_y, shift_z);
336         fprintf('Loop 2, current time: %f \n', current_time);
337         fprintf('The robot might be \"jumping\" around...\n');
338     end
339
340     %Making sure there is a hit point before trying to update it
341     if(~strcmp(hit_pt,'NaN'))
342         bounces = bounces + 1;
343         Hit_pts(bounces,:) = hit_pt';
344         hit = true;
345         fprintf('\t The boundary was hit at time: %g\n',current_time);
346     end
347
348     %We need to reset a few values that are used throughout the loops
349     x0_robot = contacts0(1);
350     y0_robot = contacts0(2);
351     x0_surf = contacts0(3);
352     y0_surf = contacts0(4);
353     psi0 = contacts0(5);
354     % Update the index:
355     index2 = index2 + 1;
356     % Update the time:
357     current_time = current_time + dt;
358     %Break out if max time is reached
359     if(current_time >= t_final); break; end
360 end %End of loop #2 (loop until boundary is hit)
361
362 if getappdata(handle,'canceling'); delete(handle); return; end
363 %update index1:
364 index1 = index2;
365 %Again, break out if max time is reached:
366 if(current_time >= t_final); break; end
367 %%
368 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
369 %%% Part eight: figure out the angle of rotation: %%%
370 prev_contact_pt = Contact_pts(index1-1,:);
371 current_contact_pt = Contact_pts(index1,:);
372 direction_vect = (current_contact_pt-prev_contact_pt)';
373 x_hit = Hit_pts(end,1); y_hit = Hit_pts(end,2); z_hit = Hit_pts(end,3);
374
375 %Getting the normal vector to the wall
376 i = 0; done = false; tmp_found1 = false; need_break1 = false;

```

```

374     for j=0:r/10
375         for k=0:r/10
376             if(k==0 && j==0)
377                 %Do nothing
378             elseif(x_hit-i<1 || y_hit-j<1 || z_hit-k<1)
379                 need_break1 = true;
380                 break;
381             elseif(WORKSPACE(x_hit,y_hit+j,z_hit+k))
382                 if(i*i+j*j+k*k >= r*r/100)
383                     done = true;
384                     break
385                 else
386                     tmp_v1 = [i;j;k];
387                     tmp_found1 = true;
388                 end
389             elseif(WORKSPACE(x_hit,y_hit-j,z_hit+k))
390                 if(i*i+j*j+k*k >= r*r/100)
391                     j = -j;
392                     done = true;
393                     break
394                 else
395                     j = -j;
396                     tmp_v1 = [i;j;k];
397                     tmp_found1 = true;
398                 end
399             elseif(WORKSPACE(x_hit,y_hit+j,z_hit-k))
400                 if(i*i+j*j+k*k >= r*r/100)
401                     k = -k;
402                     done = true;
403                     break
404                 else
405                     k = -k;
406                     tmp_v1 = [i;j;k];
407                     tmp_found1 = true;
408                 end
409             elseif(WORKSPACE(x_hit,y_hit-j,z_hit-k))
410                 if(i*i+j*j+k*k >= r*r/100)
411                     j = -j; k = -k;
412                     done = true;
413                     break
414                 else
415                     j = -j; k = -k;
416                     tmp_v1 = [i;j;k];
417                     tmp_found1 = true;
418                 end
419             end
420         end
421         if(need_break1); break; end;
422         if(done); break; end;
423     end
424     if(done)
425         v1 = [i;j;k];
426     elseif(tmp_found1)
427         v1 = tmp_v1;
428     else

```



```

429     v1 = 'NaN';
430 end
431 j = 0; done = false; tmp_found2 = false; need_break2 = false;
432 for i=0:r/10
433     for k=0:r/10
434         if(k==0 && i==0)
435             %Do nothing
436         elseif(x_hit-i<1 || y_hit-j<1 || z_hit-k<1)
437             need_break2 = true;
438             break;
439         elseif(WORKSPACE(x_hit+i,y_hit,z_hit+k))
440             if(i*i+j*j+k*k >= r*r/100)
441                 done = true;
442                 break
443             else
444                 tmp_v2 = [i;j;k];
445                 tmp_found2 = true;
446             end
447         elseif(WORKSPACE(x_hit-i,y_hit,z_hit+k))
448             if(i*i+j*j+k*k >= r*r/100)
449                 i = -i;
450                 done = true;
451                 break
452             else
453                 i = -i;
454                 tmp_v2 = [i;j;k];
455                 tmp_found2 = true;
456             end
457         elseif(WORKSPACE(x_hit+i,y_hit,z_hit-k))
458             if(i*i+j*j+k*k >= r*r/100)
459                 k = -k;
460                 done = true;
461                 break
462             else
463                 k = -k;
464                 tmp_v2 = [i;j;k];
465                 tmp_found2 = true;
466             end
467         elseif(WORKSPACE(x_hit-i,y_hit,z_hit-k))
468             if(i*i+j*j+k*k >= r*r/100)
469                 i = -i; k = -k;
470                 done = true;
471                 break
472             else
473                 i = -i; k = -k;
474                 tmp_v2 = [i;j;k];
475                 tmp_found2 = true;
476             end
477         end
478     end
479     if(need_break2); break; end;
480     if(done); break; end;
481 end
482 if(done)
483     v2 = [i;j;k];

```

