Rover Trajectory Planning via Simulation Using Incremented Particle Swarm Optimization

Ayman Hamdy Kassem
Aerospace Engineering Department, Cairo University

Abstract – This paper presents an off-line optimal trajectory planning for differential-drive rover through simulation of the dynamic model. The paper starts with the model dynamics of an actual rover built in our space science and technology lab (SSTLab) and controlled by simple PD controllers. Next, the proposed optimization technique used is presented which is called Incremented Particle-Swarm Optimization (IPSO) where the number of variable increases incrementally if the goal is not satisfied to minimize the time and CPU usage running the cost function. Different trajectories and cost functions were tested with obstacles and without it. The results show that the trajectory can be optimized efficiently using IPSO and a simple cost function based on total time and distance to final destination.

Keywords: Modeling, optimization, trajectory planning.

1 Introduction

Differential-drive rover is a two wheels configuration where the wheels are in-line with each other. The wheels are independently powered and controlled so the desired motion will depend on how these wheels are controlled. It is a simple robot which gained a lot of popularity for its many uses applications such outer space exploration and deep sea excavation, etc. It also gained a lot of popularity in robotic competitions such as NASA rover challenge and Robotic Rover Competition by The Mars Society. It also gained popularity in academic research [1-5].

For so many applications on land and under sea, there is always a need to have a planned trajectory optimized for time, distance, or obstacle-avoidance to save resources in a mission while improving its value. Off-line trajectory optimization will work perfectly for well-known terrains and will improve sensors readings for real-time trajectory planning with partially-known terrains.

In this paper, the model of an experimental differential-drive rover, built in our lab, will be presented and will be used for simulating the rover for trajectory optimization. The resulting system takes voltage as a motor input, hence the rotational speed of the wheels may vary and so will the robot’s position whose central point is represented in a Cartesian plane (XY).

A new proposed optimization technique is introduced. It is based on particle swarm optimization starting with small number of variables and then incrementing of the number of variables to improve computational cost and reach the optimum solution faster.

2 Rover Model

The differential-drive rover uses the two different controllers to control the speed of each wheel. If the speeds of both wheels are equal the rover will go straight. If they are unequal, the rover will rotate. Figure 1 shows counter clockwise rotation mechanism.

Counter Clockwise Rotation

\[ \mathbf{V}_r > \mathbf{V}_l \]

Left Wheel \[ \mathbf{V}_l \]

ROVER

Right Wheel \[ \mathbf{V}_r \]

Figure 1: Rotational mechanism for differential-drive rover

The dynamics equations for a single wheel motor are given by:

\[
\begin{align*}
\mathbf{u}_c(t) &= \mathbf{R} \mathbf{i}_c(t) + L \frac{d\mathbf{i}_c(t)}{dt} + \mathbf{e}_s(t) \\
\mathbf{e}_s(t) &= k_\omega \omega_m(t) \\
\mathbf{T}_m(t) &= k_\tau \mathbf{i}_c(t) \\
\mathbf{T}_m &= \mathbf{J} \frac{d^2\theta_m}{dt^2} + \mathbf{B} \frac{d\theta_m}{dt}
\end{align*}
\]
Where:
- \( R \): motor resistance.
- \( L \): motor inductance.
- \( e_b \): back emf.
- \( T_m \): motor torque.
- \( i \): electric current.
- \( k_b, k_i \): motor constants.
- \( \theta_b \): motor rotational angle.
- \( J_m \): motor inertia.
- \( B_m \): motor damping coefficient.
- \( \omega_m \): motor rotational speed.

Figure 2 shows the electromechanical circuit for the motor

\[
\begin{align*}
\begin{bmatrix}
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 \\
-0.45 & -0.007 & 0.51 & 0 & 0 \\
0 & -1.58 & -5.31 & 0 & 0 \\
-11.74 & 4.11 & -12.97 & 0 & 0 \\
-11.74 & -4.11 & 0 & -12.97 & 0 \\
0 & 1 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
\theta_b \\
T_m \\
e_b \\
L \end{bmatrix}
= \begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
\end{bmatrix}
= \begin{bmatrix}
0 & 0 & 12.97 & -4.11 \\
0 & 0 & 5.31 & -1.58 \\
0 & 0 & 0.51 & -0.007 \\
0 & 0 & 0.51 & -0.007 \\
0 & 0 & 12.97 & -4.11 \\
0 & 0 & 12.97 & -4.11 \\
0 & 0 & 12.97 & -4.11 \\
\end{bmatrix}
= \begin{bmatrix}
0 & 0 & 12.97 & -4.11 \\
0 & 0 & 5.31 & -1.58 \\
0 & 0 & 0.51 & -0.007 \\
0 & 0 & 0.51 & -0.007 \\
0 & 0 & 12.97 & -4.11 \\
0 & 0 & 12.97 & -4.11 \\
0 & 0 & 12.97 & -4.11 \\
\end{bmatrix}
\end{align*}
\]

The model is controlled by two PD controllers for the speed and heading rate channels. The actual rover is shown in figure 3.

