

MOTION CONTROL FOR INTELLIGENT GROUND VEHICLES BASED ON  
THE SELECTION OF PATHS USING FUZZY INFERENCE

by

Shiwei Wang

A Thesis  
Submitted to the Faculty  
of the  
WORCESTER POLYTECHNIC INSTITUTE  
in partial fulfillment of the requirements for the  
Degree of Master of Science  
in  
Electrical and Computer Engineering  
by

---

May 2014

APPROVED:

---

Professor Taskin Padir, Major Advisor

---

Professor Xinming Huang

---

Professor Fred J. Loofth III

## **Abstract**

In this paper I describe a motion planning technique for intelligent ground vehicles. The technique is an implementation of a path selection algorithm based on fuzzy inference. The approach extends on the motion planning algorithm known as driving with tentacles. The selection of the tentacle (a drivable path) to follow relies on the calculation of a weighted cost function for each tentacle in the current speed set, and depends on variables such as the distance to the desired position, speed, and the closeness of a tentacle to any obstacles. A Matlab simulation and the practical implementation of the fuzzy inference rule on a Clearpath Husky robot within the Robot Operating System (ROS) framework are provided.

## Acknowledgements

Studying at Worcester Polytechnic Institute for the past three years has been an unforgettable experience. Now, it is time to bring my studies to a conclusion.

First of all, I would like to thank my family members for their unconditional support, both financially and emotionally throughout my degree. In particular, the love and understanding shown by my father, mother and my brother is greatly appreciated.

Furthermore, I am deeply grateful and sincerely thankful for my advisor Professor. Taskin Padir, whose guidance, patience, and support throughout my studies have enabled me to understand and develop my thesis project.

I would also like to show gratitude to my committee, Professor Xinming Huang and Professor Fred J. Looft III. I deeply appreciate their questions, and suggestions.

In addition, I would like to thank all of my friends who helped me during these past three years. Thank you.

# Contents

|  |             |
|--|-------------|
| <b>List of Figures</b>   | <b>vi</b>   |
| <b>List of Tables</b>  | <b>viii</b> |
| <b>List of Symbols</b>   | <b>ix</b>   |
| <b>1 Introduction</b>  | <b>1</b>    |
| <b>2 Problem Statement</b>                                       | <b>4</b>    |
| 2.1 The Combination Method . . . . .                             | 4           |
| 2.2 The Simulation of Tentacle Path in Set 0 . . . . .           | 4           |
| 2.3 The Experiment Robot . . . . .                               | 5           |
| <b>3 Robot Kinematic Model</b>                                   | <b>8</b>    |
| 3.1 Trajectory Plotting Calculation and Steering Angle . . . . . | 8           |
| 3.2 The Robot State . . . . .                                    | 11          |
| 3.3 Obstacle and Destination Calculation . . . . .               | 12          |
| 3.4 The Vehicle Speed . . . . .                                  | 13          |
| <b>4 Selection of Path using Fuzzy Inference</b>                 | <b>14</b>   |
| 4.1 Three Factors of Tentacle Selection . . . . .                | 14          |
| 4.2 Transformation Functions . . . . .                           | 16          |
| 4.2.1 Distance Transformation Function . . . . .                 | 16          |
| 4.2.2 Closeness Transformation Function . . . . .                | 17          |
| 4.3 Fuzzy Control . . . . .                                      | 18          |
| 4.3.1 Fuzzy Control Rules . . . . .                              | 18          |
| 4.3.2 The Sequence of Membership Functions . . . . .             | 19          |
| <b>5 Obstacle Avoidance Simulation Design</b>                    | <b>21</b>   |
| 5.1 Pre-Selection . . . . .                                      | 21          |
| 5.1.1 Pre-Selection In Road Map . . . . .                        | 22          |
| 5.1.2 Pre-Selection In Obstacle Map . . . . .                    | 22          |
| 5.2 Trajectory Selection . . . . .                               | 23          |
| 5.3 Matlab Function Structure Chart . . . . .                    | 25          |

|          |   |           |
|----------|---|-----------|
| <b>6</b> | <b>Trajectory Simulation Results and Analysis</b> | <b>28</b> |
| 6.1      | Invalid Destination . . . . .                     | 28        |
| 6.2      | Close Destination . . . . .                       | 29        |
| 6.3      | Distant Destination . . . . .                     | 29        |
| 6.4      | Simulation Analysis in Different Speed . . . . .  | 31        |
| 6.5      | Simulation With Time Scale . . . . .              | 32        |
| <b>7</b> | <b>Experimental Results</b>                       | <b>36</b> |
| <b>8</b> | <b>Conclusions</b>                                | <b>38</b> |
|          | <b>Bibliography</b>                               | <b>40</b> |

# List of Figures

|     |   |    |
|-----|---|----|
| 2.1 | The simulation of one tentacle set . . . . .  | 5  |
| 2.2 | Clearpath Husky Robot . . . . .   | 6  |
| 3.1 | The steering angle of one tentacle . . . . .  | 10 |
| 3.2 | Robot steering angle and position . . . . .   | 11 |
| 4.1 | Pseudo code of the one cycle . . . . .  | 15 |
| 4.2 | The tentacles with close obstacle . . . . .   | 16 |
| 4.3 | The set of tentacles with distant obstacles covered . . . . .   | 17 |
| 4.4 | Fuzzy rule cube sketch diagram . . . . .  | 19 |
| 4.5 | The sequence of MFs and output according to the factors . . . . .   | 20 |
| 5.1 | Pre-selection method simulation in road map . . . . .   | 22 |
| 5.2 | Pre-selection method simulation in obstacle map . . . . .   | 23 |
| 5.3 | Tentacle selection method schematic diagram . . . . .   | 24 |
| 5.4 | The fuzzy rule editor, it showing 343 fuzzy rules of the complete fuzzy controller  | 25 |
| 5.5 | 3D surface representation of the velocity and distance to goal axis of fuzzy selection cube . . . . .                       | 26 |
| 5.6 | 3D surface representation of the closeness to obstacles axis vs. distance to goal axis of the fuzzy selection cube. . . . . | 26 |
| 5.7 | 3D surface representation of the closeness to obstacles axis vs. velocity axis of the fuzzy selection cube. . . . .         | 27 |
| 5.8 | Function flow chart of the whole simulation . . . . .   | 27 |
| 6.1 | Invalid destination . . . . .   | 29 |
| 6.2 | The command window in the situation that the destination close to the start position . . . . .                              | 29 |
| 6.3 | The command window of no available path . . . . .   | 30 |
| 6.4 | Schematic plan of trajectory selection in speed set 1 . . . . .   | 30 |
| 6.5 | Schematic plan of trajectory selection with random obstacles map, j=2,4,6,8   | 31 |
| 6.6 | Path selection in one map . . . . .   | 34 |
| 6.7 | Schematic plan of trajectory selection in speed set 0 with time scale . . . . .   | 35 |
| 7.1 | Output from Local Planner node in rviz . . . . .  | 37 |

7.2 The output from Polly . . . . . 37

## List of Tables

|     |   |    |
|-----|---|----|
| 2.1 | The technical specifications of Polly . . . . .               | 7  |
| 3.1 | The table of speed set value . . . . .                        | 13 |
| 6.1 | The table of duration comparison in different speed . . . . . | 32 |



# List of Symbols

| Item           | Description   |
|----------------|---|
| $j$            | The tentacle set number   |
| $k$            | The tentacle's number   |
| $rk$           | The radius of the kth tentacle in set j                                   |
| $lk$           | The length of the kth tentacle in set j                                   |
| $R$            | The resolution of the kth tentacle in set j                               |
| $os$           | The robot's steering angle of the previous state                          |
| $ox$           | The robot's x coordinate of the previous state                            |
| $oy$           | The robot's y coordinate of the previous state                            |
| $x$            | The x coordinate of the tentacle's plotting points                        |
| $y$            | The y coordinate of the tentacle's plotting points                        |
| $os'$          | The robot's steering angle of the current state                           |
| $ox'$          | The robot's x coordinate of the current state                             |
| $oy'$          | The robot's y coordinate of the current state                             |
| $pns$          | A 81 by 1 matrix, the distance from each endpoint to the desired position |
| $tns$          | A 81 by 1 matrix, the minimum distance from the tentacle to the obstacle  |
| $desiredx$     | The x coordinate of the destination position                              |
| $desiredy$     | The y coordinate of the destination position                              |
| $obx$          | The x coordinate of the obstacle position                                 |
| $oby$          | The y coordinate of the obstacle position                                 |
| $tenavail$     | A 81 by 1 matrix, the pre-selection output based on the tns value         |
| $distance\_m$  | The matrix pns  |
| $temp\_d$      | The reciprocal value of distance_m multiplied by a large number           |
| $temp\_dv$     | The maximum temp_d  |
| $temp\_vm$     | The value of temp_d divides by temp_dv                                    |
| $dist\_mtx$    | The output matrix of the distance factor                                  |
| $close\_mtx$   | The output matrix of the closeness factor                                 |
| $GoldFunction$ | A Broken Line Model   |
| $MFs$          | Membership functions  |
| $\theta$       | The steering angle of each plotting point                                 |

# Chapter 1

## Introduction

With recent advances in research and enabling technologies, intelligent robots are increasingly being fielded in applications which require the robot to navigate autonomously in unstructured environments. As a result, motion planning for nonholonomic wheeled vehicles, operating in time-varying dynamic environments, has been the focus of extensive research efforts. The overarching goal is to advance the capabilities of unmanned systems and enable their adoption in applications including defense, search and rescue, exploration, and transportation.

One approach is to use a probabilistic road map method in which the graph and map are constructed with sampling techniques [1], [2]. Another method involves implementing two rapidly-exploring random trees to configure the destination [3]. The authors of [4] describes a ripple tentacle algorithm, it is focus on the weighted coefficients [4]. Alternatively, implementations have been demonstrated using the mathematical properties and functions, for instance, one approach proposes the mathematical model with a remotely controlled mobile robot system and its neural network [5]. Also, the knowledge of calculus, integrated navigation system, and Kalman filter algorithms have been used [6]. The common characteristic of the various methods is the mathematical model needed. However, the complexity of the environment and the functional limits of a robot make path planning difficult. The unstructured environments present planning difficulties primarily because of the complexity of object identification, something which comes easily to a human. Hence, in order to

implement an obstacle avoidance function on a robot, scientists rely on several advanced sensor system, including: global position system (GPS) [7], radar technology [8], infrared detector [9] and many other kinds of sensors, such as sonar [10], wide angle sonar [11], or combinations of sensors, which are called sensor networks [12]. Although computing power has increased at an exponential rate, mathematical models of systems are still difficult to solve in real-time. To imitate a simple human eye's object recognition process, a robot needs lots of computing power working with a complex algorithm and calculation procedure. Intelligent robots still has a long way to go before they will be able to achieve satisfactory performance in dynamic environments. We need a simple and repeatable method of avoiding obstacles urgently.