```

484 elseif(tmp_found2)
485     v2 = tmp_v2;
486 else
487     v2 = 'NaN';
488 end
489 k = 0; done = false; tmp_found3 = false; need_break3 = false;
490 for i=0:r/10
491     for j=0:r/10
492         if(j==0 && i==0)
493             %Do nothing
494         elseif(x_hit-i<1 || y_hit-j<1 || z_hit-k<1)
495             need_break3 = true;
496             break;
497         elseif(WORKSPACE(x_hit+i,y_hit+j,z_hit))
498             if(i*i+j*j+k*k >= r*r/100)
499                 done = true;
500                 break
501             else
502                 tmp_v3 = [i;j;k];
503                 tmp_found3 = true;
504             end
505         elseif(WORKSPACE(x_hit-i,y_hit+j,z_hit))
506             if(i*i+j*j+k*k >= r*r/100)
507                 i = -i;
508                 done = true;
509                 break
510             else
511                 i = -i;
512                 tmp_v3 = [i;j;k];
513                 tmp_found3 = true;
514             end
515         elseif(WORKSPACE(x_hit+i,y_hit-j,z_hit))
516             if(i*i+j*j+k*k >= r*r/100)
517                 j = -j;
518                 done = true;
519                 break
520             else
521                 j = -j;
522                 tmp_v3 = [i;j;k];
523                 tmp_found3 = true;
524             end
525         elseif(WORKSPACE(x_hit-i,y_hit-j,z_hit))
526             if(i*i+j*j+k*k >= r*r/100)
527                 i = -i; j = -j;
528                 done = true;
529                 break
530             else
531                 i = -i; j = -j;
532                 tmp_v3 = [i;j;k];
533                 tmp_found3 = true;
534             end
535     end
536 end
537 if(need_break3); break; end;
538 if(done); break; end;

```

```

539     end
540     if(done)
541         v3 = [i;j;k];
542     elseif(tmp_found3)
543         v3 = tmp_v3;
544     else
545         v3 = 'NaN';
546     end
547     if (~strcmp(v1, 'NaN') && ~strcmp(v2, 'NaN') && ~strcmp(v3, 'NaN'))
548         norm_wall1 = cross(v1,v2);
549         norm_wall2 = cross(v1,v3);
550         norm_wall3 = cross(v2,v3);
551         norm_walls = [norm_wall1,norm_wall2,norm_wall3];
552         norms = [norm(norm_wall1),norm(norm_wall2),norm(norm_wall3)];
553         index_norms = find(abs(norms - max(norms)) < 1e-5,1);
554         norm_wall = norm_walls(:,index_norms);
555     elseif(~strcmp(v1, 'NaN') && ~strcmp(v2, 'NaN'))
556         norm_wall = cross(v1,v2);
557     elseif(~strcmp(v1, 'NaN') && ~strcmp(v3, 'NaN'))
558         norm_wall = cross(v1,v3);
559     elseif(~strcmp(v2, 'NaN') && ~strcmp(v3, 'NaN'))
560         norm_wall = cross(v2,v3);
561     else
562         fprintf('ERROR: The normal to the wall cannot be computed\n');
563         return
564     end
565     if(norm(norm_wall)<1e-5);
566         fprintf('ERROR: The normal to the wall is too close to (0,0,0)\n');
567         return
568     end
569     %The normalizing the normal vector to the wall:
570     norm_wall = norm_wall/norm(norm_wall);
571     if(dot(direction_vect,norm_wall)<0); norm_wall = -norm_wall; end
572     %The bouncing vector (the new direction):
573     bounce_vector_0 = direction_vect-2*dot(direction_vect,norm_wall)*norm_wall;
574     bounce_vector = bounce_vector_0;
575     bounce_vector(3) = 0; %We don't want a z-direction
576     bounce_vector = bounce_vector/norm(bounce_vector); %normalize
577     %The reorientation vector (tangent to the surface):
578     norm_wall = -norm_wall;
579     reorientation_vector = ...
580         bounce_vector_0-dot(bounce_vector_0,norm_wall)*norm_wall;
581     reorientation_vector(3) = 0; %We don't want a z-direction
582     if(reorientation_vector(1) == 0 && reorientation_vector(2) == 0)
583         reorientation_vector(1) = norm_wall(2)*sign(wy);
584         reorientation_vector(2) = -norm_wall(1)*sign(wy);
585     end
586     reorientation_vector = reorientation_vector/norm(reorientation_vector); ...
587         %normalize
588     if(isnan(reorientation_vector(1)) || isnan(reorientation_vector(2)))
589         fprintf('ERROR: There was an issue computing the reorientation vector\n');
590         return
591     end
592     wx = -bounce_vector(2);
593     wy = bounce_vector(1);

```

