The N-Robot Formation and Path Tracking Procedure

Ben Cannon, *Eric Wallace*, and *Omar Abdelkader*

***Abstract*—The main purpose of this technical report is to describe an optimal solution to the n-robot formation and path tracking procedure problem. In this problem, *n* randomly placed identical autonomous robots are tasked with locating one another, lining up in formation, and moving along a path following the "leader", or first robot.**

***Index Terms*—Dubins car, pure pursuit, differential drive, autonomous navigation, target pose.**

I. Introduction

Consider a room filled with N differential drive robots [2]. We will be required to design a communication system and control algorithm that causes the robots to form a line at an arbitrary point in the room, and then follow an arbitrary path.

The following constraints are placed on our robot formation system:

1) The robots are not assigned a trivial ordering
   - There is no physical labeling of the cars
   - The robots' ordering cannot be hardcoded into their firmware
   - No determination of order can be made by the serial number or any other physical attribute
2) No robot is trivially assigned as leader
   - Each robot has equal rights or priority to be the leader
3) All the robots communicate with each other via an exclusive network where they can share information (i.e. position and heading) with one another

In brief, this problem is comparable to parents asking their identical children with the same name to form a line and follow a trajectory.

A. Dubins Car

We will model our robots after a Dubins car. These cars have two wheels on either side of the car body, with a simple motor powering each. In our system, we will be ignoring any collisions amongst cars. We will also consider a room with no obstacles inside.

Fig:1 Dubins Car

By driving the motors at the same angular velocity, we can propel the car forward. By driving the motors at equal but opposite angular velocities, we can rotate the car in place. Combining these two forms of motion provides the robot with a third form of motion, traveling in a curved line or path.

B. State Space

The only factor that differentiates one robot from another is the robot's state. Each robot is defined by its position (x, y) and its heading angle (Θ). Taken together, we call the state vector $(x\,(t)\,, y\,(t)\,, \Theta\,(t))$ the robot's "pose" at time t.

If we define U_L and U_R to be the wheel speeds of the right and left wheels, respectively, then the robot's equations of motion are governed by:

$$\dot{x} = \tfrac{r}{2}\,(U_L + U_R)\cos(\Theta)$$
$$\dot{y} = \tfrac{r}{2}\,(U_L + U_R)\sin(\Theta)$$
$$\dot{\Theta} = \tfrac{r}{L}\,(U_L - U_R)$$

Here r is the radius of the wheels and L is the distance between the wheels. The wheel speeds U_L and U_R are the "controls", or inputs to the system, that allow us to maneuver the robot [3].

It is important to note that the differential robot is *completely controllable*, meaning that any final pose can be reached from any initial pose. Upon further inspection, this is immediately clear. Although it would not always be the most efficient means of travel, a robot can travel between any two configurations by 1) rotating itself to the goal position; 2) translating to the goal position; 3) and rotating itself to the desired final orientation.

II. Navigation Algorithm

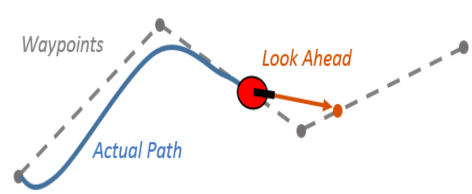

Fig:2 Lookahead Method [4]

The most obvious means of navigation between two points would be to implement the "turn and shoot" method. In this procedure, the robot orients itself so that its heading is towards the goal point and drives forwards until it reaches that coordinate. This process can be repeated as many times as necessary in order to follow a set of waypoints or path. While easy to implement in a computational software package like MATLAB, this path tracking technique requires frequent stopping and starting. As the car's heading begins to divert from its goal, the car will stop proceeding forward and begin a rotation. This can cause an "inchworm" type movement, especially when the car is turning on a sharp angle.

The adaptable pure pursuit approach solves many of the problems presented by the "turn and shoot" technique. Although there are several algorithms that may be better suited for the problem being presented, the adaptable pure pursuit approach is one of the more efficient algorithms for its level of implementation complexity.

In this algorithm, the robot looks ahead along its current trajectory and identifies a point one look-ahead distance, *D*, away from its current position. This look-ahead distance is adapted based on the robots current distance from the goal point and turning angle [1]. For example, when the robot is close to its next point, it will calculate look ahead points that are extremely close to its current position. The robot then determines its control laws (the left and rights right motor) and follows an arced path based on its angle and distance away from the look-ahead point. The following code snippet describes the methodology for determining the left and right arc drive control laws; where *rNear* is the interior wheel radius of the turn and *rFar* is the exterior wheel radius of the turn.

```
if(strcmp(direction, 'left')) % turn left
   UR = veloctiy;
   UL = (rNear/rFar)*UR;
elseif(strcmp(direction, 'right')) % turn
    right
   UL = velocity;
   UR = (rNear/rFar)*UL;
end
```

One of the issues with pure pursuit is when the robot is facing the opposite heading of where it needs to travel to. In this case, the robot will attempt to follow a extremely long, sweeping pattern to circle around to its target heading. Moreover, pure pursuit is intended to be constantly looking ahead to a goal point, and thus causes inconsistent behavior when approaching the end of a path. In order to remedy this, we formed a combination of the "turn and shoot" and adaptive look ahead pure pursuit algorithm.

We first initiate "Orient Mode", where the robot simply rotates to face the correct heading towards its first goal point. The robot then continually operates the pure pursuit algorithm to track and follow the path until it approaches the end of the path. It finally enters "Finish Mode", where the robot uses "turn and shoot" to reach its final point with great accuracy.