There is one path planning approach that use driving with tentacles method which simplifies the trajectory analysis intuitive and easy. Path selection criteria is represented in a form that is easy to represent in a computer model [13]. Another method uses fuzzy logic to analyze ground smoothness [14]. The global traversability map is represented by regions of different traversability indices by using fuzzy-logic constructs which simplifies the calculation process. The method uses multiple layers of control laws in which three factors are combined together and generate a final result for the robot [15]. One similar approach is behavior-based, neuro-fuzzy controller for mobile robot navigation which is a neuro-fuzzy controller for robot navigation with a sensor implemented [16]. Inspired by these methods, our research presents a path planning and obstacle avoidance algorithm which combines the method known as driving with tentacles, with a selection rule based on fuzzy inference. In our approach, we utilize a fuzzy inference based selection algorithm to decide on the path to be followed by the autonomous vehicle. The algorithm takes the parameters of 81 tentacles (drivable paths in the form of circular arcs emanating from robots center of turning) in each of the 16 speed sets corresponding to the vehicle speeds in the range of its minimum and maximum speeds. This generates the tentacle selection as its output. Each tentacle is implemented as a structure whose members are the identifier and range as well as the membership function, and the parameters such as length and radius. In contrast to the traditional implementation of the selection which is based on the calculation of a weighted cost function, in this research we formulate the mapping between the inputs and outputs

using fuzzy logic without the need for a mathematical model. In this way, the need for adjustment of the weights in the cost function in different environments is avoided.

The paper is organized as follows: In Section II, we will describe the path planning problem in detail. Section III shows the robot kinematic model and formula calculation, section IV covers the selection of paths using fuzzy inference. Section V we discuss the simulation design, and Section VI is the simulation results and analysis. Section VII provides an analysis of the performance of the implementation on a Clearpath Husky ground vehicle. Finally, Section VIII is the conclusion.

## Chapter 2

# Problem Statement

Our research is concerned with path planning for autonomous ground vehicles. Some methods use mathematical methods which may cause an increase of calculations. The workload of the computer will hold-up the calculation speed so the running vehicle will receive data far on in time. Hence, the completed, refined and succinct mathematical computing method is needed if we want to pursuit a good path planning system.

### 2.1 The Combination Method

We propose the idea of combining the tentacle model and fuzzy logic together to select the trajectory. The combination method achieves the trajectory planning without using complex mathematical modules and uses a simplified path planning procedure. The new method increased the robot computing speed. This allows for it to be followed smoothly and accurately without the need of additional computational considerations. The powerful, efficient dodging system can be widely applied to any robot with Robot Operating System (ROS) as its software framework, which needs to implement obstacle avoidance functionality.

### 2.2 The Simulation of Tentacle Path in Set 0

In the method 'Driving with tentacles', the paths (or called tentacles) are split into 16 sets of 81 arcs. Each arc represents a possible vehicle trajectory and the length of each

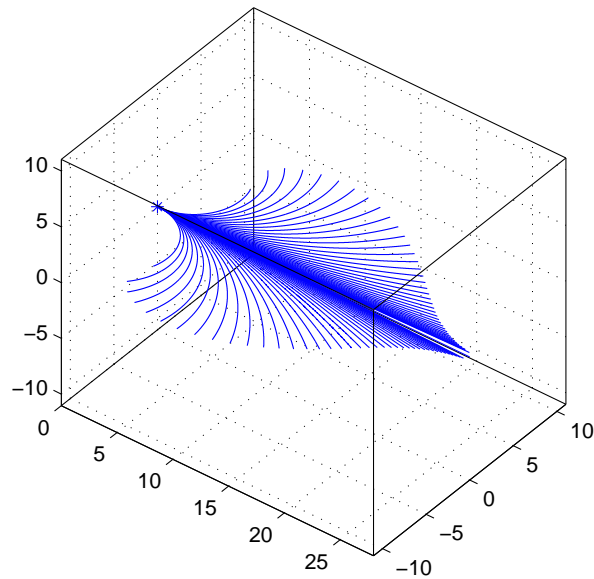


Figure 2.1: The simulation of one tentacle set

arc varies depending on the tentacle number  $k$  and the tentacle set number  $j$ . In Figure 2.1 shows the simulation of tentacle set 0, it is the sketch of the set number 0 of paths of the robot, the small star stands for the beginning point of the path set, which is the robot's position. We use two formulas presented in 'Driving with tentacles' to create the mathematical representation of the tentacles [13]. One formula calculated the length of the tentacle path  $lk$ , the other one calculated the radius of the path  $rk$ . The path length and the radius are used to calculate the distance from the vehicle to the destination or the closeness from the vehicle to the obstacle.

### 2.3 The Experiment Robot

In order to test the effectiveness of the proposed method, the algorithm has been implemented on a Husky platform (named Polly) by Clearpath Robotics. The Figure 2.2 shows the experiment robot Polly used as a testbed for this research with the ladar implemented, which will detect the obstacles. Polly is a six-wheeled vehicle compatible with the



Figure 2.2: Clearpath Husky Robot

Robot Operating System (ROS). Our testing implementations used on the ground vehicle are implemented in standard ROS form, enabling portability of the algorithm to other robot platforms. The system specifications of our experiment robot (Husky A100) are shown in Table 2.1.

Table 2.1: The technical specifications of Polly

| Item           | Value              | Units |
|----------------|--------------------|-------|
| L x W x H      | 860x600x340        | mm    |
| Clearance      | 85                 | mm    |
| Weight         | 35                 | kg    |
| Max. payload   | 40                 | kg    |
| Power          | 600                | W     |
| Max. Speed     | 1.5                | m/s   |
| Kinematics     | Differential drive | N/A   |
| Operating Time | 2                  | h     |
| Batteries      | 12,44              | V, Ah |



## Chapter 3

# Robot Kinematic Model

In this chapter, we mainly discuss about the robot kinematic model and its mathematical representations. The kinematic model represents the motion of the vehicle as the following parameters: the robot's state, position, steering angle, the distance between the vehicle and the obstacles/destination, and velocity. Note that the kinematic model has the following properties:

- This is a two dimensional (2-D) model.
- The forces that effect the motion (such as friction, air resistance) are not taken into consideration.
- It deals with the geometric model of the system.
- The model analyzes the relationship between the control parameters and the state space model ([17], [18]).

### 3.1 Trajectory Plotting Calculation and Steering Angle

The method driving with tentacles details the geometry of the tentacles which are circular arcs emanating from the robot's center of turning used as we mentioned in the previous chapter, including the tentacle set number  $j$  with range from 0 to 15, the tentacle's number  $k$  with the range from 0 to 80, the radius of each tentacle  $rk$ , the length of each tentacle  $lk$ .

The equation 3.1 calculates the radius  $rk$  of the  $k$ th tentacle in set  $j$ . Equation 3.2 lists the length of the  $k$ th tentacle in set  $j$  [13].

$$rk = \begin{cases} (1.15)^k \times \left( \frac{8+33.5 \times (\frac{j}{15})^{1.2}}{1.2 \times (\frac{\pi}{2}) \times (1.01 - (\frac{j}{15})^{0.9})} \right) & \text{if } 0 \leq k \leq 39 \\ (1.15)^{(k-40)} \times \left( \frac{8+33.5 \times (\frac{j}{15})^{1.2}}{1.2 \times (\frac{\pi}{2}) \times (1.01 - (\frac{j}{15})^{0.9})} \right) & \text{if } 41 \leq k \leq 80 \\ \infty & \text{if } k = 40 \end{cases} \quad (3.1)$$

$$lk = \begin{cases} (8 + 33.5 \times (\frac{j}{15})^{1.2}) + 20 \times (\frac{k}{40})^{0.5} & \text{if } 0 \leq k \leq 39 \\ (8 + 33.5 \times (\frac{j}{15})^{1.2}) + 20 \times (\frac{k-40}{40})^{0.5} & \text{if } 41 \leq k \leq 80 \\ 20 + 8 + 33.5 \times (\frac{j}{15})^{1.2} & \text{if } k = 40 \end{cases} \quad (3.2)$$

The tentacle set is symmetric in the design (tentacle from 0 to 39 and 41 to 80) and the symmetric axis is tentacle 40. The resolution of the arc plotting is  $R$  (pixel per meter) which is defined by us and each arc is plotted by a point. We can use the equation 3.3 to calculate the distance between two plotting points.

$$R = \begin{cases} \frac{10000 \times lk}{\pi \times rk} & \text{if } 0 \leq k \leq 80, k \neq 40 \\ \frac{10000 \times lk}{\pi} & \text{if } k = 40 \end{cases} \quad (3.3)$$

The resolution helps us to plot the path which shows as a arc where  $rk$  and  $lk$  are calculated using equations 3.1 and 3.2.

In Figure 3.1, the black arc is a tentacle with steering angle  $\theta$  marked. The steering angle corresponding to each tentacle (shows in Figure 3.1) can be calculated in equation 3.4.

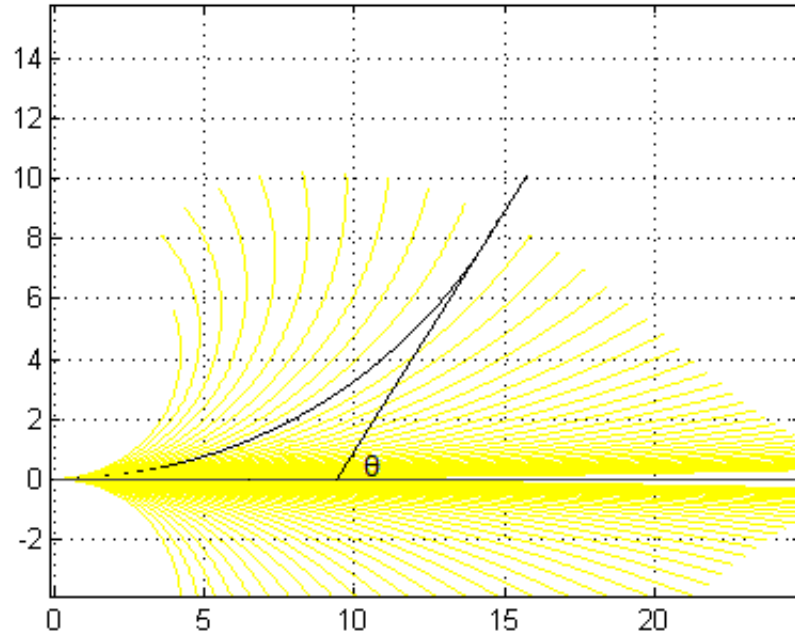


Figure 3.1: The steering angle of one tentacle

$$\theta = \begin{cases} \underbrace{(os - \frac{\pi}{2}), (os - \frac{\pi}{2} + \frac{\pi}{10000}) \dots (\frac{lk}{rk} + os - \frac{\pi}{2})}_{\text{total of } R \text{ factors}} & \text{if } 0 \leq k \leq 39 \\ \underbrace{(os + \frac{\pi}{2} - \frac{lk}{rk}), (os + \frac{\pi}{2} - \frac{lk}{rk} + \frac{\pi}{10000}) \dots (os + \frac{\pi}{2})}_{\text{total of } R \text{ factors}} & \text{if } 41 \leq k \leq 80 \\ \underbrace{0, \frac{\pi}{10000} \dots lk}_{\text{total of } R \text{ factors}} & \text{if } k = 40 \end{cases} \quad (3.4)$$

In equation 3.4,  $\theta$  is the steering angle,  $lk$  is the length,  $rk$  is the tentacle's radius, and  $os$  is the initial steering angle which will be discussed in more detail in section 3.2.