```

592     V = [0;0;0;wx;wy;0];
593
594     %Check we are not going over time:
595     if(current_time >= t_final); break; end
596
597     if getappdata(handle,'canceling'); delete(handle); return; end
598     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
599     %% Part nine: move away from the boundary                                %%
600     move = 1/(5*dt);%r/5;
601     for index3 = index1:index1+move %Loop #3: Loop a little to move away
602                                     %from boundary
603         %Again we are rolling:
604         R_psi = R(psi0);
605         R_cocf_psi = R_cocf(psi0);
606         [bigZ, bigC] = Get_bigZ_bigC(z_x, z_y, ...
607             x0_robot, y0_robot, p_hat, M_robot, K_robot, ...
608             T_robot, R_psi, R_cocf_psi, V);
609         current_time = current_time + dt;
610         t_sol(index3+1) = current_time;
611         c_sol(index3+1,:) = ((bigZ^(-1))*dt*bigC + contacts0)';
612         c_sol(index3+1,1) = contacts0(1);
613         c_sol(index3+1,2) = contacts0(2);
614         contacts0(3) = c_sol(index3+1,3);
615         contacts0(4) = c_sol(index3+1,4);
616         c_sol(index3+1,5) = contacts0(5);
617         %Shift the robot to it's new position:
618         prev_contact_pt = Contact_pts(index3,:);
619         shift_x = round(c_sol(index3+1,3)) - round(prev_contact_pt(1));
620         shift_y = round(c_sol(index3+1,4)) - round(prev_contact_pt(2));
621
622         %Locally, everything is a plane. So we can use the equation of a
623         %plane:  $z = z_x*x + z_y*y + c \Rightarrow dz = z_x*dx + z_y*dy$ 
624         shift_z = round(z_x*shift_x + z_y*shift_y);
625         if(shift_x>r/10 || shift_y>r/10 || shift_z>r/10)
626             fprintf('shift values:\t %i \t %i \t %i\n', shift_x, shift_y, shift_z);
627             fprintf('Loop 3, current time: %f \n', current_time);
628             fprintf('The robot might be \"jumping\" around...\n');
629         end
630         ROBOT = circshift(ROBOT,[shift_x, shift_y, shift_z]);
631         %Check we are not going over time
632         if(current_time >= t_final); break; end
633         %Check whether a boundary was hit.
634         %In theory there should be enough room so that it doesn't.
635         %If a boundary is hit an error message will be displayed.
636         guess = ...
            round([c_sol(index3+1,3),c_sol(index3+1,4),prev_contact_pt(3)+shift_z]);
637         if getappdata(handle,'canceling'); delete(handle); return; end
638         percentage = round(current_time/t_final*1e4)/100;
639         waitbar((current_time-t_start)/t_final,handle,sprintf('%1.2f ...
            %%',percentage));
640         [contact_pt,hit_pt,error] = ...
            Get_hit_cont_pts(ROBOT,WORKSPACE,prev_contact_pt,r,guess);
641         if(error); return; end %Checking for the error boolean value
642         Contact_pts(index3+1,:) = contact_pt';
643
644

```

```

645     %Getting the slopes z_x and z_y (del_z/del_x and del_z/del_y):
646     [z_x,z_y] = Get_slopes(WORKSPACE,contact_pt,r,z_x_old,z_y_old);
647     z_x_old = z_x; z_y_old = z_y; % update the "old" values.
648     %In the event where there is a hit point then an error message is
649     %displayed and the program jumps to plotting
650     if(~strcmp(hit_pt,'NaN'))
651         fprintf('WARNING: The boundary was hit before reorientation\n');
652         break_val = 0;
653         %break;
654     end
655     %We need to reset a few values that are used throughout the loops
656     x0_robot = contacts0(1);
657     y0_robot = contacts0(2);
658     x0_surf = contacts0(3);
659     y0_surf = contacts0(4);
660     psi0 = contacts0(5);
661     end %End of loop #3 (moving away from the boundary)
662
663     if(break_val); break; end
664     %Updating index1:
665     index1 = index3+1;
666     %Check we are not going over time:
667     if(current_time >= t_final); break; end
668
669     if getappdata(handle,'canceling'); delete(handle); return; end
670     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
671     %%% Part ten: Reorientating the robot %%%
672     % The robot now has to reorientate to be in the direction of the
673     % tangent line to the wall
674     wx = -reorientation_vector(2);
675     wy = reorientation_vector(1);
676     V = [0; 0; 0; wx; wy; 0];
677
678     % The code can now move back up to allow for the robot to roll in the
679     % new direction
680     end %End of loop #1 (loop until final allowed time)
681
682     if getappdata(handle,'canceling'); delete(handle); return; end
683     percentage = round(current_time/t_final*1e4)/100;
684     waitbar((current_time-t_start)/t_final,handle,sprintf('%1.2f %%',percentage));
685     %
686     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
687     %%% Final part: Plot %%%
688     %The first plot is the contact coordinates
689     figure(1); clf;
690     %Make the psi angle between -pi and pi as well as y_robot
691     %contacts_total(:,2) = mod(contacts_total(:,2),2*pi) - pi;
692     %contacts_total(:,5) = mod(contacts_total(:,5),2*pi) - pi;
693     plot(t_sol, c_sol(:,1), 'b', ... %x_robot
694          t_sol, c_sol(:,2), 'r', ... %y_robot
695          t_sol, c_sol(:,3), 'g', ... %x_surf
696          t_sol, c_sol(:,4), 'c', ... %y_surf
697          t_sol, c_sol(:,5), 'm', ... %psi
698          t_sol, 0); %The t-axis
699     minix = min(t_sol); %Minimum time value

```