Adding integrators to the angular and translational rates and using transformation (sine and Cosine), we can have the position co-ordinates or the rover (X, Y) as shown in figure 4.

Setting the heading angle rate command and translation velocity command, the simulator will calculate the position of the rover. By fixing the rover translation velocity command, the heading angle rate command is used to control the position of the rover and it will be used as the optimization parameters as shown in the following section.

### 3 Optimization

The optimization process is shown in figure 5. It starts with specifying the destination for the rover and any constraints (obstacles) and the rover model. The PSO module will pass randomly generated input to the rover model simulator and then the error budget is calculated and passed to the cost function to the PSO module.
PSO is an algorithm proposed by James Kennedy and R. C. Eberhart in 1995 [7, 8, 9] who were motivated by the social behavior of organisms such as bird flocking and fish schooling.

PSO shares many similarities with evolutionary computation techniques such as GAs. The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike the GA, PSO has no evolution operators such as crossover and mutation.

PSO mimics the behaviors of bird flocking in algorithmic form. Suppose the following scenario: a flock of birds is randomly searching for food in an area. There is only one piece of food in the area being searched. All of the birds do not know where the food is, but they know which bird is the closest to food in each iteration. The effective strategy is to follow the bird which is nearest to the food.

\begin{equation}
v = v + c_1 \times \text{rand} \times (p\text{best} - \text{present}) + c_2 \times \text{rand} \times (g\text{best} - \text{present}) \tag{5}
\end{equation}
\begin{equation}
\text{present} = \text{present} + v \tag{6}
\end{equation}

Here \(v\) is the particle velocity and \(\text{present}\) is the current particle (solution). \(p\text{best}\) and \(g\text{best}\) are defined as stated before. \(\text{rand}\) is a random number between (0,1). \(c_1\) and \(c_2\) are learning factors,\[7\] and usually \(c_1 = c_2 = 2\).

PSO is an extremely simple algorithm that seems to be effective for optimizing a wide range of functions.\[7,9\]. It also uses the concept of \textit{fitness}, as do all evolutionary computation paradigms. The pseudo code is shown in figure 4.

The proposed incremented particle swarm optimization IPSO is a modified version of PSO. It is best suited when the optimization parameters are functions in time like direction commands for robots. These types of optimization problems were solved numerically by assuming higher order polynomials or splines etc. and trying to find the coefficients of these polynomials or splines that satisfy the optimization problem. In IPSO, few numbers of values (control points) are assumed for design variable (in this case it is the heading angular speed). These control points, as it controls the shape of a function, are equally distributed over time and the design variable is assumed piecewise linear in between using direct linear interpolation as seen in figure 7-a. In figure 7-a, four values define the shape of the design variable over time \(F(t)\). It is the PSO task to find these values which satisfy the minimization process. If PSO failed, the number of control points is increased which allow more freedom for the shape of the design variable. Figure 7-b shows a function \(F(t)\) after incrimination with three intermediate control points (seven control points total). It can be seen that \(F(t)\) in figure 7-b has the ability to represent more complicated functions with seven control points than \(F(t)\) in figure 7-a with only four control points as shown with the dotted lines. The same procedure of increasing the number of variables is applied repeatedly. Each time the search-space representing the variables is expanded (i.e. more control points are added) and the added values are linear interpolation of the adjacent old ones. In another words, the procedure starts with a coarse grid of control points and
uses the solutions of this coarse grid as an initial guess (population) for a finer grid and continues this process until a stop criterion occurs. The idea of linear interpolation, besides simplifying the calculations, guarantees that the initial population of the expanded search-space has the same best fitness of the search-space of smaller dimensions. This guarantees convergence to better solution faster than classical PSO. It also helps in selecting the correct number of optimization variables for the trajectory as you start with small number and increase till you reach the optimum.