### 3.2 The Robot State

In this section, a simple 2D kinematic model with vehicle steering angle and position are shown in the Figure 3.2. The robot's previous state  $(ox, oy)$  and current state  $(ox', oy')$  are marked. The robot running from position  $(ox, oy)$  to  $(ox', oy')$ .  $os$  is the vehicle's steering angle with the range from  $-\pi/2$  to  $\pi/2$  in previous state. The initial  $os$  equals to 0, shown in the Figure 3.2.

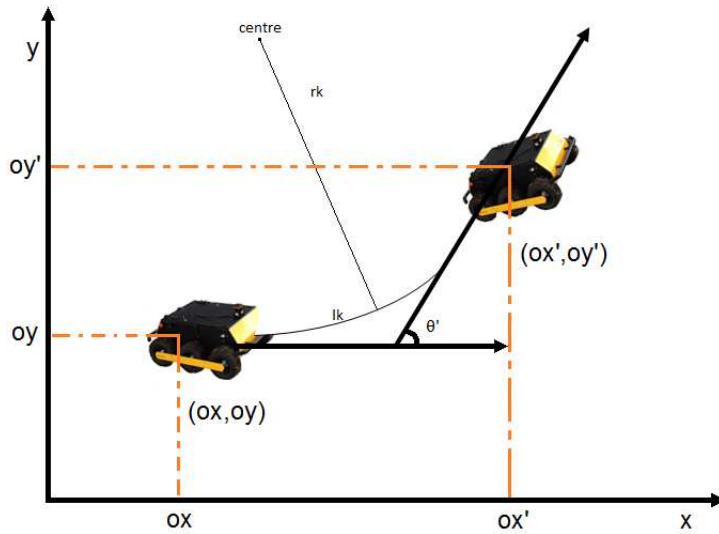


Figure 3.2: Robot steering angle and position

In Figure 3.2, the arc between  $(ox, oy)$  and  $(ox', oy')$  stands for the selected path by using our method. In the next state, the robot's current state will be the previous state, hence the next  $os$  equals to  $os'$ . In the Figure 3.2,  $os'$  equals to  $\theta'$ .

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{cases} \begin{pmatrix} (rk \times \cos \theta - rk \times \sin os) + ox \\ (rk \times \sin \theta + rk \times \cos os) + oy \end{pmatrix} & \text{if } 0 \leq k \leq 39 \\ \begin{pmatrix} (rk \times \cos \theta + rk \times \sin os) + ox \\ (rk \times \sin \theta - rk \times \cos os) + oy \end{pmatrix} & \text{if } 41 \leq k \leq 80 \\ \begin{pmatrix} \theta \times \cos os + ox \\ \theta \times \sin os + oy \end{pmatrix} & \text{if } k = 40 \end{cases} \quad (3.5)$$

The arc's (or called tentacle path) plotting points  $(x, y)$  shown in Figure 3.2 are calculated in equation 3.5.  $ox$ ,  $oy$  and  $os$  are calculated from previous state.  $rk$  and  $lk$  are

calculated in equations 3.1 and 3.2.  $\theta$  is calculated in equation 3.4.

$$\begin{bmatrix} ox' \\ oy' \\ os' \end{bmatrix} = \begin{cases} \begin{bmatrix} rk \times \cos \frac{lk}{rk} + (os - \frac{\pi}{2}) - rk \times \sin os + ox \\ rk \times \sin \frac{lk}{rk} + (os - \frac{\pi}{2}) + rk \times \cos os + oy \\ \frac{lk}{rk} + os \end{bmatrix} & \text{if } 0 \leq k \leq 39 \\ \begin{bmatrix} rk \times \cos -\frac{lk}{rk} + (os + \frac{\pi}{2}) + rk \times \sin os + ox \\ rk \times \sin -\frac{lk}{rk} + (os + \frac{\pi}{2}) - rk \times \cos os + oy \\ -\frac{lk}{rk} + os \end{bmatrix} & \text{if } 41 \leq k \leq 80 \\ \begin{bmatrix} lk \times \cos os + ox \\ lk \times \sin os + oy \\ os \end{bmatrix} & \text{if } k = 40 \end{cases} \quad (3.6)$$

The steering angle of current state  $os'$  and robot's position  $ox'$ ,  $oy'$  are shown in the equation 3.6. Same as 3.5,  $ox$ ,  $oy$  and  $os$  are calculated in last state.  $rk$  and  $lk$  are calculated in equations 3.1 and 3.2.  $\theta$  is calculated in equation 3.4.

### 3.3 Obstacle and Destination Calculation

In this section, we model the distance between the vehicle and the obstacles/destination by the following variables:  $pns$ ,  $tns$  and  $tenavail$ .  $pns$  is the distance from each endpoint to the desired position.  $tns$  is the minimum distance from the tentacle to the obstacle.  $tenavail$  is the pre-selection (please see Section 5.1) output based on the  $tns$  value.

$$pns = \min(((tx - desiredx)^2 + (ty - desiredy)^2)^{0.5}) \quad (3.7)$$

$$tns = \min((((ox + x - obx)^2 + (oy + y - oby)^2)^{0.5}) - obr(n, 1)) \quad (3.8)$$

$$tenavail(m, 1) = \begin{cases} 1 & \text{if } tns(m, 1) > 0 \\ 0 & \text{otherwise} \end{cases} \quad m = 1, 2, 3 \dots 81 \quad (3.9)$$

$pns$ ,  $tns$  are calculated in 3.7, 3.8. The variables  $pns$  and  $tns$  where  $desiredx$  and  $desiredy$  are coordinates of the destination position while  $obx$  and  $oby$  are coordinates of the obstacle position. One thing to note here is that  $tns$ ,  $pns$  are 81 by 1 vectors. If the minimum distance from the tentacle to obstacle (equation 3.8) is a negative value, the tentacle will not be selected in the pre-selection process (details in Section 5.1: Obstacle Avoidance Simulation Design), because it intersected with one or more obstacles.

### 3.4 The Vehicle Speed

From the reference [13], we can calculate the speed of each speed set (shown in Table 3.1). These speeds correspond to forward linear speed of the robot (Polly). In the simulation,  $n$  equals 16, which means 16 sets of tentacles and  $j$  stands for the speed set number name.

Table 3.1: The table of speed set value

| Speed Set Name | Speed(m/s) |        |        |         |
|----------------|------------|--------|--------|---------|
| 0-3            | 0.2500     | 0.6282 | 1.1188 | 1.6633  |
| 4-7            | 2.2460     | 2.8589 | 3.4970 | 4.1567  |
| 8-11           | 4.8357     | 5.5319 | 6.2437 | 6.9700  |
| 12-15          | 7.7095     | 8.4616 | 9.2253 | 10.0000 |

In this chapter, we presented the details of generating a tentacle set which contains a preset number of drivable paths for the robot. This is an essential part to the selection process using fuzzy logic which will be discussed next.

## Chapter 4

# Selection of Path using Fuzzy Inference

In 1965, the American mathematician professor Zadeh put forward the concept of the fuzzy set [19]. His idea 'fuzzy logic' improved the 0, 1 set concept into  $[0,1]$  interval which contains infinite values. This means fuzzy logic is using fuzzy set including infinite continuous values to discuss the fuzziness object. Fuzzy logic control offers a method of dealing with modeling problems by implementing linguistic, non-formally expressed control laws derived from expert knowledge [19] 'IF (process state) THEN (control action)'.

Fuzzy control has been used effectively to develop control methods for systems whose mathematical model is complex or difficult to obtain. It solved our difficulty of tentacle selection method.

In our tentacle selection method, the pseudo code (shows in Figure 4.1) describes the tentacle selection flow.

### 4.1 Three Factors of Tentacle Selection

The underlying goal in tentacle selection is to reach the desired point safely and as quickly as possible. Safely means without crashing, and without repeatedly having large

```

Set available_tentacle_number = 0;
While (the vehicle has not reach the desired position)
{
    if (available_tentacle_number >= 1)
    {
        vehicle is driven by function fuzzy_logic_controller;
    }
    else if (distance <5)
    {
        vehicle stop;
    }
    Update the available_tentacle_number;
}

```

Figure 4.1: Pseudo code of the one cycle

direction changes. Quickly means maintaining the highest velocity while as close to the goal as possible along the safest tentacle possible.

The factors to consider are:

- The distance between the current point of the tentacle and the desired end position: If the other conditions are same, from the available tentacle candidates, the closer to the desired position the better that tentacle's trajectory is. Range: 0 to 0.3.
- Speed: With higher speed (corresponding to a larger speed set index), the vehicle will achieve its final position as soon as possible. There are 16 speed sets name set 0 to set 15. Range: 0 to 0.3
- Closeness to obstacles: This factor considers the distance from a tentacle to an obstacle. If the obstacle is extraordinarily close to the tentacle, as in Figure 4.2, the free tentacle may not afford sufficient space for the vehicle because the robot itself has certain volume. Therefore, tentacles which are further away from obstacles are considered to be better than ones close to obstacles. Range: 0 to 0.4

In Figure 4.2 and 4.3, the coordinates axes are x,y,z respectively, they show the position of the tentacle set. Figure 4.2: the obstacle is too close to the vehicle, Figure 4.3: the



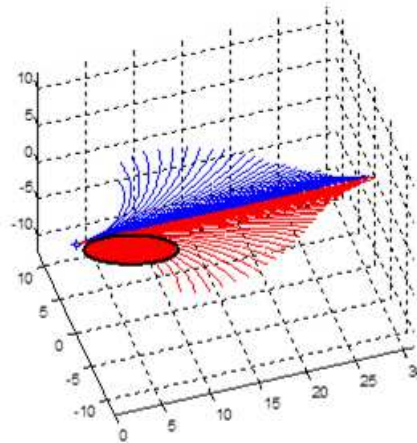


Figure 4.2: The tentacles with close obstacle

obstacles (black circles) are not that close to the vehicle, the path can be selected from several red tentacles.

## 4.2 Transformation Functions

Now we introduce the transformation functions. The functions normalized the vectors' value ( $tns, pns$ ). The reason why we need transformation functions is because the matrices obtained from calculation procedure in the Kinematic Model cannot be implemented into the fuzzy calculation function directly. In our context, we obtained two matrices  $tns, pns$  from the calculation procedure in Kinematic Model and transformed  $pns$  and  $tns$  into range 0 to 0.3 and 0 to 0.4 corresponding to the consider factors Distance and Closeness.

### 4.2.1 Distance Transformation Function

The new matrix  $distance\_m$  equals to  $pns$ , it will be the input parameter of the function. The key method of Distance Transformation Function is to get reciprocal values of each member in  $distance\_m$ . It is easy to understand that the larger distance the less we prefer, hence, get the reciprocal value will make things obvious: the smaller the distance, the larger reciprocal value we get.

