Daavid Moseley
Sensor Based Robotics
Project Report – Path Planning in a Cluttered Workspace
May 18th, 2017

Abstract

One of the primary drivers of robotics research is the "path planning problem," a fundamental topic which involves finding the correct pathway algorithm for a robot to navigate through a cluttered workspace environment. The pathway for a robot is constructed by finding the most optimal pathway from the start position of the robot to the target (or end) position via the application of one of many algorithmic methodologies (e.g., Djikstra). From this point forth, the goal shifts: Now that the pathway has been found, *how* will the robot move? For this project, a 3-link manipulator is mounted on a mobile robot which must navigate through an obstacle set to arrive at its end position. As the mobile robot navigates through this pathway, however, the mounted 3-link manipulator must also avoid another set of obstacles which are intended to deter the mobile robot from achieving its goal. The robotics simulation software V-REP is used to design the 3-link robot manipulator and the Pioneer P3DX is used as the mobile robot. V-REP is also used to construct the pathway of the robot. MATLAB is utilized to navigate the mobile robot and 3-link manipulator through the cluttered workspace to reach the end position.

Introduction

The path planning problem is quite unique in the field of robotics research literature since each robot manipulator and obstacle set is different. Hence, it follows that there exists no specific algorithmic methodology to employ for a given problem; typically, one must analyze the system as a geometric entity and choose the best algorithm that works for the environment and robot (mobile or non-mobile). The project presented here is represents dual layered path planning: A 3-link RRR robotic manipulator must avoid three cuboid obstacles whilst the mobile vehicle it is mounted upon must also avoid its own unique obstacle set *and* the robot manipulator obstacle set. Both obstacle sets are unchanging, e.g., they do not represent moving/dynamic obstacles. The obstacle sets, the 3-link RRR robotic manipulator and the mobile vehicle were all implemented in V-REP; the coding for the motion planning of the robot was done in MATLAB via V-REP's Remote API. The pathway was constructed in V-REP using their path planning module.

This report is divided into five main sections:

1. Introduction
2. Development of the Environment in V-REP
3. Motion planning/MATLAB Coding Discussion
4. Conclusion
5. Appendix (Code)

Development of the Environment

3-Link RRR Manipulator

The Revolute-Revolute-Revolute manipulator was designed in V-REP using the three revolute joints and four primitive objects (three cuboids and one cylinder). The manipulator had one main function: Avoiding any/all obstacles that would come in its immediate pathway. The obstacle set for this robot was constructed to present a difficult, yet manageable workspace for the manipulator to navigate. The decision to develop a unique manipulator rather than use a previously constructed one was based upon both the familiarity with designing a manipulator *and* the fact that the primary goal wasn't to utilize a *complicated manipulator*, but to have it navigate a *complicated pathway*.