![Figure 7: Piecewise linear function approximation using control points](image)

The cost function has three parts one for the trajectory which is the difference between the required final position and the actual final position for the rover and the second part is time which is used as a measure of power consumption. The third part is a penalty for constraints usually obstacles. The cost function is shown in table 1

<table>
<thead>
<tr>
<th>Table 1 Cost Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- Cost function for trajectory optimization:</td>
</tr>
<tr>
<td>$F_1 =</td>
</tr>
<tr>
<td>where $P_{final}$ represents rover actual final position. $P_{required}$ represents the required final position</td>
</tr>
<tr>
<td>2- Cost function for power minimization</td>
</tr>
<tr>
<td>$F_2 = \int T dt$ where T is the time in seconds.</td>
</tr>
<tr>
<td>3- Penalty for constraints</td>
</tr>
<tr>
<td>$F_3$ defined based on the constraint and will be shown in the test cases</td>
</tr>
<tr>
<td>3- Cost fitness function $F = \alpha_1 F_1 + \alpha_2 F_2 + \alpha_3 F_3$</td>
</tr>
<tr>
<td>where $\alpha_1$, $\alpha_2$, $\alpha_3$ are weighting constants to be selected by user.</td>
</tr>
</tbody>
</table>

The IPSO procedure can be summarized as follows:

<table>
<thead>
<tr>
<th>Table 2 IPSO procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- Start with n control points for each variable.</td>
</tr>
<tr>
<td>2- Run the PSO procedure (figure 4) and call the simulator to calculate the cost function as in table 1.</td>
</tr>
<tr>
<td>3- Repeat PSO procedure for m step. If satisfied go to 7</td>
</tr>
<tr>
<td>4- Increment the number of control points for input variable through linear interpolation.</td>
</tr>
<tr>
<td>5- Adjust problem size accordingly.</td>
</tr>
<tr>
<td>6- Go to step (2) until stop criterion is reached.</td>
</tr>
<tr>
<td>7- Stop.</td>
</tr>
</tbody>
</table>

The procedure for incrementing the variables is simple. The number of variables starts with two variables and adding additional one in between it became three variables then five, nine, seventeen, and so on as shown in table 3.

<table>
<thead>
<tr>
<th>Table 3 Variables distribution over time span</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two variables *</td>
</tr>
<tr>
<td>Three variables *</td>
</tr>
</tbody>
</table>
| Five variables *                           *
| Nine variables *                           *
| ←Time-----------------Span→ |

4 Test Cases

Three test cases will be shown in this section, one with no obstacles and the second one with one obstacle and the third with two obstacles.

4.1 No obstacle trajectory

In this case, the rover is required to move from point (0, 0) to point (10, 10). The speed is fixed to 1 m/s and the optimization problem is set for heading angular speed. The problem starts with two variables and the minimum is satisfied with 17. It is clear that optimal trajectory is to make a single turn to an angle 45 deg. then straight to the final destination. The following figures show the trajectory and the command for heading angular speed. The heading angle rate command has 1.57 in the first element and then zeros. So we have this triangle shape rate command which when integrated, it will give the required 45 deg. turn. This test case shows that the optimization technique is working correctly.
4.2 Circular obstacle trajectory

In this case, the rover is required to move from point (0, 0) to point (10, 10) with a circular obstacle with radius 3 m in its way. The translational speed is fixed to 1 m/s and the optimization problem is set for heading angular speed. The problem starts with two variable and incremented to three to reach the goal. The circular constraint is defined as:

If $\sqrt{(x-4)^2 + (y-4)^2} < 9$ then set $F_3$ to a non-zero value in the cost function ($F_3 = 50$ in this case)

The following figures show the trajectory and the command for heading angular speed.

4.3 Two Circular obstacles trajectory

In this case, the rover is required to move from point (0, 0) to point (10, 10) with two circular obstacles with radii 3 m and 1 m respectively, in its way. The translational speed is fixed to 1 m/s and the optimization problem is set for heading angular speed. The problem starts with two variable and incremented to five to reach the goal. The circular constraints are defined as:

If $\sqrt{(x-2)^2 + (y-1)^2} < 1$ then set $F_{31}$ to a non-zero value in the cost function ($F_{31} = 50$ in this case)

If $\sqrt{(x-7)^2 + (y-7)^2} < 4$ then set $F_{32}$ to a non-zero value in the cost function ($F_{32} = 50$ in this case)
Where $F_3 = F_{31} + F_{32}$

The following figures show the trajectory and the command for heading angular speed.

1. Run IPSO starting with 2 variables then 3, 5, 9, and 17 as shown in table 3 with 20 iterations at each stage so the total number of iterations will sum up to 100 and record the convergence curve.

2. Run PSO with 17 variables 100 iterations and record the convergence curve.

3. Run steps 1 and 2 100 times for each case and plot the average convergence curve.

The incremented case showed better convergence performance in reaching the optimal value. Figures 14, 15, and 16 show average convergence-curves for the two optimization methods for the three test cases.
5 Conclusions

A new method for off-line optimal trajectory planning for differential-drive rover through simulation is proposed. The method combines the global characteristics of particle swarm optimization with better convergence characteristics by incrementing the number of variables. It also uses a special cost function based on time and distance to final destination. The method is tested on two-dimensional rover trajectories and the optimized solution showed fast convergence and good obstacle avoidance with satisfaction of the imposed constraints.

6 References