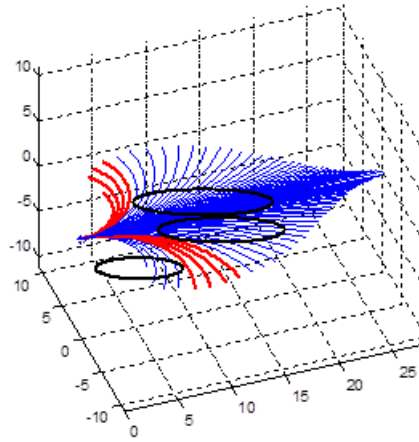


Figure 4.3: The set of tentacles with distant obstacles covered

$$temp_{dv} = \max temp_d \quad (4.1)$$

$$temp_{vm} = \frac{temp_d}{temp_{dv}} \quad (4.2)$$

$$dist_{mtx} = [ten_{avail}] \cdot [0.3 \times temp_{vm}] \quad (4.3)$$

$temp_d$  is the reciprocal value multiplied by 10000. The equations (4.1, 4.2, 4.3) are the calculation detail of the function, the output matrix is  $dist_{mtx}$ .

#### 4.2.2 Closeness Transformation Function

If each value of the distance matrix  $tns$  is larger than 1, the closeness transformation is same as the distance transformation function, otherwise,  $tns$  value needs to be separated into three conditions: less than 0.3, between 0.3 to 0.6 or larger than 0.6.  $close_{mtx}$  is the

output matrix (4.4), Gold Function is a broken line model, when the input is 0.382 times maximum input value, the output gets the maximum value.

$$close\_mtx(m, 1) = \begin{cases} \frac{tns(m,1)}{1000} & \text{if } tns(m, 1) \leq 0.3 \\ \frac{tns(m,1)}{30} & \text{if } 0.3 < tns(m, 1) \leq 0.6 \\ GoldFunction & \text{if } 0.6 < tns(m, 1) \leq 1 \end{cases} \quad m = 1, 2, 3 \dots 81 \quad (4.4)$$

The reason to choose 0.3 as the boundary is the width of robot Polly, our experiment robot, is 0.6 meters, so 0.6 divide by 2 is the length from the center to the edge of the robot, which is 0.3.

### 4.3 Fuzzy Control

Our system model is established in Matlab environment for simulation. In Matlab, the three factors for selecting a tentacle were put into the Fuzzy Inference System(FIS) editor, which returns an output value from 0 to 1. This procedure is called fuzzy control. If the output is 1, the tentacle is optimal and the fuzzy controller will always select the tentacle to be the path. This would only happen in the event of a tentacle that reaches the goal, is in the highest speed set, and is not near any obstacles. In reality the preferability is always a decimal value less than one, and the controller selects the one which is closest to 1. Thus the control law can be described as a 7\*7\*7 3-dimensional cube, shown in Figure 4.4. The three coordinates represent the distance to desired position, the speed of vehicle and the closeness to obstacles.

#### 4.3.1 Fuzzy Control Rules

Three-dimensional fuzzy rule cube has 343 combinations, shown in Figure 4.4. The three coordinates stand for three factors. Each factor has 7 values(PB, PM, PS, Z, NS, NM, NB, details in Section 4.3.1: The Sequence of Membership Functions). These 343 combinations have 7 different output as same as the input values. They are combined to be the fuzzy rule form.

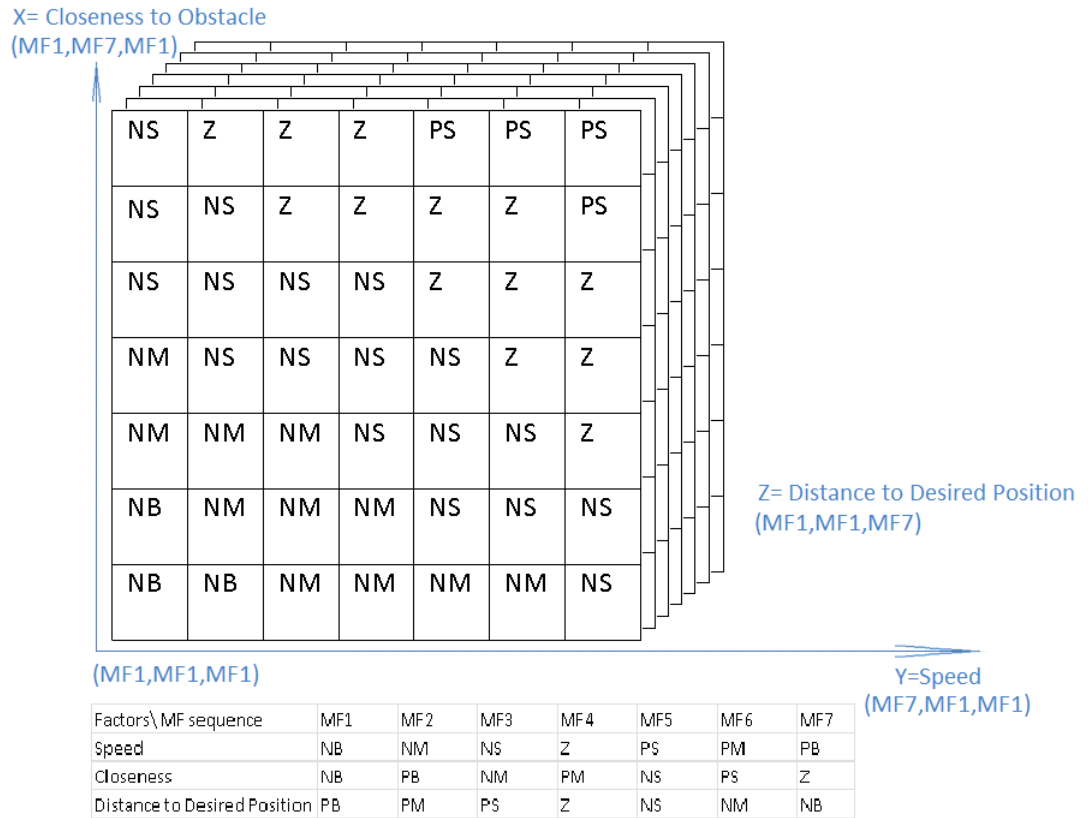


Figure 4.4: Fuzzy rule cube sketch diagram

### 4.3.2 The Sequence of Membership Functions

In the fuzzy rule cube, each factor has seven membership functions (MFs), but the representation of these membership functions is different for each axis. In Fig 4.5, numbers 1 to 7 stand for size sequence in MFs, PB stands for positive big, PM stands for positive middle, PS stands for positive small, Z stands for zero, NS stands for negative small, NM negative middle, NB negative big. In the closeness to goal MF, the shorter the distance from tentacle to the goal position the larger MF bin. In the velocity axis, the larger speed set index we use, the higher the bin. In the closeness to obstacle axis, the further the robot is from an obstacle, the higher the value in the bin is.

| <b>THE SEQUENCE OF MFs AND OUTPUT ACCORDING TO THE FACTORS</b> |    |    |    |   |    |    |    |
|--|----|----|----|---|----|----|----|
| MFs \ Factor   | PB | PM | PS | Z | NS | NM | NB |
| Distance to position   | 1  | 2  | 3  | 4 | 5  | 6  | 7  |
| Speed of vehicle   | 7  | 6  | 5  | 4 | 3  | 2  | 1  |
| Closeness to obstacle  | 2  | 4  | 6  | 7 | 5  | 3  | 1  |
| Output   | 7  | 6  | 5  | 4 | 3  | 2  | 1  |

Figure 4.5: The sequence of MFs and output according to the factors

## Chapter 5

# Obstacle Avoidance Simulation

## Design

After building the algorithm structure, the simulation procedure is ready to start. The selection algorithm can be separated into two parts, the first one is pre-selection and the second one is trajectory selection. By inputting the speed set number (from 0 to 15), the simulation result will show in the plotting diagram.

### 5.1 Pre-Selection

Before running the fuzzy controller selection algorithm, there exist a pre-selection procedure at each update cycle. Pre-selection reduces the computational complexity of the tentacle selection. We should note that the cartographic model will change in different circumstances, hence, two of many representative conditions are selected and listed below: road map and obstacle map. The road map condition means the vehicle needs to follow the road path to avoid the obstacle without touching the edge of the road and all available routes beyond the boundary of the road cannot count as free trajectories. Obstacle map condition means the vehicle needs to avoid those obstacles (yellow circles) showing on the map. The obstacles are the simulation of woodland.

### 5.1.1 Pre-Selection In Road Map

The Figure 5.1 shows the selection detail in road map condition. First, eliminate any tentacles which touched the edge of the road collide with an obstacle. The x coordinate and y coordinate stand for position, the start point in the figure is (2, 10). The green colored arcs are tentacles that pass the pre-selection process; the red arcs are tentacles that are out of the roads edge. The black segment stands for obstacle. The two blue arcs are the boundaries of tentacles that were eliminated by colliding with obstacle arcs.

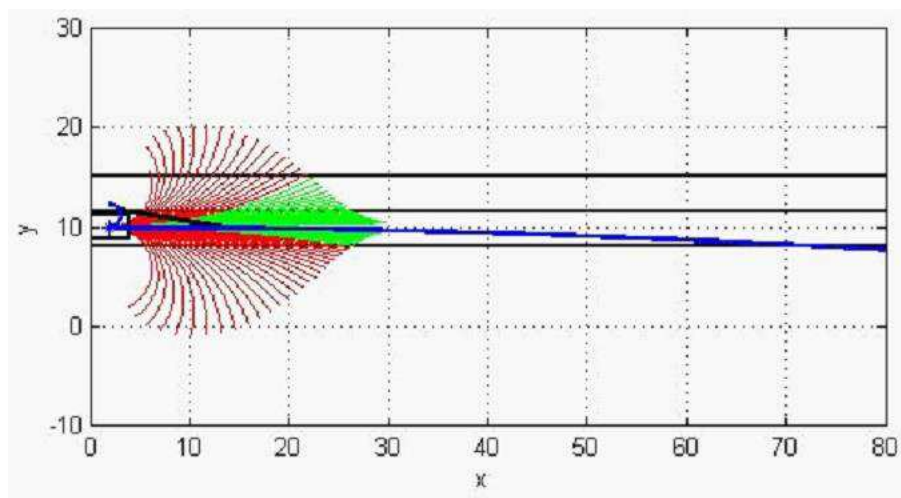


Figure 5.1: Pre-selection method simulation in road map

### 5.1.2 Pre-Selection In Obstacle Map

The Figure 5.2 shows the pre-selection in the obstacle map. The obstacle map is a map with ten random circles which stand for obstacles distributed in defined areas. The radius range of those circles is from 3 to 18 (unit length, meters).

The start point (blue star) in Figure 5.2 is (100, 0), the green colored arcs are available tentacles by pre-selection calculation; the red arcs are not selected by pre-selection because they are touching the obstacle.

