BRADLEY
UNIVERSITY

ECE 467: Robotics Design Laboratory

Lab 3: Mobile Robot Regulation

Anthony Le, Mike McGrath, and Jason Morris

Contact Information

Anthony Le
Electrical and Computer Engineering Department
Caterpillar College of Engineering and Technology
Bradley University
Jobst Hall #254
1501 W. Bradley Avenue
Peoria, IL, 61625, USA
Phone: +1 (309) 369-8462
e-Mail: ale@mail.bradley.edu

Table of Contents

1 Objective(s)

In this lab, our goal was to familiarize ourselves with regulation control problems and simulate both Cartesian and posture control law. The robot will attempt to move to the origin and in part 2, the robot also must have an orientation of 0°.

2 Background

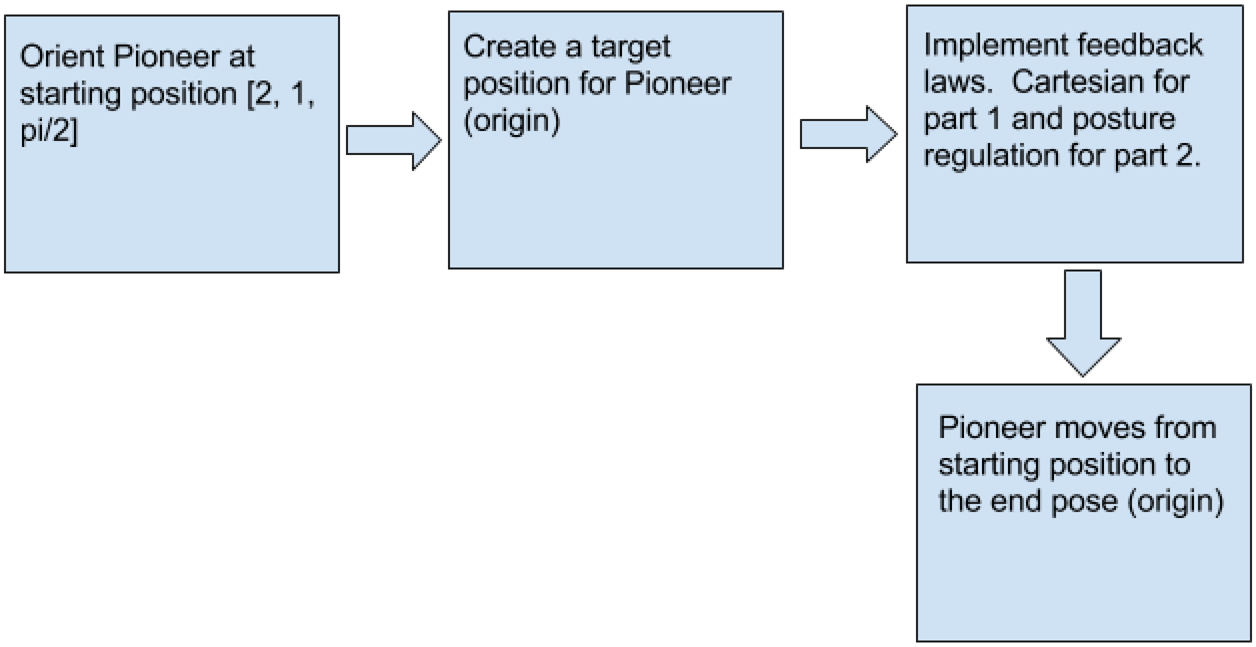
2.1 Software

- For this lab, we used visual studio 2015
- We had to make sure that we had the C++ add-ons installed
- Adept MobileRobots Advanced Robotics Interface for Applications (ARIA)
- MobileSim
- Matlab 2015

2.2 Hardware

- Pioneer P3-DX robot
- Laptop
- USB to Mini-DIN communication cable

2.3 Block Diagram


Orient Pioneer at starting position [2, 1, pi/2]
Create a target position for Pioneer (origin)
Implement feedback laws. Cartesian for part 1 and posture regulation for part 2.
Pioneer moves from starting position to the end pose (origin)

