Quasi-Harmonic Function Approach to Human-Following Robots

Navigation and Human-Following Methods Using Velocity Potential Fields Approach

Guangqi Nie

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the degree of

Master of Applied Science in Mechanical Engineering

uOttawa

University of Ottawa

Ottawa, Ontario, Canada

April, 2014

Abstract

In this thesis，an approach for robot motion control with collision avoidance and human-following is investigated. Using velocity potential fields approach in a modified, quasi-harmonic, solution, the navigation controller is developed. A quasi-harmonic function based controller uses harmonic solutions for collision avoidance and smoothly changes toward a non-harmonic solution which tends toward a zero velocity command only when approaching the goal. After the motion controller was created, human-following strategy was designed to let a non-holonomic robot have the ability to follow a human in an unknown environment with obstacles. The approach is based on velocity potential fields that permit to generate velocity vector commands that drive the robot at a safe distance with regard to the human while avoiding obstacles. The quasi-harmonic approach is investigated analytically using symbolic math solutions of MAPLE™ as well as in simulations using MATLAB™. Motion simulations of both holonomic and non-holonomic robot motion illustrate how the proposed approach operates. Experiments are also made with LEGO MINDSTROMS NXT robot to test the algorithm in environment with simple and complex obstacles.

Acknowledgment

Foremost, I would like to express my appreciations and respect to my supervisor Dr. Dan Necsulescu for his inspiration, encouragement, advices, guidance and patience. He gave me a lot of help with this thesis and made our research fruitful. He is not only a good mentor but also a helpful friend and I was so lucky to meet him at the very beginning.

I would also like to thank Dr. Jurek Sasiadek who attended my presentation and gave me significant comments and suggestions to my research. And thank him to the direction to one of our papers.

I thank to my colleagues not only for their encouragement, inspiration and advice, but also for their friendship.

Finally, I would like to thank my family, specifically my parents for their love, care and supports, and to Miss Luo, for her love, understanding and encouragement.

Contents

Abstract I

Acknowledgment II

Contents III

List of Figures VIII

List of Tables XIII

Chapter 1 Introduction 1

1.1 Background 1

1.1.1 Mobile Robots 1

1.1.2 Autonomous System of Mobile Robot 3

1.1.3 Autonomous Mobile Robot Navigation 4

1.1.4 Navigation of Non-holonomic Robot 5

1.1.5 Human-Following Robot 5

1.2 Research Objective and Approach 7

1.3 Thesis Outline 8

Chapter 2 Literature Review 10

2.1 Acoustics Based Human-Detection Robot 10

2.2 Vision-Based Human-Detection Robot 12

2.3 Infrared Camera Based Human-Detection Robot 15

2.4 Pyroelectric Infrared Sensor Based Human-Detection Robot 16

2.5 Human-Tracking Robot in an Intelligent Space 19

2.6 Other Human-Tracking Strategies 21

Chapter 3 Proposed Navigation and Human-Following Approaches 23

3.1 Quasi-Harmonic Approach 23

3.1.1 Harmonic Function and Non-Harmonic Function 24

3.1.2 Derivation of Quasi-Harmonic Potential Function 27

3.1.3 Comparison of Harmonic and Quasi-Harmonic Function Using Typical Illustrations 30

3.2 Human-Following Approach 43
Chapter 4 Experimental Setup 52
4.1 NXT Intelligent Brick 53
4.2 Sensors 55
4.2.1 Ultrasonic Sensor 56
4.2.2 Passive Infrared Sensor 60
4.2.3 Thermal Infrared Sensor 63
4.2.4 Rotation Sensor 64
4.3 Servo Motor 65
4.4 Networking Setup 67
4.5 Mechanical Structure of LEGO NXT Robot Based on Experiments. 69
Chapter 5 Description of the Software 72
5.1 MATLAB 73
5.2 NI LabVIEW 76
5.3 LEGO MINDSTORMS NXT-G 78
5.4 Comparison of NXT-G and NI LabVIEW 79
Chapter 6 Simulation Results 81
6.1 Simulation of Holonomic Robot Obstacle Avoidance 84
6.1.1 Holonomic Robot Move Around Two Obstacles 84
6.1.2 Holonomic Robot Get Through Two Obstacles 86
6.2 Simulation of Non-Holonomic Robot Obstacle Avoidance 88
6.3 Simulation of Non-Holonomic Human-Following Robot 91
Chapter 7 Experimental Results 94
7.1 Experiment of Quasi-Harmonic Function 95
7.2 Experiment of Non-Holonomic Robot with Two Adjacent Obstacles 96
7.3 Experiments of Non-Holonomic Robot with Concave Shape Obstacle 99
7.4 Experiments of Non-Holonomic Human-Following Robot with Obstacles 102
Chapter 8 Conclusions 106
8.1 Summary 106
8.2 Main Research Contributions 107

8.3 Future Work 108
References 110
Appendix A Development Environment of LEGO NXT Robot 118
A.1 LEGO MINDSTORMS NXT-G 2.0 120
A.1.1 Programming Principles of NXT-G 121
A.1.2 The Blocks Available in NXT-G 123
A.2 National Instrument LabVIEW 2012 125
A.2.1 Introduction to VI and Graphical Programming 126
A.2.2 LabVIEW for LEGO MINDSTORMS 128
Appendix B MATLAB Simulation: Holonomic Robot Collision Avoidance Controller Code 134
B.1 Main Function 134
B.2 Robot Sensors Detect Obstacles 137
B.3 Collision Avoidance Controller 139
B.4 Direction Arrow 142
B.5 Sensor View 143
Appendix C MATLAB Simulation: Holonomic Robot Pass Through Two Obstacles 144
C.1 Main Function 144
C.2 Robot Sensors Detect Obstacles 147
C.3 Collision Avoidance Controller 150
C.4 Direction Arrow 153
C.5 Sensor View 154
C.6 Estimate If the Robot Can Get Through the Two Obstalces 155
Appendix D MATLAB Simulation: Non-Holonomic Robot Collision Avoidance Controller Code 156
D.1 Main Function 156
D.2 Robot Sensors Detect Obstacles 160
D.3 Collision Avoidance Controller 163
D.4 Direction Arrow 167

D.5 Sensor View 168
D.6 Pose of the Four Wheels 169
D.6.1 Left-Front-Wheel 169
D.6.2 Right-Front-Wheel 169
D.6.3 Left-Rear-Wheel 169
D.6.4 Right-Rear-Wheel 169
D.7 Shape of the Four Wheels 170
D.7.1 Left-Front-Wheel 170
D.7.2 Right-Front-Wheel 170
D.7.3 Left-Rear-Wheel 170
D.7.4 Right-Rear-Wheel 171
Appendix E MATLAB Simulation: Holonomic Human-Following Robot 172
E.1 Main Function 172
E.2 Robot Sensors Detect Obstacles 177
E.3 Collision Avoidance Controller 179
E.4 Human Obstacle Notice 182
E.5 Human Trajectory 184
E.6 Human Detection 187
E.7 Human Position 189
E.8 Direction Arrow 190
E.9 Sensor View 191
E.10 Pose of the Four Wheels 192
E.10.1 Left-Front-Wheel 192
E.10.2 Right-Front-Wheel 192
E.10.3 Left-Rear-Wheel 192
E.10.4 Right-Rear-Wheel 192
E.11 Shape of the Four Wheels 193
E.11.1 Left-Front-Wheel 193
E.11.2 Right-Front-Wheel 193
E.11.3 Left-Rear-Wheel 193

E.11.4 Right-Rear-Wheel 194
Appendix F LabVIEW: Obstacle Avoidance 195
Appendix G LabVIEW: Human Following 200
Appendix H LabVIEW: Human Following Combine with Obstacle Avoidance 206

List of Figures

Figure 3. 1 Location of obstacles and robot.....25
Figure 3. 2 (a) Harmonic function (b) Non-harmonic function.....25
Figure 3. 3 Plot of harmonic function with obstacles and goal 27
Figure 3. 4 Plot of velocity based on harmonic function.....28
Figure 3. 5 Plot of velocity based on quasi-harmonic function 29
Figure 3. 6 Position of obstacles and goal and velocity commands of robot.....30
Figure 3. 7 Plot of harmonic function for model 1 31
Figure 3. 8 Area near the saddle point when kt=0 32
Figure 3. 9 Area near the saddle point when k_t=1.0.....33
Figure 3. 10 Velocity of harmonic function in X-Z view.....34
Figure 3. 11 Velocity of quasi-harmonic in X-Z view 36
Figure 3. 12 Velocity of quasi-harmonic in Y-Z view.....37
Figure 3. 13 Position of obstacles and goal and velocity commands of robot.....38
Figure 3. 14 Plot of harmonic function for model 2 39
Figure 3. 15 Area near the saddle point for normal velocity commands 40
Figure 3. 16 Area near the saddle point when kt=0.2 42
Figure 3. 17 Area near the saddle point when kt=0.4 42
Figure 3. 18 Area near the saddle point when kt=0.8 43
Figure 3. 19 Description of the task.....44
Figure 3. 20 robot found and far from the human.....44
Figure 3. 21 robot is getting too close to the human.....45
Figure 3. 22 Working principle of PIR sensor 45
Figure 3. 23 Relationship between relative motion and sensor value.....48
Figure 3. 24 Deployment configuration of the PIR sensor array (method 1) 49
Figure 3. 25 Deployment configuration of the PIR sensor array (method 2) 50

Figure 4. 1 Illustration of NXT intelligent brick 1 54
Figure 4. 2 Illustration of NXT intelligent brick 2 54
Figure 4. 3 Ultrasonic probe and it's symbol 57
Figure 4. 4 Theory of ultrasonic ranging 57
Figure 4. 5 LEGO ultrasonic sensor 58
Figure 4. 6 Directivity of LEGO ultrasonic sensor 59
Figure 4. 7 The design of obstacle detection structure 59
Figure 4. 8 HiTechnic NXT PIR sensor (NIS1070) 60
Figure 4. 9 The field of view of PIR NIS1070 61
Figure 4. 10 Plot of sensor's values when a human passes by 62
Figure 4. 11 PIR sensors' position on robot frame 62
Figure 4. 12 Thermal infrared sensor for LEGO® MINDSTORMS® NXT 63
Figure 4. 13 Dissection of LEGO rotation sensor 64
Figure 4. 14 Circuit board of the rotation sensor 64
Figure 4. 15 (a) NXT servo motor (b) Mechanical design of NXT servo motor 65
Figure 4. 16 NXT motor rotation speed vs. motor power level 66
Figure 4. 17 Configuration of three NXT servo motors 67
Figure 4. 18 USB connection 68
Figure 4. 19 (a) Side view of robot; (b) Front view of robot; (c) Top view of robot 69
Figure 4. 20 (a) Sensors on robot; (b) Output ports and counter weight on robot; (C) Motors and chassis of the robot 70
Figure 4. 21 TIR sensor on the robot 71

Figure 5. 1 Obstacle avoidance sub function 72
Figure 5. 2 Obstacle avoidance program with LabVIEW 77
Figure 5. 3 Quasi-harmonic function in LabVIEW 78
Figure 5. 4 Obstacle avoidance program with NXT-G 78
Figure 5. 5 Temperature measurement program with NXT-G 79

Figure 6. 1 Flowchart of main function 82
Figure 6. 2 Sub functions 82
Figure 6. 3 (a) Obstacle detection sub function; (b) Robot's pose calculation sub function 82
Figure 6. 4 Flowchart of main function 83
Figure 6. 5 Sub functions 83
Figure 6. 6 (a) Human following sub function; (b) Robot's pose calculation sub function 84
Figure 6. 7 Simulation of holonomic robot move around obstacles 85
Figure 6. 8 Simulation of holonomic robot pass through obstacles 87
Figure 6. 9 Simulation of non-holonomic robot obstacle avoidance 90
Figure 6. 10 Simulation of human-following robot 92

Figure 7. 1 Screenshots of experiment 1 every 3 seconds 96
Figure 7. 2 Experiment of non-holonomic robot travelling around two adjacent obstacles and reach the goal 98
Figure 7. 3 (a) Room temperature; (b) Heater surface temperature 99
Figure 7. 4 Experiment of non-holonomic robot with 5 obstacles 100
Figure 7. 5 Experiment of non-holonomic robot with 3 trash can obstacles 101
Figure 7. 6 (a) Room temperature; (b) Dressed human body temperature indoor 102
Figure 7. 7 (a) Outside temperature; (b) Dressed human body temperature outdoor 102
Figure 7. 8 Experiment of human-following robot 105

Figure A. 1 NXT-G development environment 121
Figure A. 2 Programming model and thread of NXT-G 122
Figure A. 3 Parallel program and no connection block 122
Figure A. 4 Blocks available in NXT-G 123
Figure A. 5 TIR block (upper) and PIR block (below) 125

Figure A. 6 Front panel of a single VI 126
Figure A. 7 (a) Frequently-used controls; (b) Frequently-used indicators..... 127
Figure A. 8 Back panel of a single VI..... 128
Figure A. 9 (a) Examples of functions; (b) structures..... 128
Figure A. 10 Functions included in LEGO NXT module patch for LabVIEW . 129
Figure A. 11 Start interface of LabVIEW for LEGO MINDSTORMS 130
Figure A. 12 MINDSTORMS competition toolkit 130
Figure A. 13 (a) NXT programming; (b) NXT I/O; (c) Behaviors; (d) Third party motors 131
Figure A. 14 Additional third party sensors and motors 132
Figure A. 15 A customized PIR sensor function 133

Figure F. 1 Flow chart of obstacle avoidance sub function 196
Figure F. 2 Obstacles on both left and right..... 197
Figure F. 3 Obstacle on the left..... 197
Figure F. 4 Obstacle on the right..... 198
Figure F. 5 No obstalce detected..... 198
Figure F. 6 Obstacles right ahead the robot case 1..... 199
Figure F. 7 Obstacles right ahead the robot case 2..... 199

Figure G. 1 Customized PIR senor program..... 200
Figure G. 2 Flow chart of human following sub function 201
Figure G. 3 PIR sensor 1 and 2 detect tempurature difference 202
Figure G. 4 PIR sensor 1, 2 and 3 detect tempurature difference 203
Figure G. 5 PIR sensor 2 detect tempurature difference..... 203
Figure G. 6 PIR sensor 2 and 3 detect tempurature difference 204
Figure G. 7 PIR sensor 1 detects tempurature difference 204
Figure G. 8 PIR sensor 3 detects tempurature difference 205
Figure G. 9 No PIR sensor detects tempurature difference 205

Figure H. 1 Combination of human following and obstacle avoidance functions 206

List of Tables

Table 3- 1 Solution of equation set 3.23 and 3.24 39

Table 3- 2 Robot motion with respect to sensors 51

Table 4- 1 Sensor usage 56

Table 4- 2 Numerical results of sensor's values when a human passes by 61

Table 4- 3 Main characteristics of the NXT servo motor 65

Table 4- 4 Meanings of bluetooth icons 68

Table 4- 5 I/O ports utilization 71

Table A- 1 NXT programming software 118

Table A- 2 Minimum technical requirements of NXT-G 120

Chapter 1

Introduction

1.1 Background

During the past years, motivation of studies in intelligent autonomous robots has grown within universities, corporations and federal agencies around the world in military, civilian and commercial applications. Autonomous robots must have the ability to make decision and action autonomously. Tele-operated and reprogrammable robots expand this technology to more exacting applications in remote unstructured and hazardous environments. A fully autonomous robot needs to have the ability to obtain information of the environment, to work for an extended period individually without human intervention, to move, either all or part of itself, throughout its operating environment with no human assistance, to avoid harmful situation to human, property, or itself.

1.1.1 Mobile Robots

Mobile robot is one the most significant parts of autonomous robots. It is a major focus of current research and there are one or more labs that focus on mobile robot research almost in every major university. It can also be found in industry, military and security environments. Mobile robots have great value in searching unknown arrears and performing tasks that are dangerous or not suitable to human. For instance, some repetitive or unpleasant tasks would cause high cost or low efficiency if human were hired for these jobs. Mobile robots' task area includes agriculture, container handling, inspection, scientific exploration, transportation, mining, waste management, etc.

Based on the environment mobile robots are working in, they can be classified into four types:

- Unmanned Ground Vehicles (UGVs). Working on land, they are wheeled, tracked or legged robots with two or more legs. Land and home robots are usually classified into this part.
- Unmanned Aerial Robots (UAVs). Working in the air, aerial robots are usually referred to this type.
- Autonomous Underwater Vehicles (AUVs) are robots work underwater.
- Polar Robots. This kind of robot is designed to navigate in icy filled environments.

Based on the device robots used to move, UGV mobile robots can be further classified into three types:

- Legged Robot: Human-like legs (humanoid robot) or animal-like legs (resembling animals or insets).
- Wheeled Robot.
- Continuous Track Robot.

Mobile robots with legs always have high degrees of freedom, they are mechanically complex and are expensive to build and manufacture. But they are more maneuverable and adaptable in rough terrain and tasks. Robot with wheels and tracks are easier and cheaper to manufacture, and they can travel at higher velocities.

In our research, wheeled mobile robot is the object of study. The wheeled mobile robot will move on two-dimensional flat terrain. The driving wheels of the robot are a pair of differential drive steering wheels. A differential wheeled robot is a mobile robot whose movement is based on two independently driven wheels placed parallel on both sides of the robot body. It gives robot good maneuvering characteristics. It determines its direction by controlling the relative rate of rotation of its wheels and does not require an additional steering motion. A differential wheeled robot has the abilities to spin in place, move in a straight line, move in a circular path, and follow prescribed trajectories. However, such a robot cannot move in all degrees

of freedom on its working space. For instance, there is no velocity in the direction of drive wheels' axis. A robot is holonomic if the controllable degrees of freedom are equal to the total degrees of freedom. Hence it is a non-holonomic robot. [1]

1.1.2 Autonomous System of Mobile Robot

An autonomous system needs to be designed for a mobile robot to determine how the robot will move when it meets different situations. When designing an autonomous system, designer should have a concept of the autonomy level for a robot. Generally, there are four levels of autonomy for autonomous vehicle system. The decision on the level of autonomy depends on the demands of the research and the budget of the project. The higher the level of autonomous systems, the more complex, expensive and time-consuming they are to be developed. The four levels of autonomous system are:

(1) Remote control and teleportation

Remote control is the first level of autonomy. Vehicles or robots are controlled by a human using a remote controller. To realize the remote control, communications need to be established between the human operator and robot so that the operator can receive the visualized information from the robot sensors and sent control command back to the robot according to the information received.

(2) Semi-autonomous

The second level of autonomy is the semi-autonomous mobile robot. This system minimizes the need for operator interaction compare with the first one. It adapts to simplest changes in the objective designed by human. It might have ability of navigation, obstacle avoidance and data fusion capabilities to realized the autonomous tasks.

(3) Platform-centric autonomous

As the third level, platform-centric autonomous robot can undertake complex missions and respond to additional commands form a controller without guidance. Robots with such a system are fully autonomous.

(4) Network-centric autonomous

This system is always used to build multi-robot systems. Robots in this level are able to sent and receive information from the communications network and incorporate it in the mission to complete tasks cooperatively.

1.1.3 Autonomous Mobile Robot Navigation

Navigation is the first and most basic component of motion planning for an autonomous robot. The task of navigation is finding a collision-free motion for the robot from configuration to configuration. Based on robot's sensor values, the robot makes a series of motions, so as to reach the goal efficiently without collision. Hence path planning and obstacle avoidance are two basic and significant competences of mobile robot navigation.

In some case, map of the space and a goal location are given, a basic path-planning problem is to produce a trajectory that connects a start configuration and a goal configuration. Path planning is a strategic problem solving process, as the robot has to decide what to do, how to do it more efficiently over the long term to achieve its goals.

On the other hand, obstacle avoidance, as the second competence, is more about reaction with respect to a real-time situation. There is no way to make a plan for obstacle avoidance. Sensors are significant component of this part of function. They sense real-time information of the environment and input the data to the processor of the robot so that it can react according to the obstacles.

The path planning and obstacle avoidance cooperate with each other for the navigation of a mobile robot. They work with each other and for each other to make sure others success.

1.1.4 Navigation of Non-holonomic Robot

Non-holonomic systems are characterized by constraint equations involving the time derivatives of the system configuration variables. [2] These constraint equations are non-inferable. Since it has less controls than configuration variables, non-holonomic system cannot be stabilized to a desired configuration via differentiable, continuous, pure-state feedback.

For a non-holonomic robot that has two controls for linear and angular velocities. In a three dimensional configuration space, any path in the configuration space does not necessarily correspond to a feasible path for the system. [2] Hence the geometric techniques used for holonomic robot's path planning cannot be directly applied to nonholonomic robot. Even in the absence of obstacle, path planning for nonholonomic robot is still not an easy task.

When obstacles are taken into consideration, it adds the difficulty of navigation of non-holonomic robot because constraints caused by both obstacles and non-holonomic system itself should be taken into account. It is necessary to combine geometric techniques addressing the obstacle avoidance and control theory techniques addressing the special structure of the non-holonomic motions.

The navigation strategies for non-holonomic robot can be treated separately. The path planning can be seen as an open loop control-planning problem. The obstacle avoidance and error correction can be considered as a close loop control strategy.

1.1.5 Human-Following Robot

Human-Following robots have numerous applications: maintenance in outdoor and in-door environments, assisting workers in the field, assisting military personnel, tracking intruders, terrorists or violent people, assisting incapacitated people etc. For achieving these tasks, the robots have to sense the human, track and follow while avoiding accidents and collisions [3].

Compare with a robot reaches a fixed goal, the navigation for human-following robot is move complicated. It is because the goal of human-following robot is a human and the human might move erratically. In the absence of obstacle, the path planning of human-following robot has to be real-time according to the new position of the human. Hence it is a close-loop control problem instead of open-loop control for which human's new positions are the feedbacks for the close-loop system.

There are various sensing systems for human detection and localization. They can be classified into several types as following.

- Acoustic Sensor System

Acoustic sensors are like ears of a robot. [4] This system enables the robot to localize a human by the voice of the human. Some advanced acoustic sensor system even able to understand some of the human's words. Using acoustic sensor to localize or track people has some advantages. It dose not require illumination and the detection area is omnidirectional. Furthermore, sound signal is a one dimensional signal requiring less computation compared to image-based systems. Finally, audio signals can travel around obstacles.

- Optical Imaging Sensor System

This type of system is based on camera(s) installed on robot. Cameras are the eyes for robot, with which robot can see and follow the human. Optical Sensor System always relies on sophisticated computer vision algorithms. One of its most critical advantages is the high accuracy of human detection. Main disadvantage for instance is complicated algorithm.

- Human Sensor System

Human sensors mainly refer to Pyroelectric Infrared Sensors (PIR). It is preferable for human detection since it is sensitive to the changes of human infrared radiation. Some studies[5][6][7][8][9] show that PIR sensors with Fresnel lens arrays have good performance on infrared motion sensing, which means human following.

- Network Based System

Network based human following system is established based on the communication between human and robot. There is a particular information source carried by the

human and the robot can recognize the unique signal from the human. Hence the accurate position of the human can be send to the robot in real time. The robot can easily follow the human in this case.

- Intelligent Space System

In this system, the robot is placed in an intelligent space in which distributed sensors are arranged. When a human step into the space, all the sensors in this space will offer position data of the human to the robot by network with a pretty high accurate and efficiency. This type of system can be used in defensive, monitoring or human-assistant purposes.

Systems are chosen according to the work environments, accuracy requirements, research grants, purposes of the systems, and so on.

1.2 Research Objective and Approach

Harmonic function is steady state solutions of Laplace equation, used in hydrodynamics [10][11][12][13][14][15]. It was researched and used numerously to control a mobile robot avoid obstacles successfully because harmonic functions and their linear combinations are known having maximum values on the boundary of a finite space and, thus do not lead to local minima as in the case of artificial potential fields. However, when the robot need to reach and stop at a goal area, there is a limitation for harmonic function because the velocity of the robot is infinite at the goal point. Proposed solution to this was the inclusion of exponential weighing functions, but this results in non-harmonic functions and, thus, to the loss of the advantages of saddle point of velocity potential fields.

In order to overcome the above contradiction, we were trying to create a function “between” harmonic function and non-harmonic function. This function will hopefully adapts the advantages of the two functions at the same time avoids the weaknesses of them. And we call it as a quasi-harmonic function. We proved the

function in both mathematical and graphical approaches and made a lot of simulations and experiments to check the feasibility of it. A quasi-harmonic function based controller that uses harmonic solutions for collision avoidance and smoothly change toward a non-harmonic solution which tends toward a zero velocity command only when approaching the goal.

Our research presents a motion control approach for both holonomic and non-holonomic mobile robots moving in an unknown environment containing different kinds of obstacles. The approach is based on local sense and control with collision avoidance, final positioning and human following. We include all these variables, type of mobile robot, type of obstacles, and type of goals, to show the flexibility of the quasi-harmonic function.

1.3 Thesis Outline

This thesis consists of eight chapters that starts with the designing of quasi-harmonic controller and end with human-following robot application of the controller.

Chapter 2 provides a literature review which examines human-following strategies in literature, and an analysis of each method is carried out.

Chapter 3 describes the creation of quasi-harmonic navigation control approach, and proofs the feasibility in both in mathematical and graphical ways.

Chapter 4 details the hardware components and the mechanical structure design of the LEGO MINDSTORMS NXT robot that is used in our experiments.

Chapter 5 introduces the two software and development environments used in this thesis to design the control programs for the LEGO NXT robot.

Chapter6 illustrates the processes of three simulations done with MATLAB in which holonomic robot, non-holonomic robot and human-following robot are included.

Chapter 7 includes all experiments need to be presented in the thesis. These experiments reappear the situations in the simulations and proof the control approach

can be utilized in practical way.

Finally, Chapter 11 provides a conclusion to this work, explains contributions of this research, and discusses future works might be done.

Chapter 2

Literature Review

2.1 Acoustics Based Human-Detection Robot

Acoustics based detection systems have the ability to sense sound sources. Acoustic sensors on a robot makes it like has ears. This system enables the robot to localize a human by the voice from the human. It does not require illumination and the detection area is omnidirectional. Furthermore, sound signal is a one dimensional signal requiring less computation compared to image-based systems, and audio signals can travel around obstacles. Hence acoustic based detection systems are always used to combine with other sensing systems to improve the detection accuracy.

In [3], the paper describes an application of passive human–robot interaction. In which a modality for localizing humans based on sound source localization has been developed. The sound source localization system is used to provide the face-tracker system with candidate regions for finding a human. In the research, using two electret microphones, a sound source localizer and human tracking system have been placed on the upper part of Intelligent Soft Arm Control for continuously localizing a person by the sounds from the person. In the fourth section of this paper, human tracking by sound source localization is discussed in three parts: objective and system configuration, time delay calculation, and the system performance. The result shows that sound source localization system can be combined with other system to develop a reliable tracking system.

In [16]，human-following robot based on vision sensor and laser range sensor might lose its target occasionally. To solve this problem, a speech system and sound source detection system are developed to achieve sound source localization and speech interaction between users and robot. The way sound source detection system works is

that during the tracking and following process, when robot gets lost, it will inform the user and wait for a sound from the user to re-localize the target's position. In the paper, sound source detection approach is introduced and realized with two pairs of microphones in the experiment. The results demonstrate the working process of the sound source detection and provide an alternative method to solve the losing target problem in human following researches.

In [17], Sound-source localization and tracking robot is the main body of the research. Interaural time difference (ITD) is used to compute the incidence angle of an acoustic sound-source on a horizontal plane so that the robot can locate and orient to the goal with the sound-source. The acoustic robot localization is realized by calculating the azimuth of goal with respect of robot and to calculate the azimuth of the goal, cross-correlation function is introduced. The algorithm of this paper is built based on the above approach. With plentiful experiments and data analysis, it can be seen that using cross-correlation and ITD is an effective method of sound-source localization, and this system is capable of locating a sound-source with an average accuracy of ±1.5 degrees.