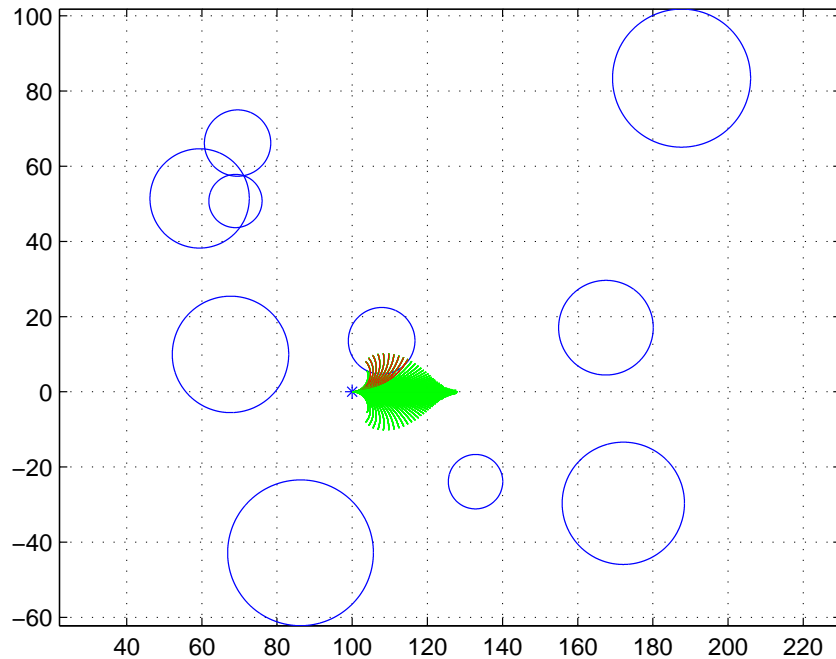


Figure 5.2: Pre-selection method simulation in obstacle map

The distance between the plotting points is  $\pi/10000$ , which is the resolution of the simulation points as we mentioned in the chapter 3. In the path trajectory simulation, the pre-selection method in the task map has been implemented with slight changes of start position (change to  $(0, 0)$ ) and obstacles area. With the implementation of pre-selection, it reduces the calculation of trajectory selection significantly.

## 5.2 Trajectory Selection

The trajectory selection runs after the pre-selection process. We have multiple potentially good path choices for the vehicle to follow as seen in Figure 4.4. We use the fuzzy-controller to select the optimized tentacle from the good tentacle candidates as in Figure 5.3.

The blue tentacle in Figure 5.3 stands for the selected tentacle which is the optimized



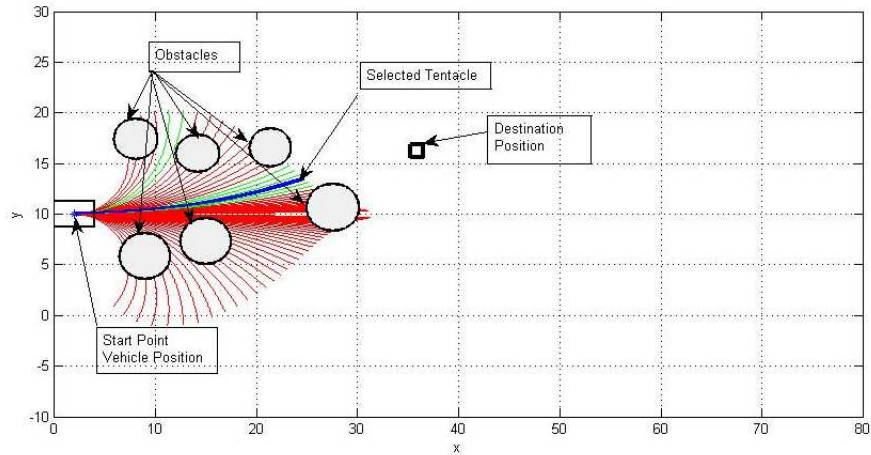


Figure 5.3: Tentacle selection method schematic diagram

tentacle among those available tentacles (green set) based on our method. The black circles represent obstacles. The number of x and y axis are the current position of the tentacle set. There are 343 fuzzy control rules in FIS rule editor as seen in Figure 5.4.

We are using the fuzzy logic toolbox in Matlab and edit those rules into the Rule Editor (Figure 5.4). The rules are following the format:

IF (*process state*) AND (*process state*) AND (*process state*) THEN (*control action*).

After editing the rules, the surface of the whole fuzzy controller can be shown from the FIS (showing in Figure 5.5, Figure 5.6, Figure 5.7).

Each free tentacle will have its own parameters such as velocity, angle, distance to desired position, etc. We apply the MFs to each good tentacle candidate to get the final ranking value for each tentacle. If there is a tie for the largest value, the algorithm will use the closeness to obstacles as the first tie-breaker, the distance to goal as the second tie-breaker, and the velocity as the final tie-breaker. In the unlikely event that all of these values are tied, then it will simply choose the tentacle with the smallest index. The pre-selection tentacle choice followed by fuzzy control selection process is repeated every update cycle until the robot reaches its goal point.

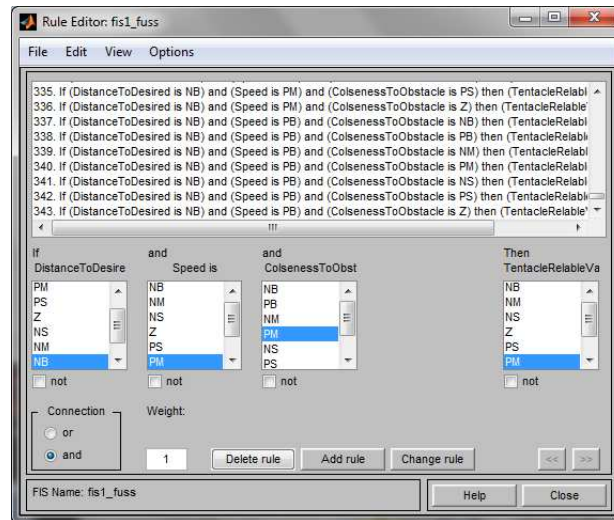


Figure 5.4: The fuzzy rule editor, it showing 343 fuzzy rules of the complete fuzzy controller

### 5.3 Matlab Function Structure Chart

This section mainly talks about the Matlab functions and the framework. Figure 5.8 lists the function flow, our simulation build based on the flow chart. The simulation starts from the obstacle model and Start/End Point, these two functions generate obstacle maps with desired destination. If the destination is valid, the simulation will get into loop. The loop is a big function to find the path selected by our algorithm and judge whether the vehicle will arrive to destination or not. The judgment comes first and basic calculation and selection come after, the figure plotting happens in every loop. In the basic calculation, we will need to input the speed set number, in the fuzzy selection function, we will need to implement and convert three factor matrices into fuzzy calculation function with fuzzy control function. We can use fuzzy logic tool box to create and edit it by simply typing 'fuzzy' in the command window, after saving the fuzzy control function, we need to erase the 'bad' tentacle from the pre-selection result. If the vehicle gets no path to follow, the command window will show need backup warning and the whole simulation exits. Otherwise the vehicle will follow the selected path until the vehicle 'arrive' (close desired position within 5 units of length) to destination.

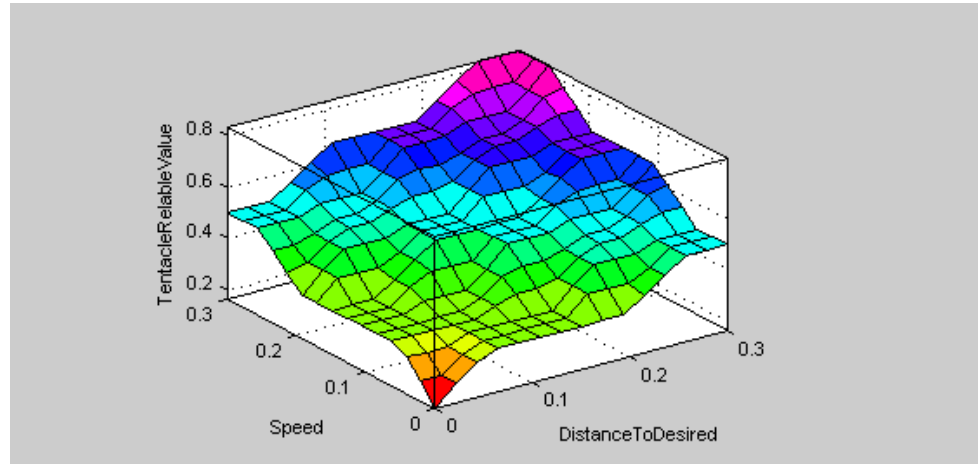


Figure 5.5: 3D surface representation of the velocity and distance to goal axis of fuzzy selection cube

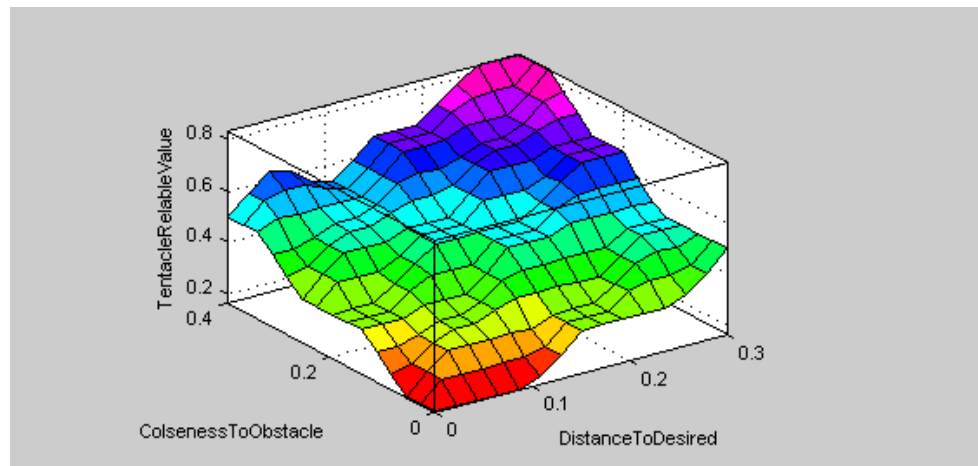


Figure 5.6: 3D surface representation of the closeness to obstacles axis vs. distance to goal axis of the fuzzy selection cube.

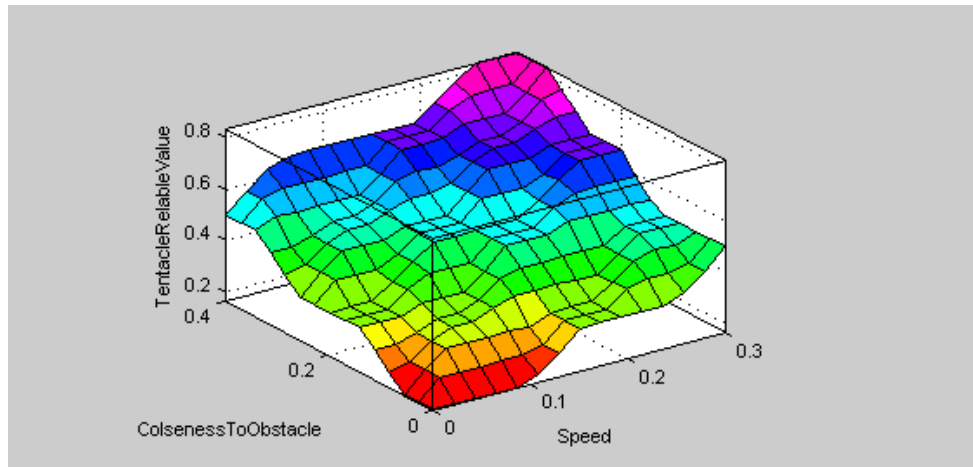


Figure 5.7: 3D surface representation of the closeness to obstacles axis vs. velocity axis of the fuzzy selection cube.