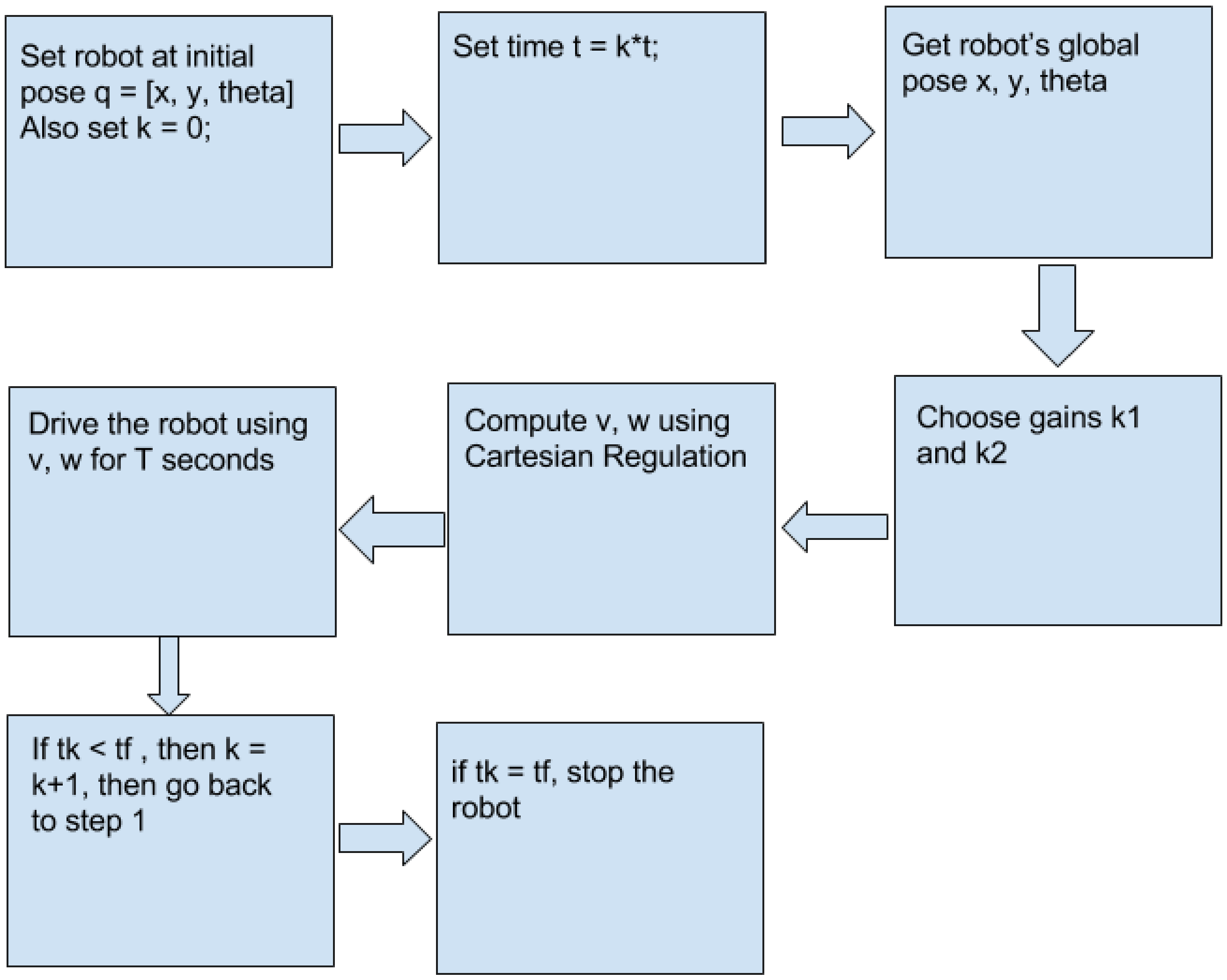
3 Implementation

3.1 Steps

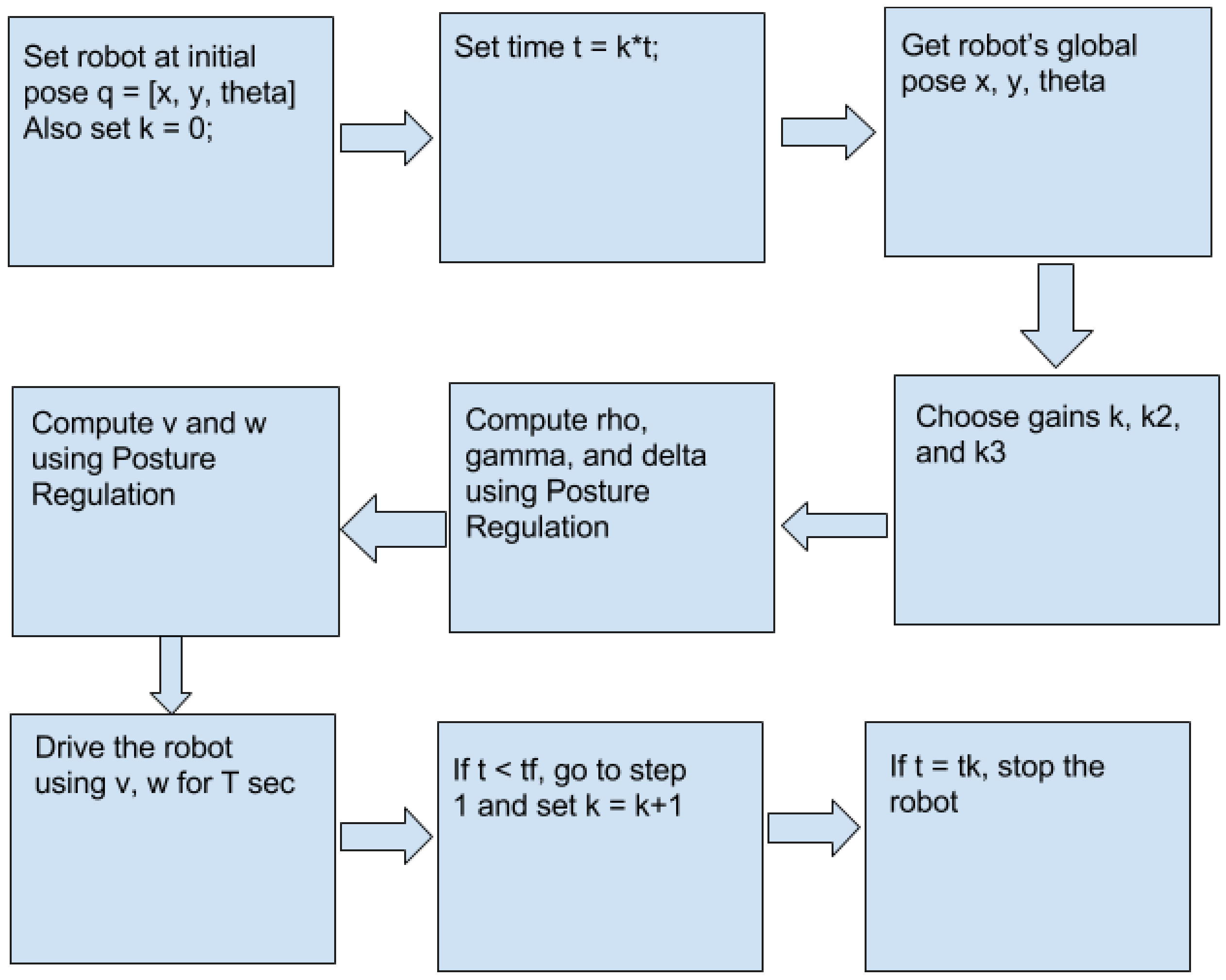
1. Copied the visual studio project file from the previous lab. This was so we wouldn't have to add all of the dependencies and Linker files.
2. Wrote the code that implemented the Cartesian Regulation law for part 1.
3. Simulated Cartesian Regulation (Part 1) on MobileSim
4. Wrote the code that implemented the Posture Regulation law for part 2.
5. Simulated Posture Regulation (Part 2) on MobileSim.
6. Demonstrated the final code using Pioneer to the TA.

3.2 Code

Part 1 Flow Chart


Set robot at initial pose q = [x, y, theta] Also set k = 0;
Set time t = k*t;
Get robot's global pose x, y, theta
Choose gains k1 and k2
Compute v, w using Cartesian Regulation
Drive the robot using v, w for T seconds
If tk < tf , then k = k+1, then go back to step 1
if tk = tf, stop the robot

Part 2 Flow Chart


Set robot at initial pose q = [x, y, theta] Also set k = 0;
Set time t = k*t;
Get robot's global pose x, y, theta
Choose gains k, k2, and k3
Compute rho, gamma, and delta using Posture Regulation
Compute v and w using Posture Regulation
Drive the robot using v, w for T sec
If t < tf, go to step 1 and set k = k+1
If t = tk, stop the robot