```

700 maxix = max(t_sol); %Maximum time value
701 miniy = min(min(c_sol)); %Minimum contacts value
702 maxiy = max(max(c_sol)); %Maximum contacts value
703 %Note: the two min/max are necessary because contacts_total is a matrix
704 %and not just a vector
705 range = [minix*1.1, maxix*1.1, miniy*1.1, maxiy*1.1];
706 %Set the range of the y-axis:
707 axis(range)
708 legend('x-{robot}', 'y-{robot}', 'x-{surface}', 'y-{surface}', ...
709        'angle \psi', 'Location', 'best');
710 title('Contact coordinates of rolling robot');
711 xlabel('time');
712
713 %Figure 1, but split in 2 to see better.
714 figure(2); clf; hold on;
715 subplot(2,1,1)
716 plot(t_sol, c_sol(:,1), 'b', ...
717       t_sol, c_sol(:,2), 'r', ...
718       t_sol, c_sol(:,5), 'm', ...
719       t_sol, 0);
720 legend('x-{robot}', 'y-{robot}', 'angle \psi', 'Location', 'best');
721 title('Robot Contact coordinates of rolling robot');
722 xlabel('time');
723 subplot(2,1,2)
724 plot(t_sol, c_sol(:,3), 'g', ...
725       t_sol, c_sol(:,4), 'c', ...
726       t_sol, 0);
727 legend('x-{surface}', 'y-{surface}', 'Location', 'best');
728 title('Surface Contact coordinates of rolling robot');
729 xlabel('time');
730
731 %The third plot is the robot with the final "workspace" coordinates (not
732 %centered at 0)
733 %Since the robot is always a sphere, this plot is kind of useless, but just
734 %in case something goes wrong it is nice to have it.
735 figure(3); clf; hold on;
736 [x_robot,y_robot,z_robot]=ind2sub(size(ROBOT), find(ROBOT));
737 plot3(x_robot,y_robot,z_robot,'b. ');
738 %Over plot the contact point:
739 contact_robot = robot(c_sol(:,1)', c_sol(:,2)');
740 contact_robot = contact_robot + (n/2+1)*ones(size(contact_robot));
741 %plot3(contact_robot(1), contact_robot(2), contact_robot(3), ...
742        '%r.','markersize', 6);
743 xlabel('x-axis'); ylabel('y-axis'); zlabel('z-axis')
744 title('Mapping x-{robot} and y-{robot} back to the sphere rolling on the plane')
745 view(160,10); axis equal; box on; colormap cool;
746 hold off;
747
748 %The fourth plot is the surface with the two boundaries as well as the
749 %contacts over plotted
750 figure(4); clf; hold on;
751 [x_workspace,y_workspace,z_workspace]=ind2sub(size(WORKSPACE), find(WORKSPACE));
752 plot3(x_workspace,y_workspace,z_workspace,'b.','markersize',0.1);
753 xlabel('x-axis'); ylabel('y-axis'); zlabel('z-axis')
754 %Over plot the contact points:

```

```

755 x_contact_surf = Contact_pts(:,1);
756 y_contact_surf = Contact_pts(:,2);
757 z_contact_surf = Contact_pts(:,3);
758 plot3(x_contact_surf, y_contact_surf, z_contact_surf, ...
759       'r.','markersize', 6);
760 title('Mapping x_{surface} and y_{surface} back to the surface')
761 view(-15,65); axis equal; box on; colormap cool;
762 hold off;
763
764 %The fifth plot is a nicer version of the fourth plot:
765 figure(5); clf; hold on;
766 clear('ROBOT'); clear('WORKSPACE');
767 if(selection == 1) %Surface and walls were selected separately
768     %Plotting the surface:
769     Surf=false(n,n);
770     Z_surf = zeros(n,n);
771     for i=1:n
772         filename=[char(Bitmaps_dir) '/' char(surf_str) '/' int2str(i) '.png'];
773         Surf(:,:)=imread(filename);
774         Z_surf(Surf') = i;
775     end
776     [X_surf,Y_surf] = meshgrid((1:n),(1:n));
777     mesh(X_surf,Y_surf,Z_surf);
778     %Plotting the first wall:
779     Wall1=false(n,n);
780     Z_wall1 = zeros(n,n);
781     for i=1:n
782         filename=[char(Bitmaps_dir) '/' char(wall1_str) '/' int2str(i) '.png'];
783         Wall1(:,:)=imread(filename);
784         Z_wall1(Wall1') = i;
785     end
786     [X_wall1,Y_wall1] = meshgrid((1:n),(1:n));
787     waterfall(X_wall1,Y_wall1,Z_wall1);
788     %Plotting the second wall:
789     Wall2=false(n,n);
790     Z_wall2 = zeros(n,n);
791     for i=1:n
792         filename=[char(Bitmaps_dir) '/' char(wall2_str) '/' int2str(i) '.png'];
793         Wall2(:,:)=imread(filename);
794         Z_wall2(Wall2') = i;
795     end
796     [X_wall2,Y_wall2] = meshgrid((1:n),(1:n));
797     waterfall(X_wall2,Y_wall2,Z_wall2);
798
799 else %The full workspace was entered at once
800     Workspace = false(n,n);
801     Z_workspace = zeros(n,n);
802     for i=1:n
803         filename=[char(Bitmaps_dir) '/' char(workspace_str) '/' int2str(i) '.png'];
804         Workspace(:,:)=imread(filename);
805         Z_workspace(Workspace') = i;
806     end
807     [X_workspace,Y_workspace] = meshgrid((1:n),(1:n));
808     waterfall(X_workspace,Y_workspace,Z_workspace);
809 end

```

```

810
811 %Over plot the contact points:
812 plot3(x_contact_surf, y_contact_surf, z_contact_surf, ...
813       'r.', 'markersize', 6);
814 title('Mapping x_{surface} and y_{surface} back to the surface')
815 view(-15,65); axis equal; box on; colormap cool;
816 xlabel('x-axis'); ylabel('y-axis'); zlabel('z-axis')
817 hold off;
818
819 fprintf('The robot bounced a total of %i times \n', bounces);
820 delete(handle);
821 clear variables;
822 toc

```

## 2 Get\_bigZ\_bigC.m