In [18], sound detection extends the searching areas and increases the detecting accuracy of human detection for robotic urban search and rescue. During search and rescue tasks, rescue people sometimes stop all activity to listen to a shouting person. Sounds detection is effective in this condition. However in this research, since the researchers think that there are some limitations for sound detection, robot will not make a decision according to the sound. It will send the sound information to a computer and let a human determine whether or not the sound is what they are looking for.

In [19], sound source detector is used in an intelligent human activity analysis robot. The robot can track a human by ensuring an optimal observability of the person's activity in complex and cluttered environments. This paper introduces a framework with multiple human detectors (legs detector, face detector, upper body detector and sound source detector) based on all kinds of sensors embedded in the mobile robot and the fuzzy logic mechanisms to make the robot track humans. An array of

microphones integrated with the Kinect combined with the HARK software is used in this research to detect the sound source's location. The system has the ability to locate the sounds with an accuracy of 5 degrees. The sound sources in this paper are from humans, especially from talking and footsteps.

In [20], this paper describes a mobile robot equipped with a real-time sound-source localization system. The sound localization method in this paper is based on the precedence effect of the human auditory system to deal with echoes and reverberations. Researchers developed a robot auditory system and proofed its ability to localize sound source and navigate robot in reverberant environments. Sound localization and robot navigation experiments are made with fixed obstacles and a sound-source goal. Results show that the robot is capable of localizing sound sources and approaching the objects without collisions, even when the sound sources are behind obstacles.

2.2 Vision-Based Human-Detection Robot

This type of system is based on camera(s) connected to robot. Cameras are the eyes of the robot, with which robot can see and follow the human. Optical Sensor System always relies on sophisticated computer vision algorithms. One of its most critical merits is the high accuracy of human detection.

In [21], a human following robot (ApriAttenda^{TM}) that finds a specified person using visual tracking functions and follows the human while avoiding obstacles, has been developed. The robot obtains the image of target person by two CCD cameras on its head with pan/tilt motions. Using stereovision, it can get higher-resolution images in real time than in the case of using an omnidirectional camera. The robot in this research tracks a specified people by using a developed proprietary image-processing algorithm. The algorithm recognizes specific individual by registering the color and texture of the human's clothing, distinguishing the target from cluttered backgrounds, and calculating the distance between the target and robot and following the target. Moreover, a Laser Range Finder (LRF) is used as another sensor for the tracking

performance gain to offer highly accurate measurement information. In this paper, an improvement method based on the idea of Vision-LRF sensor fusion system is proposed. One feature of this system is the fusion rate changes with respect to the information of environments. The experimental results confirm the efficiency of Vision-LRF human-following method.

In [22], the research object is a mobile robot that interacts with humans. The robot has to follow specific people and distinguish among different people with the help of vision functions. For the task of person recognition and tracking, a method of inter-classifier feedback is used to track both uniquely identifying characteristics of a person such as face of a human, and more frequently visible, but perhaps less uniquely identifying characteristics such as the clothing. The inter-classifier feedback enables merging multiple, heterogeneous sub-classifiers designed to track and associate different characteristics of a person being tracked[22]. By identifying additional online training data for one another, heterogeneous sub-classifiers give feedback to each other, and hence improving the performance of the overall tracking system.

In [23], a monocular vision-based human-following robot was designed. From two aspects, image based visual servoing (IBVS) and position-based visual servoing (PBVS), visual servo techniques were discussed and position-based visual servo control approach is used in this paper to realize the human-following task.

Position-based visual servo strategy consists of three phases: target recognition, target pose estimation and controller calculations. In this research, a generalized target tracking system is designed; a hybrid control is designed which combines the benefits of direction-based motion control and point-to-point motion; and a human-following system incorporating human orientation is carried out.

In [24], vision based gesture-driven interface for human-robot interaction is designed. The human–following procedure is studied in an environment without obstacle. Human detection and tracking are realized by reorganization of a skeleton model using NITE middleware in OpenNI framework. In this work, the robot can distinguish point located in the center of the object and skeleton model which is set of

characteristic points connected by lines. Point tracking algorithm is used in human-following process while skeleton model is used to recognize gestures of human. Vision based interface with gesture recognition mechanism is realized in this paper by applying the above approach.

In [25], an agent for people detection and tracking through stereo vision is presented. A vision system is installed on a robot to perform the human tracking procedure. An initially created map that registers the motionless characteristics will be used to detect objects in motion in the environment and humans are searched using face detector. If a human has been spotted, the robot is capable of tracking the human target through the stereovision system. In addition, Kalman filter is used to help the robot track human more robustly. Human head and arms are the targets which robot focuses on.

In [26], an autonomous vision based human-following mobile robot is developed and introduced. A wireless camera is the image capturing facility of the robot. To realize the human following algorithm, MATLAB is used to process image capturing and robot controlling. To differentiate a human in a picture, the foreground and background are separated. The foreground is used to determine whether the object is a human or not. If the system can make sure the object in an image is a human, then a classification algorithm is utilized to find the centroid of the human. Later on, the centroid will be compared with the center of the picture to locate the human with respect to the camera view. Date for the centroid of human is presented by the Graphical User Interface.

In [27], a human-following mobile robot platform based on color vision combination and laser range scanning is introduced. Using a highly actuate, modified Support Vector Machine (SVM), real-time face detection in color images becomes possible. The SVM has shown its excellent face detection performance even in difficult conditions and thus seems to be a good choice in this context. After a human face is detected, laser scanner is used to track the human. The laser scanner measures the distances between human and robot and transforms the distance into scan points lying in a robot-centered Cartesian coordinate system, the scan points are clustered by

proximity and the cluster centers are calculated. The scan cluster then is selected as the human cluster to be followed for the robot. Researchers of this paper proofed the approach by some effective experiments.

In [28], a multiple distributed-camera system is created for tracking coarse human models from sequences of synchronized monocular gray scale images. The tracking process starts form a single camera view. When the active camera no longer have a view good enough to show the target, tracking will be switched to the adjacent camera which provide a better view to continue tracking. The human body is located by matching the points of middle line of the human image using Bayesian classification schemes. The paper demonstrates the feasibility of an end-to-end person tracking system that uses a unique combination of motion analysis on three-dimensional geometry in different camera coordinates.

2.3 Infrared Camera Based Human-Detection Robot

Strictly speaking, robot using infrared camera can also be treated as vision-based robot. However since infrared camera is not an ordinary camera and studied specially as a classification by researchers, the literatures related to infrared camera will be summarized in a stand-alone section. Compare with regular camera, an infrared camera will not lose its information in gentle light or even dark environment.

In [29], an infrared camera mounted on a mobile robot and a laser range finder are used for the indoor environment human tracking. To track a human, mobile robot with infrared camera always needs to combine with a filter. In this article, the Kalman filter is combined with a curve matching framework to enhance prediction accuracy of target tracking. The authors named the new filter Curve Matched Kalman Filter (CMKF). The CMKF method predicts the next possible motion of the human by taking into consideration not only the human's present motion characteristics, but also the history of target behavior patterns. For instance, if the human suddenly changes direction, the Kalman Filter may not catch up with the human immediately, but for the CMKF, due to its prediction ability, it can catch up the target.

In [30], a completely packaged infrared imaging sensor brick base on the concept of modular sensor brick is built. The infrared sensor brick consists of an acquisition block to capture images, a processing and fusion block to deal with the obtained images, a communication block to transfer data between the sensor brick and the computer, and a power block.

In [31], an approach of real-time human detection through processing video captured by a thermal infrared camera on the autonomous mobile platform is introduced. The approach starts with static analysis for the detection of human targets through some classical image processing techniques such as image normalization and thresholding. Then, a dynamic image analysis phase based in optical flow or image difference is processed. When the robot is moving, optical flow is used, while when the mobile robot is still, image difference is the preferred method. The results of both phases will be compared to emphasize the human segmentation by infrared camera. Indeed, both optical flow and image difference can enhance the foreground hot spot areas obtained at the initial human candidates' detection.

2.4 Pyroelectric Infrared Sensor Based Human-Detection Robot

Pyroelectric Infrared Sensors (PIR) is preferable for human detection since it is sensitive to the changes of human infrared radiation. A system based on PIR sensors has no limitation on the environment light conditions. It can work without visible light. Another merit for PIR sensor system is its simple algorithm. Unlike infrared camera, there is no need to design complicated vision algorithms for PIR sensor. Furthermore, the low cost of PIR sensor is also a critical reason for researches choosing to use it. PIR sensors are always used in human detection tasks in a specific room. In the room, a series of PIR sensors are installed on fixed positions with a pre-designed array. When a human is walking in the room, him position will be detected by the triggered PIR senor(s). Some researchers transplant and improve the above system so that it can be adapted to a human-detection robot.

In [4], an infrared motion sensing system for human-following robots based on PIR sensor is presented. This human motion sensing system consists of two layers in which a geometric sensing layer and a cooperative sensing layer are included. The geometric sensing layer generates bearing measurements of a moving human target from multiple perspectives through a modulation of the field of view of PIR sensor. The design objective for this layer is to build the bearing-sensitive PIR sensor arrays and the disposition strategy for creating the bearing measurements from multiple perspectives. The cooperative sensing layer's task is to localize the target human and predict the moving direction of the target so that the robot can make related actions. The design task of the layer is to establish the fusion mechanism of the bearing measurements for conjecturing the human motion location. This new sensing paradigm achieves highly sophisticated sensing capability with simple sensing patterns, which often requires with much complicated visibility patterns and is hardly implemented with a flat structure RST sensing paradigm.

In [18], PIR sensor is used in Urban Search and Rescue (USAR) robot for catastrophe and man-made disaster rescue operations. After a disaster, there might be wounded in disaster area, hence rescue tasks are needed. However some of the tasks are too dangerous and risky for rescuers or trained dogs to undertake. A USAR robot will be utilized in this case. PIR sensors can be an efficient detector to help the USAR robot make the discrimination between human and non-human presence. It is the most commonly sensor used for this application. Since the environment of disaster area is very complicated, it's impossible for a rescue robot to detect the right target with only one or two kinds of sensors. PIR sensors are always combined with other sensors in such a system due to its low cost and high efficiency.

In [3], application of passive human–robot interaction is the research emphasis. Human localization is an importance application in such a field. Hence two modalities for localizing humans based on sound source localization and infrared motion detection have been developed and integrated with the face-tracker system. The sound source localization has been presented in Section 2.1. Here the infrared motion detection part of the paper will be introduced. Five low-cost PIR motion detectors are

placed on the body of the robot to detect and track a person walking around. Each PIR sensor has a sensitivity region. The robot should decide which sensors detect a human body in which regions. The tracking algorithm makes use of the intersections and unions of these sensitivity regions. The results of the research show that the infrared motion detector provides a robust human localization function to the whole human-robot reaction system.

In [32], a lightweight and robust infrared motion localization for human-following robots is studied. The proposed localization system is capable of directly generating the bearing measurements of the moving target, which is lightweight to the environmental changes. The target's position is localized through fusing the bearing measurements from multiple perspectives with the least squares method, which does not involve complex computation. The results show that the proposed infrared motion sensing system has significant merits in the human-tracking applications.

In [33], a real-time identification system using the pyroelectric infrared sensors and hidden Markov models (HMMs) is proposed. PIR sensors are arranged into an array masked with Fresnel lens arrays. This detector array can generate digital sequential data that can represent a human motion characteristic. HMMs are taught to statistically model the motion features of individuals through an expectation-maximization (EM) learning process. Using the maximum-likelihood (ML) criterion, Human subjects are recognized by evaluating a set of new feature data against the trained HMMs. In this paper, a digital feature based system for closed-set human identification is created. PIR detector arrays are used for generating digital sequential data to represent human motion features. There are many advantages of the digital feature, for example its less rigid training process, decreased sensitivity to walking speeds, effectiveness in the path independent identification mode, and high data compression ratio for wireless data transmission.

2.5 Human-Tracking Robot in an Intelligent Space

Robots are placed in an intelligent space in which distributed sensors are arranged in this system. When a human steps into the space, all the sensors in this space will offer position data of the human to the robot by network with a pretty high accurate and efficiency. This type of system can be used in defensive, monitoring or human-assistant purposes.

In [34], a localization of mobile robot using the images by distributed intelligent networked devices (DINDs) in intelligent space (ISpace) is proposed. The system combines information from the observed position using dead-reckoning sensors and estimated position of walking human using images of moving object. The information guides the moving direction of the human-following robot. First, the position estimation of the mobile robot is quantitatively represented by the uncertainty ellipsoid. Then the real position of the human will be substituted into geometric constraint equations in the coordinates system for a given robot position. The control algorithm was proposed for the mobile robot using the linear constraint equations and the Kalman filtering technique in order to estimate and correct its position recursively and enable it to follow a walking human whose position was incompletely estimated. The proposed approach is verified by computer simulation and experiment and applied to a mobile robot in ISpace to show the reduction of uncertainty in the determining of the location of the mobile robot.

In [35] and [36], mobile robots exist in the intelligent space as physical agents, which provide human with services. Mobile robot is controlled by the Intelligent Space through its resources to follow walking human as stably and precisely as possible. The position of the human and the mobile robot in ISpace is measured with DINDs. The position dependability of the measurement errors is calculated, and the features of this noise are discussed. To reduce the measurement error, Kalman filter is utilized, and measurement error is sharply reduced. To track target objects in a wide area and control mobile robots based on environmental measurement, cooperation of the DINDs and effective communication are required. A human-following robot is

based on the measurement infrastructure of the iSpace with multiple DINDs. A proposed control with a virtual spring model was implemented as a new module in each DIND. The result shows that human following is easily achieved in the iSpace.

In [37], the research emphasis is a localization method based on interactive communication between a mobile robot and a networked laser range scanner installed in an intelligent space. Human-following control of a mobile robot is achieved with this method. The proposed system performs mutual exchange of position and heading information between the mobile robot and the networked laser range scanner. The networked laser range scanner searches the target human and the robot will estimate the position by itself. Communication between robot and the intelligent system enables the robot to receive the detection results from the networked laser range scanner. Then, the estimated position is updated and reference velocities for human-following control are calculated from the results. Estimation errors of measurement in the robot and unstable target tracking by the networked laser range scanner are compensated with this system.

In [38], mobile robots are introduced in the Intelligent Space as actuators in order to be able to provide physical services. The network of distributed sensors in the space can therefore be utilized to provide data from the space needed for the control of the robot. In the paper, implementation of an Intelligent Space system that uses spatially distributed laser range finders is introduced for tracking the mobile robot and humans in the space and building the map of the space. It is shown that laser range finders are an easy solution for these tasks. Furthermore, this paper also gives a description of the calibration and fusion processes. Finally, the controller of mobile robot is developed based on these measurements acting as physical agent of the Intelligent Space.

In [39], an approach of human-tracking in Intelligent Spaces with ubiquitous distributed sensors and actuators is described. In order to be able to implement services to humans, a reliable tracking method is needed for the robot to go to the positions of targets. Two types of sensors are installed in the intelligent space which are fixed sensors distributed in space and mobile sensors. A mobile sensor is equipped

with the mobile robot, which can be used to improve the estimate. An implementation is proposed based on laser range finders. Combining with a Kalman filter, positions of humans as well as the mobile robot are tracked. Characteristics of such a tracking system are analyzed and tested with experiments. Experimental results are shown in the paper.

In [40], a mobile robot navigation system investigate which can localize and navigate the mobile robot based on observation of walking human. The robot is designed to operate the human in the shared space with minimal disturbance to humans. Many intelligent devices are embedded in the environment to realize the system. The human walking paths are obtained from distributed vision sensors and frequently used paths in the environment are extracted. The system also supports the mobile robot navigation based on observation of human. The position and orientation of the mobile robot are estimated from wheel encoder and 3D ultrasonic positioning system using extended Kalman filter. The mobile robot moves along the frequently used paths navigated by the system.

2.6 Other Human-Tracking Strategies

There are some unique and alternative human-tracking approaches. They are rarely studied by most researchers, but they are interesting and useful in some special applications. For instance, a robot tracks a given human carrying on a unique signal source which can only be recognized by the robot. The unique source can be a WIFI emitter, Bluetooth or a light source and it gives robot the precise position of the human. Such a system is normally used as human assistant system.

In [41], a new approach in detecting position of a mobile robot using an infrared camera is presented. A Wii camera, which captures four groups of IR-LEDs installed on the robot, is attached on a human. When the camera catches the sight of all of four IR-LEDs on the mobile robot, the position and orientation of the camera, which is also the human, with respect to the mobile robot’s coordinate can be calculated. After getting camera’s position and orientation information, the information is wirelessly

transferred to a processor to guide the movement of the robot. Hence a virtual link between human and robot, which could be the distance from the robot to human, is created. The robot will move the same distance as human moved to maintain this virtual link.

In [42], the robot with a camera looks at a human with a light-emitting device attached. In order to measure the distance between the robot and the human, two LEDs are fixed on a stick and carried by the human. By taking an image of the LEDs, the robot is able to calculate the distance to the human according to the interval between the two LEDs in the image. The direction of the human can also be calculated according to the distance between the lights and the central point of the vertical axis of the image. Then such data can control the robot to follow the trajectory of the human. Results of experiments are presented in the paper to evidence the approach.

Chapter 3

Proposed Navigation and Human-Following Approaches

In this chapter, proposed navigation and human-following methods will be introduced. The proposed navigation controller will be discussed in section 3.1 and 3.2, based on quasi-harmonic potential functions. Section 3.3 will focus on the human-following control approaches of a human-following robot equipped with Passive Infrared Sensors.

3.1 Quasi-Harmonic Approach

In this thesis research, a navigation controller is needed to control the motion of mobile robots moving in an unknown environment. The approach is based on local sensing and reactive motion control with collision avoidance and final positioning at a known fixed point or an unknown moving point, which is a human followed by the robot . This thesis proposed a quasi-harmonic function based controller that uses harmonic solutions for collision avoidance and smoothly change toward a non-harmonic solution which tends toward a zero velocity command only when approaching the goal.

Quasi-harmonic function is not a harmonic function and can be seen as a function changing between harmonic function and non-harmonic function. It can be better used to control either holonomic or non-holonomic robots because it avoids weaknesses of harmonic function and non-harmonic function and combine the strengths of both two together.

This section will compare the differences among quasi-harmonic function, harmonic function and non-harmonic function to show how the navigation controller

based on quasi-harmonic function is created. And also will show different type of environment models to illustrate the applicability of the quasi-harmonic function.

3.1.1 Harmonic Function and Non-Harmonic Function

Mathematically, a real harmonic function should satisfy the Laplace equation 3.1

$$\nabla^2\Phi = 0 \tag{3.1}$$

It can be expressed as a second order differential equation

$$\frac{\partial^2\Phi}{\partial x_1^2} + \frac{\partial^2\Phi}{\partial x_2^2} + ... + \frac{\partial^2\Phi}{\partial x_n^2} = 0 \tag{3.2}$$

in which $x_1...x_n$ are variables of the function Φ.

Harmonic functions with spherical symmetry, depend only on the distance from the origin to a point and can be used to build artificial potentials for robot control. From equation 3.1, a polar coordinate equation can be derived as equation 3.3[12].

$$\nabla^2\Phi = \Phi_{rr} + \frac{n-1}{r}\Phi_r = 0 \tag{3.3}$$

where r represents the distance from the origin, Φ_{rr} is the second partial derivative of Φ with respect to r, Φ_r is the first partial derivative with respect to r, n is the dimension of vector space.

For n=2, the solution of equation 3.3 is [12]

$$\Phi = c_1 \ln r + c_2 \tag{3.4}$$

For n>2, the solution of equation 3.3 is [12]

$$\Phi = \frac{c_3}{r^{n-2}} + c_4 \tag{3.5}$$

All the simulations and experiments in this thesis can be treated as 2 dimensional cases, hence equation 3.4 can be utilized as the harmonic function model. For example, a harmonic function for 2 dimensions can be $-\ln r$.

If a function is not a harmonic function, it is a non-harmonic function. An example of a non-harmonic function for 2 dimensions is 1/r. From equation 3.4 and 3.5 it can be seen that 1/r is harmonic for n>2 but not harmonic for n=2.

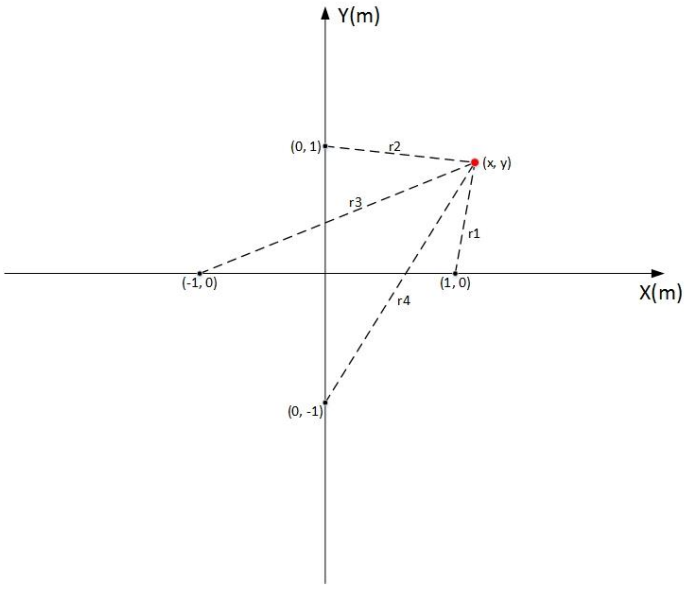


Figure 3. 1 Location of obstacles and robot

Suppose there are 4 point obstacles at point (1, 0), (0, 1), (-1, 0) and (0, -1) as shown in Figure 3.1[12]. Assume a robot's position is (x, y), then r_1, r_2, r_3 and r_4 are the distances from the robot to three point obstacles.

With the harmonic function $-\ln r$, the artificial potential field can be expressed as

$$\phi_1(x, y) = -\ln r_1 - \ln r_2 - \ln r_3 - \ln r_4 \tag{3.6}$$

And with the non-harmonic function 1/r, the artificial potential will be

$$\phi_2(x, y) = \frac{1}{r_1} + \frac{1}{r_2} + \frac{1}{r_3} + \frac{1}{r_4} \tag{3.7}$$

The potential fields based on equation 3.6 and 3.7 are shown in Figure 3.2 (a) and (b) [12].

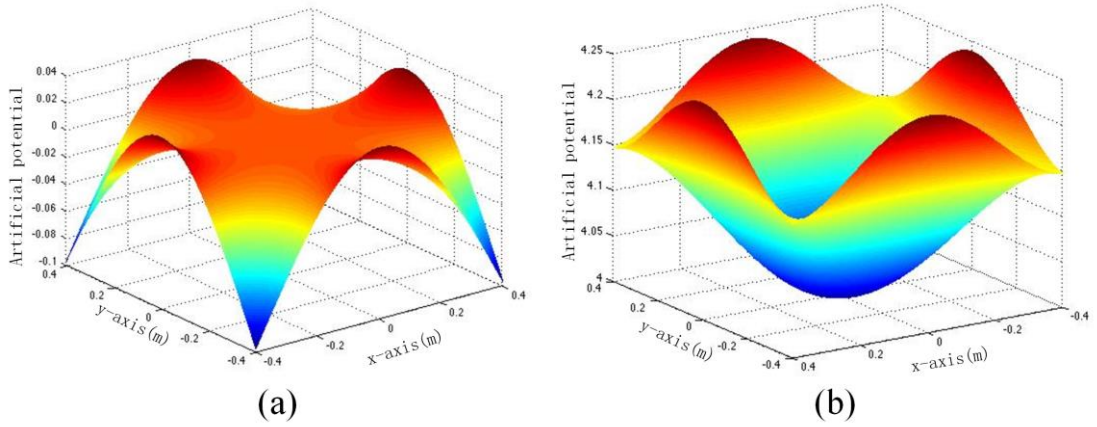


Figure 3. 2 (a) Harmonic function (b) Non-harmonic function

From the figures, it can be seen that there is a local minimum at (0, 0) in Figure 3.2 (b) for the non-harmonic function. For Figure 3.2 (a) the harmonic function, there is no local minimum and the origin (0, 0) corresponds to a stagnation point in hydrodynamics. This stagnation point is an unstable point, called saddle point.

When using artificial potential field model control a robot, obstacles are always assumed as sources and the goal is imaged as a sink in a bathtub. The sources represents repulsive forces so that robot will avoid obstacles when it gets closed to them. The sink has an attractive effect and hence the robot could move towards the goal. In this model, the robot can be imaged as a boat on a stream flows from sources to sink. There should be one and only one global minimum point which is the sink.

The reason why we do not chose a non-harmonic function for robot control function is because the possible local minimum point. In Figure 3.3, there is a goal on point (0, 10), and obstacles' positions are as in Figure 3.2. A non-harmonic function is used to compare the global minimum and local minimum. The artificial potential is

$$\phi(\mathrm{x},\mathrm{y}) = \frac{1}{r_1} + \frac{1}{r_2} + \frac{1}{r_3} + \frac{1}{r_4} - \frac{1}{r_g} \tag{3.8}$$

Where r_g is the distance from robot to goal.

We can also imagine artificial potential field in geomorphology: obstacles are peaks and goal is a valley. Assume that the robot is a little ball which placed in this environment. Then the little ball will roll to the relative low direction, namely, should be the goal. However, when we use non-harmonic function, there is a local minimum position among the three peaks even through it has not as low as the goal position. When the little ball passes through the obstacles, it may fall into the local minimum among the three peaks and never goes out again since this point is stable. Hence the little ball or, in our case, the robot will never reach the goal point.

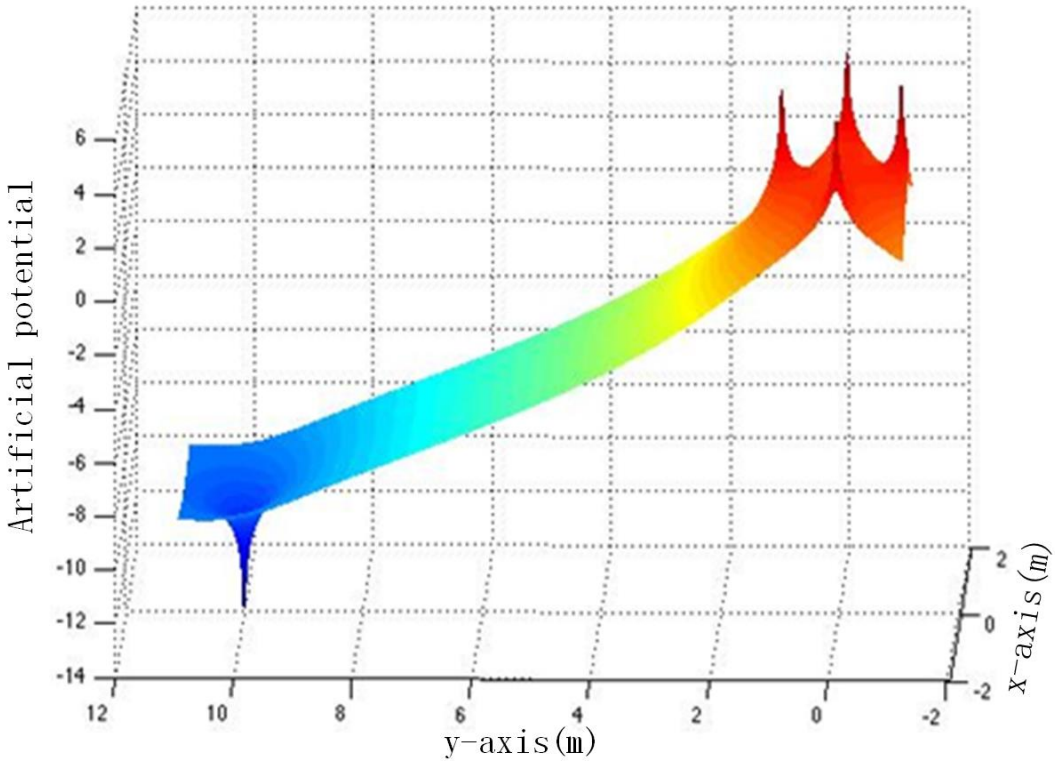


Figure 3. 3 Plot of harmonic function with obstacles and goal

3.1.2 Derivation of Quasi-Harmonic Potential Function

Harmonic functions investigated for using to control robots. However, there will be a problem when the robot is getting close to the goal using a harmonic function. The velocity field can be written as

$$V = -\nabla\Phi \tag{3.9}$$

A proposed harmonic function can be $\Phi = k\ln r$, hence for k=-1, the velocity of robot can be expressed as

$$v = -\nabla(-\ln r) = \frac{1}{r} \tag{3.10}$$

The plot of such a function is shown as Figure 3.4

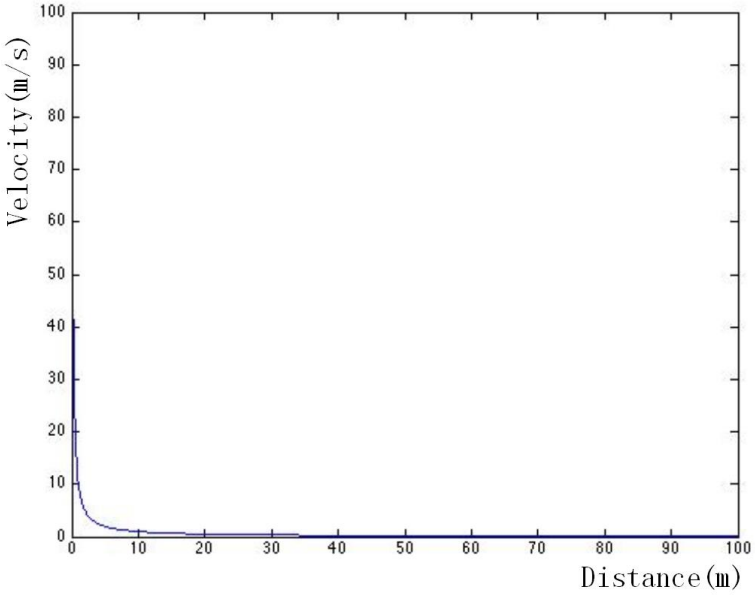


Figure 3. 4 Plot of velocity based on harmonic function

From Figure 3.4 it can be seen that when the robot gets close to the goal, the velocity of the robot is increasing, and when the robot reaches the goal, the magnitude of the velocity vector command reaches infinity. It means that the robot cannot stop at goal represented as a sink if the controller is based on harmonic function.