III. Formation Algorithm

The first major task we are faced with is the forming of a line. We will have the robots form their initial line starting at the beginning of the path needed to be followed.In order to form the line, we first let each robot calculate its respective distance from the beginning of the path. Each robot then determines its order in the line based on how far it is from the initial point. For example, the robot that is closest to the start of the path will be assigned the first position in the line. The robot who is the second closest to the initial point will be assigned to the second position in the line.

The robots will then create a line at the start of the path. The robot who is in position one will travel to the start of the path. We then compute the slope of the line connecting the first waypoints along the path. The remaining robots will position themselves on 1 meter intervals along that tangential vector. This will allow the robots to efficiently begin tracking the target path and the arbitrary lead robot.

IV. Path Tracking and Robot Following

Once each robot is in position, a signal flag will be raised to indicate that the robots have successfully gotten into position. The first robot, who is assigned to be the "leader", will receive that flag and begin to follow the described adaptive look-ahead pure pursuit algorithm and track the path using the described control system.

All other robots will be assigned to be "followers". These robots, rather than following the given path, will be tracking the position of the robot ahead of it. These robots will also follow the adaptive look ahead pure pursuit algorithm, except using the last known position of the robot in front of it as its goal points. In this manner, each robot is tracking the robot directly in front of it.

V. The Simulation

We used MATLAB to create a simulation of the described problems. We will consider five different scenarios/problems in our simulation.

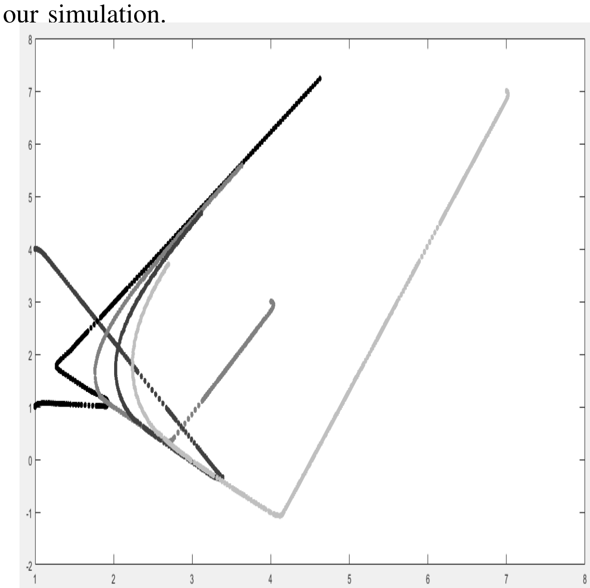

Fig:3 Simulation

The picture above shows an active screen shot of the simulation. The robots trajectories are mapped on an x,y grid. The adaptive pure pursuit look-ahead algorithm is used to track the desired path.

The code used to generate this simulation is available on GitHub at the following link https://github.com/omikader/robot-formation

1) An open loop control system. In this scenario, the robot won't use the pure pursuit algorithm, it will simply demonstrate different driving mechanics. The robot drives straight, makes a turn, and then continues straight on a different trajectory.
2) Navigation to a target pose. In this scenario, the robot must maneuver to a target point and orient to a target heading. We use the adaptive look ahead pure pursuit algorithm to navigate the robot to the point.
3) An arbitrary target path. In this scenario, the robot will following the described navigation algorithm to reach

the start of the path, track the path's trajectory, and position itself on the final point.

4) Two robots that follow an arbitrary target path. In this scenario, the robots will determine their line position and navigate to their respective positions in the line. The ”leader” robot will then track the path trajectory, while the ”following” robot will track the position of the leader.
5) N robots follow an arbitrary target path. The robots will calculate their order, create the line formation, and assign a leader and followers. The leader will then track the path, while each follower is tracking the robot directly in front of it.

VI. Conclusions and Suggestions for Further Research

In this procedure, we provide a method for a group of N robots to get into a single file line formation and track a path trajectory as a group. We have presented a method to track path's using a combination of the ”turn and shoot” and the adaptive look ahead pure pursuit algorithms. Furthermore, we have described a method for the robots to determine their relative position in line, and calculate their respective initial starting position. Using a combination of intercommunication and control systems, we were able to create a complex simulation of N-robots organizing themselves and tracking a leader's trajectory.

In this paper, we have ignored any collisions that could have occurred amongst robots. We also have ignored any obstacles and assumed robots are free to move over any area in the room. Creating an improved simulation that takes advantage of these factors will provide more realistic applications for the described problems. Other following algorithms could be implemented that use a PID based controller to more accurately track the leading robot.

Acknowledgment

We would like to thank Dr. Blankenship for the guidance throughout the semester in our Signals and Systems Course.

VII. References

[1] Giesbrecht, J., D. Mackay, J. Collier, and S. Verret. ”Path Tracking for Unmanned Ground Vehicle Navigation.” Defence Research and Development Canada, December 2005.

[2] Lavelle, Steven M. ”A Differential Drive Robot.” Algorithms. Accessed December 12, 2016. http://planning.cs.uiuc.edu/node659.html.

[3] Lundgren, Martin. ”Path Tracking for a Miniature Robot.” Master's thesis, Umea University, 2003. Accessed December 12, 2016. http://www8.cs.umu.se/kurser/TDBD17/VT06/utdelat/Assignment Papers/Path Tracking for a Miniature Robot.pdf.

[4] Mathworks. ”Pure Pursuit Controller.” Pure Pursuit Controller - MATLAB & Simulink. Accessed December 12, 2016. https://www.mathworks.com/help/robotics/ug/pure-pursuit-controller.html.