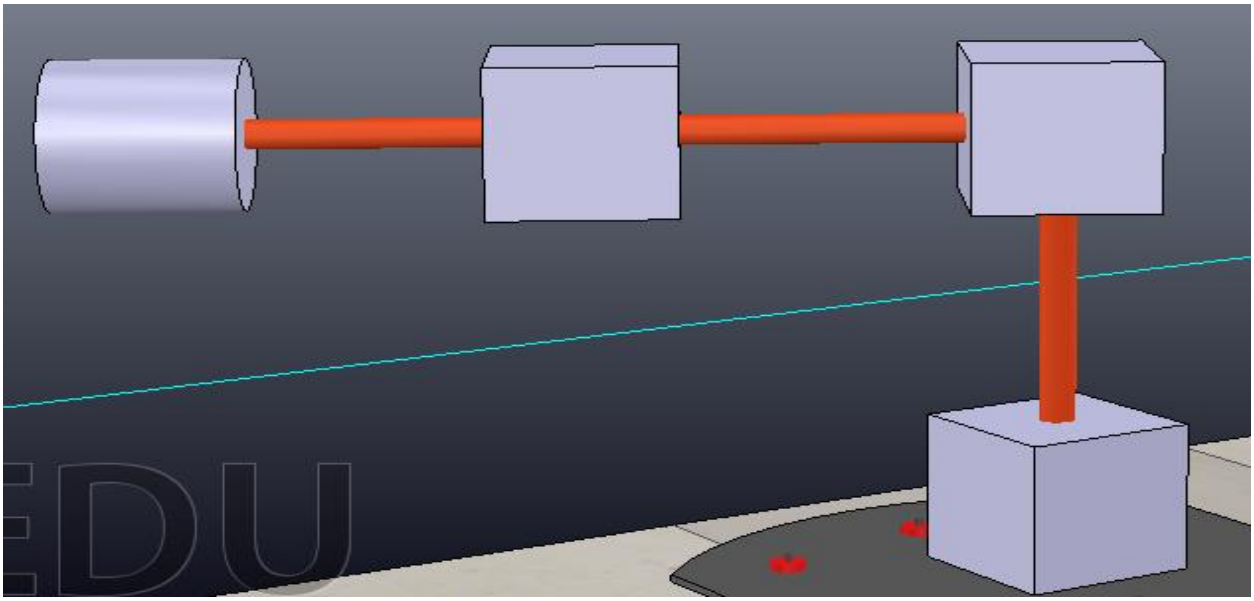

Figure 1. View of the robot manipulator.

Pioneer P3DX Robot

The pioneer P3DX mobile robot is a standard V-REP model. Originally, the project was only supposed to feature one (or two) robot manipulators coordinating with one another through an obstacle set, but this idea proved to be not as intriguing as the concept of having a mounted robot avoid an obstacle set *while another robot* avoided its own, distinct obstacle set. Hence, after experimenting with several mobile robots, the P3DX model was chosen for its wheel steering capability and its ability to balance the weight of the robotic manipulator.

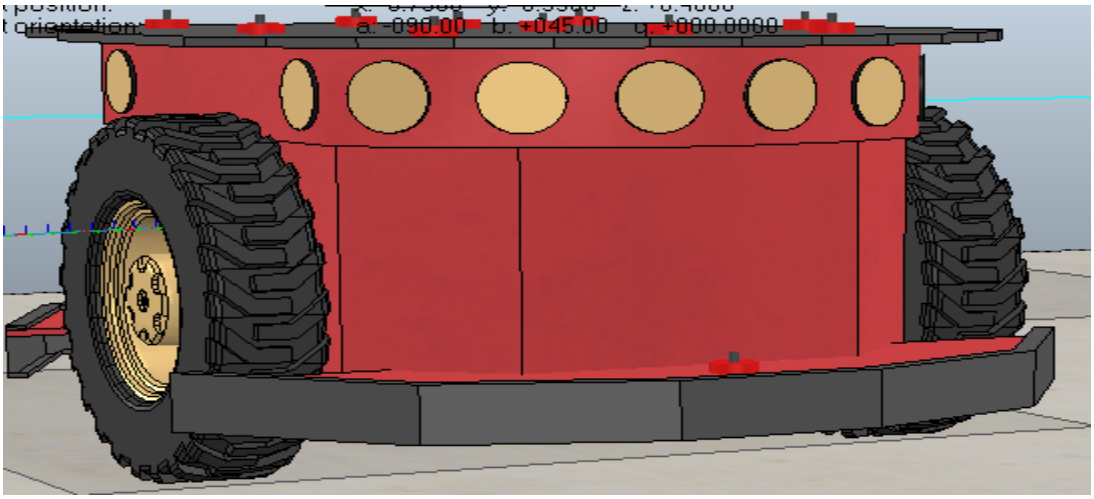

Figure 2. Pioneer P3DX Mobile Robot.

Obstacle Sets

The obstacle sets were created in a manner to create a technically challenging workspace environment for both robots. To be precise, the cluttered workspace needed to present navigational difficulty for the mobile robot (e.g., sudden sharp turning, avoidance of random obstacles as it redirected its trajectory, adjusting wheel speed to account to maintain balance of the robot manipulator) and obstacle avoidance for the robot manipulator as its companion robot engaged the environment. This setup created two distinct, yet integral obstacle sets shown below.