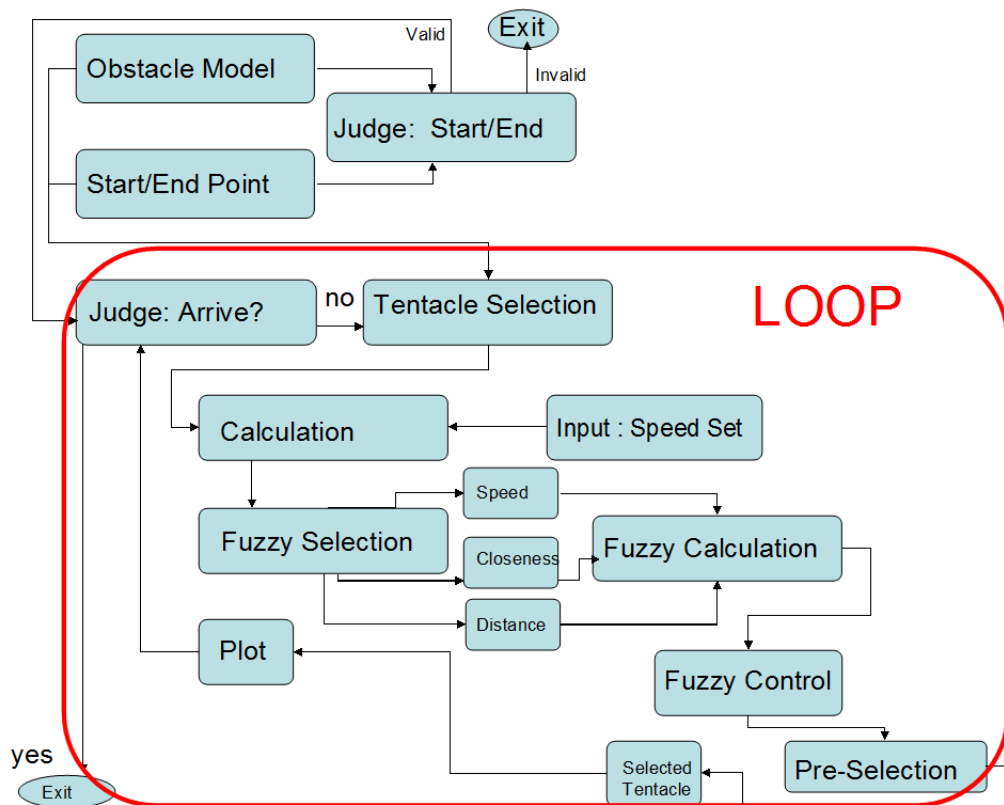


Figure 5.8: Function flow chart of the whole simulation

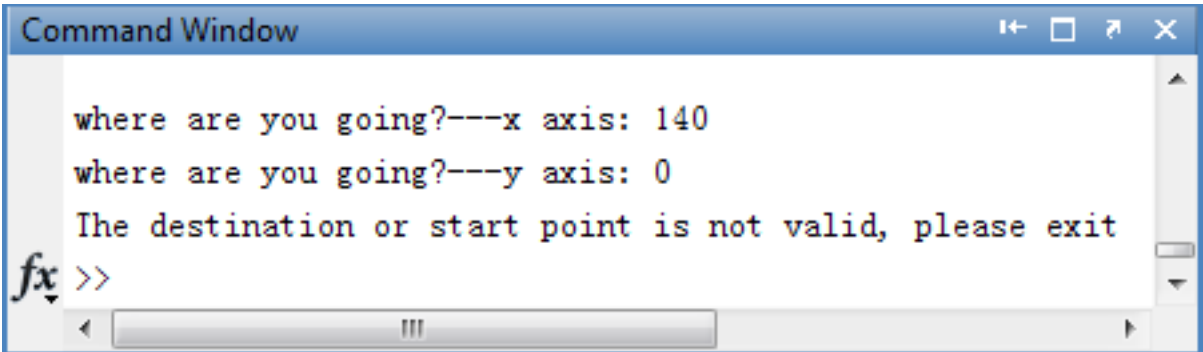
## Chapter 6

# Trajectory Simulation Results and Analysis

In this chapter, we show the trajectory simulation result utilizing our combined method, fuzzy algorithm. In the simulation, the vehicle's task is to reach the desired position given random start point and random obstacles. We demonstrate the simulation results under five different conditions: invalid destination, close destination, distant destination in slow speed set, simulation by using large speed set, simulation with time scale. The simulation is based on Matlab.

### 6.1 Invalid Destination

The invalid destination means the destination point is inside of an obstacle. In this case, the plot of obstacles map will first display but the system will not run the simulation, instead, a warning message will be shown in the command window (see Figure 6.1) and the simulation will exits. For example, in Figure 6.1, the destination coordinates, given by the user input, is an invalid destination. Therefore, the error message pops up.



```

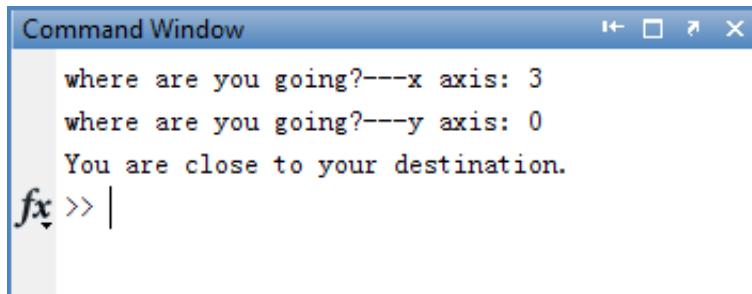
Command Window
where are you going?---x axis: 140
where are you going?---y axis: 0
The destination or start point is not valid, please exit
fx >>

```

Figure 6.1: Invalid destination

## 6.2 Close Destination

When the start position and valid destination are in close proximity, which means the distance from the start point to the desired position is smaller than or equals to 5 by our definition, we consider it to be an arrival to the destination. The obstacles may change because they are generated randomly and the command window will show the hint: 'You are close to your destination' (Figure 6.2).



```

Command Window
where are you going?---x axis: 3
where are you going?---y axis: 0
You are close to your destination.
fx >> |

```

Figure 6.2: The command window in the situation that the destination close to the start position

## 6.3 Distant Destination

Several trajectory simulations in distant destination are discussed in the current section, in these simulations, the speed values ( $j$  value) are from 0 to 8. The advantages to driving with small velocity are: good maneuvering, easy turning and fast braking. One disadvantage

is driving slowly is a time consuming way to arrive to the desired position.

Another situation, when no path is found to the destination, the command window shows the warning: Wrong way, stop, and need backup! (Figure 6.3).

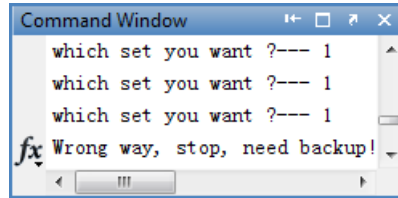


Figure 6.3: The command window of no available path

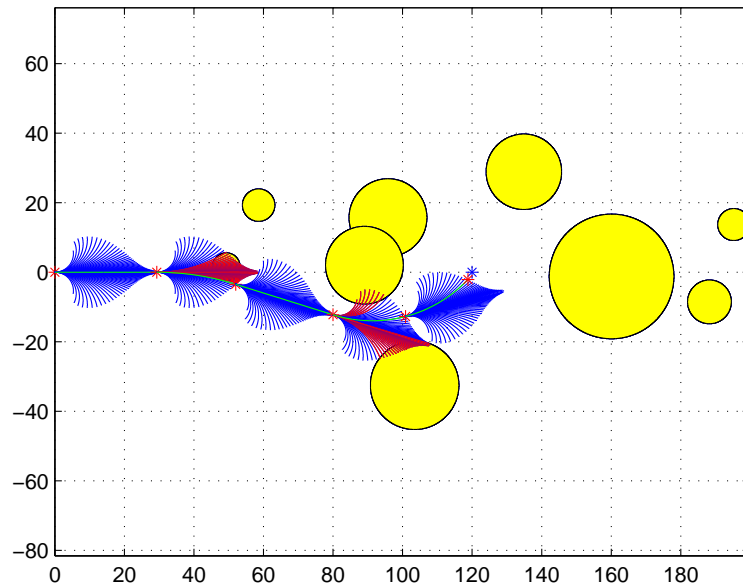


Figure 6.4: Schematic plan of trajectory selection in speed set 1

One of the successful simulations is shown in Figure 6.4. The small red star at (0,0) designates the vehicle starting point and the blue star at (120,0) is the goal point. The vehicle keeps a certain velocity ( $j$  equals to 1) in the simulation process. Blue tentacles are the pre-selected paths, red arcs are blocked paths, and those green arcs are the selected tentacles. A red star is plotted at the end of each selected tentacle. The yellow circles stand

for obstacles.

Other simulation results with the same start point and desired position as in Figure 6.4 are shown in Figure 6.5, the desired position among those subplots in Figure 6.5 are same, which is  $(250, 0)$ ,  $j$  equals to 2,4,6,8 respectively.