```

1 function [bigZ, bigC] = Get_bigZ_bigC(z_x, z_y, ...
2     x0_robot, y0_robot, p_hat, M_robot, K_robot, T_robot, ...
3     R_psi, R_cocf_psi, V)
4 %Returns the big Z matrix as well as the big C (right hand side) vector
5
6 % - x0_robot and y0_robot are the current values for the x and y parameters
7 % of the robot
8 % - p_hat is used especially when computing Ad_g matrices and it is easier
9 % to have it as a parameter passed to the function rather than compute it
10 % here
11 % M_robot is the metric tensor of the robot
12 % K_robot is the curvature tensor of the robot
13 % T_robot is the torsion form of the robot
14 % R_psi is another matrix necessary for the computations
15 % R_cocf_psi is there for the same reason as R_psi
16 % V is the velocity vector
17
18 %% The surface:
19 % The surface is approximated to a plane:
20 % @(x,y) [x+cst; y+cst; z_x*x + z_y*y + cst]
21
22 %%
23 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
24 %% Initializing a few matrices and vectors before getting          %%
25 %% to rewriting the system of equations                            %%
26 I_surf = [1+z_x^2, z_x*z_y; z_x*z_y, 1+z_y^2]; %The first fundamental form
27 M_surf = sqrtm(I_surf); %The metric tensor
28 R_fcf = [-sin(x0_robot)*cos(y0_robot), -sin(y0_robot), -cos(x0_robot)*cos(y0_robot)
29         -sin(x0_robot)*sin(y0_robot), cos(y0_robot), -cos(x0_robot)*sin(y0_robot)
30         cos(x0_robot) 0 -sin(x0_robot)];
31 Adg_fcf = [R_fcf, p_hat*R_fcf; zeros(3), R_fcf];
32
33
34 %%
35 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
36 %% Part three: Rewriting the system of equations                %%

```



```

37 A = (M_robot^(-1))*(K_robot)^(-1);
38 B = (M_surf^(-1))*R_psi*(K_robot)^(-1);
39 C_f = T_robot*M_robot;
40 RR = [R_cocf_psi, zeros(3); zeros(3), R_cocf_psi];
41 J = [M_surf; zeros(4,2)];
42 L = eye(6)-((Adg_fcf)^(-2));
43 Q1 = [0 0 0 0 -1 0
44       0 0 0 1 0 0];
45 Q2 = [1 0 0 0 0 0
46       0 1 0 0 0 0];
47 Q3 = [0 0 0 0 0 1];
48 Q4 = [0; 0; 0; 0; 0; 1];
49 S1 = Q1*L*RR*J;
50 S2 = Q2*L*RR*J;
51 S3 = Q3*L*RR*J;
52 T1 = Q1*L*Q4;
53 T2 = Q2*L*Q4;
54 T3 = Q3*L*Q4;
55 V1 = Q1*((Adg_fcf)^(-1))*V;
56 V2 = Q2*((Adg_fcf)^(-1))*V;
57 V3 = Q3*((Adg_fcf)^(-1))*V;
58
59
60 %%
61 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
62 %% Part four: Getting the big Z matrix and the big C vector          %%
63 bigZ = [eye(2), -A*S1, -A*T1
64         zeros(2), eye(2)-B*(S1+K_robot*S2), -B*(T1+K_robot*T2)
65         -C_f, -S3, 1-T3];
66 bigC = [A*V1; B*(V1+K_robot*V2); V3];
67
68
69 end

```

### 3 Get\_hit\_contact\_pts.m

```

1 function [contact_pt, hit_pt, error] = Get_hit_cont_pts(ROBOT, WORKSPACE, ...
2     prev_contact_pt, r, guess)
3 done = false;
4 trial = 0; %Trials to shift the robot around
5 while(~done)
6     done = true; %Shouldn't need to loop around
7     %Returns the current contact point (with the surface) along with the
8     %current hit point (if such hit point exists) as well as with a boolean
9     %type error variable
10
11     error = 0; %If all goes well, there is no error.
12     num_contacts = 1; %The 'theoretical' number of contact points (1 or 2)
13
14     %Because of the irregularities in the bitmaps (one cannot make a
15     %perfect sphere out of cubes), there might be a few contact points
16     %where in theory (with a perfect sphere) there would only be one; hence

```