C++ CODE

```
/*
TestProgram.cpp by Jason Morris and Mike McGrath
This simple program shows what is needed for a program to control the
Aria based robot.
*/
//C++ class to allow communication of the program to a user
#include "../00_nav_library/nav_aria_iface.h"
#include "../00_nav_library/nav_random.h"
#include "../00_nav_library/nav_map.h"
#include "../00_nav_library/nav_robot2map.h"
```

```
#include "../00_nav_library/nav_field_potential.h"
//#include "nav_robot2lines.h"
#include <Windows.h>
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <iomanip>
#include <conio.h>
#include <sstream>
#include <complex>

#include <math.h>       /* atan2 */
#include <stdlib.h>     /* abs */
using namespace std;
//This class is what allows a robot
//object to be created to communicate to an Aria based robot.




void Follow_Circle(NAV_Robot & R) {

      char file[] = "posePioneer.csv";
      std::ofstream data;
      data.open(file);

      char input;
      int k = 0;
      const double tf = 60;
      const double T = 50;
      double time = 0;
      const double Rad = 1;
      double gamma, delta;
      double rho, xdt, ydt, thetadt, xd_dott, yd_dott, vdt, x = 2, y
= 1, theta = M_PI / 2, tmp, k1, k2, k3, v, w, e1, e2, e3,
xd_dot_dott, yd_dot_dot;

      //do{
      while (time<tf) {

            xdt = 0;
            ydt = 0;
            x = R.getX() / 1000;
            y = R.getY() / 1000;
            theta = R.getTh()*(M_PI / 180);
            if (theta > 2 * M_PI)
            {
                  theta = theta - (2 * M_PI);
            }

            if (theta < 0)
            {
                  theta = theta + (2 * M_PI);
            }
            cout << "time: " << time << endl;
            rho = sqrt(y*y+x*x);
            gamma = atan2(y,x) - theta + M_PI;
            delta = atan2(y,x) + theta;



            //print error
            k1 = .4;
            k2 = .4;
            k3 = .4;






            v = k1*rho*cos(gamma);
            w =
k2*gamma+k1*(sin(gamma)/gamma)*(cos(gamma))*(gamma+k3*delta);

            //R.unlock();
            //cout << w << " " << v << endl;
            R.lock();
            //R.doRotateAngle(w);
            //R.doDriveSpeed(v);
            R.doDriveSpeed(v * 1000);
            R.doRotateSpeed(w*(180 / M_PI));
            R.unlock();
            R.Sleep(T);
            data << time << " " << x << " " << y << " " << theta << "
" << v << " " << w << endl;


            //R.lock();
            k += 1;
            time = k*T / 1000;
            //} while (time<tf);
      }

      //R.doDriveSpeed(0);
      //R.unlock();
      return;
}



int main() {

      /* We create an object R of class NAV_Robot that will be used
to control the robot
      the robot is in a local area network and will be connected
through the IP
      192.168.4.4 with port 8101 */
      //NAV_Robot R("localhost", 8101);
      NAV_Robot R("COM9");

      /////////////   NAV_Robot R("localhost", 8101); //This Creates
the object R that will be used to

      //control the robot in this case it will be connected
      //to the local port 8101 which MobileSim defaults to
      //listen on.
      /* You may want to connect the robot through COMx, where x is
the COM port number
      when the robot is connected using a USB to serial cable*/
      ////////////NAV_Robot R("COM1",8101);
      //This Creates the object R that will be used to
      //control the robot in this case it will be connected
      //to the COM1 port of your computer.
      if (!R.is_connected()) {
           // This checks if the robot is connected and returns a
 bool value
           std::cout << "Error in connecting" << std::endl;
           return(0);
      }


      R.moveCoords(ArPose(2000, 1000, 90));//This sets the robot to
 be at 0
      Follow_Circle(R);

      return(0);

 }
```

MATLAB CODE

```
%prelab part3 parta%
% mainGenerateDesiredTrajectory.m   Script for simulating mobile
robot's
% reference trajectory

function mainApproxmiate

% Created by S. Miah on Feb. 01, 2016.
close all
clear all
clc

disp('Please wait while the simulation runs ...');

% ---------------------
% SIMULATION PARAMETERS
% ---------------------

% Simulation time
t0 = 0; tf = 60; % initial and final simulation time [s]
T = .5;  % Sampling time [s]
tsteps = floor((tf-t0)/T); % number of time steps
dt = T*(0:tsteps)'; % Discrete time vector (include time t0 to plot
initial state too)
```