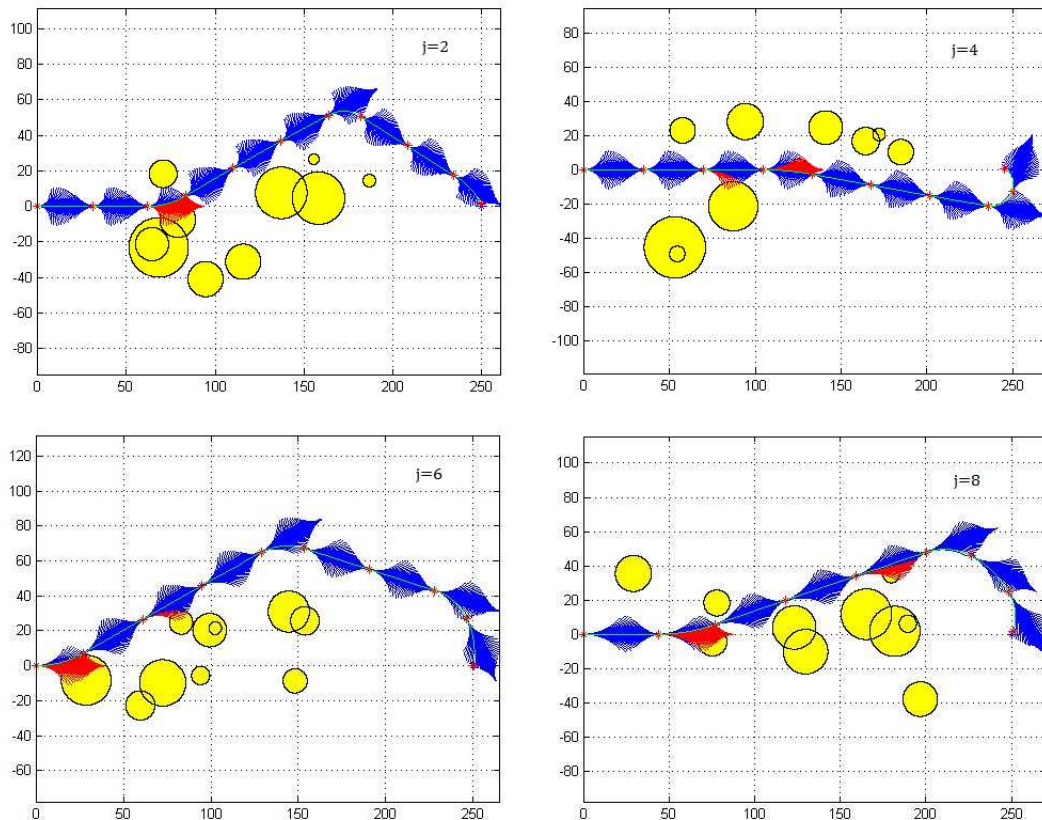


Figure 6.5: Schematic plan of trajectory selection with random obstacles map,  $j=2,4,6,8$

## 6.4 Simulation Analysis in Different Speed

The simulation in Figure 6.6 displays the selected paths in different values of  $j$  with the same obstacles map. The destination is  $(200, 0)$  and the speed set is 6, 3.4970m/s. The selected trajectory is not the shortest one because the shortest path is the straight line between two points: start position and the destination, other simulations in speed sets



0,1,4,5 have as same destination as simulation  $j=6$ , nearly all of them are selected closer path trajectory compared with the path in the last simulation shown in Figure 6.6 because they are all closer to the straight line between start point and end point, compared with the simulation  $j=6$ . Putting speed factor into consideration, the speed set 6 gets the minimum duration to arrive the desired position.

From the simulation results shown in Figure 6.6 we can obtain each subplot figure's simulation path length, speed set value and loop number. Each path length divided by its own speed value comes out with the duration of the simulation. The Table 6.1 shows that with the increasing of the speed, the duration reduces significantly as well as calculation loop times. Meanwhile, the path length basically showed a similar tendency as the speed.

Table 6.1: The table of duration comparison in different speed

| Speed set | Speed (m/s) | Duration(s) | Path length(m) | Loop number |
|-----------|-------------|-------------|----------------|-------------|
| 0         | 0.2500      | 860.8076    | 215.2019       | 9           |
| 1         | 0.6282      | 327.7057    | 205.8647       | 8           |
| 4         | 2.2460      | 92.4026     | 207.5363       | 7           |
| 5         | 2.8589      | 77.6952     | 222.1228       | 7           |
| 6         | 3.4970      | 61.7619     | 215.9815       | 7           |

## 6.5 Simulation With Time Scale

We can see the simulation result from Figure 6.7, the vehicle obtains selected tentacle from the output of algorithm and follows the selected path until the end point of the tentacle. During the path following procedure, the vehicle gets into the next loop to calculate the next optimized path, the start point of the next loop is the end point of the selected tentacle in the current loop.

There is another way to simulate the path selection process: plot the path by using points. The distance between two points is the 'mapping resolution' as we defined in the chapter 'Robot Kinematic Model'. One disadvantage of this method is that it does not need to obtain a calculation result before each plotting point in every loop. If the mapping accuracy is 0.0003, the mission is to move forward 30 (unit length), the method needs to run the path selection algorithm 100,000 times with 99,999 pauses. The pause occurs because

the robot needs to get the calculation result before plotting the next point.

In our solution, the selection algorithm only calculates one time, thus we can complete the next path selection result during the vehicle following the current selected tentacle path. The pre-selection procedure provides free tentacles set. As a result, if we need to achieve the goal 'move forward 30 (unit length)', our simulation needs to calculate the time with same amount of pauses. It depends on the vehicle's speed and the distance to the destination as well.

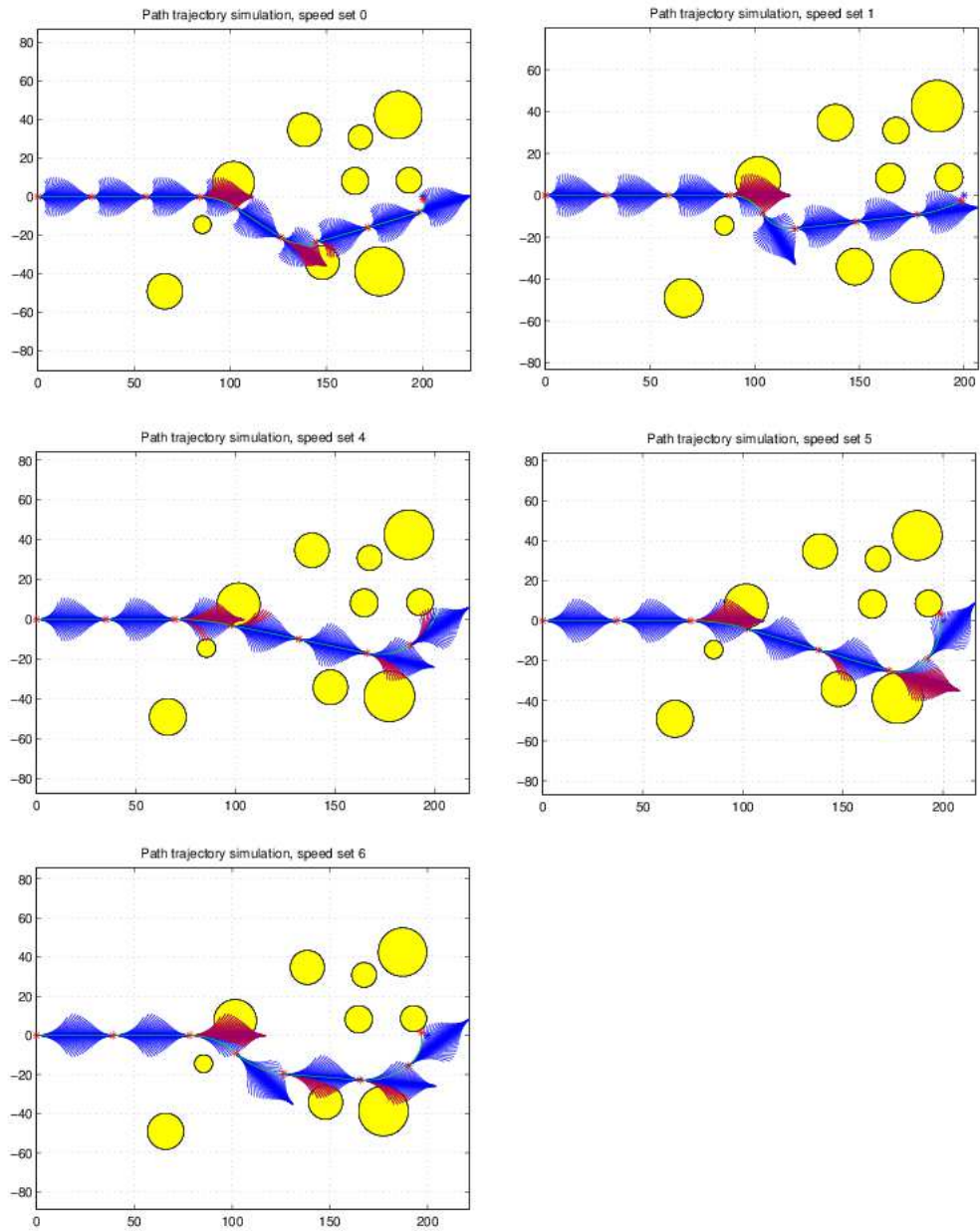


Figure 6.6: Path selection in one map

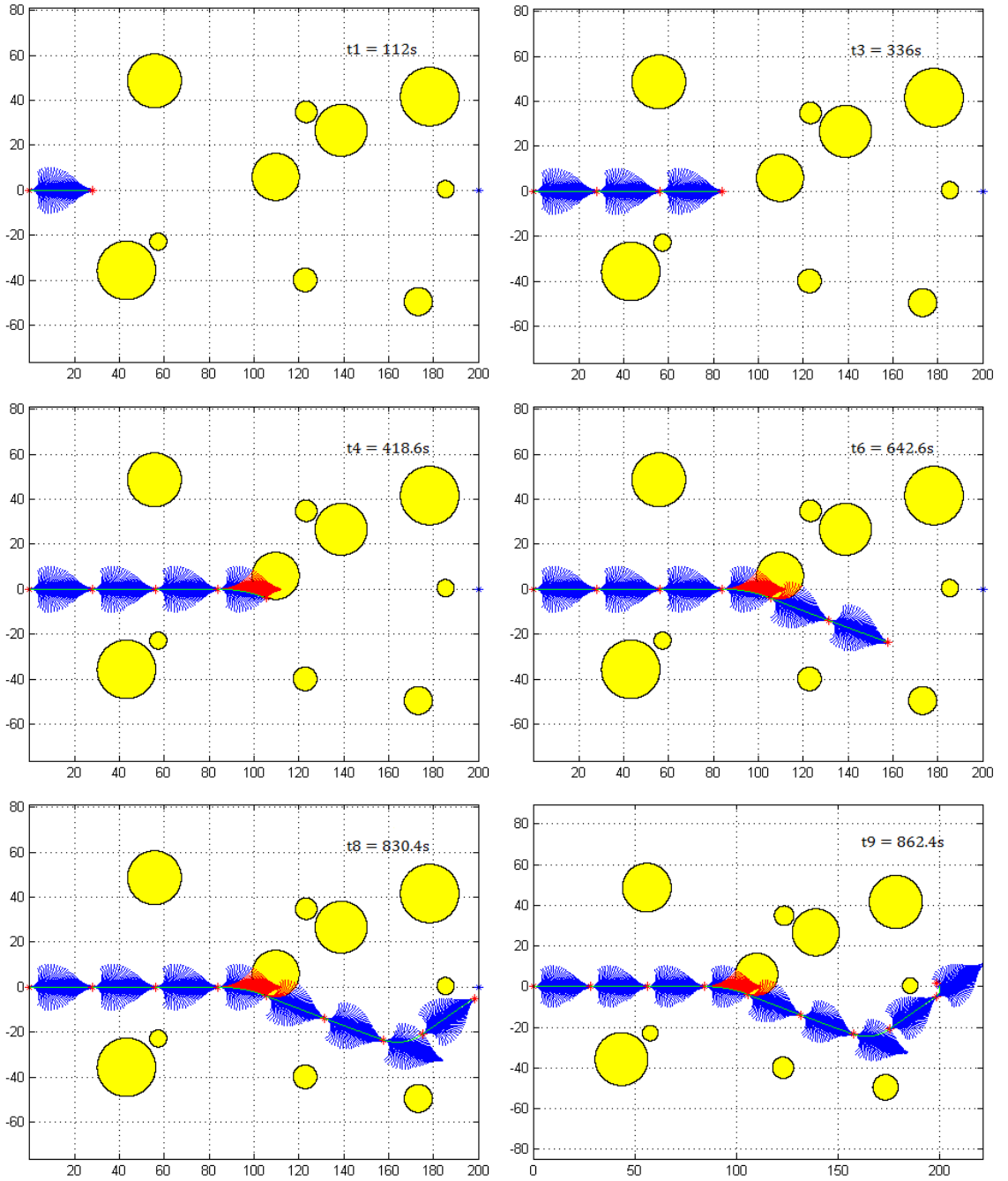


Figure 6.7: Schematic plan of trajectory selection in speed set 0 with time scale

## Chapter 7

# Experimental Results

After the simulation, we implemented the algorithm on a Clearpath Robotics Husky A100 robot running the Robot Operating System (ROS), using the C++ programming language. The data output of our tests are LIDAR scan results (Figure 7.1). They are stored and recorded in Polly during the experiment (Figure 7.2)

By using the ROS visualization tool rviz, we got the diagram result in Figure 7.1 shows obstacle avoidance with our algorithm implemented. When the obstacle appears in front of the vehicle, the system will choose a tentacle to avoid the obstacle. The smooth arc is the selected tentacle, the other data points are obstacle information from the LIDAR scan.