```

17 %the distinction between num_contacts (the 'theoretical' number of
18 %contact points) and the true number of contact points found.
19 [contacts_x,contacts_y,contacts_z] = ind2sub(size(ROBOT),find(ROBOT ...
20 & WORKSPACE));
21 if(length(contacts_x)~=length(contacts_y) || ...
22    length(contacts_y)~=length(contacts_z) ...
23    || length(contacts_z)~=length(contacts_x))
24     fprintf('WARNING: vectors length differ in contact matrix\n');
25 end
26 %Vector length, and tolerance level of spacing between points that
27 %should in theory be touching each other
28 len = length(contacts_x); delta = r/10;
29 %Checking that there is at least one contact point (the robot is not
30 %flying around).
31 %If we get unlucky and the sphere does not land right on the surface,
32 %we can try and move it very slightly.
33 if(len < 1)
34     if(trial > 0)
35         fprintf('ERROR: contact has been lost\n');
36         contact_pt = zeros(3,1); hit_pt = zeros(3,1); error = 1;
37         return
38     end
39     delta_move = round(r/10);
40     finished = false;
41     for i=0:delta_move
42         for j=0:delta_move
43             for k=0:delta_move
44                 tmp_ROBOT = circshift(ROBOT,[i, j, k]);
45                 tmp_index = find(tmp_ROBOT & WORKSPACE,1);
46                 if(~isempty(tmp_index))
47                     finished = true;
48                     break;
49                 end
50                 tmp_ROBOT = circshift(ROBOT,[-i, j, k]);
51                 tmp_index = find(tmp_ROBOT & WORKSPACE,1);
52                 if(~isempty(tmp_index))
53                     finished = true;
54                     i = -i;
55                     break;
56                 end
57                 tmp_ROBOT = circshift(ROBOT,[i, -j, k]);
58                 tmp_index = find(tmp_ROBOT & WORKSPACE,1);
59                 if(~isempty(tmp_index))
60                     finished = true;
61                     j = -j;
62                     break;
63                 end
64                 tmp_ROBOT = circshift(ROBOT,[i, j, -k]);
65                 tmp_index = find(tmp_ROBOT & WORKSPACE,1);
66                 if(~isempty(tmp_index))
67                     finished = true;
68                     k = -k;
69                     break;
70                 end
71                 tmp_ROBOT = circshift(ROBOT,[i, -j, -k]);

```

```

72         tmp_index = find(tmp_ROBOT & WORKSPACE,1);
73         if(~isempty(tmp_index))
74             finished = true;
75             j = -j; k = -k;
76             break;
77         end
78         tmp_ROBOT = circshift(ROBOT,[-i, j, -k]);
79         tmp_index = find(tmp_ROBOT & WORKSPACE,1);
80         if(~isempty(tmp_index))
81             finished = true;
82             i = -i; k = -k;
83             break;
84         end
85         tmp_ROBOT = circshift(ROBOT,[-i, -j, k]);
86         tmp_index = find(tmp_ROBOT & WORKSPACE,1);
87         if(~isempty(tmp_index))
88             finished = true;
89             i = -i; j = -j;
90             break;
91         end
92         tmp_ROBOT = circshift(ROBOT,[-i, -j, -k]);
93         tmp_index = find(tmp_ROBOT & WORKSPACE,1);
94         if(~isempty(tmp_index))
95             finished = true;
96             i = -i; j = -j; k = -k;
97             break;
98         end
99     end
100     if(finished); break; end;
101 end
102 if(finished); break; end;
103 end
104 if(~finished);
105     fprintf('ERROR: No contact has been found within range\n');
106     contact_pt = zeros(3,1); hit_pt = zeros(3,1); error = 1;
107     return
108 end
109 shift_x = i;
110 shift_y = j;
111 shift_z = k;
112 ROBOT = circshift(ROBOT,[shift_x, shift_y, shift_z]);
113 done = false;
114 trial = trial + 1;
115 else
116     jump_index = 0; %To track where the discontinuity is located
117     for i=1:len-1
118         if(abs(contacts_x(i+1)-contacts_x(i)) > delta ...
119             && abs(contacts_y(i+1)-contacts_y(i)) > delta ...
120             && abs(contacts_z(i+1)-contacts_z(i)) > delta)
121             jump_index = i;
122             num_contacts = num_contacts + 1; %2 contacts/hit
123         end
124     end
125     %Checking that there are no more than 2 theoretical contact points:
126     if(num_contacts > 2)

```

```

127         fprintf('ERROR: there are more than 2 theoretical contact points\n');
128         fprintf('\t\t num_contacts = %i\n', num_contacts);
129         contact_pt = zeros(3,1); hit_pt = zeros(3,1); error = 1;
130         return
131     end
132     %There are two cases: either there is 1 theoretical contact point
133     %or there are two. Those must be taken care of separately.
134     if(num_contacts == 1) %Only 1 contact point. Note: jump_index == 0
135         if(~strcmp(guess, 'NaN') ...
136             && WORKSPACE(guess(1), guess(2), guess(3)) ...
137             && ROBOT(guess(1), guess(2), guess(3)))
138             contact_pt = guess;
139         else
140             %Where the theoretical contact point is located in the
141             %contact vectors:
142             contact_index = ceil(len/2);
143             contact_pt = [contacts_x(contact_index); ...
144                 contacts_y(contact_index); contacts_z(contact_index)];
145         end
146         hit_pt = 'NaN';
147     else %2 contact points (more complicated because I need to
148         %differentiate between the contact vectors and the hit vectors
149
150         %Making sure a second contact point is allowed:
151         if(strcmp(prev_contact_pt, 'NaN'))
152             fprintf('ERROR: there cannot be a second contact point at this ...
153                 point \n');
154             contact_pt = zeros(3,1); hit_pt = zeros(3,1); error = 1;
155             return
156         end
157
158         %The first part of the contacts vectors
159         contacts_x1 = contacts_x(1:jump_index);
160         contacts_y1 = contacts_y(1:jump_index);
161         contacts_z1 = contacts_z(1:jump_index);
162         len1 = length(contacts_x1);
163         contact_index1 = ceil(len1/2);
164         contact_pt1 = [contacts_x1(contact_index1); ...
165             contacts_y1(contact_index1); contacts_z1(contact_index1)];
166
167         %The second part of the contacts vectors
168         contacts_x2 = contacts_x(jump_index+1:end);
169         contacts_y2 = contacts_y(jump_index+1:end);
170         contacts_z2 = contacts_z(jump_index+1:end);
171         len2 = length(contacts_x2);
172         contact_index2 = ceil(len2/2);
173         contact_pt2 = [contacts_x2(contact_index2); ...
174             contacts_y2(contact_index2); contacts_z2(contact_index2)];
175
176         %Find which contact_pt is the real one and which is the hit_pt
177         d1 = sum((prev_contact_pt-contact_pt1).^2);
178         d2 = sum((prev_contact_pt-contact_pt2).^2);
179         if(d1<d2) %contact_pt1 is closer to the previous contact point)
180             contact_pt = contact_pt1;
181             hit_pt = contact_pt2;
182         else
183             contact_pt = contact_pt2;

```

