Homework: Week 6

Jasen Carroll

MEM 380 – Wheeled Robotics

2/3/2011

Table of Contents

[Task 1: 3](#_Toc285041606)

[Task 2: 4](#_Toc285041607)

# Task 1:

%% This script drives a differential-drive robot subject to velocity (v)

%% and turn rate (omega) inputs in USARSim

% The robot kinematics are assumed to be

% xdot = v cos(theta);

% ydot = v sin(theta);

% thetaDot = omega;

% The robot's state (or pose) vector is a 6x1 vector of the form [x; y; z;

% thetaX; thetaY; thetaZ] where (x, y, z) denotes the world coordinates of

% the robot's position and thetaX, thetaY, thetaZ denotes the orientation

% of the robot w.r.t. the X, Y, Z axes respectively with 0 <= theta < 2pi

close all % closes all figure windows

clear all % clears all the workspace variables

clc % clears the command window

%% The following variables are specified by the user and must be

%% initialized each time the simulator runs

% the robot pose vector is [x; y; z; thetaX; thetaY; thetaZ] in global coordinates

startRobPose = [-5; -5; 1.8; 0; 0; 0]; % robot's initial state/pose

goalRobPose = [0; 0; 1.8; 0; 0; pi/2]; % robot's final state/pose

%% Initializes USARSim and spawns a robot at the given location

rob = initializeRobot('Rob', 'P2AT', startRobPose(1:3)', startRobPose(4:6)');

pause(5); % pauses for 5 seconds so we start getting a reading from INS sensor

%% internal variables used during the simulation

wheelR = 0.13; % wheel radius for P2AT in meters

wheelB = 0.415; % wheel base for P2AT in meters

maxWheelSpd = 5.385; % maximum wheel rotational speed in rad/s

Tol = 0.5; % Tolerance to determine whether robot has reached goal pose

v = 0; % current input velocity

omega = 0; % current input omega

robotTraj = startRobPose'; % vector to store all the history of the robot's poses

% unfortunately since we don't know how many

% iterations it will take to get to the goal,

% this vector will have to grow over time

err = norm(startRobPose(1:2)-goalRobPose(1:2)) + angleDifference(goalRobPose(6),startRobPose(6)); % difference between start and goal poses

%% Task 1: Open-loop control

% Define two vectors, vOpenLoop and omegaOpenLoop that contains all the

% sequence of open-loop control inputs for the robot

vOpenLoop = .53; % Found theoretically

omegaOpenLoop = -.145; % Originally found theoretically but then manipulated to fit expirement

insRdgs = getINSReadings(rob);

robPose = [insRdgs.Position; insRdgs.Orientation];

%% Main Control Loop

while err > Tol

%% Task 1:

v = vOpenLoop; % assign v to the respected vOpenLoop value

omega = omegaOpenLoop; % assign w to the respected omegaOpenLoop value

robState = getVehicleState(rob);

robState.TimeStamp;

%% converts v and omega inputs into motor commands

mLspd = (v - 0.5\*wheelB\*omega)/wheelR;

mRspd = (v + 0.5\*wheelB\*omega)/wheelR;

mLspd = 100\*mLspd/maxWheelSpd; % converts the wheel speeds to between -100 and 100

mRspd = 100\*mRspd/maxWheelSpd;

motorCmd = [max(min(mLspd,100),-100) max(min(mRspd,100),-100)]; % saturates the values at -100 and 100

sendDriveCommand(rob, motorCmd, 'differential');

pause(0.5);

% obtains new pose from robot's sensor

insRdgs = getINSReadings(rob);

robPose = [insRdgs.Position; insRdgs.Orientation];

% computes new error

err = norm(robPose(1:2)-goalRobPose(1:2)) + angleDifference(goalRobPose(6),robPose(6)); % difference between start and goal poses

% store new robot pose

robotTraj = [robotTraj; robPose'];

end

sendDriveCommand(rob, [0 0], 'differential'); % stops the robot

plotRobotTrajectory(startRobPose, goalRobPose, robotTraj)



***Figure 1:*** *Robot with open loop system*

# Task 2:

%% This script drives a differential-drive robot subject to velocity (v)

%% and turn rate (omega) inputs in USARSim

% The robot kinematics are assumed to be

% xdot = v cos(theta);

% ydot = v sin(theta);

% thetaDot = omega;

% The robot's state (or pose) vector is a 6x1 vector of the form [x; y; z;