To solve this problem, we create the quasi-harmonic function

$$\Phi_{qh}(\mathrm{x},\mathrm{y}) = k_g \ln(\mathrm{r}) + \frac{1}{2} k_g Ei(1, \mathrm{kr}^2) \tag{3.11}$$

where r is the distance between goal and robot, k_g is a constant coefficient of the function, and Ei is an exponential integral function.

MAPLE solution shows that the second differential of $\Phi(\mathrm{x},\mathrm{y})$ is not equal to zero

$$\frac{\partial^2 \Phi_{qh}}{\partial x^2} + \frac{\partial^2 \Phi_{qh}}{\partial y^2} = \frac{(x+y+1)\left[\left(2x^2+2y^2+4y+3\right) * e^{-x^2-(y+1)^2} - 1\right]}{\left[x^2+(y+1)^2\right]^{\frac{3}{2}}} \neq 0 \tag{3.12}$$

indicating that equation 3.11 is a quasi-harmonic function but not harmonic. It can become, however, a harmonic function in specific conditions, and consequently has both characteristics of harmonic and non-harmonic in various areas of motion.

The velocity potential according to the quasi-harmonic function is

$$v_{qh} = -\nabla\Phi_{qh}(\mathrm{x},\mathrm{y}) = k_g(1-e^{-kr^2})\frac{1}{r} \tag{3.13}$$

in which kg and k are constant coefficients and r is the distance between goal and robot in polar coordinates.

This function can help the robot to smoothly slow down when it is getting close to the goal, because when r gets close to 0, the limit of vg equals to 0.

$$\lim_{r\to 0}\frac{k_g(1-e^{-kr^2})}{r} = 0 \tag{3.14}$$

When the distance increases, the Function 3.13 tends towards a harmonic function $v_g = k_g\frac{1}{r}$, corresponding to a velocity potential field $\Phi = \ln(\mathrm{r})$.

The plot of Equation 3.13 is shown below in Figure 3.5 when k_g = 3000 and k=1/400

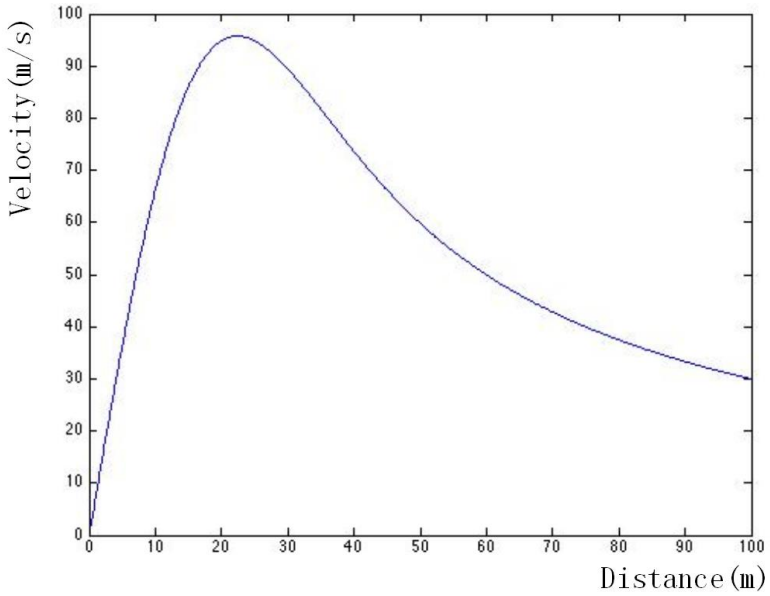


Figure 3. 5 Plot of velocity based on quasi-harmonic function

Quasi-harmonic function can also solve the saddle point problem when using harmonic function, which will be presented in section 3.1.3.

3.1.3 Comparison of Harmonic and Quasi-Harmonic Function Using Typical Illustrations

Two obstacle models which may cause local minimum of artificial potential field approach will be shown below. Mathematical and graphic results will be provided to better illustrate the comparison.

1. Two close-by obstacles

In this model the focus is on investigating a solution that solves the problem local minimum of potential field approach. The approach is presented for the simple case of a robot avoiding two close-by obstacles, too close to allow the robot to pass in-between, as shown in Figure 3.6. There are two obstacles, which are very close to each other, at (-0.1, 0) and (0.1, 0) and the goal is at (0, 1). The robot, has the coordinates (x, y), and starts moving from a position on the other side of the obstacles with regard to the goal.

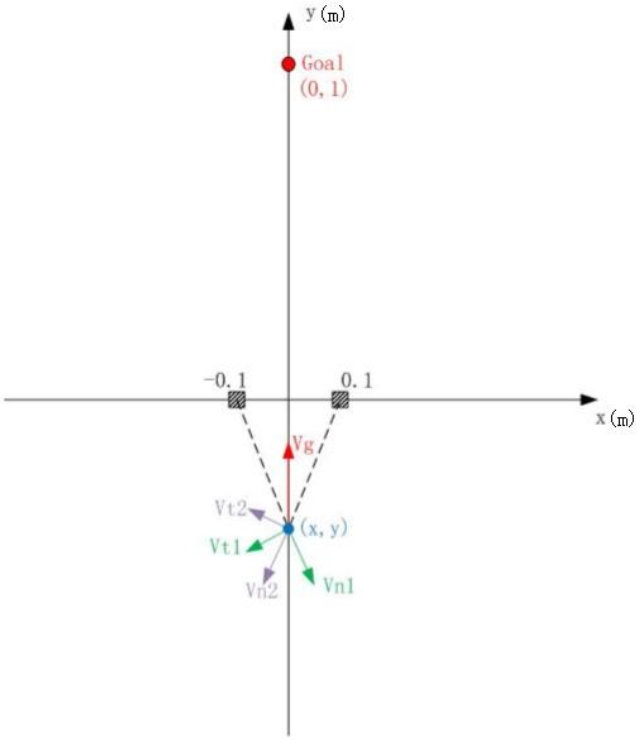


Figure 3. 6 Position of obstacles and goal and velocity commands of robot

The velocity potential field's negative gradient results in velocity vector commands Vn1 and Vn2 which are normal vectors directing the robot away from the obstacles.

The obstacles are positioned very close and the associated velocity potential fields do not allow the robot to pass in-between them. The potential field of this model using harmonic function is shown in Figure 3.7. Velocity potential fields for the case of vortexes were introduced in section III to help robot move around the obstacles and result in tangent velocity commands, Vt1 and Vt2.

From Function 3.6, the negative gradient equation set consists of partial differentials with respect to x and y and can be derived and give the first two ratios in the right hand side of Function 3.15 and 3.16, the velocity commands Vn1 and Vn2, in Cartesian coordinates, for avoiding the two obstacles. This is inspired from the case of a source in fluid dynamics [43]. The last component in Equation 3.15 and 3.16 corresponds to the velocity command to move toward the goal, inspired from a sink in hydrodynamics.

$$\frac{\partial \Phi(x,y)}{\partial x} = -\frac{x-0.1}{(x-0.1)^2+y^2} - \frac{x+0.1}{(x+0.1)^2+y^2} - \frac{x}{x^2+(y+1)^2} \tag{3.15}$$

$$\frac{\partial \Phi(x,y)}{\partial y} = \frac{y}{(x-0.1)^2+y^2} + \frac{y}{(x+0.1)^2+y^2} - \frac{y+1}{x^2+(y+1)^2} \tag{3.16}$$

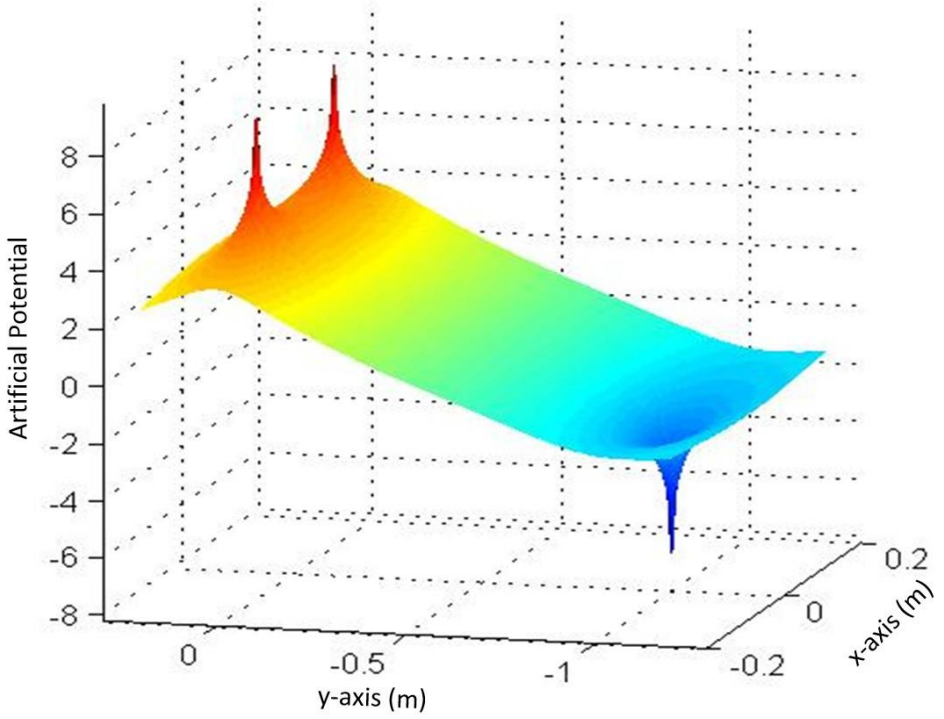


Figure 3. 7 Plot of harmonic function for model 1

When the resulting equations 3.15 and 3.16 are set to zero, the solution gives the point (x, y) whose velocity is zero, which means it can be either a minimum point or a saddle point. The resulting coordinates of this point, obtained with MAPLE, were: x=0, y=0.004987562112

Figure 3.8 illustrates that this point is a saddle point not a local minimum point.

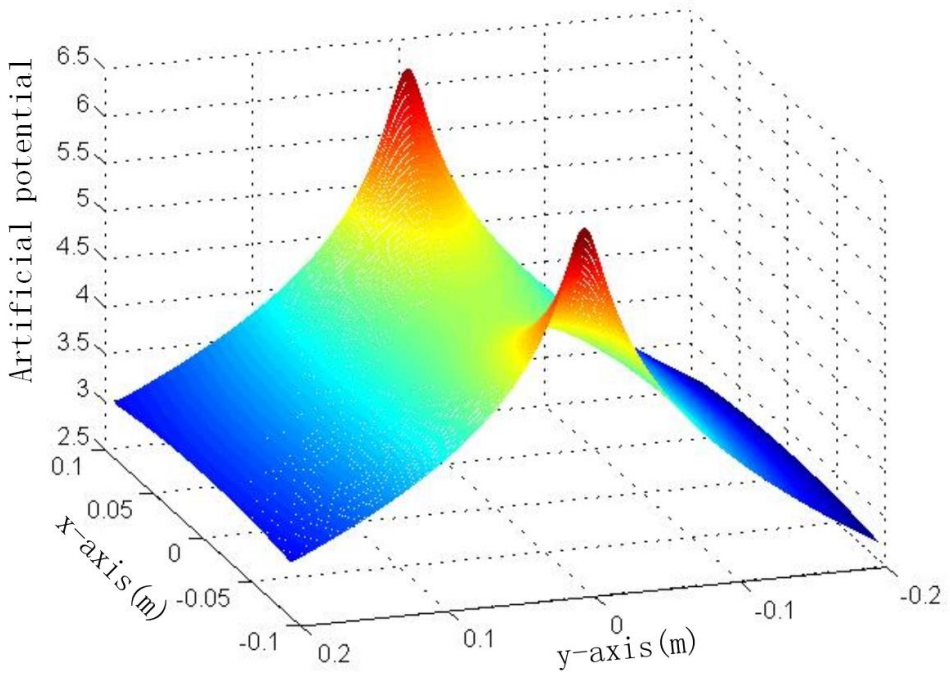


Figure 3. 8 Area near the saddle point when kt=0

The robot is in an unstable equilibrium in the saddle point. To avoid possible long duration of staying in this point and to command the motion in a desired direction, tangent velocities were added. This solution is inspired from the spiral vortex in hydrodynamics, which is known as a harmonic function, i.e. it also verifies Laplace equation [43][44].

The differential equation set becomes

$$\frac{\partial \Phi(x,y)}{\partial x} = -\frac{x-0.1}{(x-0.1)^2+y^2} - \frac{(x+0.1)}{(x+0.1)^2+y^2} - \frac{x}{x^2+(y+1)^2} - k_t \frac{y}{(x-0.1)*\sqrt{(1+\frac{y^2}{(x-0.1)^2})}*\sqrt{(x-0.1)^2+y^2}} \quad (3.17)$$

$$+k_t \frac{y}{(x+0.1)*\sqrt{(1+\frac{y^2}{(x+0.1)^2})}*\sqrt{(x+0.1)^2+y^2}} = 0$$

$$\frac{\partial \Phi(x,y)}{\partial y} = \frac{y}{(x-0.1)^2+y^2} + \frac{y}{(x+0.1)^2+y^2} - \frac{y+1}{x^2+(y+1)^2} + k_t \frac{1}{\sqrt{(1+\frac{y^2}{(x-0.1)^2})} * \sqrt{(x-0.1)^2+y^2}} \quad (3.18)$$

$$-k_t \frac{1}{\sqrt{(1+\frac{y^2}{(x+0.1)^2})} * \sqrt{(x+0.1)^2+y^2}} = 0$$

in which kt is the weight of tangent velocity component relative to the normal component.

Tangent velocity can be changed by changing the value of kt in Equation 3.17 and 3.18. Figure 3.9 shows the results when kt equals to 1.0. This solution is inspired from the spiral vortex in hydrodynamics [43][45].

In Figure 3.9 it appears clearly that there is a saddle point and no minimum except on the boundaries. This is a consequence of using harmonic functions.

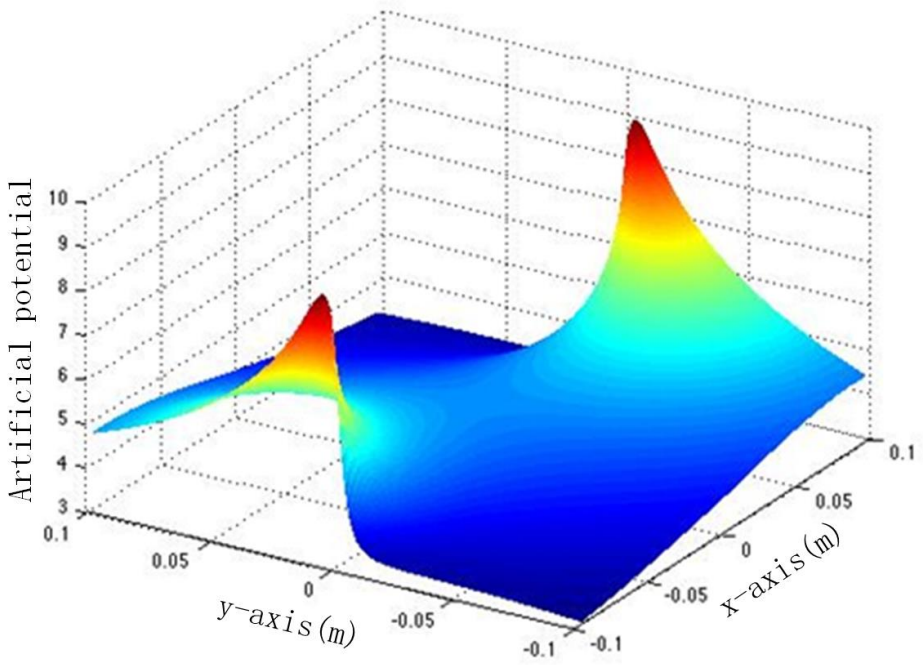


Figure 3. 9 Area near the saddle point when k_t=1.0

MAPLE solution of the equation set 3.17 and 3.18 when kt equals 1 indicates that there is the real root at (0.00249, 0.00249), near the key area and that the minimum velocity is very close to zero at this point. This non-zero value can be explained by the approximations of this numerical solution. Tangent velocities commands can set

the direction of the motion of the robot in this point. The MATLAB plot of the velocity is shown in Figure 3.10.

The coordinate of minimum velocity point moved from y-axis into the first quadrant. Both features facilitate obstacle avoidance by the robot.

A consequence of using harmonic functions inspired from hydrodynamics is that the robot would not have zero velocity command when arriving at the goal in case of using velocity potential function corresponding to an uniform flow or to a sink in hydrodynamic, $\Phi = \ln(r)$ in polar coordinates with the distance r with respect to goal [43][44][46][47] . This is one reason, in this thesis, we change the harmonic function with respect to goal into the Function 3.11.

For MATLAB plotting, it is required to define numerical values of the parameters. In Function 3.13, letting kg = 8, and k = 1, it becomes

$$v_{qh} = -\nabla\Phi_{qh}(x, y) = 8(1 - e^{-r^2})\frac{1}{r} \tag{3.19}$$

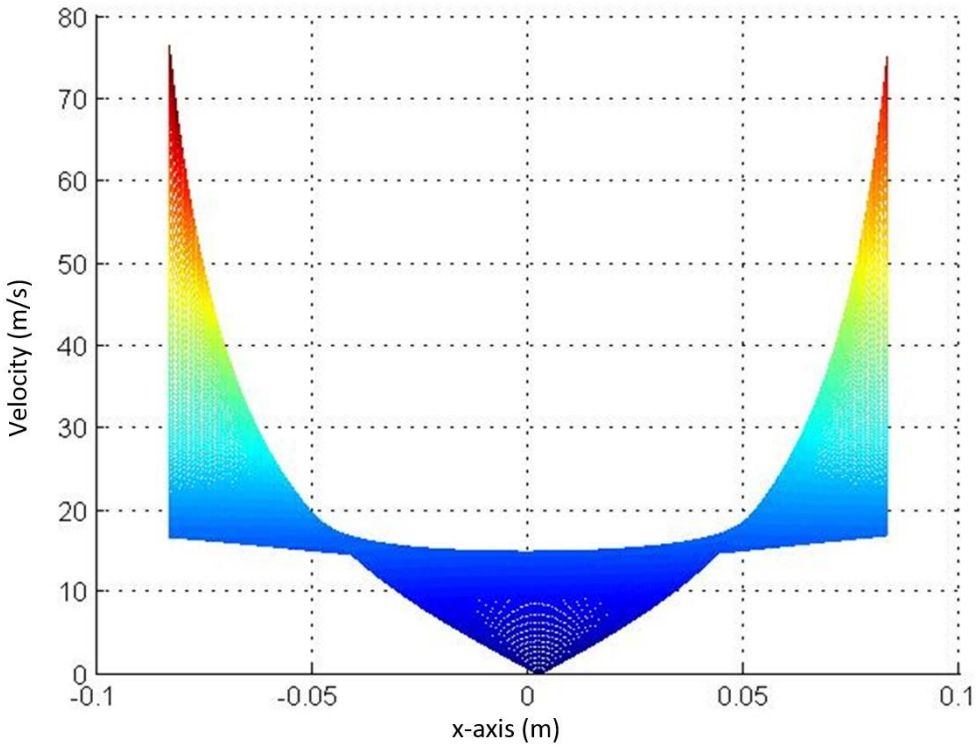


Figure 3. 10 Velocity of harmonic function in X-Z view

The resulting components of the velocity command vector in Cartesian

coordinates, resulting from the quasi-harmonic function, are

$$v_x = -\frac{x-0.1}{(x-0.1)^2+y^2} - \frac{x+0.1}{(x+0.1)^2+y^2} - 8\frac{1-e^{-r^2}}{r\sqrt{1+\frac{(y+1)^2}{x^2}}} - \frac{y}{(x-0.1)*\sqrt{(1+\frac{y^2}{(x-0.1)^2})}*\sqrt{(x-0.1)^2+y^2}} + \frac{y}{(x+0.1)*\sqrt{(1+\frac{y^2}{(x+0.1)^2})}*\sqrt{(x+0.1)^2+y^2}} = 0 \quad (3.20)$$

$$v_y = \frac{y}{(x-0.1)^2+y^2} + \frac{y}{(x+0.1)^2+y^2} - 8\frac{1-e^{-r^2}}{r\sqrt{1+\frac{(y+1)^2}{x^2}}} + \frac{1}{\sqrt{(1+\frac{y^2}{(x-0.1)^2})}*\sqrt{(x-0.1)^2+y^2}} - \frac{1}{\sqrt{(1+\frac{y^2}{(x+0.1)^2})}*\sqrt{(x+0.1)^2+y^2}} = 0 \quad (3.21)$$

in which kg = 8 is a constant coefficient giving the relative weight of the goal attractive component versus the obstacle avoidance components.

MAPLE solution of Equations 3.20 and 3.21 shows that there is no real root for this equation set. This is expected, because the equations are quasi-harmonic, but not strictly a harmonic function. The velocity in the goal point (0,-1) tends in fact toward zero. However, when the robot gets close to the goal, velocities commands caused by obstacles are very small but not equal to zero because we only replaced the harmonic function with respect to goal with a quasi-harmonic function, while functions with respect to obstacles are still harmonic functions.

To visualize the result, the velocity command from quasi-harmonic approach, in X-Z view, plotted with MATLAB, is presented in Figure 3.11. This can be used to find out if there is a point whose velocity is zero. This figure shows that there is no point where velocity is zero.

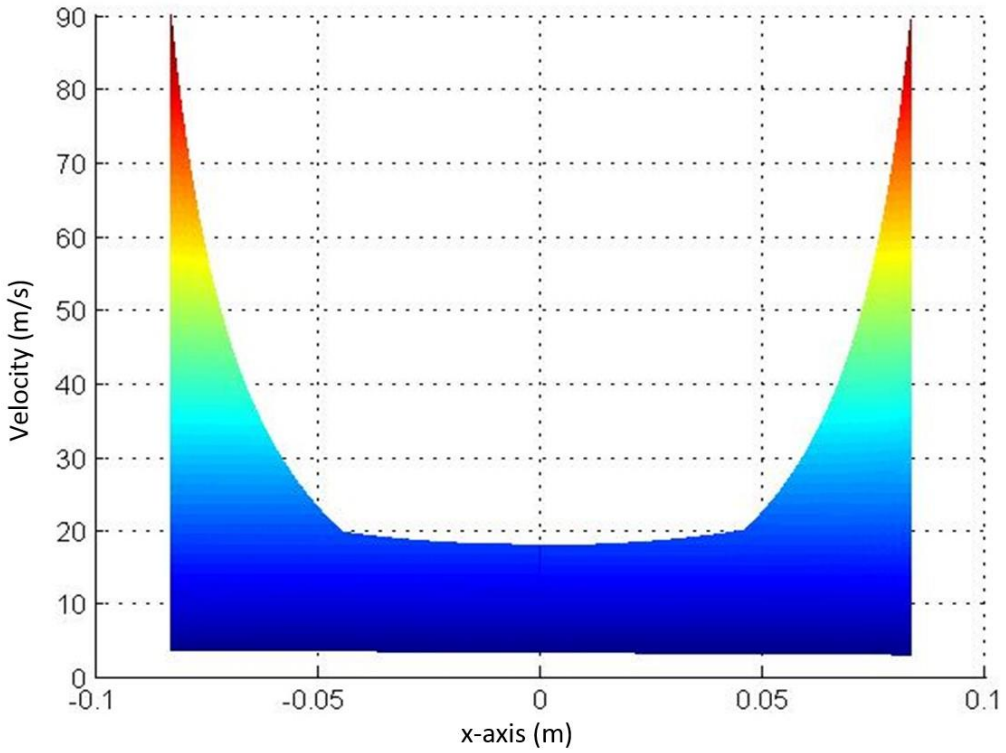


Figure 3. 11 Velocity of quasi-harmonic in X-Z view

In conclusion, both the numerical and graphic solutions indicate that there is no real number solution for the Equations 3.20 and 3.21, which means that there is no saddle point for this case.

The purpose of choosing the quasi-harmonic function was to command the robot slow down smoothly and finally stop at the goal. Figure 9 shows the velocity in Y-Z view, with low value but non-zero velocity command at the goal.

As explained above, velocities commands caused by obstacles are very small away from the obstacles but not zero because we only replaced the harmonic function with respect to goal with a quasi-harmonic function, while functions with respect to obstacles are still influencing to some extent the resulting command near the goal.

In practice, this problem can be solved by defining a range for the harmonic functions with respect to the obstacles. For instance, by imposing that when the distance between robot and obstacle is larger than a given value or the range of sensor can detect, the velocity command caused by obstacle becomes zero. In this case the velocity of robot will be zero at the goal given that Equation 3.14 and Figure 3.5 shows that the limit of velocity is zero when robot reaches the goal.