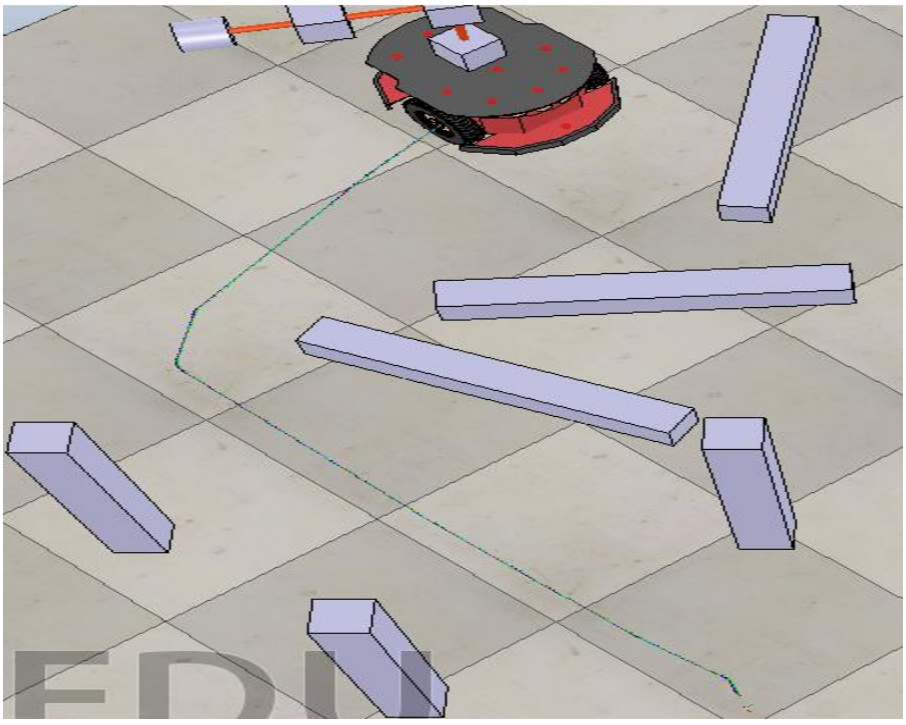

Figure 3. The cuboids laying on the ground in front of the mobile robot represent the obstacle set for the mobile robot, whereas the vertical cuboids represent the obstacle set for the manipulator.

Pathway Development

The pathway was developed thanks to V-REP's in built path planning module. First, the mobile robot obstacle set was created and grouped as a collection. Two dummies (start and end) were created thereafter to signify the starting/end points of the pathway. The "start" dummy was stationed at the position of the mobile robot, whereas the "end" dummy was positioned outside of the obstacle set. The path planning module was then executed, using the Nonholonomic settings since the model of the robot is subject to differential constraints/non-integrable. The appropriate settings were modified and several pathways were automatically developed in V-REP. Of all pathways, the shortest and most optimal pathway was chosen. It also happened to be the most difficult pathway. One pathway, for example, included just moving *around* the entire obstacle set rather than actively engage it; this was far too trivial to implement.

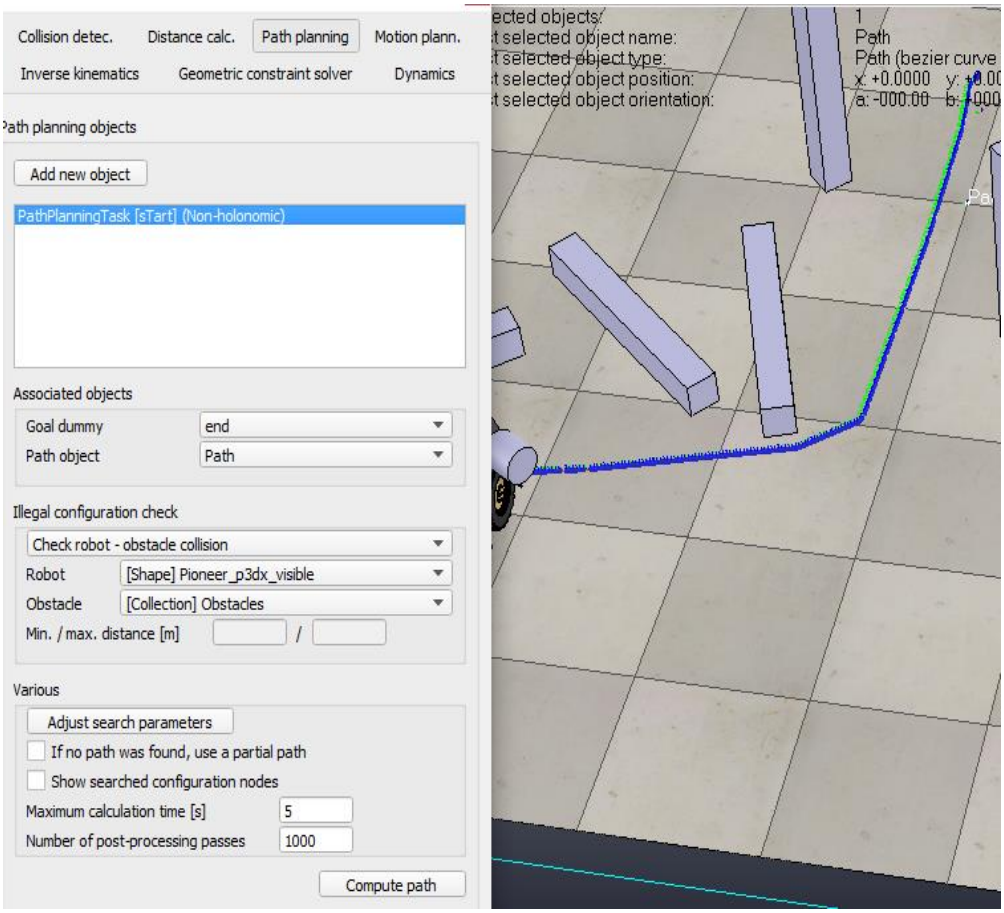

Figure 4. Path Planning module & highlighted pathway.