The top graph is the robots X/Y position, the middle graph is its linear velocity, and the bottom graph is its rotational velocity. The angular velocity graph is noisy due to the inaccuracies of the encoders, but the large excursions represent obstacle avoiding. The X-axis for all graphs is in seconds.

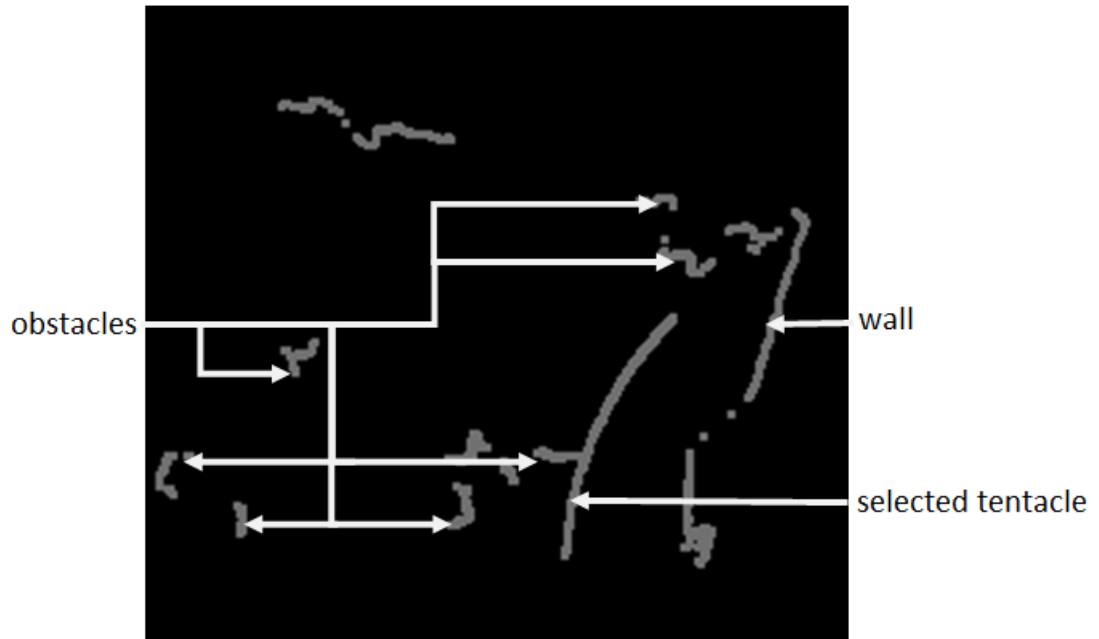


Figure 7.1: Output from Local Planner node in rviz

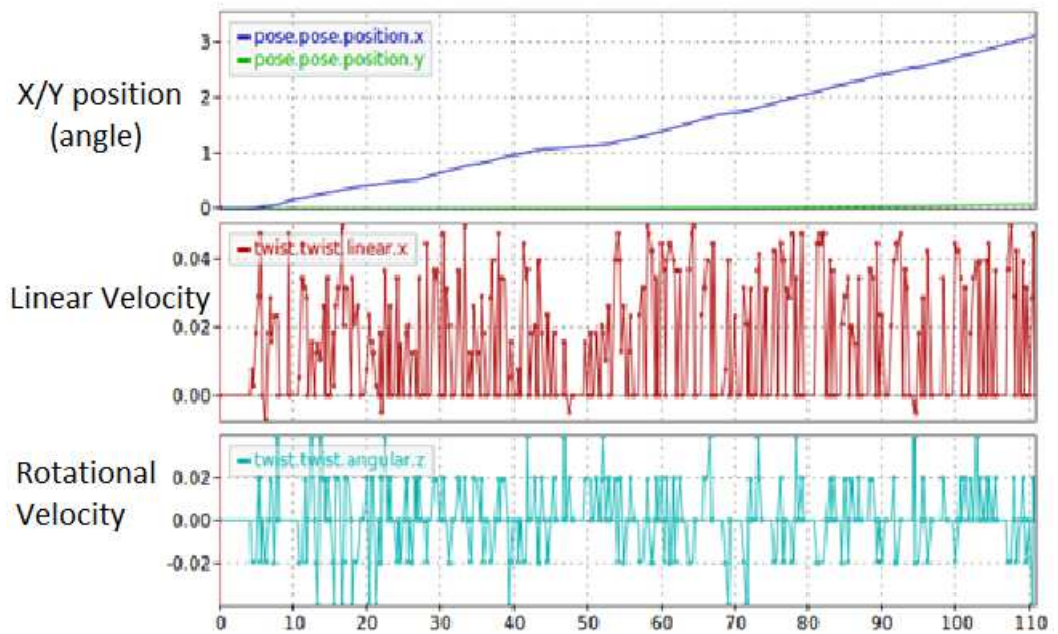


Figure 7.2: The output from Polly

## Chapter 8

# Conclusions

We proposed a simple path planning method using only LIDAR information for obstacle avoidance. The approach can be used in situations without global positioning system data or other localization techniques as it is only dependent on basic odometry data that does not need to be very accurate. The obstacle avoidance behavior itself is completely decoupled from localization as it is performed in the robot-frame. The algorithm was successfully implemented both in simulation and on actual hardware, and in both instances successfully drove autonomously and avoided obstacles while moving towards a goal point. The simulation successfully demonstrated tentacle sets modeling, obstacles avoidance and tentacle selection using fuzzy logic. We proposed three dimensional fuzzy rules with weight for each factor, and produced a fuzzy controller. In the experiment portion, we implemented the fuzzy-tentacle algorithm on our Clearpath robot Polly using the UTM-30LX LIDAR installed on the robot for our environmental sensor. The testing environment experimental results indicated the feasibility and efficiency of the fuzzy-tentacle algorithm. Meanwhile, several extensions are possible to improve the algorithm. One of the interesting ideas would be all tentacles reach obstacles. In this situation the vehicle could still run with low speed and modified bad-tentacle criteria, as well as increased LIDAR scan frequency to pass through very closely spaced obstacles. In addition, the global planner is necessarily as simple as the point was to demonstrate the feasibility of the obstacle avoidance behavior. More complex global planners could improve tentacle selection by strategically placing the

goal point at various points along a basic pre-computed optimal path that is updated at a much slower rate for computational efficiency. Finally, the algorithm currently is only in 2D space (x,y). However it could be easily expanded to 3D space by extending both the tentacle generation algorithm and the local planner into 3D.



# Bibliography

- [1] Peter Leven and Seth Hutchinson. A framework for real-time path planning in changing environments. *The International Journal of Robotics Research*, 21(12):999–1030, 2002.
- [2] Elmer G Gilbert and Daniel W Johnson. Distance functions and their application to robot path planning in the presence of obstacles. *Robotics and Automation, IEEE Journal of*, 1(1):21–30, 1985.
- [3] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.
- [4] Hongxiao Yu, Jianwei Gong, Karl Iagnemma, Yan Jiang, and Jianmin Duan. Robotic wheeled vehicle ripple tentacles motion planning method. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pages 1156–1161. IEEE, 2012.
- [5] Bruno Apolloni and Adrian Moise. Mobile robot navigation using the mathematical model of the sensing system. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, pages 2625–2630. IEEE, 2000.
- [6] Robert M Rogers. *Applied mathematics in integrated navigation systems*, volume 1. Aiaa, 2003.
- [7] AJ Mannucci, BD Wilson, DN Yuan, CH Ho, UJ Lindqwister, and TF Runge. A global mapping technique for gps-derived ionospheric total electron content measurements. *Radio science*, 33(3):565–582, 1998.

- [8] David Titterton, John Weston, et al. Strapdown inertial navigation technology. 2-nd edition. *The Institution of Electrotechnical Engineers, Reston USA*, 2004.
- [9] Gines Benet, Francisco Blanes, José E Simó, and Pascual Pérez. Using infrared sensors for distance measurement in mobile robots. *Robotics and autonomous systems*, 40(4):255–266, 2002.
- [10] Alberto Elfes. Sonar-based real-world mapping and navigation. *Robotics and Automation, IEEE Journal of*, 3(3):249–265, 1987.
- [11] Hans P Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 116–121. IEEE, 1985.
- [12] Maxim A Batalin, Gaurav Sukhatme, and Myron Hattig. Mobile robot navigation using a sensor network. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 636–641. IEEE, 2004.
- [13] Felix Von Hundelshausen, Michael Himmelsbach, Falk Hecker, Andre Mueller, and Hans-Joachim Wuensche. Driving with tentacles: Integral structures for sensing and motion. *Journal of Field Robotics*, 25(9):640–673, 2008.
- [14] Ayanna Howard, Homayoun Seraji, and Barry Werger. A terrain-based path planning method for mobile robots. In *7th International Conference on Automation Technology*. Citeseer, 2003.
- [15] Homayoun Seraji and Ayanna Howard. Behavior-based robot navigation on challenging terrain: A fuzzy logic approach. *Robotics and Automation, IEEE Transactions on*, 18(3):308–321, 2002.
- [16] Petru Rusu, Emil M Petriu, Thomas E Whalen, Aurel Cornell, and Hans JW Spoelder. Behavior-based neuro-fuzzy controller for mobile robot navigation. *IEEE Transactions on Instrumentation and Measurement*, 52(4):1335–1340, 2003.
- [17] Carlos Canudas De Wit, Georges Bastin, and Bruno Siciliano. *Theory of robot control*. Springer-Verlag New York, Inc., 1996.

- [18] Gregory Dudek and Michael Jenkin. *Computational principles of mobile robotics*. Cambridge university press, 2010.
- [19] Stanislaw H Zak. *Systems and control*. Oxford University Press New York, 2003.