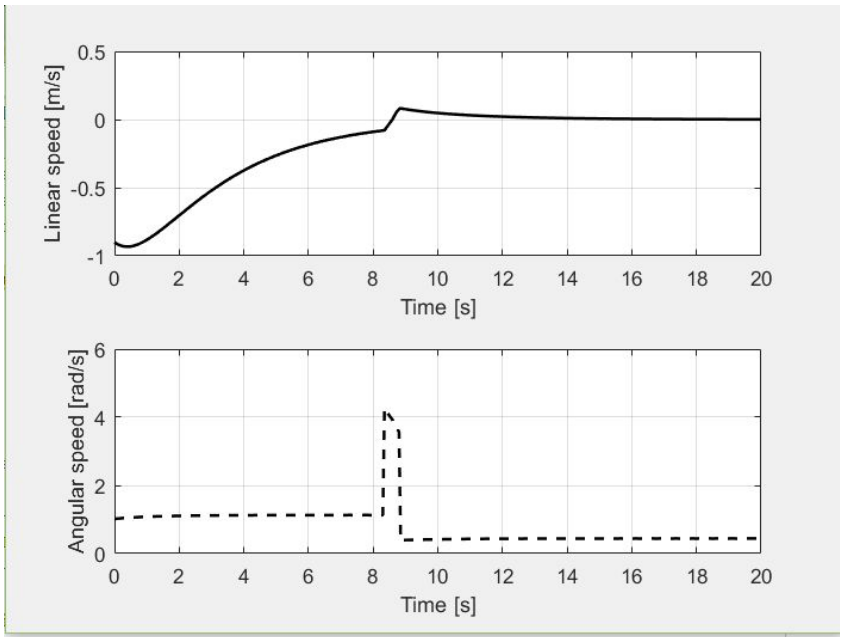
```
qInit=[2,1,pi/2];

qd=[0,0,0];

k1=.115;
k2=k1;
k3=k1;
% x_yd=[0;0]
% t=interp1(t,dt);

% Run the simulation POSTURE
%pt=sqrt(q(1).^2+q(2).^2);
%lambt= atan2(q(2),q(1))-q(3)+pi;
%delt=lambt+q(3)
%pt = interp1(dt,pt);

%v = k1*pt.*cos(lambt);
%vdt = k1*pt*cos(lambt);
%w = k2*lambt+k1*(sin(lambt)/lambt)*cos(lambt)*(lambt+k3*delt)

%POSTURE



% v=-k1*(q(1)*cos(q(3))+q(2)*sin(q(3)))
% w=k2*(atan2(q(2),q(1))-q(3)+pi);

[tout,qTemp] = ode45(@(t,q) stateEqError(dt,q,qd, k1,k2,k3),[t0
tf],qInit);
%[t,qTemp] = ode45(@(t,q) stateEqError( q,v, w), [t0 tf], qInit );
% solve the robot's actual pose
qTemp = interp1(tout,qTemp,dt);

row = sqrt(((qTemp(1)).^2) + ((qTemp(2)).^2));
gamma = atan2(qTemp(2), qTemp(1))-qTemp(3)+pi;
delta = row + qTemp(3);

 v = k1.*row.*cos(gamma);
 w=k2.*row+k1.*((sin(row))/row)*row.*(row+k3.*delta);

actualPose = qTemp;

e= [qTemp(:,1), qTemp(:,2)];
generatePlots(length(dt), dt, actualPose, actualPose, e, [v w]);
```

```
disp('... done.');


function qdot = stateEqError(t,q,qd, k1,k2,k3)
row = sqrt(((q(1)).^2) + ((q(2)).^2));
gamma = atan2(q(2), q(1))-q(3)+pi;
delta = row + q(3);

vt=k1*row*cos(gamma);
wt = k2*row+k1*((sin(row))/row)*cos(row)*(row+k3*delta);

row_dot = cos(gamma);
gamma_dot = (vt/row)*sin(gamma)-wt;
delta_dot = gamma_dot+wt;


x_dot=vt*cos(q(3));
y_dot=vt*sin(q(3));
theta_dot = wt;
qdot = [x_dot;y_dot;theta_dot];
% qdot = [row_dot;gamma_dot;delta_dot];



% Reference robot kinematic model
function qddot = stateEqDesired(t,qd,dt,vdt,omegadt)

vd = interp1(dt,vdt,t);
omegad = interp1(dt,omegadt,t);
thetad = qd(3);
xddott = vd*cos(thetad);
yddott = vd*sin(thetad);
thetaddott = omegad;

qddot = [xddott;
         yddott;
         thetaddott];



function generatePlots(Tn, t, actualStates, desiredStates, error, u)
```

```
close all; % close all opened figures
% Tn = number of discrete time/path parameter points
% t = time/path parameter history, dimension = Tn x 1
% actualState = actual state of the system, dimension = Tn x n
% desiredState = desired state of the system, dimension = Tn x n
% error = desired state - actual state, dimension =  Tn x n
% u = control inputs, dimension = Tn x m


% Plot the robot's  velocities,
figure
subplot(2,1,1)
plot(t,u(:,1), 'k-','LineWidth', 1.5);
xlabel('Time [s]');
ylabel('Linear speed [m/s]');
grid on

subplot(2,1,2)
plot(t,u(:,2), 'k--','LineWidth', 1.5);
xlabel('Time [s]');
ylabel('Angular speed [rad/s]');
grid on

savefilename = ['OUT/controlInputs'];
saveas(gcf, savefilename, 'fig');
print('-depsc2', '-r300', [savefilename, '.eps']);

% Create a movie of the simulation (path/trajectory)


xmax = max(desiredStates(:,1));
xmin = min(desiredStates(:,1));
ymax = max(desiredStates(:,2));
ymin = min(desiredStates(:,2));

vid = VideoWriter('OUT/trajectory.avi');