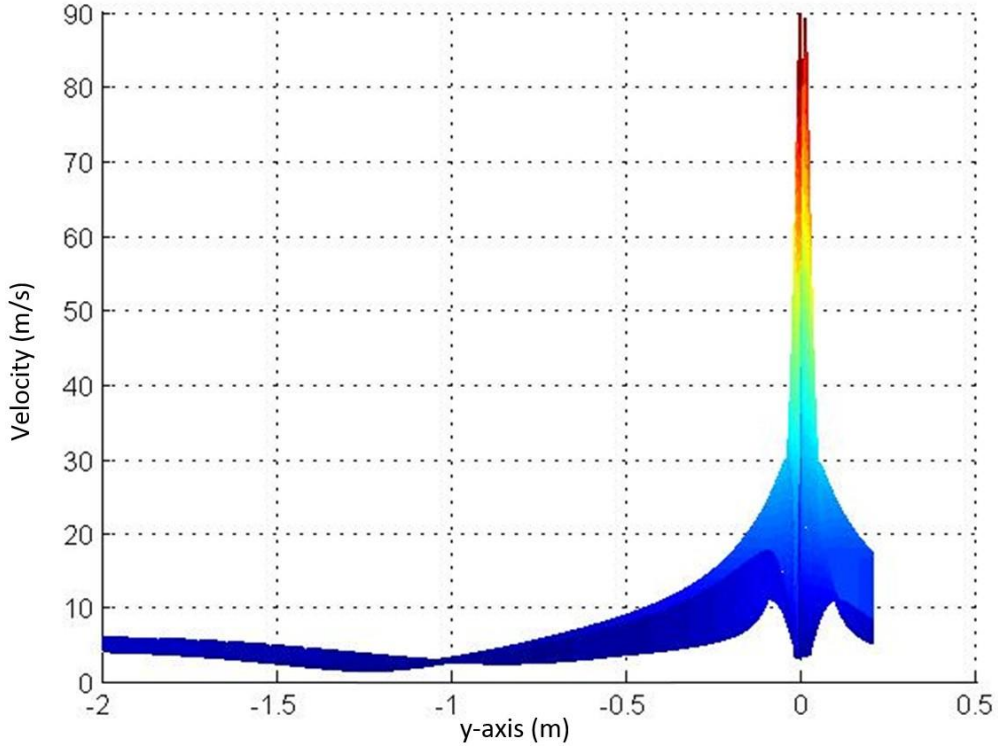


Figure 3. 12 Velocity of quasi-harmonic in Y-Z view

This change was actually implemented in the MATLAB program simulations presented in the next section.

2. Concave obstacles

The model of this environment is shown in Figure 3.13. We model the concave shape obstacles with 5 point obstacles, which will be better to simulate with MATLAB. There are 5 obstacles which are very close to each other, at (-0.2, -0.05), (-0.1, 0), (0, 0.05), (0.1, 0) and (0.2, -0.05). The goal is at (0, 1). The robot, with the coordinates (x, y), starts from a position opposed to the goal with regard to the obstacles.

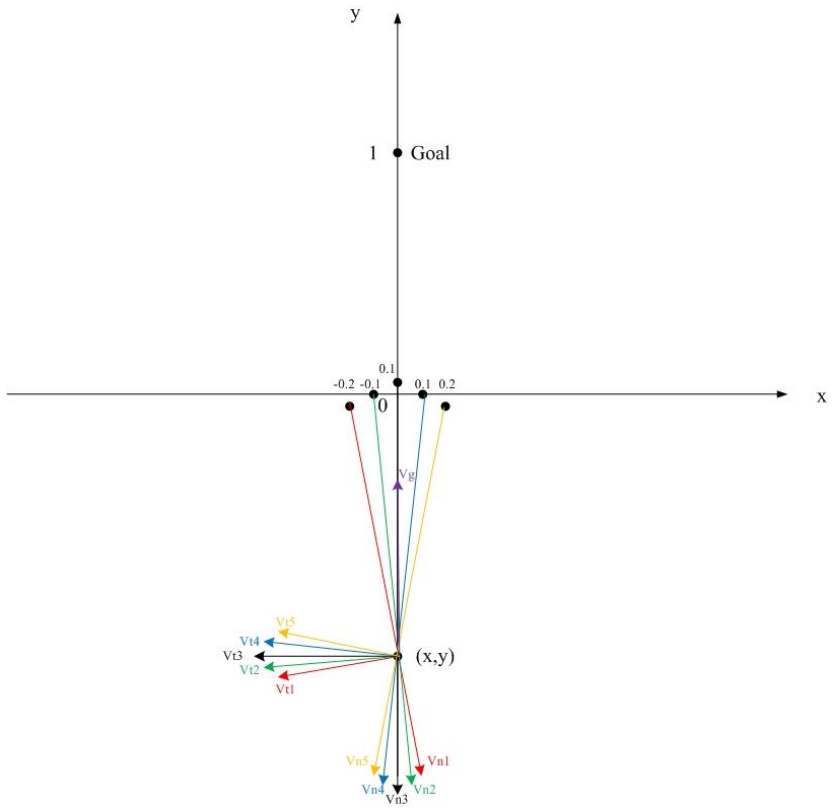


Figure 3. 13 Position of obstacles and goal and velocity commands of robot

The negative gradient of the velocity potential field results in normal velocity vector commands Vn1, Vn2, Vn3, Vn4 and Vn5 directing the robot away from the obstacles. The obstacles are positioned very close and the associated velocity potential fields do not allow the robot to pass in-between them. The potential field of this model using harmonic function is shown in Figure 3.14 where can be seen the picks of associated with the five obstacles and a sink associated with the goal. Vortexes are introduced to help the robot move around the obstacles and result in tangent velocity commands, Vt1, Vt2, Vt3, Vt4 and Vt5.

Corresponds to Equation 3.6, the controller based on harmonic function is

$$\phi(\mathrm{x},\mathrm{y}) = -\ln r_1 - \ln r_2 - \ln r_3 - \ln r_4 - \ln r_5 + \ln r_6 \tag{3.22}$$

The negative gradient equation set consists of partial differentials with respect to x and y can be derived and give the first two ratios in the right hand side of Function 3.23 and 3.24.

$$\frac{\partial \Phi(x,y)}{\partial x} = -\frac{x+0.2}{(x+0.2)^2+(y+0.05)^2} - \frac{x+0.1}{(x+0.1)^2+y^2} - \frac{x}{x^2+(y-0.05)^2} - \frac{x-0.1}{(x-0.1)^2+y^2} - \frac{x-0.2}{(x-0.2)^2+(y+0.05)^2} + \frac{x}{x^2+(y-1)^2} \quad (3.23)$$

$$\frac{\partial \Phi(x,y)}{\partial y} = -\frac{y+0.05}{(x+0.2)^2+(y+0.05)^2} - \frac{y}{(x+0.1)^2+y^2} - \frac{y-0.05}{x^2+(y-0.05)^2} - \frac{y}{(x-0.1)^2+y^2} - \frac{y+0.05}{(x-0.2)^2+(y+0.05)^2} + \frac{y-1}{x^2+(y-1)^2} \quad (3.24)$$

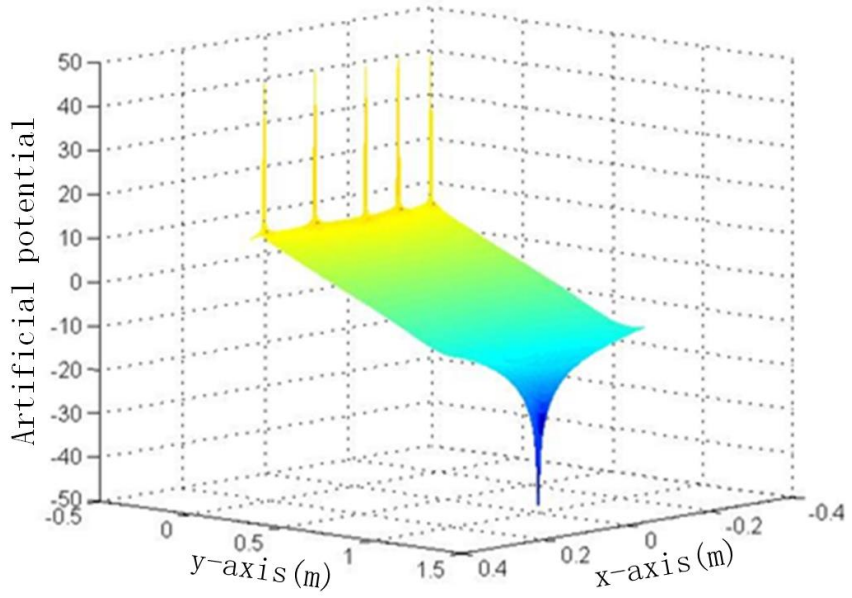


Figure 3. 14 Plot of harmonic function for model 2

The last components in Equation 3.23 and 3.24 correspond to the velocity command to move toward the goal, corresponding to a sink in hydrodynamics.

Let Equation 3.23 and 3.24 equal to zero and the solutions of this equation set give the point (x, y) whose velocity is zero, which can be associated to either a minimum point or a saddle point. The coordinates of this point, obtained with MAPLE, are:

Table 3- 1 Solution of equation set 3.23 and 3.24

	X	**Y**
POSITION 1	0.04848299620	0.01395929577
POSITION 2	0.1624623214	-0.03564531224

POSITION 3	-0.04848299620	0.01395929577
POSITION 4	-0.1624623214	-0.03564531224

As illustrated in Figure 3.15, these four points are saddle points not a local minimum points.

The saddle point corresponds to an unstable equilibrium. To avoid too long time of staying in this point, tangent velocities commands were added inspired from the spiral vortex in hydrodynamics[14].

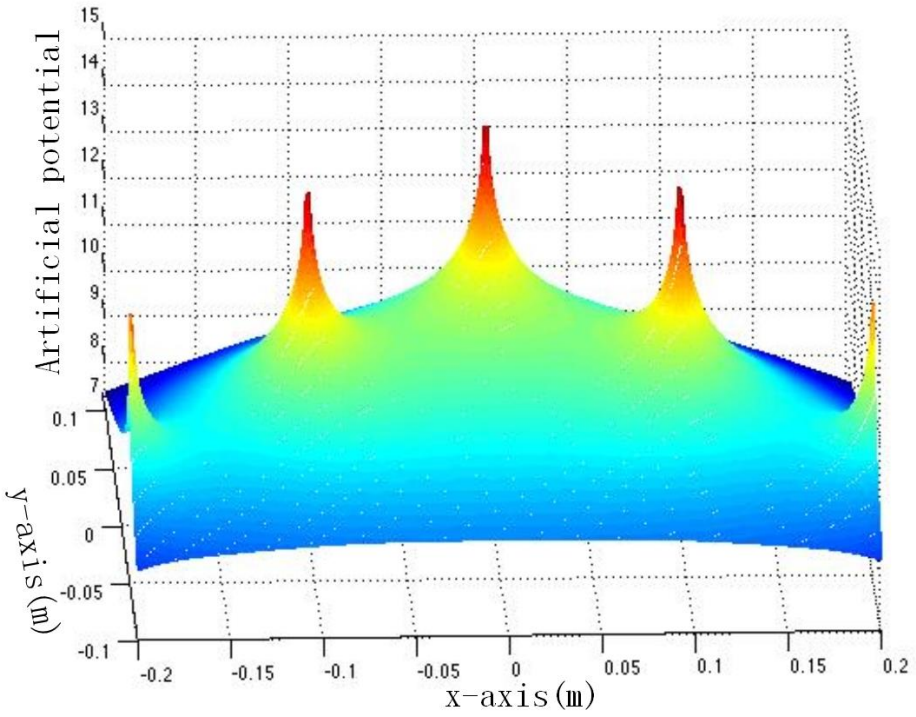


Figure 3. 15 Area near the saddle point for normal velocity commands

The differential equation set for five obstacles becomes very complicated when tangent velocities are taken into consideration. Furthermore, a similar but simpler situation had shown in first part of section 3.1.3 using a case with one saddle point. Hence we will not method the mathematical calculation again in this this part of the section.

After apply the quasi-harmonic function to the controller, the velocity potential becomes Equation 3.25

$$
\begin{aligned}
\Phi &= -\ln(r_1) - \ln(r_2) - \ln(r_3) - \ln(r_4) - \ln(r_5) + \ln(r_6) + \frac{1}{2} * Ei(1, r_6^{\,2}) \\
&+ k_1 \ln(r_1)\left[\sin|\theta_2 - \theta_1| + \sin|\theta_3 - \theta_1| + \sin|\theta_4 - \theta_1| + \sin|\theta_5 - \theta_1|\right] \\
&+ k_2 \ln(r_2)\left[-\sin|\theta_1 - \theta_2| + \sin|\theta_3 - \theta_2| + \sin|\theta_4 - \theta_2| + \sin|\theta_5 - \theta_2|\right] \\
&+ k_3 \ln(r_3)\left[-\sin|\theta_1 - \theta_3| - \sin|\theta_2 - \theta_3| + \sin|\theta_4 - \theta_3| + \sin|\theta_5 - \theta_3|\right] \\
&+ k_4 \ln(r_4)\left[-\sin|\theta_1 - \theta_4| - \sin|\theta_2 - \theta_4| - \sin|\theta_3 - \theta_4| + \sin|\theta_5 - \theta_4|\right] \\
&+ k_5 \ln(r_5)\left[-\sin|\theta_1 - \theta_5| - \sin|\theta_2 - \theta_5| - \sin|\theta_3 - \theta_5| - \sin|\theta_4 - \theta_5|\right] \\
&+ \left[\ln(r_6) + \frac{1}{2} * Ei(1, r_6^{\,2})\right]\left[\begin{array}{l}\sin(\theta_1 - \frac{\pi}{2} - \theta_6) + \sin(\theta_2 - \frac{\pi}{2} - \theta_6) + \sin(\theta_3 - \frac{\pi}{2} - \theta_6) \\ + \sin(\theta_4 - \frac{\pi}{2} - \theta_6) + \sin(\theta_5 - \frac{\pi}{2} - \theta_6)\end{array}\right]
\end{aligned}
\tag{3.25}
$$

where r_n (n=1~5) denotes the distance between robot and the nth obstacle, r_6 denotes the distance between robot and goal, $Ei(1, r_6^{\,2})$ is an integral function with respect to r_6 and θ_n denotes the direction of robot's normal velocity with respect to obstacles or goal.

The partial differential of equation 3.25 with respect to either x or y is more than one page can show, hence it will not appear in this paper. And there is no analytical solution for this equation set. Hence only graphic results will be shown.

Figures 3.16-3.18 show the potential filed of the quasi-harmonic function with different coefficients of tangent velocity. From these figures it can be seen that the slope will getting shaper when the tangent velocity getting larger.

Figure 3.18 indicates a significantly higher slope on one side. This higher slope directs the robot along the tangential velocity command and the robot will not stay any more for a while in an unstable equilibrium in the middle between the two obstacles.

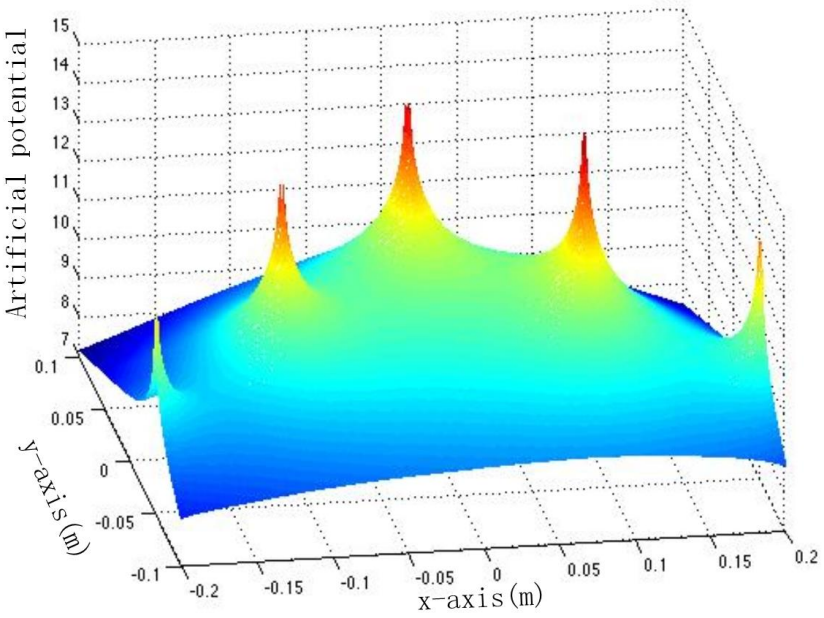


Figure 3. 16 Area near the saddle point when kt=0.2

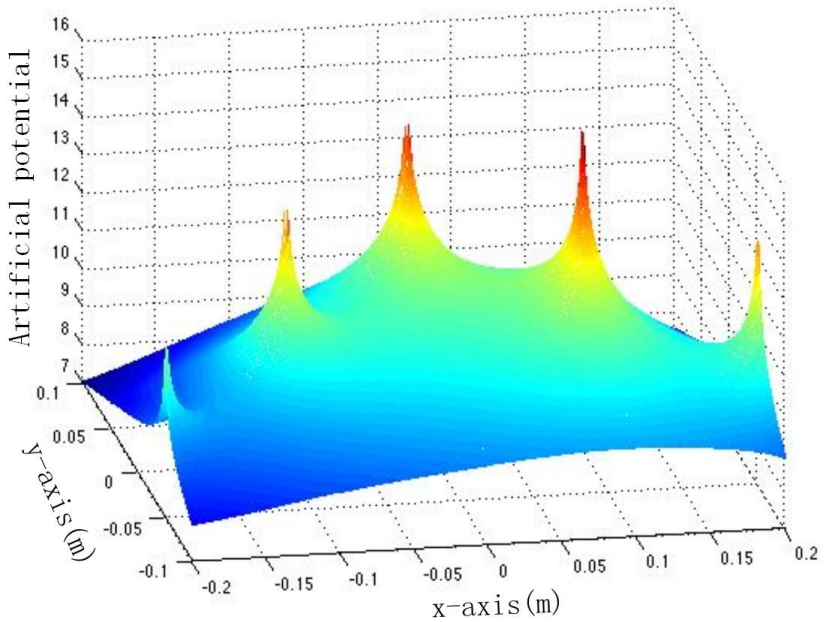


Figure 3. 17 Area near the saddle point when kt=0.4

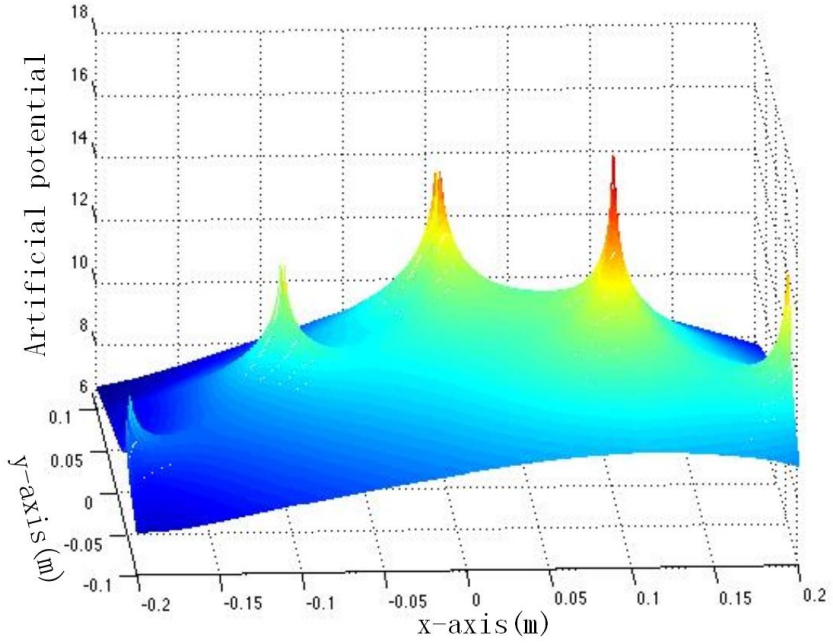


Figure 3. 18 Area near the saddle point when kt=0.8

From the section 3.2 it can be shown that quasi-harmonic function can be better used to control a robot. In chapter 7, simulations for both holonomic and non-holonomic robots will be presented using quasi-harmonic function.

3.2 Human-Following Approach

In this thesis, our objective is to let a non-holonomic robot have the ability to follow a human in an unknown environment with obstacles. The above section has shown that robot is capable to avoid obstacles in complicated environments when using a quasi-harmonic function. Now let's assume the following task: a human is going to finish some activity in a goal area and only the human knows where the goal is. The robot's task is to detect and find where the human is and follow the human until it reaches the goal area. To finish this task successfully the robot should first find the human and then catch up with the human rapidly. When the robot gets closed to the human, it has to slow down. At the same time, the robot needs to avoid obstacles to make sure it can continue following the human without colliding with obstacles.

The figure of this task is shown below in Figure 3.19, where the magenta circle and diamond represent the goal area and goal, the car-like frame represents the non-holonomic robot, a small circle represents the human, the obstacles by squares and the goal by a diamond.

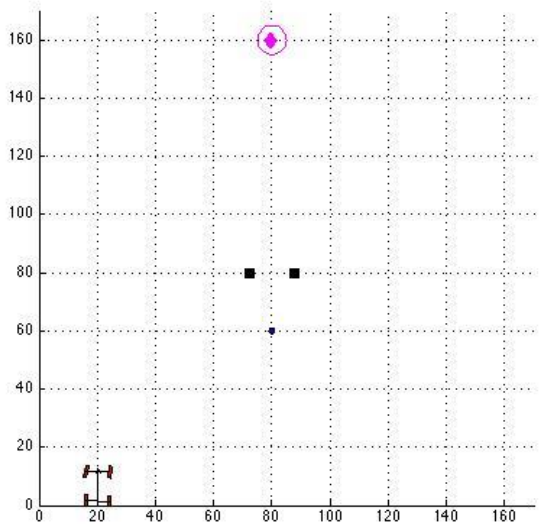


Figure 3. 19 Description of the task

The human is a goal when the robot finds him and is far from him, but becomes to an obstacle when the robot is getting too close, as figure 3.20 and 3.21 show.

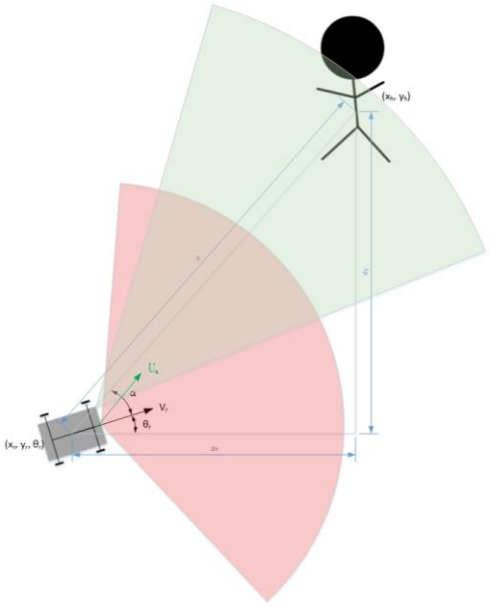


Figure 3. 20 robot found and far from the human

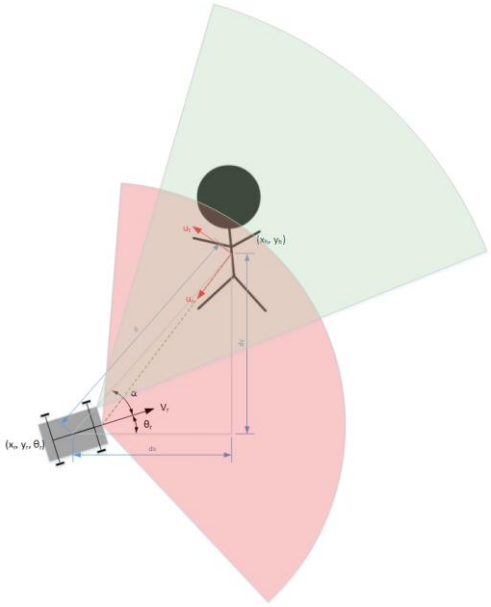


Figure 3. 21 robot is getting too close to the human

Where the four-wheel vehicle represents the robot, the green area represents the Passive Infrared Sensor's (PIR sensor) detection range and the red area is the ultrasonic sensor's detection range.

The working principle can be simply described as Figure 3.22 [48].

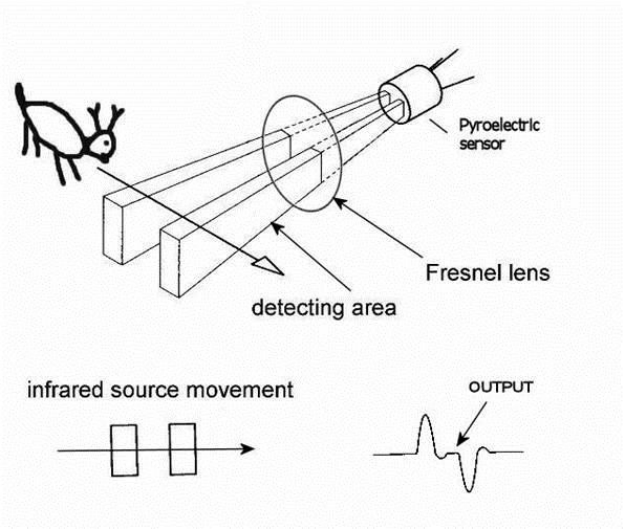


Figure 3. 22 Working principle of PIR sensor

To detect the human, a passive Infrared Sensor (PIR Sensor) is used.

All objects with a temperature above absolute zero emit heat energy in the form of radiation. In most conditions this radiation is infrared wavelengths and hence is

invisible to the human eye. To detect such a radiation, electronic devices such as PIR sensor is needed. It is important to note that PIR sensors don't detect or measure "heat", instead they detect the Infrared radiation emitted from an object which is different from but often associated/correlated with the object's temperature [48].

Since the PIR sensor detects the change of temperature, if a human have a relative velocity with the robot, there should be several conditions as follows:

(1) When the human moves to the right relative to the robot, as Figure 3.23(a), the PIR sensor will react as a positive value.

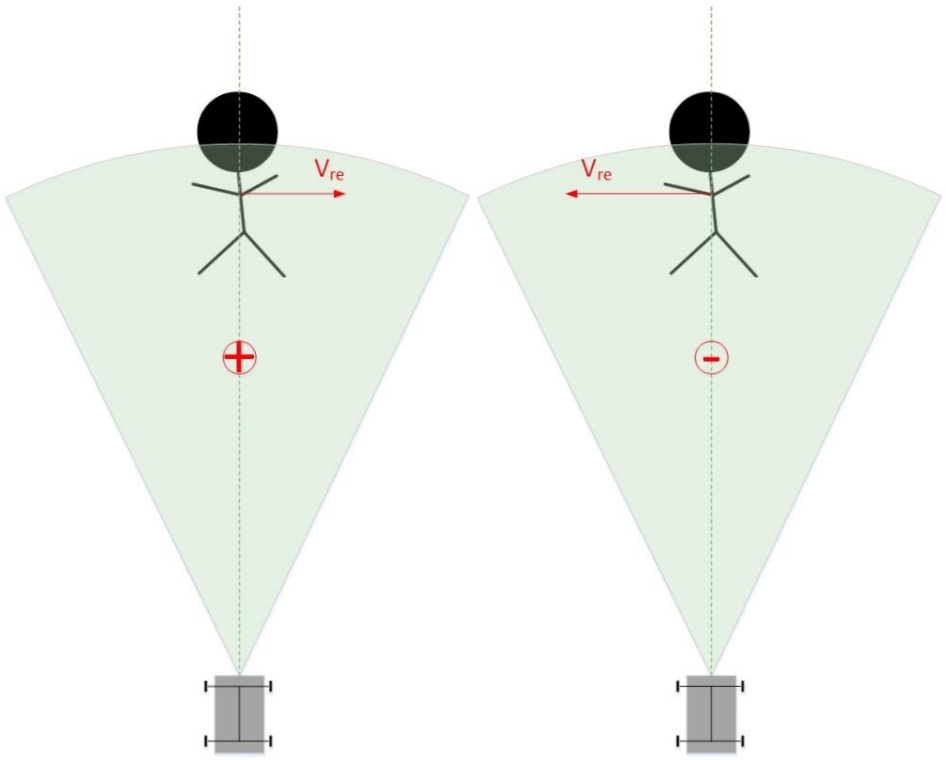
(2) When the human moves to the left relative to the robot, as Figure 3.23(b), the PIR sensor will react as a negative value.

(3) When the human is detected by the PIR sensor with the right zone of the sensor and moving forward relative to the robot, as Figure 3.23(c), the PIR sensor will react positive.