Motion Planning/MATLAB Code Discussion

Once the robots and obstacle sets were appropriately developed, MATLAB was used for the navigational programming via V-REP's Remote API. Making use of the simxSetJointTargetVelocity for the right/left wheels of the p3dx robot and setting them to varying speeds at different intervals (e.g., sharp and sudden turning), successful navigation of the pathway proved to be time consuming, but manageable. The primary dilemma was controlling the lag offset in the communication protocol between MATLAB and V-REP. During certain intervals of the code, the communication between programs would lag and deem my navigational code useless, as it would collide with several objects. This communication offset was what originally prevented the code from being fully operational and allowing full testing. Lag reduction was finally achieved upon stopping movement of the mouse and halting the usage of any unnecessary background programs from running as the code was implemented.

The secondary dilemma was *timing*. Although V-REP has an in-built *path planning* module, it removed the *path following* module that it once exhibited. If a robot (mobile or non-mobile) must follow a certain path, this must be developed in a V-REP child script *or* in the remote software. There are two primary methodologies to employ to have a robot follow a directed pathway: (a) Development of a controller to follow a chosen pathway or (b) manually develop an algorithm to follow a chosen pathway. For this project, option #2 was chosen since option #1 worked conceivably better in the child script of V-REP in its predetermined language, Lua. Since option #2 was chosen, the timing of the robot was crucial so that it didn't collide with another obstacle whilst avoiding another one. After fixing the lag offset, this took some patience/trial and error.

Mirroring V-REP Assignment #2, the decision was made to monitor the joint position via the simxGetJointPosition command.

```
% Motor Pathway

[returnCode] = vrep.simxSetJointTargetVelocity(clientID, left_Motor, 2 ,vrep.simx_opmode_bl

pause(1.1);

[returnCode] = vrep.simxSetJointTargetVelocity(clientID, left_Motor,0,vrep.simx_opmode_bloc
[returnCode] = vrep.simxSetJointTargetVelocity(clientID, right_Motor, 2, vrep.simx_opmode_b
[returnCode] = vrep.simxSetJointTargetVelocity(clientID, left_Motor, 2.6 ,vrep.simx_opmode_

pause(1.2);

[returnCode] = vrep.simxSetJointTargetVelocity(clientID, right_Motor,0,vrep.simx_opmode_blo
[returnCode] = vrep.simxSetJointTargetVelocity(clientID, left_Motor,0,vrep.simx_opmode_bloc
[returnCode] = vrep.simxSetJointTargetVelocity(clientID, right_Motor,5,vrep.simx_opmode_blo
[returnCode] = vrep.simxSetJointTargetVelocity(clientID, left_Motor,2.3,vrep.simx_opmode_bl


pause(1);
```

Figure 5. Snippets of the code.

Conclusion

This project was somewhat time consuming since a manual algorithm had to be developed for the given pathway. In addition, although the mobile robot and robotic manipulator were following the path, for a sequential stage of this project, it would be advisable to write a function that would calculate how much from the center the mobile robot veered from the pathway. In addition, the implementation/creation of a controller for the mobile robot and robotic manipulator would've potentially enhanced the accuracy/timing of the navigational movement. Nonetheless, the robots in this project avoided all obstacles as it traveled along its predetermined trajectory, thereby making all aforementioned issues improvements rather than drastic changes.

In the future, this project will be revisited and all recommended changes outlined earlier will be implemented. Furthermore, a far more complex obstacle set will be constructed with the coordination of two (or more) robotic manipulators and mobile robots. Navigating a pathway with a dynamic obstacle set (e.g., three mobile vehicles moving haphazardly in the workspace) would be an ideal environment for technical complexity.