vid.Quality = 100;
vid.FrameRate = 5;
open(vid)

fig=figure;
clf reset;
```

```
    for i = 1:Tn,
        clf;
        box on;
        axis([xmin-5 xmax+5 ymin-5 ymax+5]);
        axis equal;
        axis manual;
        [Xa,Ya] = plot_DDMR(actualStates(i,:),axis(gca)); %
        hold on;
        desired =
plot(desiredStates(1:i,1),desiredStates(1:i,2),'LineWidth',1.5);
        hold on
        actual = plot(actualStates(1:i,1),actualStates(1:i,2),'k--');
        fill(Xa,Ya,'r');
        hold off;
        xlabel('x [m]');
        ylabel('y [m]');
        F = getframe(fig);
        writeVideo(vid,F);
    end
    [Xa,Ya] = plot_DDMR(actualStates(1,:),axis(gca)); % DDMR =>
Differential drive mobile robot
    grid on
    hold on
    plot(Xa,Ya);
    hold on
    %legend([actual desired],'actual', 'desired');
    savefilename = 'OUT/trajectory';
    saveas(gcf, savefilename, 'fig');
    print('-depsc2', '-r300', [savefilename, '.eps']);

    close(vid);

    % Create the movie and save it as an AVI file
    % winopen('OUT/trajectory.avi')




function [X,Y] = plot_DDMR(Q,AX)
% PLOT_UNICYCLE   Function that generates lines for plotting a
unicycle.
%
%    PLOT_UNICYCLE(Q,AX) takes in the axis AX = axis(gca) and the
```

```
unicycle
%    configuration Q and outputs lines (X,Y) for use, for example,
as:
%       fill(X,Y,'b')
%    This must be done in the parent script file.

x     = Q(1);
y     = Q(2);
theta = Q(3);

l1 = 0.02*max([AX(2)-AX(1),AX(4)-AX(3)]);
X =
[x,x+l1*cos(theta-2*pi/3),x+l1*cos(theta),x+l1*cos(theta+2*pi/3),x];
Y =
[y,y+l1*sin(theta-2*pi/3),y+l1*sin(theta),y+l1*sin(theta+2*pi/3),y];
```



3.3 Division of Labor

Anthony Le - Helped write the code for Matlab and Pioneer, block diagram, put comments in the code

Jason Morris - Helped write the code for Pioneer, objective, background, implementation in lab report

Mike McGrath - Helped write the code for Pioneer, flow chart, table of contents, and Summary and Conclusion in lab report

Of course, throughout the lab all three of us consulted with each other and helped each other when necessary. Most of the time we were all working on the same thing at the same time. Everybody did roughly the same amount of work.

3.4 Summary and Conclusion

We were able to implement Cartesian Regulation for the Pioneer robot. The Pioneer went from the starting pose to the origin in 20 seconds. We were also able to implement part 2 (Posture regulation). The robot moved from the starting pose to the origin with an orientation of 0° in 60 seconds.

In summary, we learned how to use Cartesian Regulation and Posture Regulation to move the robot to a desired point and pose. We implemented these both by simulating them using Matlab and MobileSim and also by testing them in the lab using the actual robot.