Robot Driver

Matlab Based Code

Robot Driver Code Sample (Matlab)

```
function [robot,pose,vel] = drive_robot(F,T,pose,vel)
%UNTITLED2 Summary of this function goes here
   dt=.1;  % time step
   %robot parameters
   size_x=1.50;   %(m)
   size_y=2;   %(m)
   mass=25;    %(kg)
   %mass moment of inertia of a disk
   inertia=mass*((size_x)^2+(size_y)^2)/12;
   %newtons second law
   accel(1)=F/mass; %translation
   accel(2)=T/inertia; %rotation
   %saturation limits (lin. acceleration)
   accelLim=[10,.5]; %(m/s^2) - 1 g
   if accel(1)>accelLim(1);
      accel(1)=accelLim(1);
   elseif accel(1)<-accelLim(1);
      accel(1)=-accelLim(1);
   end
   %saturation limits (ang. acceleration)
   velLim=[11 1];  %(m/s) - approx 25 mph
   if accel(2)>accelLim(2);
      accel(2)=accelLim(2);
   elseif accel(2)<-accelLim(2);
      accel(2)=-accelLim(2);
   end
   %integrate lin. acceleration to lin. velocity (EULERS's integration)
   vel(1)=vel(1)+accel(1)*dt;
   %integrate ang. acceleration to ang. velocity
   vel(2)=vel(2)+accel(2)*dt;

   %saturation limits (lin. velocity)
   if vel(1)>velLim(1);
```

```
    vel(1)=velLim(1);
elseif vel(1)<-velLim(1);
    vel(1)=-velLim(1);
end
%saturation limits (ang. velocity)
if vel(2)>velLim(2);
    vel(2)=velLim(2);
elseif vel(2)<-velLim(2);
    vel(2)=-velLim(2);
end
%integrate rotational vel into theta
pose(3)=pose(3)+vel(2)*dt;
%calc x,y components at robot theta
vX=vel(1)*cos(pose(3));
vY=vel(1)*sin(pose(3));
%integrate velocity to position
pose(1)=pose(1)+vX*dt;
pose(2)=pose(2)+vY*dt;
%build the patch
robot.vertices=[-size_x/2,0
                size_x/2,0
                0,size_y];
robot.faces=[1,2,3];
%rotate the patch
R=[cos(pose(3)-pi/2) -sin(pose(3)-pi/2)
  sin(pose(3)-pi/2) cos(pose(3)-pi/2)];
robot.vertices=R*robot.vertices';
robot.vertices=robot.vertices';
%translate the patch
for g=1:length(robot.faces)
    robot.vertices(g,1)=robot.vertices(g,1)+pose(1);
    robot.vertices(g,2)=robot.vertices(g,2)+pose(2);
```

Map Builder

```
%mapbuilder for roboSim V1.3
%save maps as text file using ginput;
%currently maps are square (v1_3)
function [xs,ys,myHeight,myWidth] = map_builder()

   myFile=input('what would you like to name you map file? ','s');
   myID=fopen(myFile,'w');

   myWidth=input('Choose the Arena Width. ');
   myHeight=input('Choose the Arena Height. ');

   fprintf(myID,'%i %i 0 0 0 0 0 0 \n',myHeight,myWidth)

   arena=[0,myWidth,myWidth,0,0
     0,0  ,myHeight,myHeight,0];

   figure(1);
   plot(arena(1,:),arena(2,:),'b-'); hold on
   axis([-5 myWidth+5 -5 myHeight+5]);
   axis equal
   disp('Place your objects on the figure. Each object must have 4 vertices.')
   adding=1;
   ctr=0;
   while adding
      ctr=ctr+1;

      [xs(ctr,:),ys(ctr,:)]=ginput(4);
      fprintf(myID,'%f %f %f %f ',xs(ctr,1),xs(ctr,2),xs(ctr,3),xs(ctr,4));
      fprintf(myID,'%f %f %f %f ',ys(ctr,1),ys(ctr,2),ys(ctr,3),ys(ctr,4));
      fill(xs(ctr,:),ys(ctr,:),'g')
      axis([-5 myWidth+5 -5 myHeight+5]);
      axis equal
      % dont put a newline at the end of the last line
      adding=input('Would you like to add another object? (1/0) ');
      if adding
         fprintf(myID,'\n');
      end

   end

   fclose(myID)
```

Sensor Reader

```
function [sen_dist,sen_angle] = read_sensor(obsx,obsy,pose)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here


   sen_angle=[90,60,30,0,-30,-60,-90]*(pi/180)+pose(3); %senor directions
   max_range=20;          % max range for sensor
   sen_offset=0;        % offset for sensor data
   idx1=[1,2,3,4];
   idx2=[2,3,4,1];
   for l=1:length(sen_angle)   %loop over all sensors
      sen_dirx=cos(sen_angle(l)); sen_diry=sin(sen_angle(l)); % dir. of current sensor
      t_current=max_range;    %initial sensor distance (t) to max range
      for j=1:length(obsx(:,1))  % loop over all obstacles
         for k=1:4            % loop over all sides
            p1x=obsx(j,idx1(k)); p1y=obsy(j,idx1(k));
            p2x=obsx(j,idx2(k)); p2y=obsy(j,idx2(k));
            v1x=p2x-p1x;v1y=p2y-p1y;
            sx=pose(1); sy=pose(2);
            A=[sen_dirx, -v1x; sen_diry, -v1y];
            b=[p1x-sx; p1y-sy];
            if abs(det(A))>=1e-8    %check for intersecting lines
               ts=inv(A)*b;
            else
               ts=[100,100];
            end
            if (ts(2)>=0 && ts(2)<=1 && ts(1)>=0 && ts(1)<t_current)
               t_current=ts(1); %update current sensor distance
            end
         end % end side k
      end % end obstacle j
      sen_dist(l)=t_current-sen_offset;   % assign sensor distance l (subtract offset)
   end
%end sensor angle l
```