Final Report - Lab #2 - Introduction to ROS

Daniel Engelsman - 300546173
Tom Smadar - 203374061

The Script is encapsulating the controller (enables us to give the robot multiple destinations and to plot its trail and heading) is given by:

```
rosshutdown

%% Initialization
close all
clear all
clc

rosinit
global robot_path robot_az rate runNum b c
%% Topics
Odom_topic = rossubscriber('/pioneer/odom');
Vel_topic = rospublisher('/pioneer/cmd_vel');
rate = 0.2;
%% Goals
Goal(1).loc = [-0.9 -0.2 0]';
Goal(1).Orien = deg2rad(2.4);
Goal(2).loc = [-0.9 1 0]';
Goal(2).Orien = deg2rad(0);
Goal(3).loc = [1 1 0]';
Goal(3).Orien = deg2rad(-90);
Goal(4).loc = [1 -1 0]';
Goal(4).Orien = deg2rad(90);


%% Execute
SafetyHalt(Vel_topic);
b = 1;
c = 1;
for k=1:length(Goal)
        MyController(Goal(k),Odom_topic,Vel_topic);
end

%% Plot
figure(1)
hold on
box on
grid on
scatter(robot_path(1,:),robot_path(2,:))
set(gca,'FontSize',14)
title('Robot Path','FontSize',20)
xlabel('X [m]','FontSize',14)
ylabel('Y [m]','FontSize',14)

figure(2)
hold on
box on
grid on
plot(robot_az(2,:),rad2deg(robot_az(1,:)),'LineWidth',2)
set(gca,'FontSize',14)
title('Robot Azimuth','FontSize',20)
xlabel('Time [s]','FontSize',14)
ylabel('Azimuth [Deg]','FontSize',14)
```

And the functions in use are (given PD Block not included):