```

181             hit_pt = contact_pt1;
182         end
183     end
184 end
185 end
186 end

```

## 4 Get\_slopes.m

```

1 function [z_x, z_y]=Get_slopes(WORKSPACE, contact_pt, r, z_x_old, z_y_old)
2 %Returns the partial derivatives (slopes) of the tangent plane at the
3 %contact_pt
4 x = contact_pt(1); y = contact_pt(2); z = contact_pt(3);
5 i = 0; j = 0; k1 = 1; k2 = 1; done1 = false; done2 = false;
6 while(~done1)
7     if(WORKSPACE(x+i,y,z+k1))
8         done1 = true;
9     elseif(WORKSPACE(x-i,y,z+k1))
10        i = -i; done1 = true;
11    elseif(WORKSPACE(x+i,y,z-k1))
12        k1 = -k1; done1 = true;
13    elseif(WORKSPACE(x-i,y,z-k1))
14        i = -i; k1 = -k1; done1 = true;
15    else
16        k1 = k1+1;
17        if(k1 > r/5); k1 = 0; i = i+1; end
18    end
19    if(i > r/5)
20        fprintf('WARNING: Distance between points is large\n');
21    end
22 end
23 z_x = k1/i;
24
25 while(~done2)
26     if(WORKSPACE(x,y+j,z+k2))
27         done2 = true;
28     elseif(WORKSPACE(x,y-j,z+k2))
29         j = -j; done2 = true;
30     elseif(WORKSPACE(x,y+j,z-k2))
31         k2 = -k2; done2 = true;
32     elseif(WORKSPACE(x,y-j,z-k2))
33         j = -j; k2 = -k2; done2 = true;
34     else
35         k2 = k2+1;
36         if(k2 > r/5); k2 = 0; j = j+1; end
37     end
38     if(k2 > r/5)
39         fprintf('WARNING: Distance between points is large\n');
40     end
41 end
42 z_y = k2/j;
43 if(i == 0)

```

```

44     fprintf('WARNING: z_x = Inf, z_x was not updated\n');
45     z_x = z_x_old;
46 end
47 if(j == 0)
48     fprintf('WARNING: z_y = Inf, z_y was not updated\n');
49     z_y = z_y_old;
50 end
51
52 end

```

## 5 Bitmaps.m

```

1 clear ('variables'); close all;
2
3 %The "size" of the workspace
4 %When dealing with a flat plane, this corresponds to the length of a square
5 %in which the sphere is located at the center
6 size_workspace = 10;
7 %The number of pixels used on one side of the square box defining the full
8 %workspace of the robot
9 n = 500; %1000; %1100; %1250;
10 %For the main program, n/2 needs to be an integer
11 %Thus, n should be even:
12 if(mod(n,2)); fprintf('ERROR: n is not even'); return; end
13 %For better readability of the main program the value n is saved to a file
14 f = fopen('n.txt','w'); fprintf(f,'%i',n); fclose(f);
15
16 %The robot (sphere):
17 tic
18 % delete('Sphere_robot/*');
19 rho = n/size_workspace; %Radius of the sphere in pixels
20 % %Again, for better readability of the main program the value of rho is
21 % %saved to a file:
22 % f = fopen('rho.txt','w'); fprintf(f,'%i',rho); fclose(f);
23 % centerX = n/2;
24 % centerY = centerX;
25 % for i=1:(2*rho)
26 %     z = -rho + i - 1;
27 %     %fprintf('true radius = %f \t', sqrt(rho*rho - z*z));
28 %     radius = round(sqrt(rho*rho - z*z));
29 %     %fprintf('radius = %i \n', radius);
30 %     d = (5 - radius * 4)/4;
31 %     x = 0;
32 %     y = radius;
33 %     Z = false(n);
34 %     while(x<=y)
35 %         Z(centerX + x, centerY + y) = 1;
36 %         Z(centerX + x, centerY - y) = 1;
37 %         Z(centerX - x, centerY + y) = 1;
38 %         Z(centerX - x, centerY - y) = 1;
39 %         Z(centerX + y, centerY + x) = 1;
40 %         Z(centerX + y, centerY - x) = 1;

```