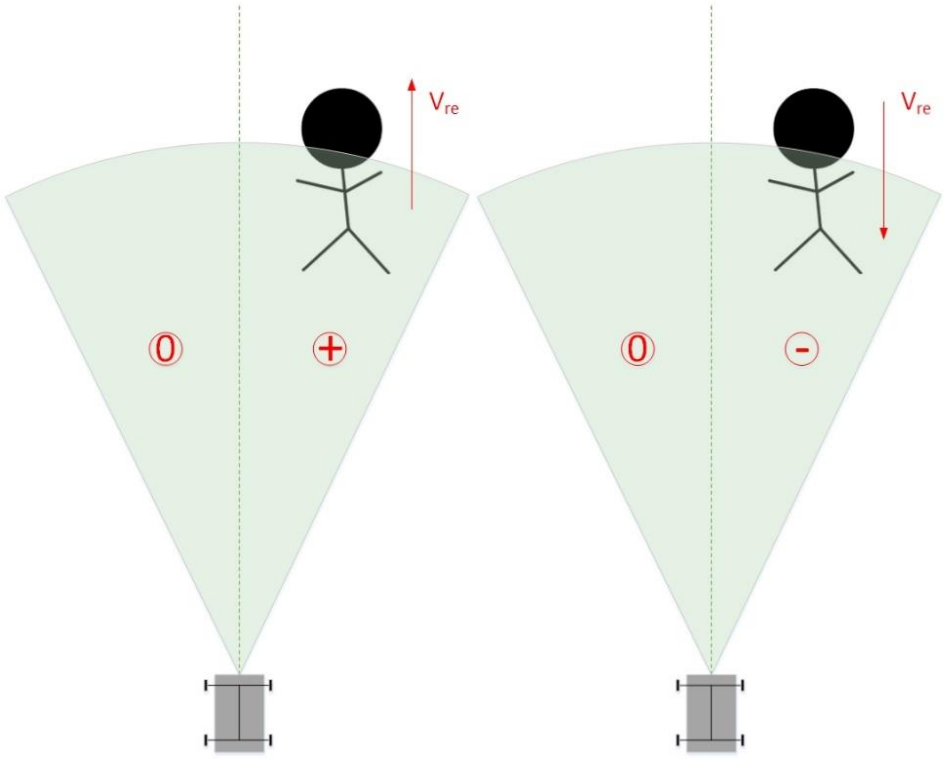
(4) When the human is detected by the PIR sensor with the right zone of the sensor and moving backward relative to the robot, as Figure 3.23(d), the PIR sensor will react negative.

(5) When the human is detected by the PIR sensor with the left zone of the sensor and moving forward relative to the robot, as Figure 3.23(e), the PIR sensor will react positive.

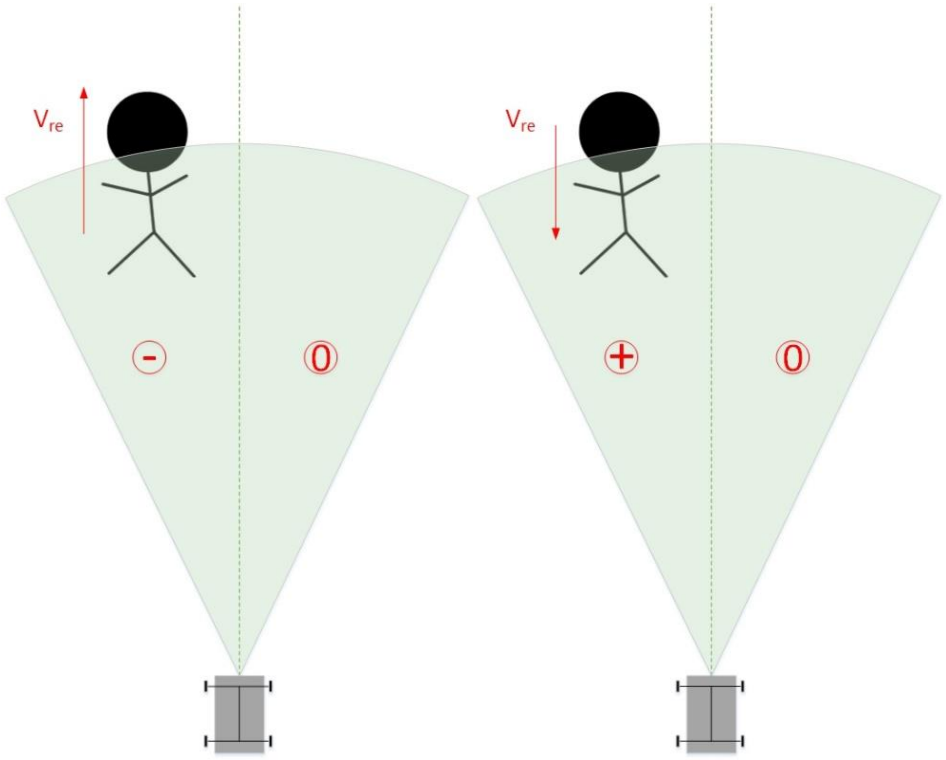
(6) When the human is detected by the PIR sensor with the left zone of the sensor and moving backward relative to the robot, as Figure 3.23(f), the PIR sensor will react positive.



(a) (b)



(c) (d)



(e) (f)

Figure 3. 23 Relationship between relative motion and sensor value

From the above introduction, we know that the robot can follow a human according to the relative motion between human and robot even if there is only one PIR sensor on the robot.

However, if the robot needs to keep a safety distance from the human and also have a stronger and more accurate ability to detect and follow the human, a PIR sensor array need to be used.

In this thesis work, we used a three-PIR sensor array to detect human. Three PIR sensors divide the detection range into several parts. According to these parts, the robot will get the pose information of the human and decide what kind of motion it will take next. Hence the human motion localization will be addressed in the robot frame.

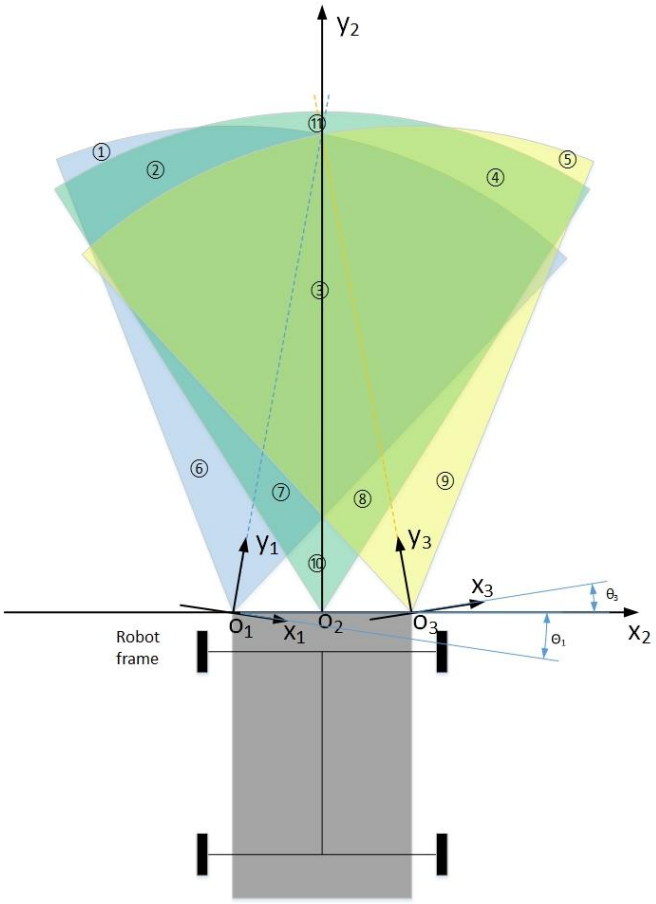


Figure 3. 24 Deployment configuration of the PIR sensor array (method 1)

In the following, the deployment of PIR sensor array will be introduced in the robot frame. There are two kinds of deployment for PIR sensors, as Figure 3.24 and Figure 3.25 show. We will compare the two kinds of configurations to see which is better for our research. Assume the blue, green and yellow areas in these two figures represent the search coverage of PIR senor 1, 2 and 3 (left, middle and right). The difference between the configurations in figure 3.24 and 3.25 is the angle of installation of PIR sensor 1 and 3. In Figure 3.24, the installation angle θ_1 is negative and θ_3 is positive, while in Figure 3.25, the installation angle θ_1 is positive and θ_3 is negative.

The PIR sensor array in Figure 3.24 divides the detection area into 11 parts, which can help robot get a more accurate position of the human. According to the relative position of human and robot, the robot can make an accurate reaction in this model. However, the disadvantage is that the detection zone is not wide enough which is bad for searching for the human in an unknown environment.

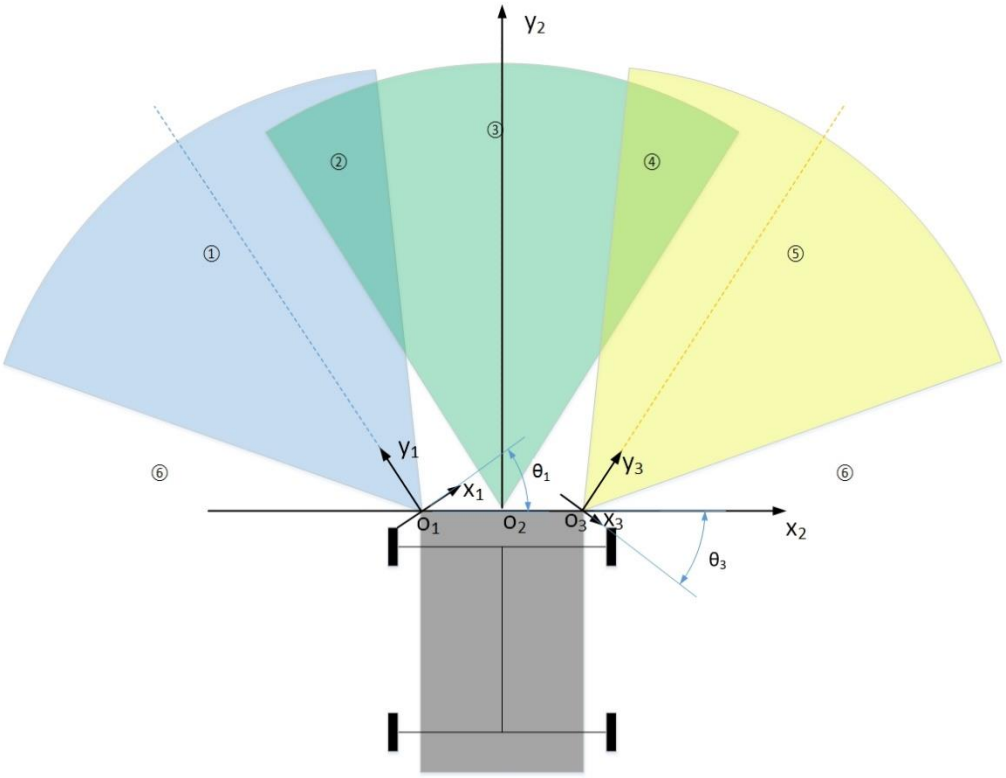


Figure 3. 25 Deployment configuration of the PIR sensor array (method 2)

The PIR sensor array in Figure 3.25 is much simpler as it divides the detection area into only 6 parts. However it has a wider detection range compared with the last configuration.

Both of the two configurations have their merits and weaknesses. Either of them can be chosen as human-following robot PIR sensor array according to different requirement and external conditions. In this thesis, the second model is chosen due to the limitation of experimental equipment. The detection range of the PIR sensor we used is very small, and human cannot be treated as a mass point, instead human is a very large objective compared with the detection area. Hence if model in Figure 3.24 is chosen, the accurate divided 11 areas will not be accurate any more since human may be sensed in several areas at the same time. On the other hand, model in Figure 3.25 can be used in such a facility condition. Details about experiments will be presented later in chapter 8.

The sense and control consists in this case of open loop commands for the robot, based on configuration in Figure 3.25, which are as follow.

When only PIR sensor 1 (left side) detects the human, human is in area 1 in Figure 3.25, and robot turns to left with a steering angle 1.

If both PIR 1 and PIR 2 detect the human, it means human's position is get close to the middle detection area (area 3 in Figure 3.25) of the robot. Under this condition, robot turn left with a smaller steering angle 2.

When human is detected only by PIR sensor 2, it means the human stands in area 3 in Figure 3.25, hence the robot will go straightly towards to the human.

Similarly, when only PIR sensor 3 detects the target, robot turns right with angle 1, and when PIR 2 and PIR 3 detect the target, robot turns right with angle 2.

A special condition is that when all three PIR sensors detect the surface of human body. Since the human is not a mass point in the experiment, the human body might be appear in all of the 2, 3 and 4 areas from Figure 3.25. This means the robot is too close to the human. Since there should be a safe distance between human and robot, in this situation, the robot needs to break (stop) until the distance between human and robot is getting larger than the safe distance.

If none of the PIR sensor detect the target, then the human is not in the robot's detection space. In this case, robot will move and search around until it detect the human.

Table 3.2 shows the robot motions with respect to sensors.

Table 3- 2 Robot motion with respect to sensors

Enabled Sensor(s)	Robot Motion
None	Rotation
PIR Sensor 1	Turn Left with Angle 1
PIR Sensor 2	Turn Right with Angle 1
PIR Sensor 3	Go straight
PIR Sensor 1 and 2	Turn Left with Angle 2
PIR Sensor 2 and 3	Turn Right with Angle 2
PIR Sensor 1 and 3	——
PIR Sensor 1, 2 and 3	Break

Chapter 4

Experimental Setup

Hardware was chosen and designed to realize robot control methods and algorithms thorough experiments. According to our control method and equipment in our laboratory, LEGO MINDSTORMS NXT 2.0 robot is chosen given that it can achieve our objectives.

LEGO MINDSTORMS NXT 2.0 is a programmable robotics kit released by LEGO on 5 August 2009. It contains 619 pieces (includes sensors and motors), two Touch Sensors, an Ultrasonic Sensor, a new Color Sensor. The NXT 2.0 uses Floating Point operations whereas earlier versions use Integer operation.[48] There are also some third-party companies produce components, especially sensors, for LEGO NXT robot. Hence numerous demands can be realized with LEGO NXT 2.0 even if it is a toy robot.

Another significant strength is the mechanical structure of LEGO robot. Not like other robots, LEGO's mechanical structure is easy to design, architecture and change according to different experiment demands because of the classical LEGO BRICK design.

Furthermore, LEGO offers a friendly programming environment for hobbyists, students, researchers or any other users. Its programming is command box programming, rather than code programming. However code programming is also welcomed if in request. LEGO supports many kinds of languages such as [49]:

- RCX Code (included in the MINDSTORMS consumer version sold at toystores)
- ROBOLAB (based on LabVIEW and developed at Tufts University)

Also some popular third-party languages are supported [49]:

- GNAT GPL: Allows programming NXT using the Ada language for real-time and embedded programming.
- LeJos: A port of Java

- Not eXactly C: (NXC), an open source C-like high-level programming language, download compiler from http://bricxcc.sourceforge.net/nxc/
- Not Quite C: (NQC)
- RoboMind: Simple educational scripting language for virtual and LEGO NXT robots.
- ROBOTC: C-Based Programming Language with an Easy-to-Use Development Environment.
- Simulink: Graphical Signal Processing and Control Design tool from which C code is auto-generated and deployed onto the NXT.
- pbFORTH: Extensions to Forth
- pbLua: Version of Lua
- Visual Basic: Via the COM+ interface supplied on the CD
- C# with Microsoft Robotics Developer Studio

This chapter describes the hardware components of LEGO NXT 2.0 robot platform as well as the mechanical structure design with respect to our experiments. Section 4.1 gives an introduction of the NXT Intelligent Brick of LEGO robot. Section 4.2 describes the sensors which were chosen and utilized in experiments. Section 4.3 is a brief introduction of robot's three servo motors. Section 4.4 describes the networking setup which enables LEGO NXT to connect to a personal computer. Finally, section 4.5 gives an overview of the robot mechanical structure design and architecture.

4.1 NXT Intelligent Brick

NXT Intelligent Brick AKA (Ciara) is the brain of the mobile robot. It's a brick-shaped computer which is the most significant component in the kit. Figure 4.1 and 4.2 [50] illustrate details of NXT Intelligent Brick.

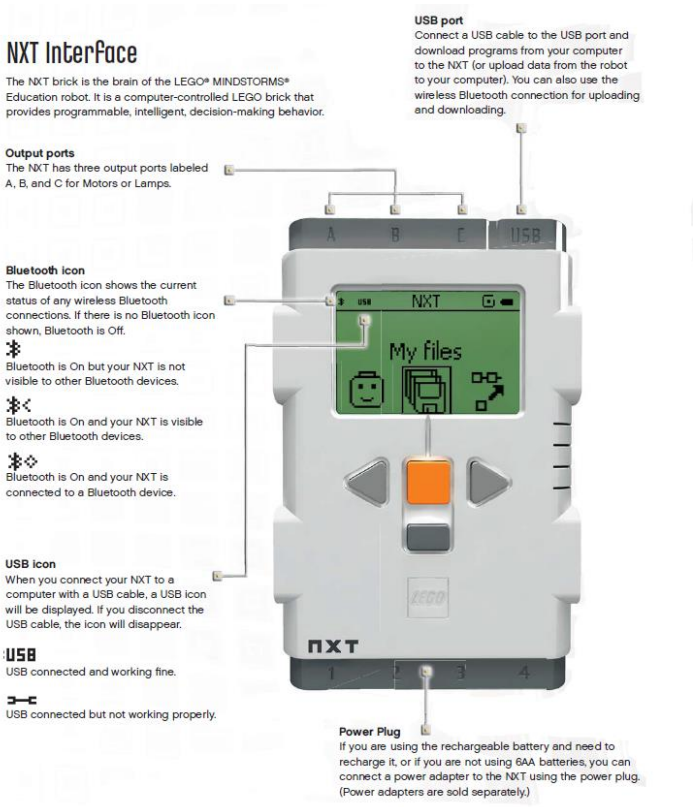


Figure 4. 1 Illustration of NXT intelligent brick 1

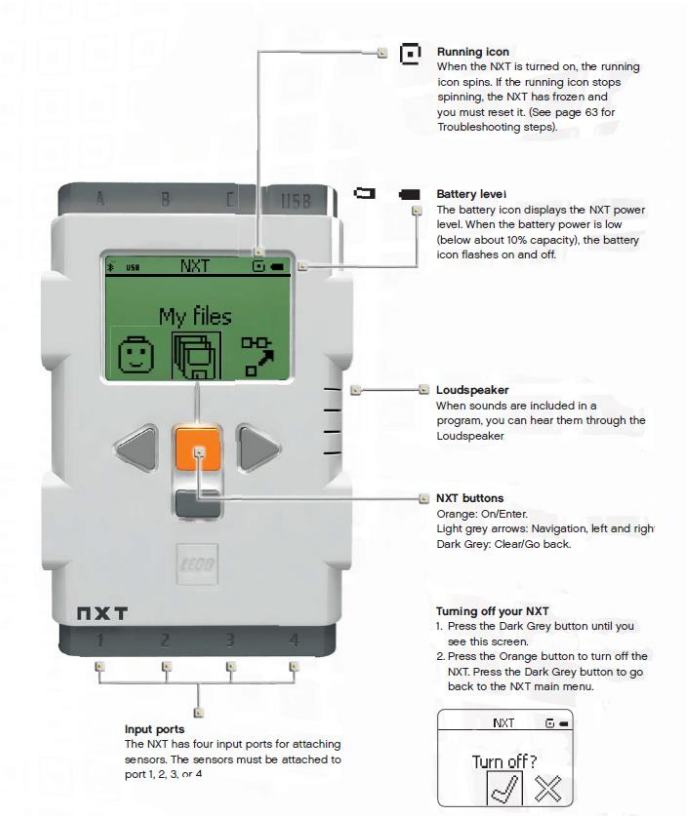


Figure 4. 2 Illustration of NXT intelligent brick 2

The details regarding NXT Intelligent Brick's functions are listed as follow [51]:

- 32-bit Atmel AT91SAM7S256 main microcontroller (256 KB flash memory, 64 KB RAM)
- 8-bit Atmel ATmega48 microcontroller @ 4 MHz (4 KB flash memory, 512 Bytes RAM)
- 100x64 pixel LCD Screen
- four 6-pin input ports (ports 1-4)
- three 6-pin output ports (ports A-C)
- USB Port
- Bluetooth Class II V2.0
- Loudspeaker - 8 kHz sound quality, 8-bit resolution, 2–16 kHz sample rate
- Four Push Buttons
- Orange button: On/Enter
- Light grey arrows: moving left and right in the NXT menu
- Dark grey button: Clear/Go back
- Powered by six AA batteries or the NXT Rechargeable DC Battery

The NXT Intelligent Brick can take input from up to four sensors and control up to three motors. In the experiments for the thesis, four input ports are connected to 3 PIR sensors and an ultrasonic sensor and all three output ports are connected to 3 servo step motors. In subsidiary experiment, one of the input ports is also connected to a Thermal Infrared Sensor (TIR Senor). Details of sensors will be introduce in the following section. During experiment, the LCD screen is used to display important data captured from sensors.

4.2 Sensors

Sensors have the ability to detect external environment, collect information and then input theses data to a processing unit of an instrument so that the instrument can make corresponding reaction according to the input data. For an autonomous robot, the sensors are crucial since they guide the robot in an unknown and dynamic

environment to complete tasks. For an analogy, sensors to a mobile robot are as the sense organs to a human being.

There is a series of exteroceptive sensors which can collect information from surrounding environment for LEGO NXT robot. In the original kit, only color sensor, touch sensor and ultrasonic sensor are included. Additionally, the LEGO Company also officially produces light sensor, sound sensor, compass sensor, accelerometer sensor, RF-ID sensor and so on. Furthermore, some third-party companies also produce unofficial bricks, especially sensors, for LEGO NXT robot. For instance, the HiTechnic Company create a series of NXT sensors in which angle sensor, acceleration/tilt sensor, barometric sensor, force sensor, gyro sensor, magnetic sensor, PIR sensor, etc. are included. The Dexter Industries Company produced Thermal Infrared Sensor, dThermometer sensor, dPressure sensor, dFlex sensor, etc. for LEGO NXT robot.

In the experiments for this paper, 4 kinds of, 8 sensors are utilized actively or passively. Table 4-1 list the sensors' usage conditions.

Table 4- 1 Sensor usage

Sensor Name	Quantity Used	Usage
Ultrasonic Sensor	1	Obstacle Avoidance
PIR Sensor	3	Detect Temperature Difference
TIR Sensor	1	Detect Temperature
Rotation sensor	3	Measuring how far motor has turned. Built into Servo Motors.

Detailed presentation of these sensors will be done in the next sections.

4.2.1 Ultrasonic Sensor

Sonic wave is created by the vibration of objects. The frequency range that human can hear is between 20 Hz-20 kHz. If the frequency is higher than 20 kHz, it turns into ultrasonic wave. The ultrasonic frequency used most is from tens of kHz to tens of MHz. There are refraction and reflection phenomenon when ultrasonic wave meets

different medium. Ultrasonic sensor is deigned based on the reflection feature of the ultrasonic wave. Figure 4.3 shows the features of an ultrasonic sensor.

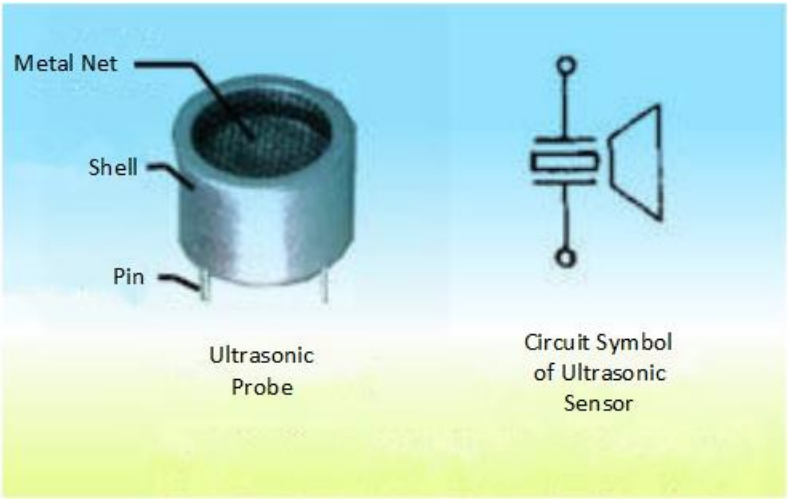


Figure 4. 3 Ultrasonic probe and it's symbol

An ultrasonic sensor is composed by an ultrasonic transmitter, an ultrasonic receiver, control circuit and power source. The foundation of ultrasonic ranging is echo sounding method, as Figure 4.4 shows.

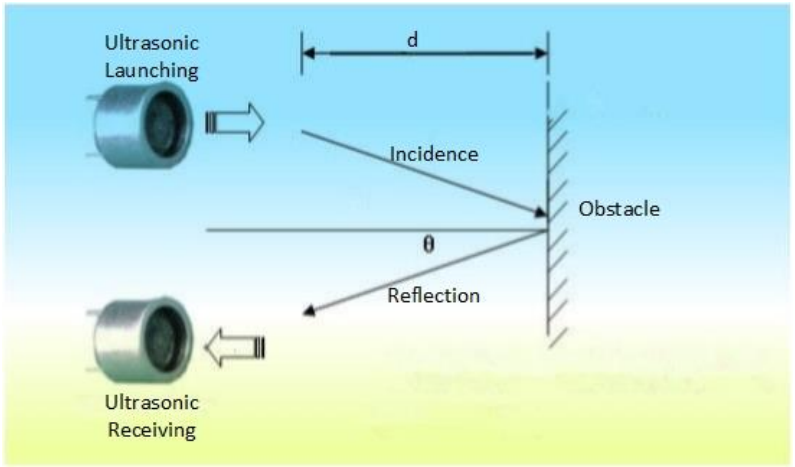


Figure 4. 4 Theory of ultrasonic ranging

Ultrasonic transmitter launches ultrasonic wave in a given direction and at the same time the timer starts to reckon by time. When the ultrasonic wave meets with a different medium, in other words, an obstacle, it reflects and spread back to the ultrasonic receiver. When the ultrasonic receiver catches the reflected wave, the timer stops immediately. Since the speed of ultrasonic wave is known in the air medium, the

distance between obstacle and the ultrasonic transmitter d can be calculated by Equation 4.1.

$$d = \frac{v * t}{2} \tag{4.1}$$

where t is the time period measured by the timer and v is the velocity of ultrasonic wave in the air medium and can be expressed as Function 4.2. [52]

$$v = 331.3 + 0.606T \tag{4.2}$$

where T is the air temperature in degrees Celsius.

The LEGO ultrasonic sensor is shown in Figure 4.5. The detection range for this sensor is theoretically from 1 to 250 cm, and the directivity, as Figure 4.6 shows, is approximately 30 degrees.

Figure 4. 5 LEGO ultrasonic sensor

In the experiments for this work, we need the field view of robot's obstacle had to be maximum 120 degrees, which is from -60 degree to 60 degree. Hence at least 3 ultrasonic sensors are needed. However, there was only one ultrasonic sensor available in laboratory, only 4 input ports on the NXT Intelligent Brick and, in any case, 3 of the 4 input ports are used as PIR sensor inputs.