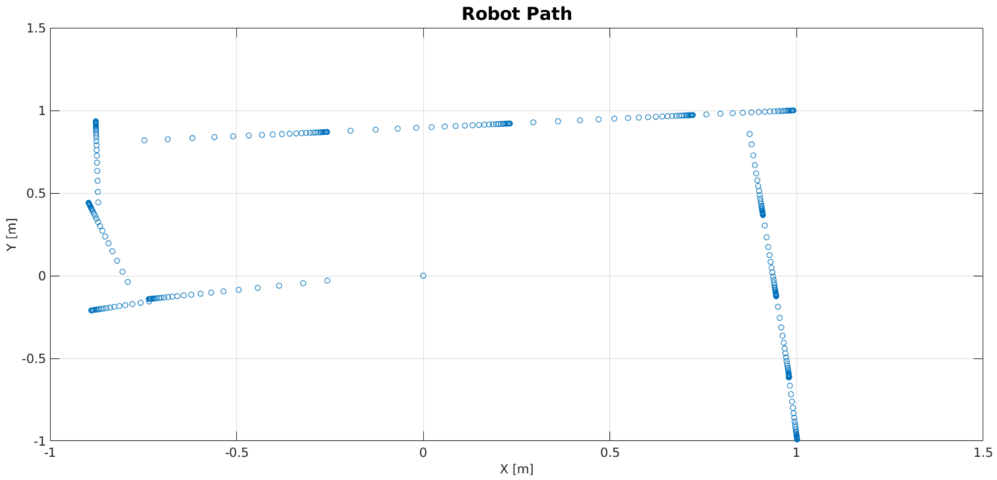
```
function [] = MyController(Goal,Odometry_Topic,Velocity_Topic)
        %% Initialization
        OdoSub = Odometry_Topic;    % Odometry subscriber
        VelPub = Velocity_Topic;  % Velocity command publisher
        epsLoc = 0.1;                % Permitted location error [m]
        step = 0.5;                  % Step length [m]
        %% Movement
        while(1)
        odo = receive(OdoSub,2);      % Recieve odometry data
        r = odo.Pose.Pose.Position;
        Rc = [r.X r.Y r.Z]';  % Current Position
        LocErr = norm(Goal.loc - Rc); % Current distance from goal [m]
        % Breaking Conditions
        if LocErr < epsLoc
        SetLOS(Goal.Orien,OdoSub,VelPub);   % If we've reached goal, turn to requested azimuth and break
        SafetyHalt(VelPub);
        return;
        end

        % Movement
        SetLOS(CalcLOS(Goal.loc,OdoSub),OdoSub,VelPub); % for each step, turn towards goal
        StepFwd(min(step,LocErr),OdoSub,VelPub);        % step forward
        end
end

function [] = SafetyHalt(VelPub)
%SafetyHalt Safely halts the Pioneer
vCmd_msg = rosmessage(VelPub);  % Create vel. cmd. message
vCmd_msg.Linear.X = 0;
vCmd_msg.Linear.Y = 0;
vCmd_msg.Linear.Z = 0;
vCmd_msg.Angular.X = 0;
vCmd_msg.Angular.Y = 0;
vCmd_msg.Angular.Z = 0;
send(VelPub,vCmd_msg)
end

function [Az1] = CalcLOS(R1,OdoSub)
% This function calculates the required LOS for the Pioneer robot


        odo1 = receive(OdoSub,2);   % Recieve odometry

        r = odo1.Pose.Pose.Position;
        R0 = [r.X r.Y r.Z]'; % Current position
        Vec = R1-R0;                 % Delta to goal
        Az1 = -atan2(Vec(2),Vec(1));   % Goal azimuth
end
```

```matlab
function [] = SetLOS(Az1,OdoSub,VelPub)
% this function turns the Pioneer robot around to a required azimuth
global rate b c robot_path robot_az
        odo = receive(OdoSub,2);   % Recieve odometry
        t0 = odo.Header.Stamp.Sec; % Initial time
        a=1;        % index
        vCmd = 0;             % Initial velocity command
        maxAzErr = deg2rad(0.5);  % Permitted angular error (epsilon)

        while(1)

        odo = receive(OdoSub,2);   % Recieve odometry
        pause(rate)
        r = odo.Pose.Pose.Orientation;
        ang = [r.X r.Y r.Z r.W]; % Current orientation
        ang = ang./norm(ang);
        ang = quat2eul(ang,'XYZ');                    % Conver to Euler angles
        Az0 = ang(1);                    % current azimuth
        disp(rad2deg(Az0));
        Err(a) = Az1-Az0;                % Azimuth difference
        disp(rad2deg(Err(a)))
        %Recording:
        rpos = odo.Pose.Pose.Position;
        R1 = [r.X r.Y r.Z]'; % Current position
        robot_path(:,b) = R1;
        b=b+1;
        robot_az(1,c) = Az0;
        t1 = odo.Header.Stamp.Sec; % Current timestamp
        robot_az(2,c) = t1;
        c=c+1;
        % /Recording
        if abs(Err(a))>pi                % Ensure shortest turn
        if Err(a) > 0
        Err(a) = Err(a) - 2*pi;
        else
        Err(a) = Err(a) + 2*pi;
        end
        end
        if abs(Err(a)) < maxAzErr        % if required azimuth reached, break
        SafetyHalt(VelPub);
        return;
        end
        dt = t1-t0;                      % Time step
        if dt<1
        dt=0.5;
        end
        t0=t1;                           % Remember previous timestamp
        vCmd = PD_Block([0.5 0.001 200],dt,Err,vCmd); % Calculate angular velocity command
        if abs(vCmd) > 0.7               % Verify angular velocity command does not exceed permitted maximum
        vCmd = 0.7*sign(vCmd);
        end
        vCmd_msg = rosmessage(VelPub);        % Create velocity command message
        vCmd_msg.Angular.Z = -vCmd;
        send(VelPub,vCmd_msg);                % Send velocity command message to robot
        a=a+1;
        end
End
```

```
function [] = StepFwd(step,OdoSub,VelPub)
% This function makes the Pioneer robot mover forward, a specified distance
global rate robot_path b robot_az c
        odo1 = receive(OdoSub,2);   % Recieve odometry
        t0 = odo1.Header.Stamp.Sec; % Initial time
        a=1;        % index
        vCmd = 0;             % Initial velocity command
        maxErr = 0.01;  % Permitted step error (epsilon)
        r = odo1.Pose.Pose.Position;
        R0 = [r.X r.Y r.Z]'; % Starting Position
        while(1)
        odo1 = receive(OdoSub,2);   % Recieve odometry
        pause(rate)
        r = odo1.Pose.Pose.Position;
        R1 = [r.X r.Y r.Z]'; % Current position
        %Recording:
        robot_path(:,b) = R1;
        b=b+1;
        r = odo1.Pose.Pose.Orientation;
        ang = [r.X r.Y r.Z r.W]; % Current orientation
        ang = ang./norm(ang);
        ang = quat2eul(ang,'XYZ');                  % Convert to Euler angles
        robot_az(1,c) = ang(1);
        t1 = odo1.Header.Stamp.Sec; % Current timestamp
        robot_az(2,c) = t1;
        c=c+1;
        % /Recording
        disp(R1);
        Err(a) = step - norm(R1-R0);  % Remaining distance to complete step
        if abs(Err(a)) < maxErr
        SafetyHalt(VelPub);
        return;                  % If we've completed the step, break
        end
        t1 = odo1.Header.Stamp.Sec; % Current timestamp
        dt = t1-t0;              % Time step
        if dt<1
        dt=0.5;
        end
        t0=t1;                   % Remember previous timestamp
        vCmd = PD_Block([0.6 0.05 100]',dt,Err,vCmd); % Calculate velocity command
        if abs(vCmd) > 1                 % Verify velocity command does not exceed permitted maximum
        vCmd = 1*sign(vCmd);
        end
        vCmd_msg = rosmessage(VelPub);  % Create vel. cmd. message
        vCmd_msg.Linear.X = vCmd;
        send(VelPub,vCmd_msg);       % Send velocity cmd. message to robot
        a=a+1;

        end
end
```