```

41 %      Z(centerX - y, centerY + x) = 1;
42 %      Z(centerX - y, centerY - x) = 1;
43 %      if(d < 0)
44 %          d = d + 2 * x + 1;
45 %      else
46 %          d = d + 2 * (x - y) + 1;
47 %          y = y-1;
48 %      end
49 %      x = x + 1;
50 %      end
51 %      imwrite(Z,['Sphere-robot/' int2str(i) '.png'],'png');
52 % end
53 % Z = false(n);
54 % for i=(2*rho+1):n
55 %     imwrite(Z,['Sphere-robot/' int2str(i) '.png'],'png');
56 % end
57 % clear Z;
58 % toc
59 tic
60 %The surfaces and walls:
61 %Store the surfaces and walls in parametric equations form
62 f_functions = {@(s,t)s %Flat surf
63               @(s,t)s %Tilted surf x
64               @(s,t)s %Tilted surf y
65               @(s,t)s %Tilted surf x,y
66               @(s,t)s %Sine wave x, surf
67               @(s,t)s %Sine wave y, surf
68               @(s,t)s %Sine wave x,y, surf
69               @(s,t)s %Gaussian dome, surf
70               @(s,t)s %Sine wall 1
71               @(s,t)s %Sine wall 2
72               @(s,t)s %Sine tilted wall 1
73               @(s,t)s %Sine tilted wall 2
74               @(s,t)s %Tilted wall 1
75               @(s,t)s}; %Tilted wall 2
76 g_functions = {@(s,t)t %Flat surf
77               @(s,t)t %Tilted surf x
78               @(s,t)t %Tilted surf y
79               @(s,t)t %Tilted surf x,y
80               @(s,t)t %Sine wave x, surf
81               @(s,t)t %Sine wave y, surf
82               @(s,t)t %Sine wave x,y, surf
83               @(s,t)t %Gaussian dome, surf
84               @(s,t)rho*sin(s/rho/4)+7.1*rho %Sine wall 1
85               @(s,t)rho*sin(s/rho/4)+2.9*rho %Sine wall 2
86               @(s,t)s+2.5*rho+rho*sin(s/rho/2) %Sine tilted wall 1
87               @(s,t)s-2.5*rho-rho*sin(s/rho/2) %Sine tilted wall 2
88               @(s,t)s+3*rho %Tilted wall 1
89               @(s,t)s-3*rho}; %Tilted wall 2
90 h_functions = {@(s,t)n/2+1+0*s+0*t%Flat surf
91               @(s,t)s %Tilted surf x
92               @(s,t)t %Tilted surf y
93               @(s,t)s/2+t/2 %Tilted surf x,y
94               @(s,t)rho*sin(s/rho/4)+rho+1 %Sine wave x, surf
95               @(s,t)rho*sin(t/rho/4)+rho+1 %Sine wave y, surf

```

```

96         @(s,t)rho*sin((s+t)/rho/8)+rho+1 %Sine wave x,y, surf
97         @(s,t)rho*sqrt(rho)*(exp(-(s-n/2).^2+(t-n/2).^2)/rho^3.8))-ceil((rho*(sqrt(rho)-7
          %Gaussian dome, surf
98         @(s,t)t %Sine wall 1
99         @(s,t)t %Sine wall 2
100        @(s,t)t %Sine tilted wall 1
101        @(s,t)t %Sine tilted wall 2
102        @(s,t)t/3.5 %Tilted wall 1
103        @(s,t)t/3.5}; %Tilted wall 2
104 %The names of all the different surfaces and walls
105 names = {'Flat_Surf'; 'Tilted_x_Surf'; 'Tilted_y_Surf'; 'Tilted_xy_Surf'; ...
106         'Sine_x_Surf'; 'Sine_y_Surf'; 'Sine_xy_Surf'; 'Gaussian_dome_Surf'; ...
107         'Sine_Top_Wall'; 'Sine_Bot_Wall'; 'Sine_Tilted_Top_Wall'; ...
108         'Sine_Tilted_Bot_Wall'; 'Tilted_Top_Wall'; 'Tilted_Bot_Wall'};
109 num_surf = length(names);
110 %for k = 1:num_surf
111 for k=13:14
112     tic
113     name = names{k};
114     mkdir(name);
115     delete([name '/' *]);
116     %Grab the correct functions
117     f = f.functions{k}; g = g.functions{k}; h = h.functions{k};
118     I = false(n,n,n); % Stores the surface in 3D matrix form
119     s = (1:n); t = (1:n); % Parametric vectors
120     [S,T] = meshgrid(s,t); % Corresponding parametric matrices
121     X = f(S,T); Y = g(S,T); Z = h(S,T); % Surface in 3 matrices form
122     X(X<1) = 1; Y(Y<1) = 1; Z(Z<1) = 1; % Values must be > 1
123     X(X>n) = n; Y(Y>n) = n; Z(Z>n) = n; % Values must be < n+1
124     %Reshape the matrices into vector form
125     x = round(reshape(X,[1,numel(X)]));
126     y = round(reshape(Y,[1,numel(Y)]));
127     z = round(reshape(Z,[1,numel(Z)]));
128     v = unique([x',y',z'],'rows'); %Remove duplicates
129     x = v(:,1); y = v(:,2); z = v(:,3);
130     linIndex = sub2ind(size(I), x, y, z); %Transforming into linear index
131     I(linIndex) = true; % Store the surface into the 3D matrix
132     for i = 1:n
133         imwrite(I(:,:,i),[name '/' int2str(i) '.png'], 'png');
134     end
135     %Save the picture
136     handle = figure;set(handle, 'Visible', 'off');
137     mesh(round(X),round(Y),round(Z));
138     view(150,130); axis equal; box on; colormap cool;
139     xlabel('x-axis'); ylabel('y-axis'); zlabel('z-axis');
140     saveas(handle,[int2str(k) '.fig']);
141     %In order to display the figure, one needs to do the following:
142     %openfig('figure.fig', 'Visible');
143     toc
144 end
145 toc
146 clear ('variables'); close all;

```