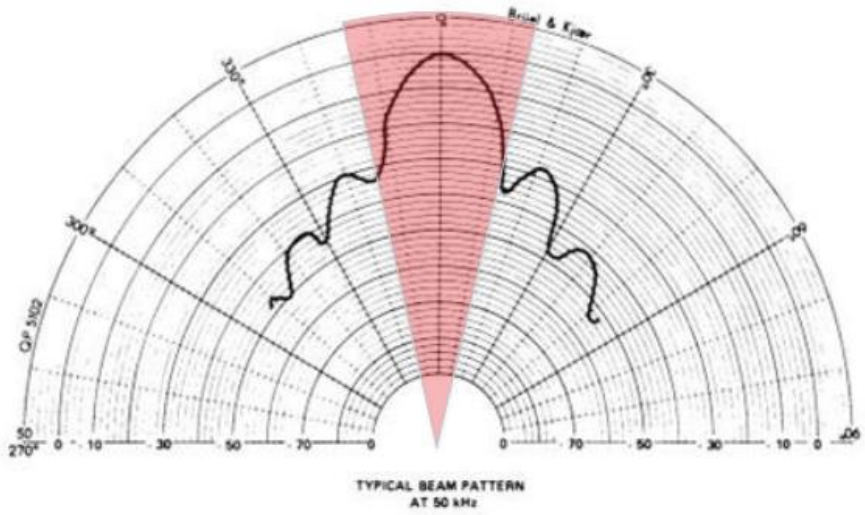


Figure 4. 6 Directivity of LEGO ultrasonic sensor

To solve this limitation, a mechanism was designed as shown in Figure 4.7 shows. The ultrasonic sensor is turned by a LEGO servo motor. By keeping rotating clockwise and counter-clockwise, the servo motor drives the ultrasonic sensor changing its detection angle all the time. In this way, the robot has the ability to detect obstacles in a range of 120 degrees.

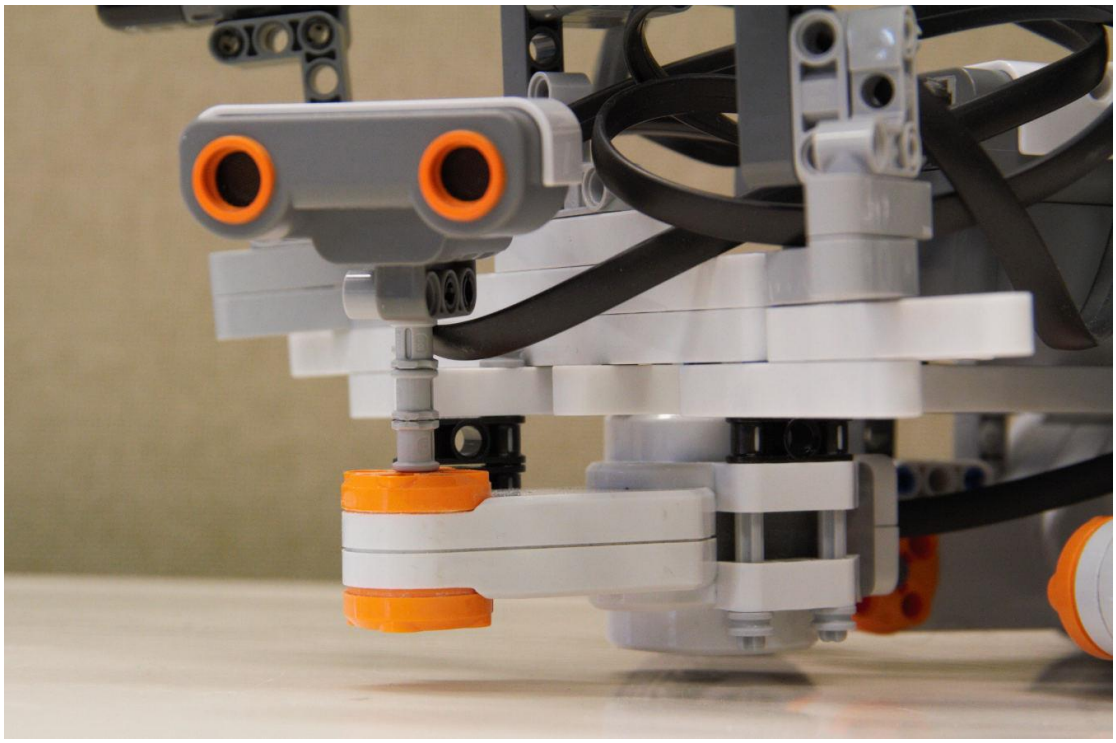


Figure 4. 7 The design of obstacle detection structure

4.2.2 Passive Infrared Sensor

Objects with temperature above absolute zero emit heat energy in the form of radiation. The warmer the object is the more radiation it emits and hence the easier it will be to be detected by a Passive Infrared Sensor (PIR sensor). PIR sensor can be used to detect infrared radiation generated by humans, animals, or any other objects. When a PIR sensor is used to detect a human, it is also called a human sensor.

The PIR sensor used in this thesis study is the HiTechnic NXT PIR Sensor (NIS1070) produced by a third-party company, as Figure 4.8 shows. This sensor can be used to detect when something warm enters, or moves, within its field of view. It works with people, animals or any objects which are warm. The ideal temperature for the subject is 85 - 120F (30 - 50C). [53]

It is important to note that PIR sensors don't detect or measure "heat", instead they detect the Infrared radiation emitted from an object which is different from but often associated/correlated with the object's temperature. Hence the PIR sensor is only sensitive to the changes of thermal input, which means if a warm object remains stationary in front of the sensor’s detection area, the contribution of that objects thermal radiation will slowly fade.

Figure 4. 8 HiTechnic NXT PIR sensor (NIS1070)

The field of view is divided into two zones, a left, positive, zone and a right, negative, zone, as shown in Figure 4.9 [53]. The sensor value is based on the relative

change in the infrared radiation from these two zones. If the infrared radiation increases in the left zone, there will be a positive effect on the sensor value. On the other hand, a decrease in radiation in left zone will cause a negative effect. The right zone has the opposite result on the sensor value compared the left zone. Details of illustration had been shown in Section 3.3 of Chapter 3.

Figure 4. 9 The field of view of PIR NIS1070

Figure 4.10 is a picture from the producer's website which shows what happens when a human walks in front of the senor. [53]

The numerical results are shown in Table 4-2. The reason there is a slight overshoot is because the person has left the negative zone which causes a net positive affect on the sensor value. [53]

Table 4- 2 Numerical results of sensor's values when a human passes by

Position	Sensor Value
1	0
2	60
3	-90
4	0

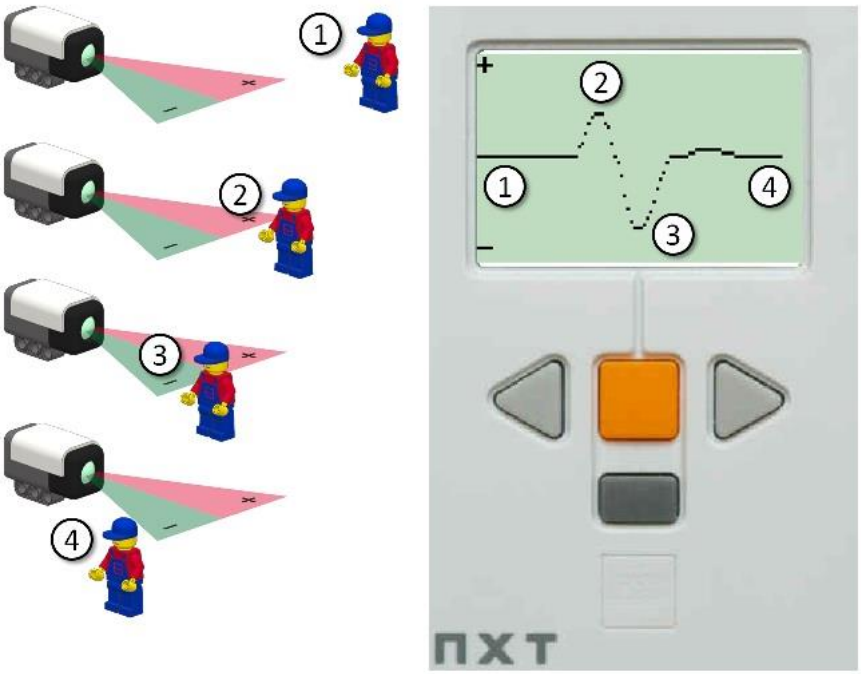


Figure 4. 10 Plot of sensor's values when a human passes by

In our experiments, 3 PIR sensors work together as an array to increase the detection area of the robot and to divide the detection area into more accurate smaller areas. Details of the PIR senor array were introduced in Chapter 3. Figure 4.11 shows the installation positions and angles of PIR sensor on the robot frame.

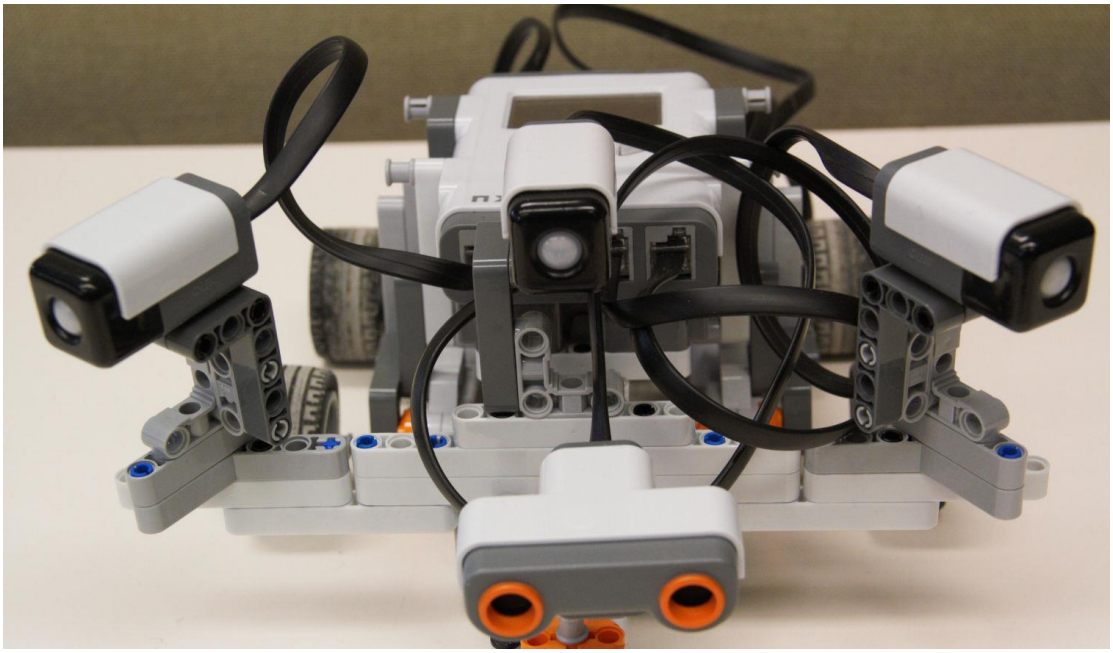


Figure 4. 11 PIR sensors' position on robot frame

4.2.3 Thermal Infrared Sensor

Different from PIR sensor, which is based on relative change in the infrared radiation, thermal infrared sensor (TIR sensor) is used to measure surface temperature value of an object. TIR sensor has a high accuracy and even NASA added it to the satellite missions to measure land surface temperature and Earth's thermal energy. In our research, TIR sensor is used in some separate experiments to support main experiments. The full name of the TIR sensor we used is Thermal Infrared Sensor for LEGO® MINDSTORMS® NXT. It is also a product of a third-party company instead of LEGO itself. Figure 4.12 shows the structure of the sensor.

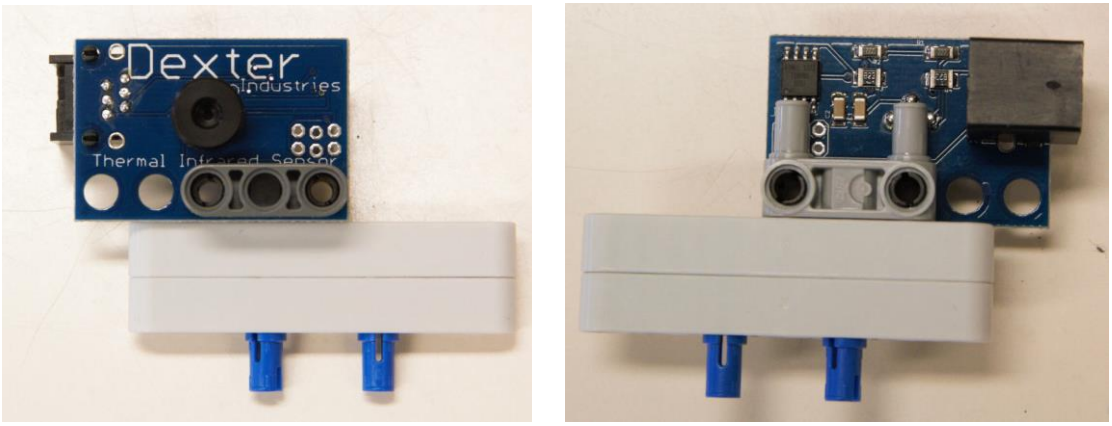


Figure 4. 12 Thermal infrared sensor for LEGO® MINDSTORMS® NXT

The Thermal Infrared Sensor reads both the ambient temperature (the temperature of the air around the sensor) and the surface temperature of the object that the sensor is pointed towards. [54] The properties of the Thermal Infrared Sensor for LEGO® MINDSTORMS® NXT is as follow.

- Measures and detects temperatures from a distance.
- Reads object temperatures between -70C and 380C.
- Has high accuracy of 0.5C and a resolution of 0.02C.
- Detects object surface temperature and ambient temperature.

4.2.4 Rotation Sensor

The rotation sensor is used to detect number of turns of the rotation of an object, rotation speed, angle, angular velocity, etc.

LEGO Rotation Sensor was built into LEGO Servo Motor, for measuring how much the motor turns. This is unique, because it measures based on the turn of the gears inside, rather than the motor itself. It is useful for robots that will coast and act based on distance rolled [55]. Lego rotation sensor is a nice little device that enables RCX to measure rotation of an axle with good resolution: 16 steps per turn [56]. Figure 4.13 is a dissection of LEGO Rotation Sensor and Figure 4.14 shows the circuit board of the rotation sensor [56].

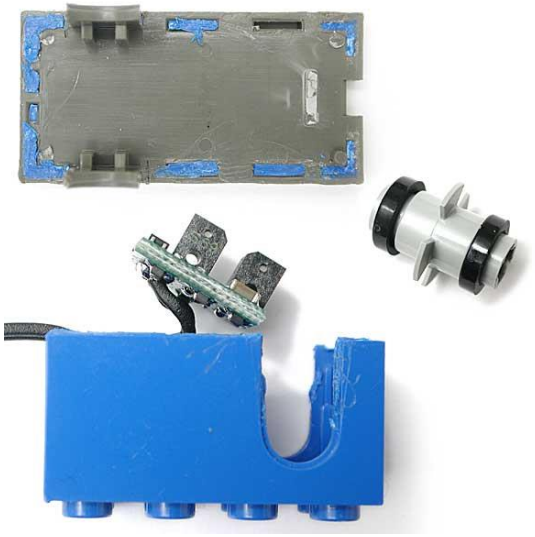


Figure 4. 13 Dissection of LEGO rotation sensor

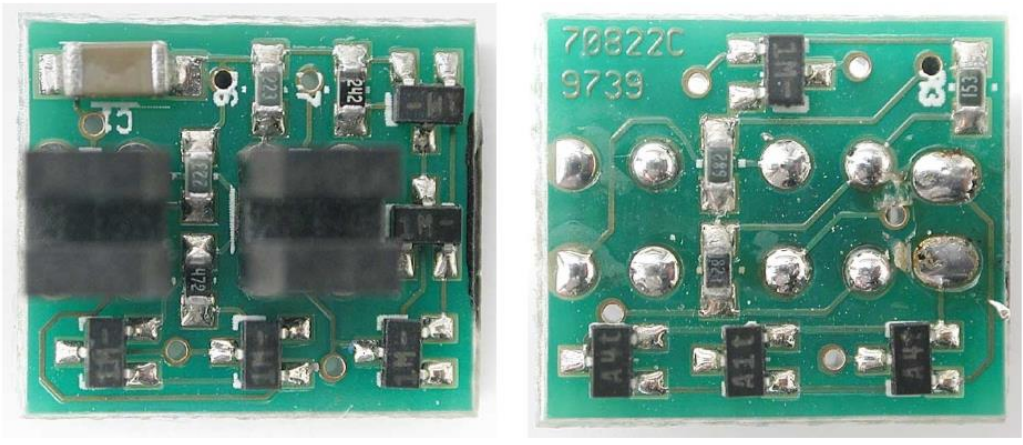


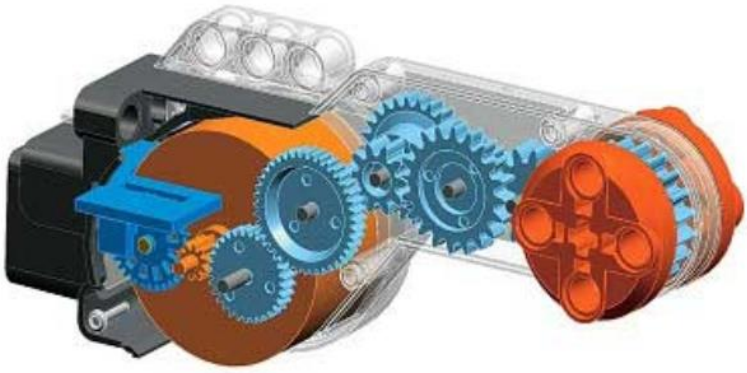
Figure 4. 14 Circuit board of the rotation sensor

4.3 Servo Motor

LEGO NXT ROBOT provides 3 DC NXT servo motors. This kind of motor is specific for LEGO NXT Series and includes a rotation encoder, whose function is feedback to the NXT the position of the shaft with 1 °resolution. The structures of the motor are shown in Figure 4.15 (a) and (b).

Table 4-3 gives some main characteristics of the NXT servo motor and these value were collected from some unofficial experiments and statistics.

(a)



(b)

Figure 4. 15 (a) NXT servo motor (b) Mechanical design of NXT servo motor

Table 4- 3 Main characteristics of the NXT servo motor

Characteristics		Values
Weight		80g
No-load	Rotation Speed	170rpm
	Current	60mA
Stalled	Torque	50N*cm

	Current	2A
Loaded Characteristics (at 9V Voltage)	Torque	16.7N*cm
	Rotation Speed	117rpm
	Current	550mA
	Mechanical Power	2.03W
	Electrical Power	4.95W
	Efficiency	41%

The curves in Figure 4.16 [57] below shows NXT motor rotation speed (Rotations per Minute) vs. motor power level (supply duty cycle).

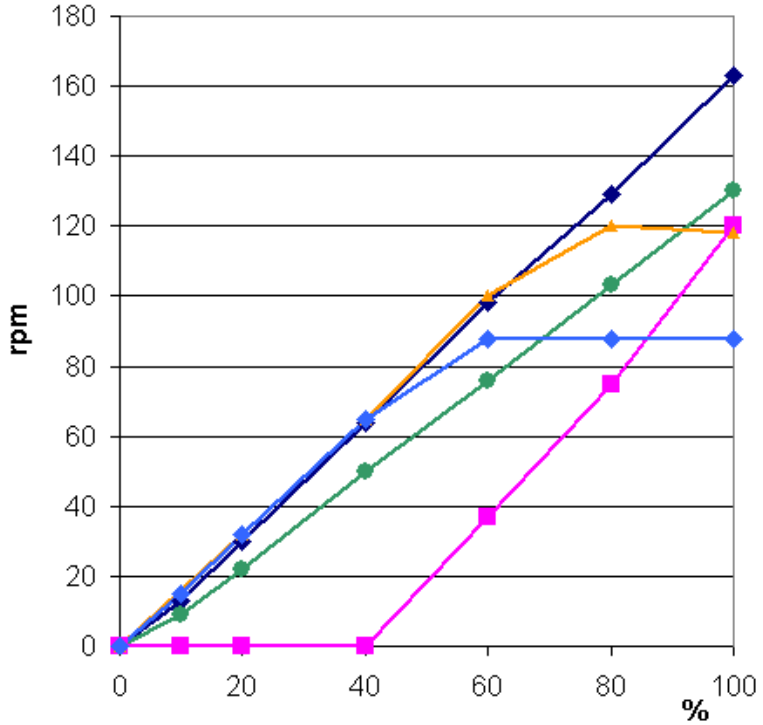


Figure 4. 16 NXT motor rotation speed vs. motor power level

represents motor not loaded, 9V NXT power.

shows motor not loaded, 7.2V NXT power.

is when motor behavior with a 11.5 N.cm load applied, no Power Control, 9V NXT power.

is the curve for motor loaded with a 11.5 N.cm, 9V NXT power.

denotes the motor loaded with a 11.5 N.cm, 7.2V NXT power.

In our experiment, all three NXT servo motors are used. One is used to drive the ultrasonic sensor as described in Section 4.2.1. The other two are used to drive the two driving front wheels. Figure 4.17 illustrates how the three motors configured.

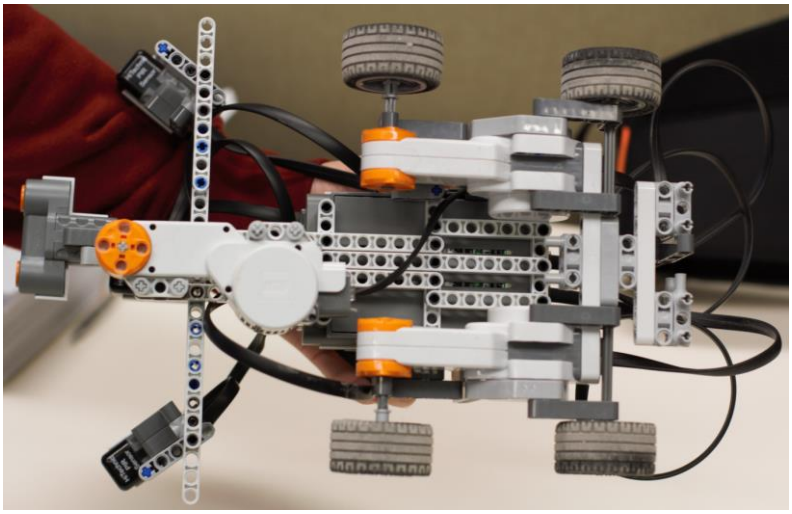


Figure 4. 17 Configuration of three NXT servo motors

4.4 Networking Setup

There are two ways for LEGO NXT to connect to a PC, wired and wireless connection.

There is an USB port on NXT Intelligent Brick, from which a PC and the robot can be connected by an USB cable. USB connection is used to download programs and data from a PC to a NXT device. Since an USB cable has to be used in the whole process, real-time update and control are not possible. The USB connection is shown in Figure 4.18.

Figure 4. 18 USB connection

For wireless connection, Bluetooth is supported by LEGO NXT. If the PC has a Bluetooth driver, information can travel through wireless connection, new programs and data can be downloaded, and mid-program, real-time updates and control are allowed. Bluetooth can be also used for communication with other Bluetooth devices, such as other NXT units, mobile phones and so on. Hence even a cellphone can be a controller of the NXT robot. Once the Bluetooth connection is set up, these features can be realized:

- Downloading programs from PC without using a USB cable.
- Sending programs from devices other than PC, including your own NXT.
- Sending programs to various NXT units either individually or in groups. A group can contain up to three NXT devices.

The meanings of Bluetooth icons on NXT screen are shown in Table 4-4

Table 4- 4 Meanings of bluetooth icons

Table 4-4 Meaning of Bluetooth Icons	
	Bluetooth is On but NXT is not visible to other Bluetooth devices.
	Bluetooth is On and NXT is visible to other Bluetooth devices.
	Bluetooth is On and NXT is connected to a Bluetooth device.
No Such Icon	Bluetooth is Off

Details of how to set Bluetooth connection between a NXT device a PC, a cellphone or other NXT devices please refer to the user guide of LEGO NXT robot.

4.5 Mechanical Structure of LEGO NXT Robot Based on Experiments.

The mechanical structure of robot was designed to meet requirements of experiments done for this thesis research. Mechanical structure of the robot is easy to design, architecture and change according to different experiment demands because of the smart LEGO BRICK design. The overview of the robot for human-following is shown in Figure 4.19 (a), (b) and (c) and Figure 4.20 (a), (b) and (c).

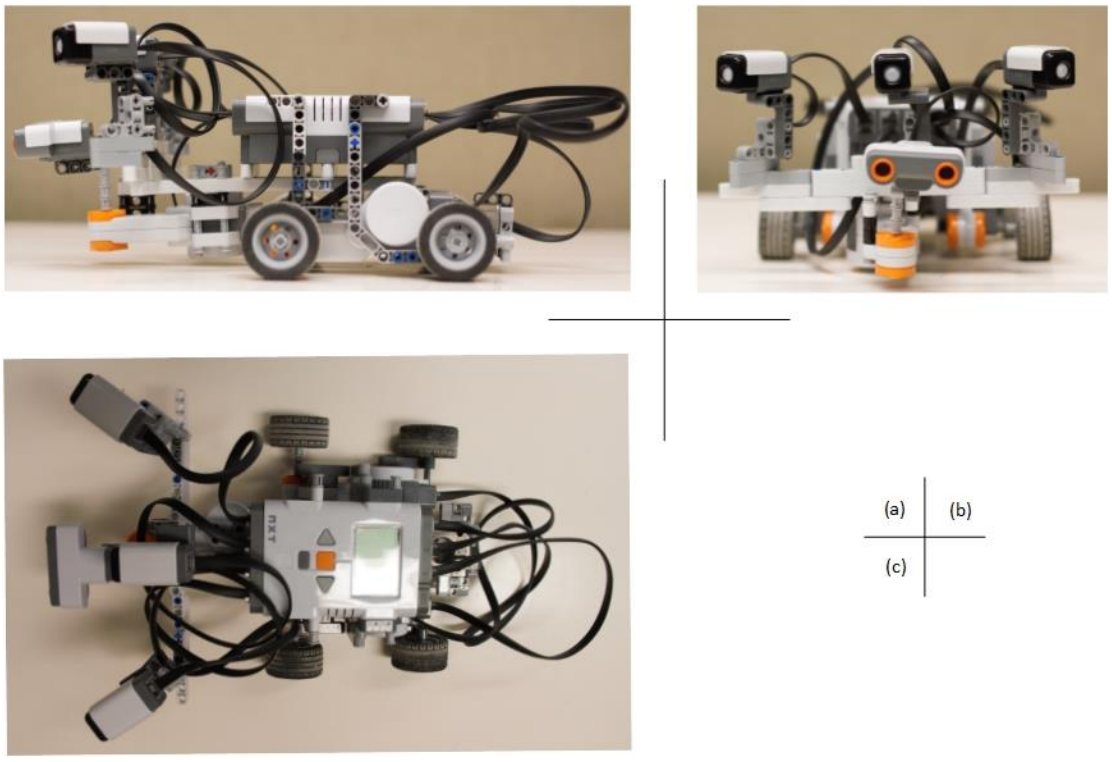


Figure 4. 19 (a) Side view of robot; (b) Front view of robot; (c) Top view of robot

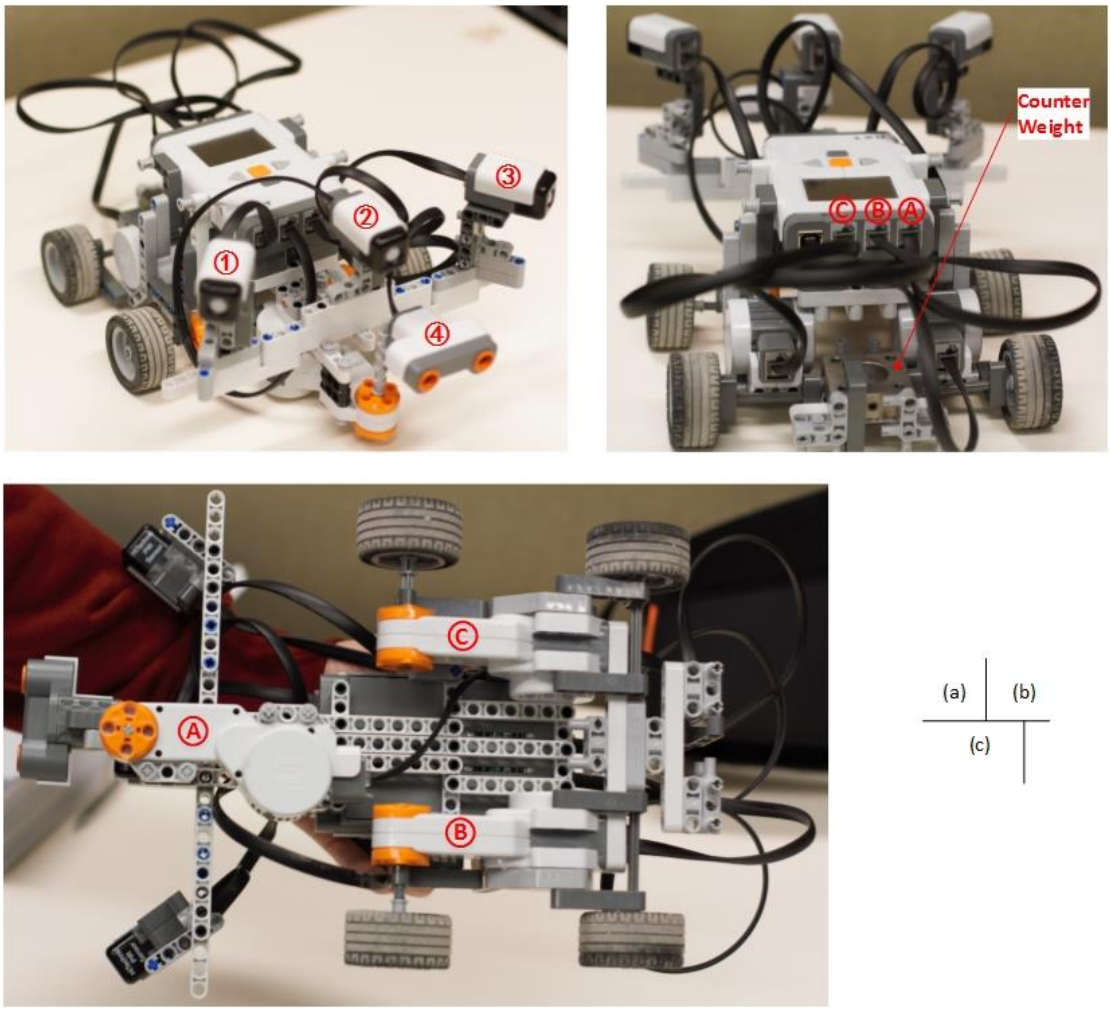


Figure 4. 20 (a) Sensors on robot; (b) Output ports and counter weight on robot; (C) Motors and chassis of the robot

In Figure 4.19 and 4.20, label 1, 2, 3 and 4 represent four sensors and their corresponding input ports on NXT Intelligent Brick. Label A, B and C represent three servo motors and their corresponding output ports on NXT Brick.