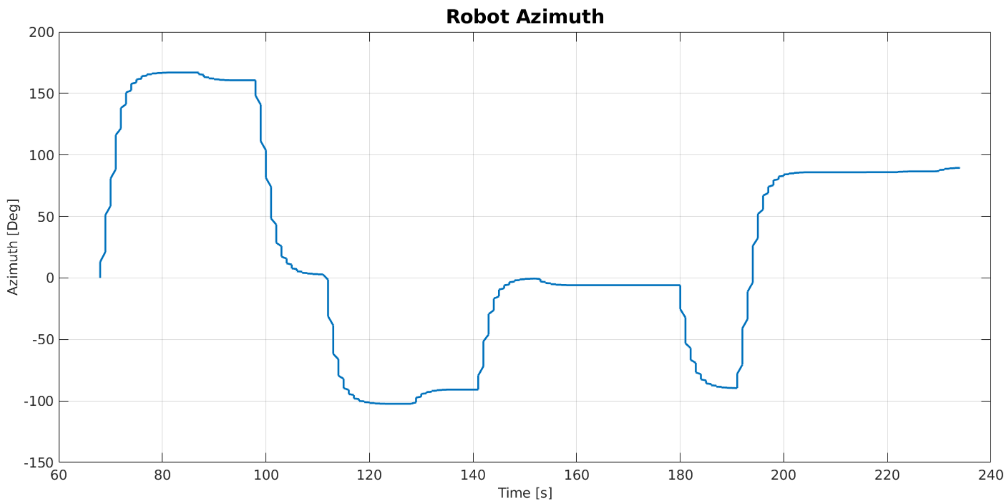
Results and conclusions

The route is comprised of numerous “goal points”, that are predefined in the envelope script. We then launch it, and execution occurs in a for-loop (per goal point) all time that the desired approximation error has not been achieved.



Robot Path in the XY plane [m]

The 4 given goal points dictate the robot’s path, and are segmented as function of the desired “step” size (here: 0.5 [m]). The density of the sampling points are explained by the properties of the given PD block, whose intensity is influenced by the magnitude of the Error.



Azimuth vs. Time [s]

The Second graph shows the currnet Robot’s Azimuth while seeking to reduce the Error. The curves are when error is big, and when parallelism is obtained, the robot is vacant to move on to the StepFwd function, that moves him forward towards the current goal point.