% thetaX; thetaY; thetaZ] where (x, y, z) denotes the world coordinates of

% the robot's position and thetaX, thetaY, thetaZ denotes the orientation

% of the robot w.r.t. the X, Y, Z axes respectively with 0 <= theta < 2pi

close all % closes all figure windows

clear all % clears all the workspace variables

clc % clears the command window

%% The following variables are specified by the user and must be

%% initialized each time the simulator runs

% the robot pose vector is [x; y; z; thetaX; thetaY; thetaZ] in global coordinates

startRobPose = [-5; -5; 1.8; 0; 0; 0]; % robot's initial state/pose

goalRobPose = [0; 0; 1.8; 0; 0; pi/2]; % robot's final state/pose

%% Initializes USARSim and spawns a robot at the given location

rob = initializeRobot('Rob', 'P2AT', startRobPose(1:3)', startRobPose(4:6)');

pause(5); % pauses for 5 seconds so we start getting a reading from INS sensor

%% internal variables used during the simulation

wheelR = 0.13; % wheel radius for P2AT in meters

wheelB = 0.415; % wheel base for P2AT in meters

maxWheelSpd = 5.385; % maximum wheel rotational speed in rad/s

Tol = 0.5; % Tolerance to determine whether robot has reached goal pose

v = 0; % current input velocity

omega = 0; % current input omega

robotTraj = startRobPose'; % vector to store all the history of the robot's poses

% unfortunately since we don't know how many

% iterations it will take to get to the goal,

% this vector will have to grow over time

err = norm(startRobPose(1:2)-goalRobPose(1:2)) + angleDifference(goalRobPose(6),startRobPose(6)); % difference between start and goal poses

%% Task 2: closed-loop control

% Set the controller gains for the closed-loop controller

% Recall that Krho > 0, Kbeta < 0, and Kalpha+(5/3)\*Kbeta-(2/pi)\*Krho > 0

% for this controller converge

Krho = .115; % K values found expirementally

Kbeta = -0.0001;

Kalpha = .073378;

% obtains robot's current pose

insRdgs = getINSReadings(rob);

robPose = [insRdgs.Position; insRdgs.Orientation];

%% Main Control Loop

while err > Tol

%% Task 2:

% computation of required variables for closed-loop feedback control

rho = norm(insRdgs.Position(1:2)-goalRobPose(1:2)); % distance from robot to goal

dx = insRdgs.Position(1)-goalRobPose(1); % x direction of rho

dy = insRdgs.Position(2)-goalRobPose(2); % y direction of rho

alpha = insRdgs.Orientation(3)-mod(atan2(dx,dy),2\*pi); % angle from orientation to rho

beta = goalRobPose(6)-insRdgs.Orientation(3); % angle from goal to orientation

v = Krho \* rho;

omega = Kalpha\*alpha + Kbeta\*beta;

%% converts v and omega inputs into motor commands

mLspd = (v - 0.5\*wheelB\*omega)/wheelR;

mRspd = (v + 0.5\*wheelB\*omega)/wheelR;

mLspd = 100\*mLspd/maxWheelSpd; % converts the wheel speeds to between -100 and 100

mRspd = 100\*mRspd/maxWheelSpd;

motorCmd = [max(min(mLspd,100),-100) max(min(mRspd,100),-100)]; % saturates the values at -100 and 100

sendDriveCommand(rob, motorCmd, 'differential');

pause(0.5);

% obtains new pose from robot's sensor

insRdgs = getINSReadings(rob);

robPose = [insRdgs.Position; insRdgs.Orientation];

% computes new error

err = norm(robPose(1:2)-goalRobPose(1:2)) + angleDifference(goalRobPose(6),robPose(6)); % difference between start and goal poses

% store new robot pose

robotTraj = [robotTraj; robPose'];

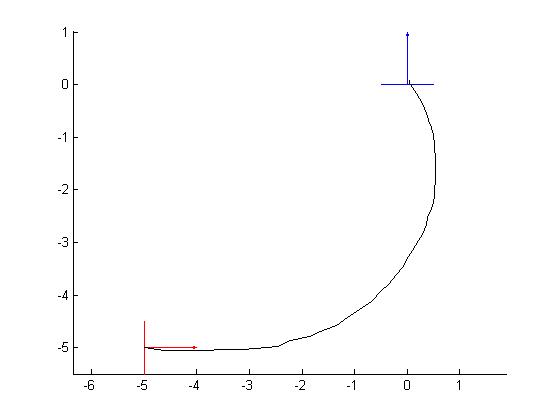
end

sendDriveCommand(rob, [0 0], 'differential'); % stops the robot

plotRobotTrajectory(startRobPose, goalRobPose, robotTraj)

shutdownRobot(rob)

***NOTE:*** *This code was tested several times with a lower success rate.* The problem was found to be that if the robot is not in the correct orientation when he reaches -5,-5 the while loop continues and the robot continues to turn back to the starting position and will eventually end up on a wall near the start. Much better success with the open loop version.



***Figure 2:*** *Robot with closed loop system*