A counter weight was allocated and illustrated in Figure 4.20 (b) to balance the weight of sensors, motor A and structures in the front part of the robot. Without the counter weight, motor A can frequently hit the ground when the robot is in decelerating phases, as Figure 4.19(a) shows.

In experiments separate from the main ones, TIR sensor is used to detect temperatures of human, heater and external environment in either indoor or outdoor conditions. The installation position of the TIR sensor is shown in Figure 4.21.

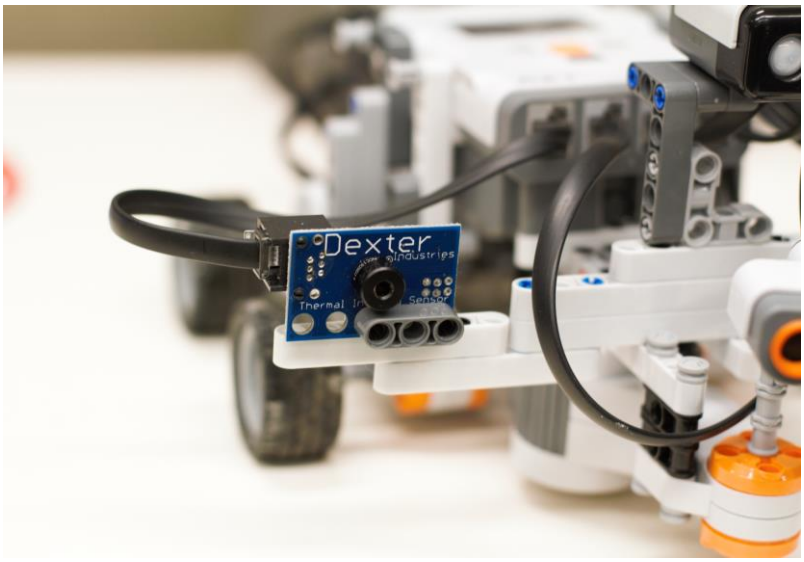


Figure 4. 21 TIR sensor on the robot

Table 4-5 gives the I/O ports of the NXT Brick corresponding to sensors and servo motors.

Table 4- 5 I/O ports utilization

I/O Ports	Equipment
Input 1	HiTechnic NXT PIR Sensor (NIS1070)
	Dexter Thermal Infrared Sensor for LEGO® MINDSTORMS® NXT (Separate Experiments)
Input 2	HiTechnic NXT PIR Sensor (NIS1070)
Input 3	HiTechnic NXT PIR Sensor (NIS1070)
Input 4	LEGO NXT Ultrasonic sensor
Output A	9V NXT DC Servo Motor
Output B	9V NXT DC Servo Motor
Output C	9V NXT DC Servo Motor

Chapter 5

Description of the Software

In this chapter, software used in the research will be introduced briefly with some specific examples. Three kinds of software were mainly used in this thesis, in which MATALB was used as a simulation tool, LabVIEW and NXT-G are the development environment of experiments based on LEGO NXT robot.

For each software, an example is given to make a better description. To compare the differences between the three software, all three examples are trying to realize the same thing, which is obstacle avoidance. The flow chart of obstacle avoidance sub function is shown in Figure 5.1.

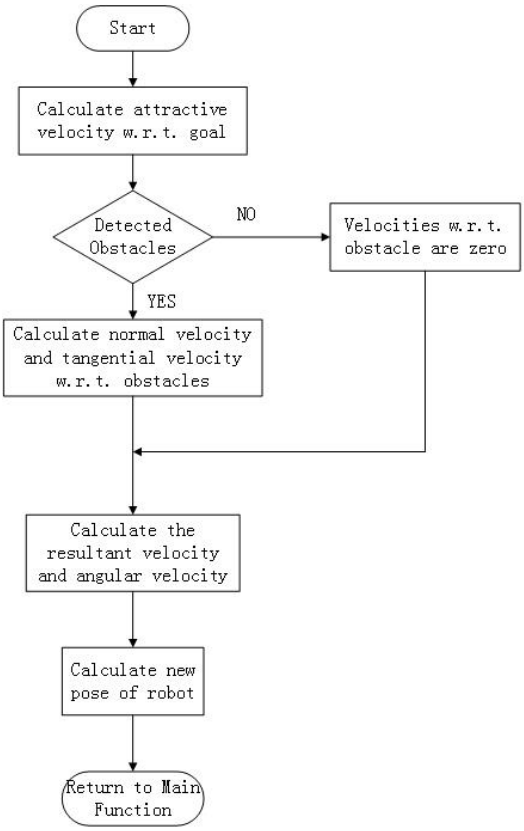


Figure 5. 1 Obstacle avoidance sub function

In Section 5.1, a MATLAB simulation program is introduced. Section 5.2 illustrates an experiment program realized with LabVIEW. Section 5.3 introduces a NXT-G

based experiment program. Finally, in Section 5.4, a comparison between NXT-G and NI LabVIEW is made.

5.1 MATLAB

The tool for simulation is MATLAB 2013. MATLAB is a numerical computing environment and high-level programming language developed by MathWorks. It is popular used in engineering and science area to analyze data, plot functions and data (as 3D-plots presented in Chapter 4), develop and test algorithms, create applications and interface with programs written in other languages such as C, C++ and Java. MATLAB is capable of robotic simulations due to its powerful engineering simulation and application ability. In this research, MATLAB simulations are used to analyze the feasibility of the algorithms in different situations so that we can process our experiments in the next step based on these algorithms.

The following codes are the function used to realize obstacle avoidance in MATLAB. The name of the function is "new_position". The inputs of the "new_position" function are shown in the brackets in the 1st statement. Using these inputs, the function can return an output to the main function, which is the new position of the robot in the 92nd statement.

All the simulation results are shown in Chapter 6, and all programs of MATLAB are shown in Appendix with respect to the simulations in Chapter 6.

```
1  function [ new_position ] =
   obstacle_controller( robot,goal,max_vel,max_ang_vel,go
   al_radius,obst_radius,obst_dist,obst_angle,T )

2  %  Constants
3  k_a=2;
4  k_n=0.5;
5  k_t=0.5;

6  % Calculate the x and y components of the distance to goal
7  delta_x=goal(1)-robot(1);
8  delta_y=goal(2)-robot(2);
```

```
9  % Calculate the distance to the goal
10 dist_goal=sqrt(delta_x^2+delta_y^2);

11 % Calculate angle to goal
12 theta=atan(delta_y/delta_x);

13 % Calculate the change in the angle between the current
   and goal position
14 delta_theta=theta-robot(3);

15 % Calculate the distance relation
16 dist_relation=exp(-dist_goal/goal_radius);

17 % Calculate attractive velocity
18 u_a=k_a*max_vel*(1-dist_relation);
19 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
20 % Find new theta value
21 close_to_zero=0.2;
22 if(abs(delta_theta)<close_to_zero)
23       theta=theta;
24 elseif(delta_theta>=0&&abs(delta_theta)<pi)
25       theta=robot(3)+max_ang_vel*T;
26 elseif(delta_theta>=0&&abs(delta_theta)>=pi)
27       theta=robot(3)-max_ang_vel*T;
28 elseif(delta_theta<=0&&abs(delta_theta)<pi)
29       theta=robot(3)-max_ang_vel*T;
30 elseif(delta_theta<=0&&abs(delta_theta)>=pi)
31       theta=robot(3)+max_ang_vel*T;
32 end
33 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
34 % Calculate repulsive velocity
35 % If no obstacle is in view
36 % Set normal and tangent values to zero
37 % If an obstacle is in view
38 % Calculate the normal and tangent values
39 if(obst_dist==0)
40    normal=0;
41    tangent=0;
42    obst_relation=0;
43    normal_angle=0;
44    tangent_angle=0;
45    theta_obst=0;
46 else
```

```
47    normal=1/obst_dist;
48    tangent=1/obst_dist;
49    obst_relation=exp(-obst_dist/obst_radius);

50 % Calculate normal angle
51 normal_angle=obst_angle+pi;
52 if(normal_angle>3*pi/2)
53     normal_angle=normal_angle-2*pi;
54 end

55 % Calculate tangent angle
56 tangent_angle=normal_angle-pi/2;
57 if(tangent_angle>3*pi/2)
58    tangent_angle=tangent_angle-2*pi;
59 end

60 % Calculate obstacle angle
61 theta_obst=(normal_angle+tangent_angle)/2;
62 if(theta_obst<0)
63    theta_obst=theta_obst+2*pi;
64 end

65 % Calculate change in angle between the robot and obstacle
66 delta_theta_obst=theta_obst-robot(3);
67 if(abs(delta_theta_obst)<close_to_zero)
68    theta_obst=theta_obst;
69 elseif(delta_theta_obst>=0&&abs(delta_theta_obst)<pi)
70    theta_obst=robot(3)+max_ang_vel*T;
71 elseif(delta_theta_obst>=0&&abs(delta_theta_obst)>=pi)
72    theta_obst=robot(3)-max_ang_vel*T;
73 elseif(delta_theta_obst<0&&abs(delta_theta_obst)<pi)
74    theta_obst=robot(3)-max_ang_vel*T;
75 elseif(delta_theta_obst<0&&abs(delta_theta_obst)>=pi)
76    theta_obst=robot(3)+max_ang_vel*T;
77 end

78 % Ignore goal while obstacle is in view
79 u_a=0;
80 theta=0;
81 end

82 % Calculate normal and tagential velocity components
83 u_n=k_n*normal*obst_relation;
84 u_t=k_t*tangent*obst_relation;
```

```
85 % Calculate the sum of all velocity components
86 u_totX=u_a*cos(theta)+u_n*cos(normal_angle)+u_t*cos(ta
   ngent_angle);
87 u_totY=u_a*sin(theta)+u_n*sin(normal_angle)+u_t*cos(ta
   ngent_angle);

88 % Calculate the changle in x and y distance for the current
   time step
89 delta_d=[u_totX*T,u_totY*T];
90 delta_theta=theta+theta_obst;

91 % Send the robot's next position to the main function
92 new_position=[robot(1)+delta_d(1),robot(2)+delta_d(2),
   delta_theta,u_totX,u_totY];
93 end
```

5.2 NI LabVIEW

Laboratory Virtual Instrument Engineering Workbench (LabVIEW) is a development environment and system-design platform for a virtual programming language researched and developed by National Instrument. In our experiments, LabVIEW 2012 were used to make programs for LEGO NXT robot. For the details about LabVIEW 2012, please refer to Appendix A.2.

The program of LabVIEW is called a virtual instrument (VI). VI consists of three parts which are front panel, back panel and connector panel. In the back panel, graphical programs are programmed. In this section, an obstacle avoidance program for LEGO robot based on LabVIEW 2012 is illustrated in Figure 5.2, in which

represents the ultrasonic sensor.

represents the DC servo motor.

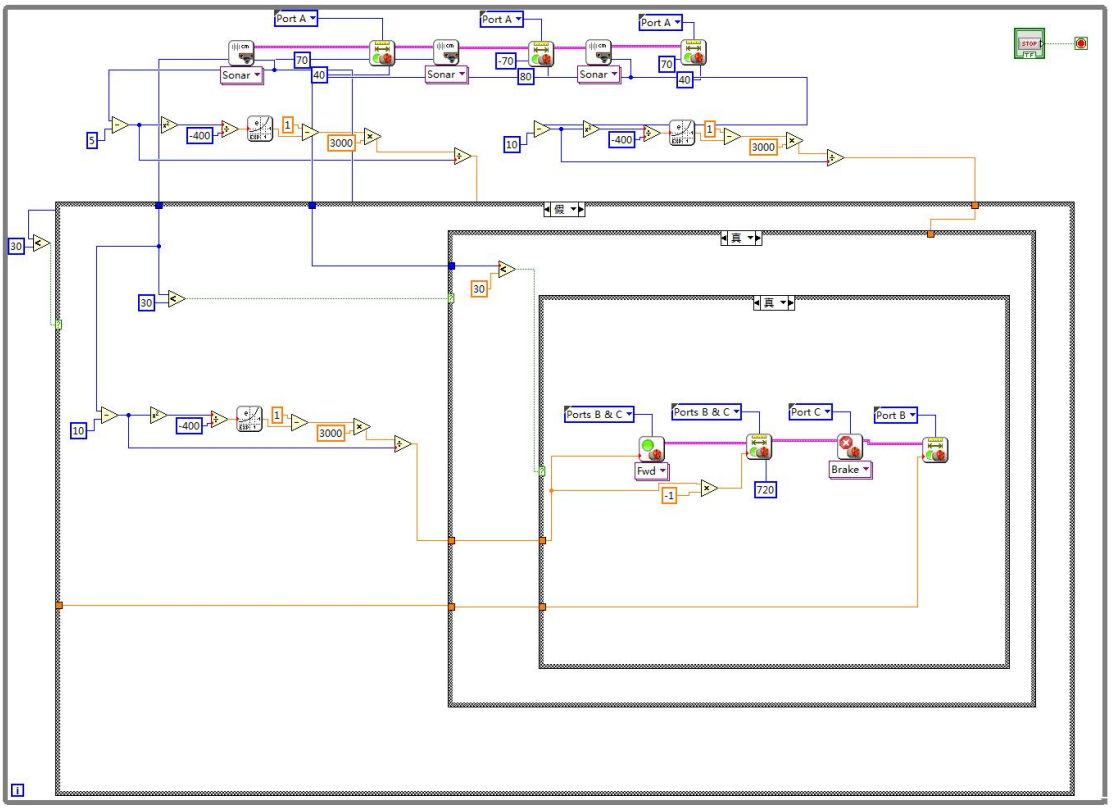


Figure 5. 2 Obstacle avoidance program with LabVIEW

The inputs are the distances detected by the ultrasonic sensor. From Figure 5.2 it can be seen that there are three ultrasonic sensor icons in the program, each followed with a motor icon. This is because the ultrasonic sensor is turned by a LEGO servo motor as Section 4.2.1 introduced. By keeping rotating clockwise and counter-clockwise, the servo motor drives the ultrasonic sensor changing its detection angle all the time.

The outputs are the velocities of motors which connected with the robot's driving wheels. Using three case structures, the outputs are classified into six situations to realized obstacle avoidance. All the six cases are shown in Appendix F. Based on the input data, the robot makes the right decisions by controlling the velocities of the two differential wheels.

Quasi-harmonic functions are used in the program to calculate the velocity based on the distance detected by the ultrasonic sensor. The quasi-harmonic function is presented in Figure 5.3.

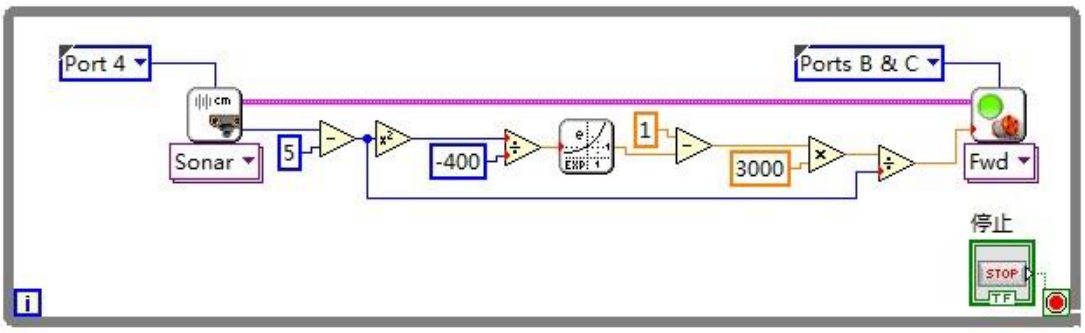


Figure 5. 3 Quasi-harmonic function in LabVIEW

5.3 LEGO MINDSTORMS NXT-G

NXT-G is a graphical programming environment which LEGO created it specifically for LEGO MINDSTORMS NXT robot based on NI LabView. In Appendix A, LEGO MINDSTORMS NXT-G is introduced in great detail include meaning of each icon.

Figure 5.4 and 5.5 show an obstacle avoidance function for LEGO robot programmed with NXT-G 2.0. It almost realizes the same result as program in LabVIEW environment except the quasi-harmonic controller since there is no complicated function calculation model in NXT-G environment.

Figure 5. 4 Obstacle avoidance program with NXT-G

With the program in Figure 5.4, the robot can only give a certain preset velocity when it is in a certain given situation. Hence in fact LabVIEW was used in most cases during this research. NXT-G was used mainly to deal with subsidiary experiments due to its abundant LEGO sensor models. For instance Figure 5.5 shows the program of environmental and surface temperature measurement, in which temperature captured by TIR senor is the input and the numerical value display on LEGO robot's screen is the output. This experiment was very helpful in the preparation stage of the human-following experiment.

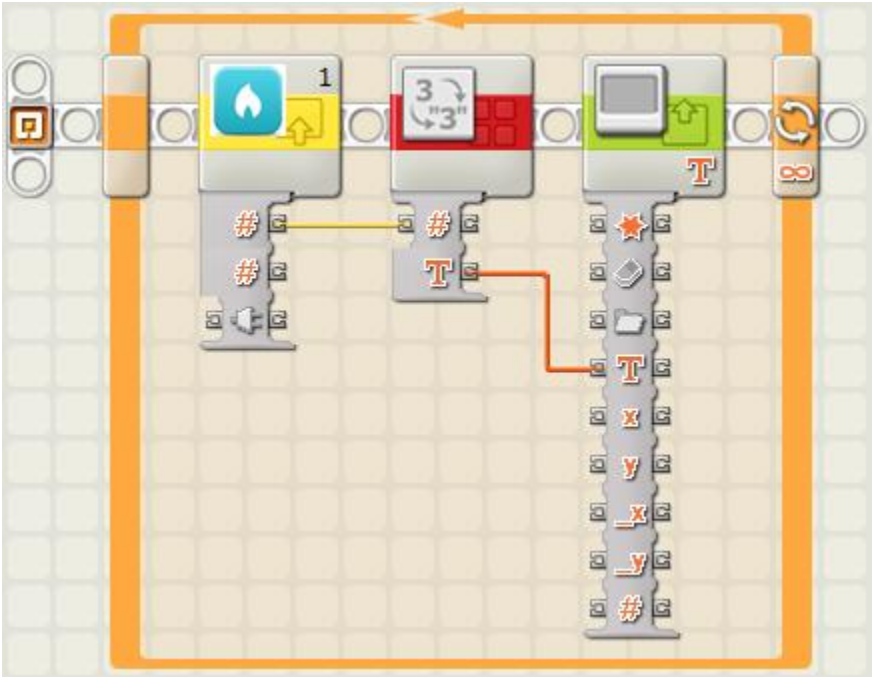


Figure 5. 5 Temperature measurement program with NXT-G

5.4 Comparison of NXT-G and NI LabVIEW

Details of comparison between NXT-G and LabVIEW are shown in Appendix A. From the comparison it can be seen that every development environment has its advantages and disadvantages and NXT-G and LabVIEW are no exception. In most situations, there is no absolute good or bad for a development environment or language. The only thing that determines which kind of environment should be used is

the user experience. Different users might have different talents, which means they may be adept at different tools.

For NXT-G, most kinds of third party NXT components and devices exist since it is the official development environment of NXT robot. However, the way it programs is more or less fixed to the NXT-G model more or less. For example, when the speed of the robot need to be controlled by quasi-harmonic function, it is hard for NXT-G to make a quasi-harmonic function using its blocks.

On the other hand, using LabVIEW researchers can help develop programs in a more flexible and customized way. The same robot speed control model can be built more easily with LabVIEW. But for LabVIEW, the limitation is the lack of third party device functions. As shown in Figure 5.15, some functions have to be made customized.

In fact, both of NXT-G and LabVIEW are good graphical programming software. They have their own strengths and weaknesses. Hence we use both software in different environment to avoid their weakness.

In main experiments, we use LabVIEW to realize more accurate and quantitative control of the robot. In separate experiment, used to support main experiments, we use NXT-G and its abundant third party NXT blocks.

Details of experiments will be shown in the Chapter 7.

Chapter 6

Simulations Results

The research is to design and realize the obstacle avoidance and human-following of a mobile robot based on quasi-harmonic function. This chapter concentrates on all the simulation works done with MATLAB in different situations.

The tool for simulation is MATLAB 2013. MATLAB is a numerical computing environment and high-level programming language developed by MathWorks. It is popular used in engineering and science area to analyze data, plot functions and data (as 3D-plots presented in Chapter 4), develop and test algorithms, create applications and interface with programs written in other languages such as C, C++ and Java. MATLAB is capable of robotic simulations due to its powerful engineering simulation and application ability. In this research, MATLAB simulations are used to analyze the feasibility of the algorithms in different situations so that we can process our experiments in the next step based on these algorithms.

Dozens of simulations were created during our research and they can be categorized into three types. Hence only three typical simulations will be presented in this Chapter. Section 6.1 refers to simulations of a holonomic robot using quasi-harmonic function to avoid obstacles. Section 6.2 introduces simulations of non-holonomic robot avoiding concave obstacles based on quasi-harmonic function. Section 6.3 illustrates a simulation of a non-holonomic human-following robot in an unknown environment with two point obstacles. All the programs codes of the simulations will be given in appendix of this thesis. Flowcharts of simulations in Chapter 6 and experiments in Chapter 7 are shown below.

(1) Flowcharts of simulation in Section 6.1 and Section 6.2

Figure 6.1 shows the flowchart of the main function of the simulation. Figure 6.2 shows the calling sequence of sub functions in main function. Figure 6.3 (a) and (b) give the flowcharts of two core sub functions.

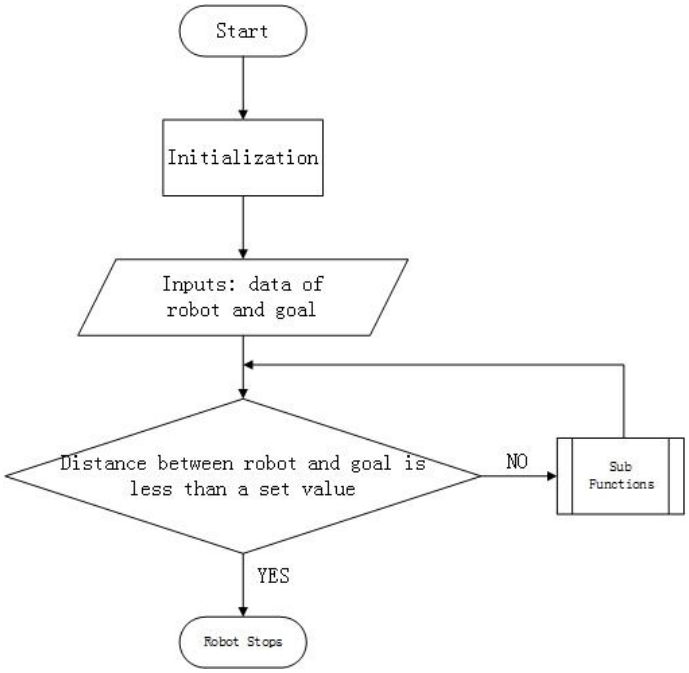


Figure 6. 1 Flowchart of main function

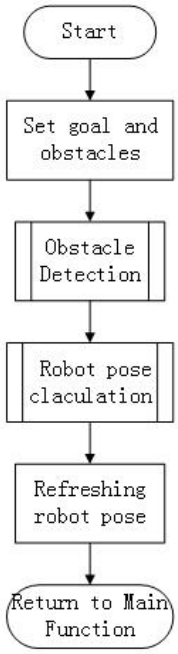


Figure 6. 2 Sub functions

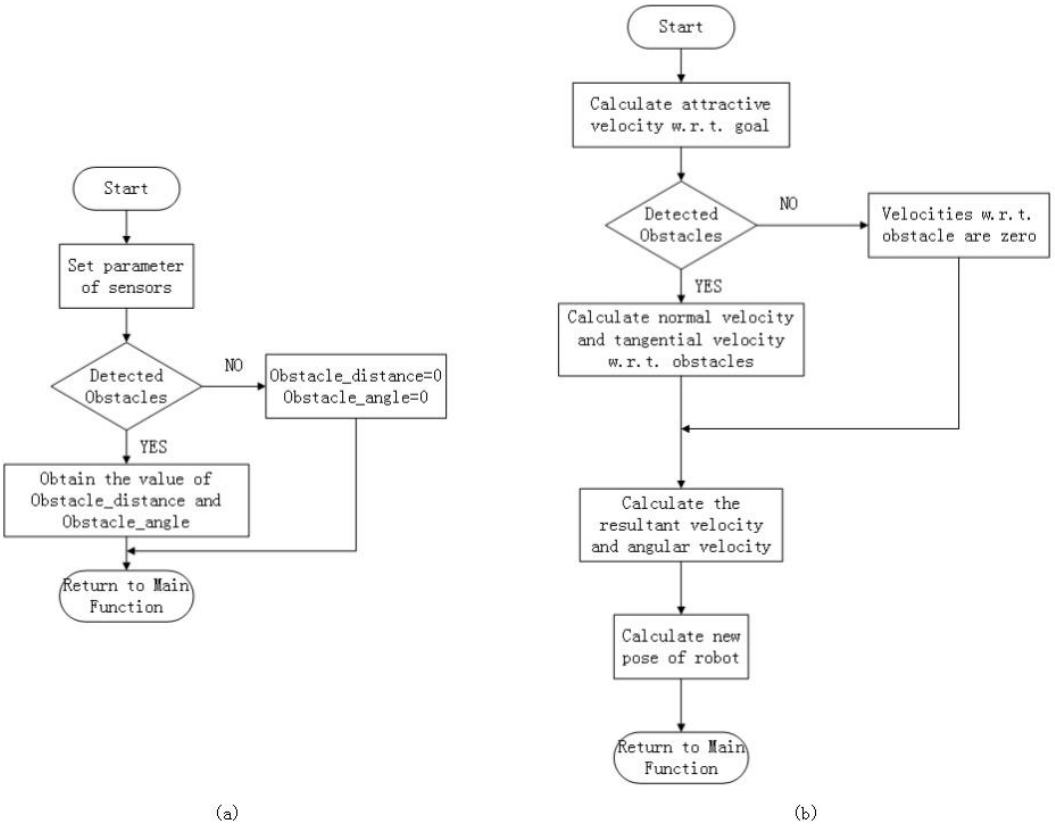


Figure 6. 3 (a) Obstacle detection sub function; (b) Robot's pose calculation sub function

(2) Flowcharts of simulation in Section 6.3

The human following robot also needs to avoid obstacles. However, since obstacle sub function has been presented in Figure 6.3 (a), here we did not repeat it any more.