Appendix

```
vrep=remApi('remoteApi');

vrep.simxFinish(-1);
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);

if (clientID>-1)
    disp('Connected')

    double angle_1;
    double angle_2;

    angle_1 = (-.7)*pi;
    angle_2 = (-.49)*pi;

    % Naming Joint Variables


[returnCode,joint_1]=vrep.simxGetObjectHandle(clientID,'Revolute_joint',vrep.
simx_opmode_blocking);

[returnCode,joint_2]=vrep.simxGetObjectHandle(clientID,'Revolute_joint0',vrep
.simx_opmode_blocking);

[returnCode,link_1]=vrep.simxGetObjectHandle(clientID,'Cuboid0',vrep.simx_opm
ode_blocking);

[returnCode,link_2]=vrep.simxGetObjectHandle(clientID,'Cuboid1',vrep.simx_opm
ode_blocking);
```

```
    % Naming Motor Variables


[returnCode,left_Motor]=vrep.simxGetObjectHandle(clientID,'Pioneer_p3dx_leftM
otor',vrep.simx_opmode_blocking)

[returnCode,right_Motor]=vrep.simxGetObjectHandle(clientID,'Pioneer_p3dx_righ
tMotor',vrep.simx_opmode_blocking)

    % Motor Pathway

    [returnCode] = vrep.simxSetJointTargetVelocity(clientID, left_Motor, 2
,vrep.simx_opmode_blocking) % Initial turn

    pause(1.1);

    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
left_Motor,0,vrep.simx_opmode_blocking)
    [returnCode] = vrep.simxSetJointTargetVelocity(clientID, right_Motor, 2,
vrep.simx_opmode_blocking)
    [returnCode] = vrep.simxSetJointTargetVelocity(clientID, left_Motor, 2.6
,vrep.simx_opmode_blocking)

    pause(1.2);

    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
right_Motor,0,vrep.simx_opmode_blocking)
    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
left_Motor,0,vrep.simx_opmode_blocking)
    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
right_Motor,5,vrep.simx_opmode_blocking)
    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
left_Motor,2.3,vrep.simx_opmode_blocking)



    pause(1);

    %Start: Avoid Cuboid #1 with robot manipulator.
    [returnCode] =
vrep.simxSetJointTargetPosition(clientID,joint_1,angle_1,vrep.simx_opmode_blo
cking);
    % End
    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
right_Motor,3,vrep.simx_opmode_blocking);
    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
left_Motor,2.4,vrep.simx_opmode_blocking);

    pause(1);

    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
right_Motor,1,vrep.simx_opmode_blocking);
    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
left_Motor,2,vrep.simx_opmode_blocking);
```

```
    pause(0.5)

    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
left_Motor,1.2,vrep.simx_opmode_blocking);
    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
right_Motor,2,vrep.simx_opmode_blocking);



    pause(1)

    % Start: Avoid Obstacle/Cuboid #2 before stopping.
    [returnCode] =
vrep.simxSetJointTargetPosition(clientID,joint_1,angle_2,vrep.simx_opmode_blo
cking);
    % End

    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
left_Motor,5,vrep.simx_opmode_blocking);

    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
right_Motor,5,vrep.simx_opmode_blocking);

    pause(1)

    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
left_Motor,0,vrep.simx_opmode_blocking);

    [returnCode] = vrep.simxSetJointTargetVelocity(clientID,
right_Motor,0,vrep.simx_opmode_blocking);

    %Motor Pathway Done

    %Position Reporting Initial
    [returnCode,position_1]=vrep.simxGetObjectPosition(clientID, link_1, -1,
vrep.simx_opmode_streaming);
    [returnCode,position_2]=vrep.simxGetObjectPosition(clientID, link_2, -1,
vrep.simx_opmode_streaming);
    [returnCode,joint_Pos1]=vrep.simxGetObjectPosition(clientID, joint_1, -1,
vrep.simx_opmode_streaming);
    [returnCode,joint_Pos2]=vrep.simxGetObjectPosition(clientID, joint_2, -1,
vrep.simx_opmode_streaming);



    for i=1:15000
    [returnCode,position_1]=vrep.simxGetObjectPosition(clientID, link_1, -1,
vrep.simx_opmode_buffer);
    [returnCode,position_2]=vrep.simxGetObjectPosition(clientID, link_2, -1,
vrep.simx_opmode_buffer);
    [returnCode,joint_Pos1]=vrep.simxGetObjectPosition(clientID, joint_1, -1,
vrep.simx_opmode_buffer);
    [returnCode,joint_Pos2]=vrep.simxGetObjectPosition(clientID, joint_2, -1,
vrep.simx_opmode_buffer);
    end
```

```
    disp('First Joint Position')
    disp(joint_Pos1)
    disp('Second Joint Position')
    disp(joint_Pos2)
    disp('First Link Position: ')
    disp(position_1)
    disp('Second Link Position: ')
    disp(position_2)

end

vrep.delete()
```