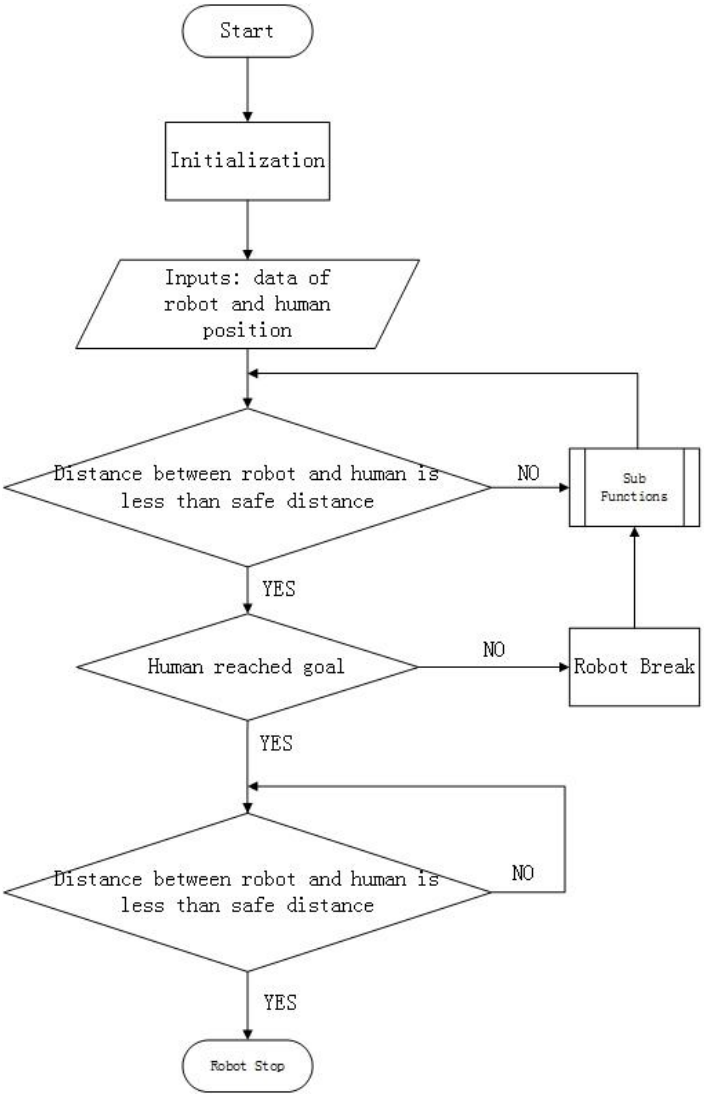


Figure 6. 4 Flowchart of main function

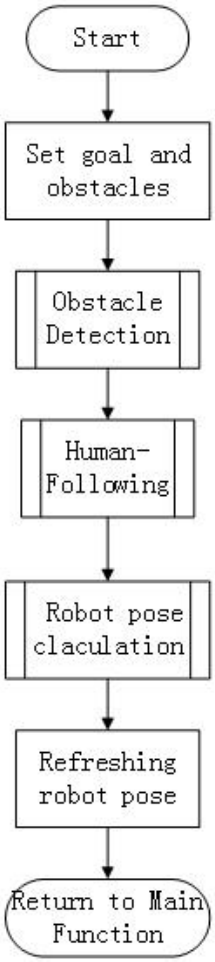


Figure 6. 5 Sub functions

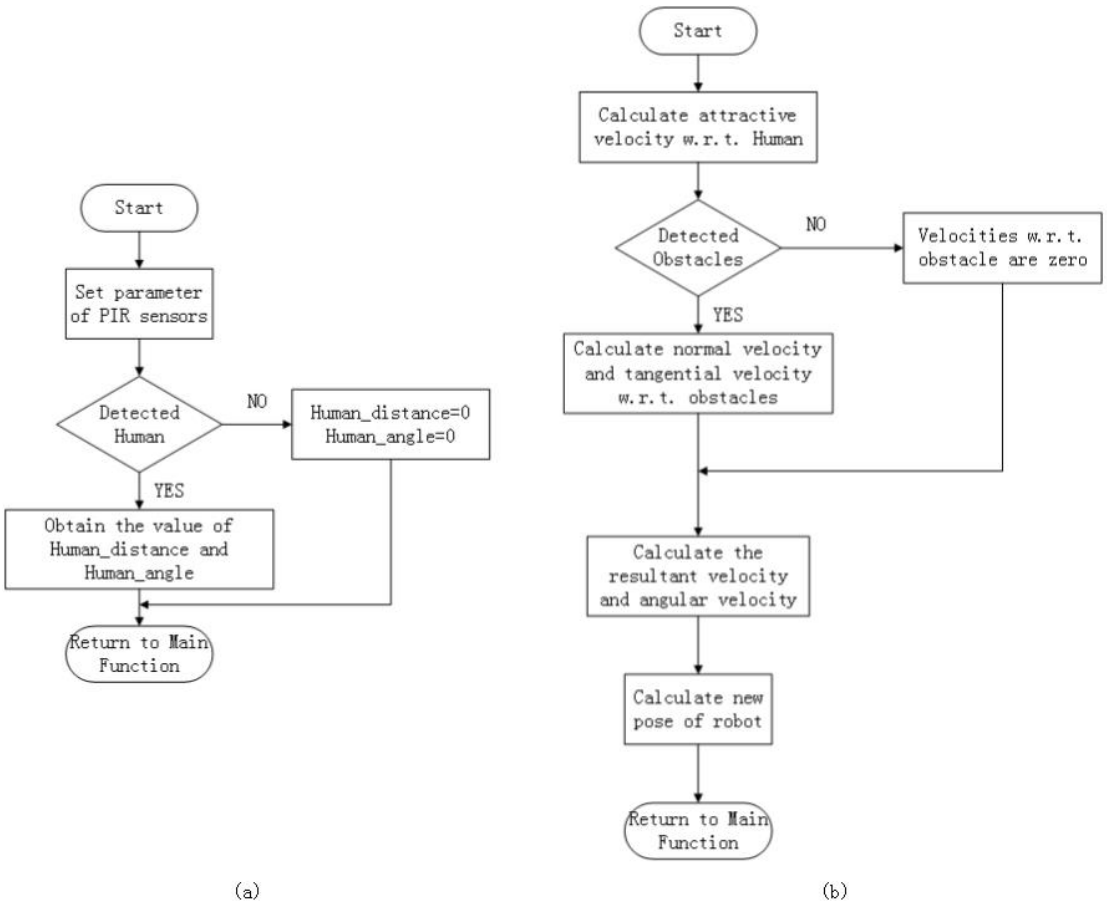


Figure 6. 6 (a) Human following sub function; (b) Robot's pose calculation sub function

6.1 Simulation of Holonomic Robot Obstacle Avoidance

6.1.1 Holonomic Robot Move Around Two Obstacles

The environment conditions of this simulation has been shown in Figure 3.6. In this simulation, a holonomic robot will start from a point on the negative part of y-axis and go towards to the goal on the positive part of y-axis. If the robot detects an obstacle, a vortex is created in the obstacle position help the robot avoid collision with the obstacle. After the robot goes around obstacles, it moves towards to the goal until it gets close enough to the goal point where the simulation stops. The simulation is based on the proposed quasi-harmonic approach and done with MATLAB. Figure 6.7 (a-f) present the results of the simulation with the following symbols:

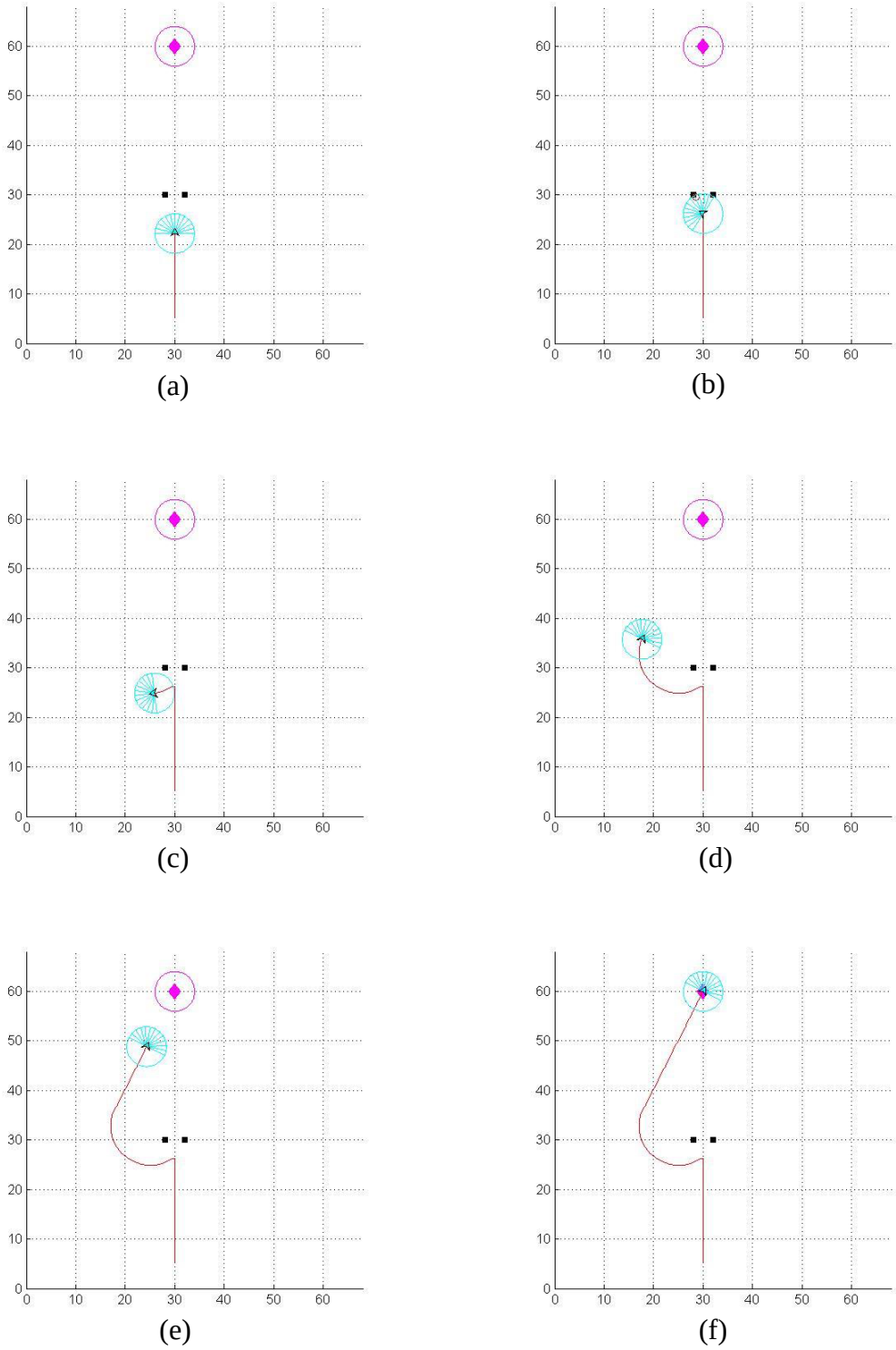


Figure 6. 7 Simulation of holonomic robot move around obstacles

The magenta diamond: the goal.

The magenta circle: the area of goal.

Black points: obstacles which are too close to get through.

The cyan arrow: current direction of robot.

The cyan circle: the detection range of robot.

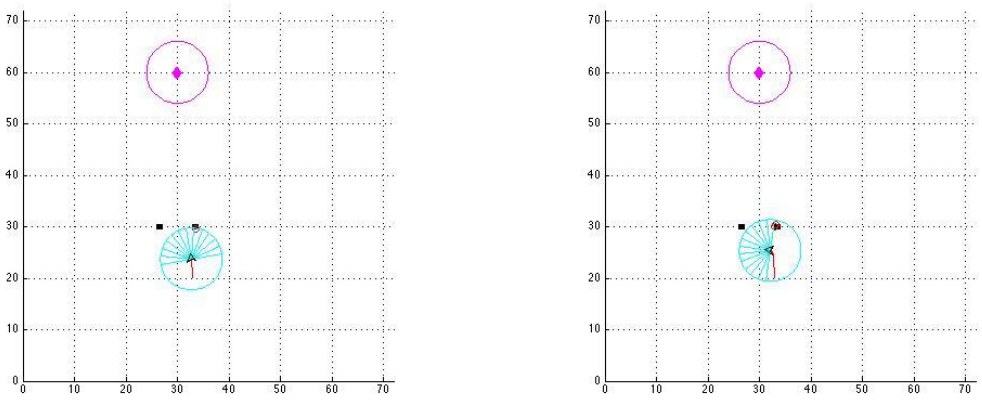
Cyan radials: sensors of the robot.

The red line: trajectory of robot.

In the simulation, harmonic functions were used with respect to the obstacles and away from the goal and a non-harmonic function was used in the vicinity of the goal in an overall quasi-harmonic approach. The investigation radius of sensors is set to 4 [m]. When the robot cannot detect any obstacle, the velocity command away from obstacle is set to zero. Hence there is only a velocity command toward the goal remains when sensors detect no obstacle near the robot, and the robot will stop smoothly at the goal. The simulation results denote that the robot can be controlled effectively to avoid obstacles and reach goal by including a quasi-harmonic function approach instead of only harmonic functions approach.

6.1.2 Holonomic Robot Get Through Two Obstacles

The only difference between the environment of simulation in Section 6.1.1 and 6.1.2 is the distance between the two point obstacles. In Section 6.1.1, the distance between the two obstacles is too closed to pass through. While in Section 6.1.2, the gap between obstacles is wide enough for the robot to pass through. The simulation results are shown in Figure 6.8.



(a) (b)

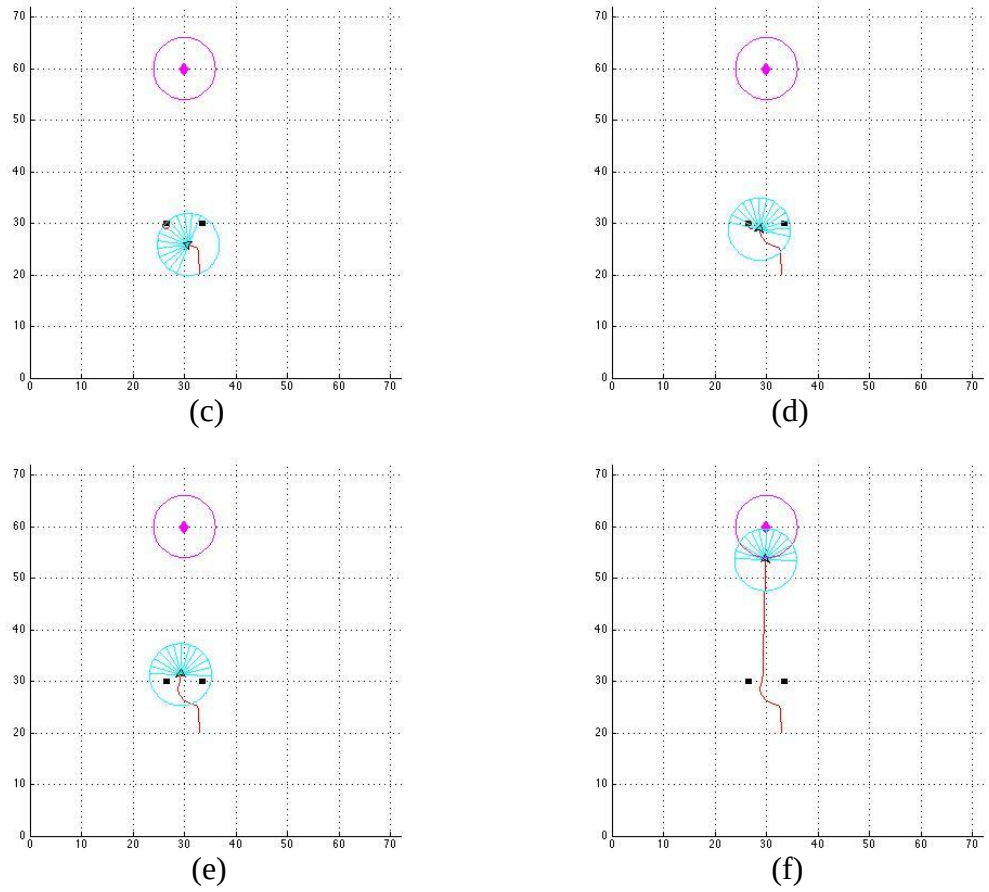


Figure 6. 8 Simulation of holonomic robot pass through obstacles

In this simulation, the initial position of the robot was (32, 20), the goal position is still at (30, 60). The detection radius of the robot is 8 meters. In Figure 6.8 (a), the robot detected the obstacle on the right side and in Figure 6.9 (c), it detected the obstacle on the left side. Then the robot calculated the distance between the two obstacles, which was 6 units in the coordinate system, and compared with size of robot, which was 3 units. [58] Since the distance between obstacles was wide enough to let the robot pass, the robot changed the safe detection radius value from 8 units to 2 units. As a result, the robot passed through the two obstacles safely and gave a shorter trajectory compare with moving around the obstacles.

6.2 Simulation of Non-Holonomic Robot Obstacle Avoidance

In this case, the robot is a non-holonomic robot. The description of environment has been shown in Figure 3.13 in Chapter 3. The numerical and graphical analysis of quasi-harmonic control in this environment has been presented in the second part of Section 3.1.3. In this section, the simulation of case from the second part of Section 3.1.3 will be illustrated.

For a front wheel driving and steering car-like robot, its kinematic model can be simplified as a bicycle. [59] The coordinates for the robot are

$$q^T = \begin{bmatrix} x & y & \theta & \phi \end{bmatrix} \tag{6.1}$$

where, x, y define the position of the middle point of front wheels, θ is robot body orientation, φ is the steering angle.

The kinematic equations are

$$\frac{dx}{dt} = v\cos(\theta + \phi) \tag{6.2}$$

$$\frac{dy}{dt} = v\sin(\theta + \phi) \tag{6.3}$$

$$\frac{d\theta}{dt} = \frac{v}{l}\sin\phi \tag{6.4}$$

$$\frac{d\phi}{dt} = \omega \tag{6.5}$$

where l is the distance between the centers of front wheels and rear wheels, ω is the velocity of the steering and v is velocity of the front wheels.

In matrix form, the kinematic model of a car-like non-holonomic robot is

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} r\bullet\cos(\theta + \phi) \\ r\bullet\sin(\theta + \phi) \\ \frac{r}{l}\sin\phi \\ 0 \end{bmatrix} \omega_r + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \omega \tag{6.6}$$

where r denotes the radius of wheels and ω_r is the angular velocity of front wheel.

Figure 6.8 (a-j) present the results of the simulation with the following symbols:

The magenta diamond: the goal.

The magenta circle: the area of goal.

Black points: obstacles which are too close to get through.

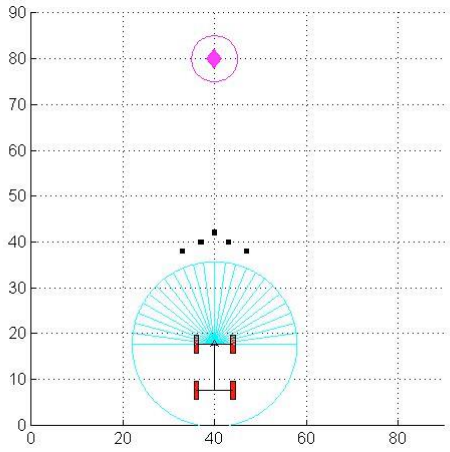
The red arrow: current direction of robot.

The cyan circle: the detection range of robot.

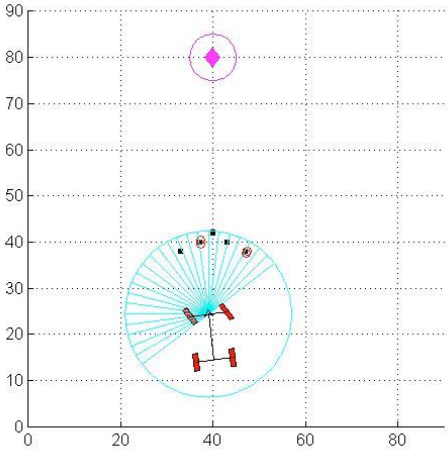
Cyan radials: sensors of the robot.

The black frame: frame of car-shape robot.

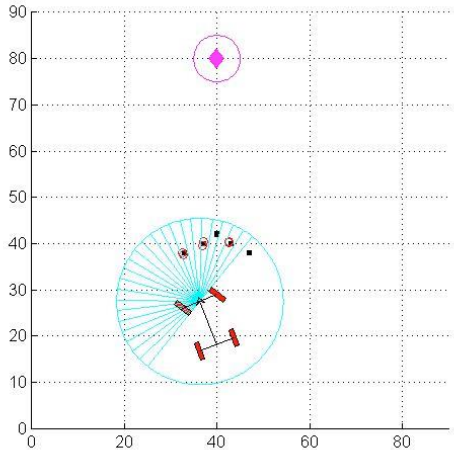
Red rectangles: four wheels of the robot



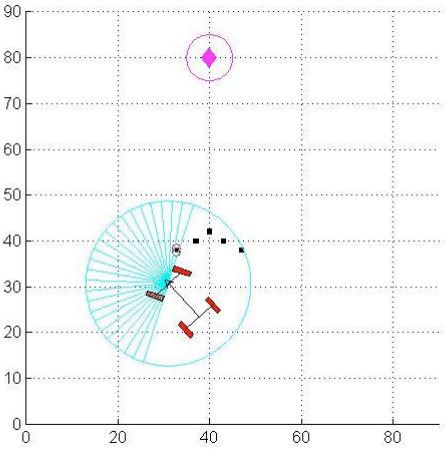
(a)



(b)



(c)



(d)

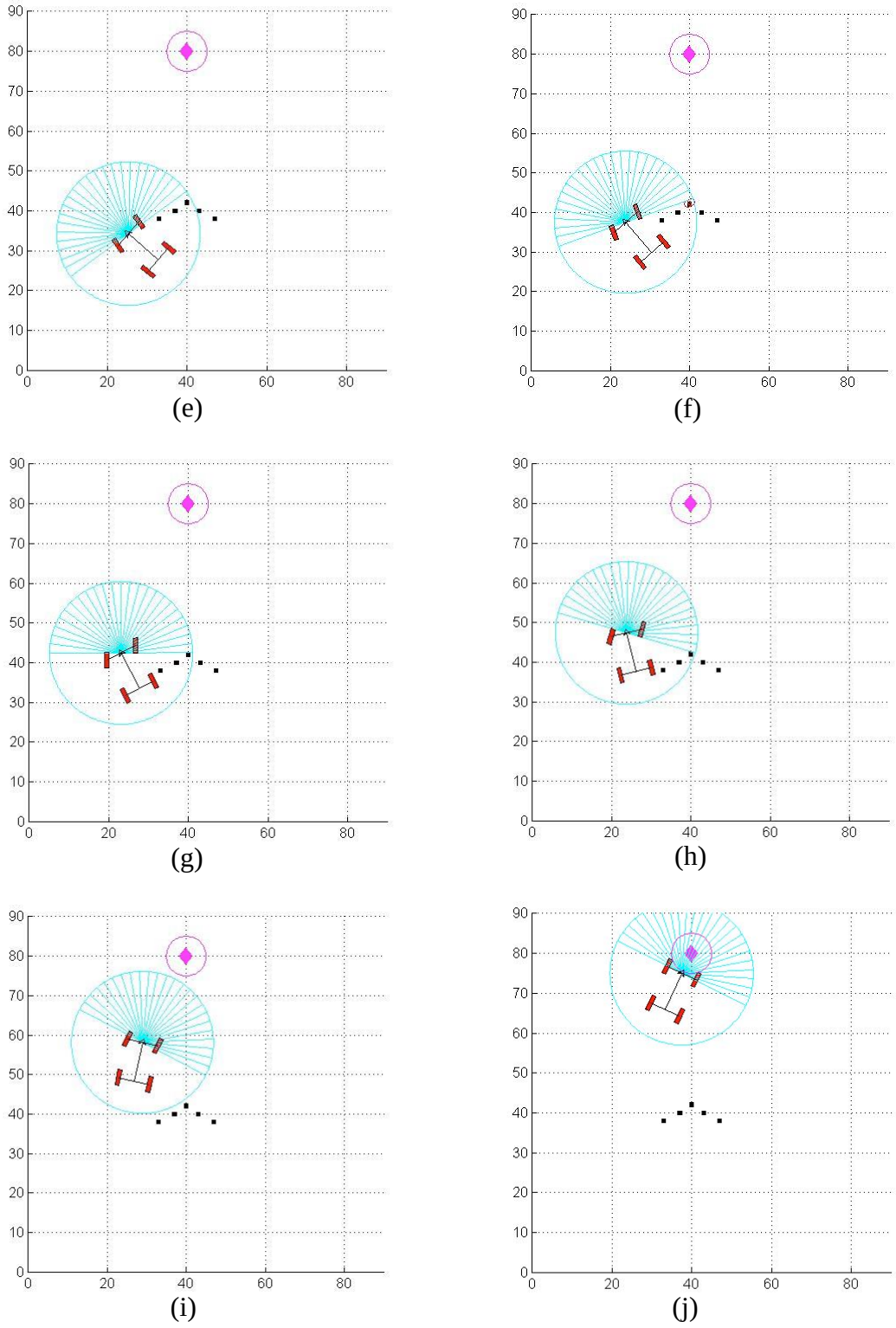


Figure 6. 9 Simulation of non-holonomic robot obstacle avoidance

In the simulation, as in fig. 6.8, harmonic functions were used with respect to the obstacles and away from the goal and a non-harmonic function was used in the vicinity of the goal in an overall quasi-harmonic approach. The results of this

simulation show that the quasi-harmonic approach can be used efficiently to control a non-holonomic robot to move around obstacles when they form a concave block which does not permit the robot to pass in-between. The proposed controller makes the non-holonomic robot smoothly slow down when it get close to the goal and to arrive at a final velocity that tends to zero at the goal.

6.3 Simulation of Non-Holonomic Human-Following Robot

The algorithm of human-following robot has been introduced in Section 3.3. A simulation will be made based on that algorithm. The human will be treated as either a goal or an obstacle depending on the distance between human and robot. To verify the strategy, we design a brief simulation, in which we ignore the human detection process and assume the robot has already sensed human's position. In other words, in the simulation, human's position is input data for the robot.

The dynamic model of the robot is as in Section 6.2. The investigation radius of sensors is set to 18.5 [m] which is also the car-like robot's Minimum Turning Radius given by Equation 6. 7.

$$R_{\min} = R_0 + \frac{d}{2} = \frac{l}{\sin\theta_{\max}} + \frac{d}{2} \tag{6.7}$$

Where l is the distance between the center of front wheels and the center of rear wheels, $\theta_{\max}$ is the maximum steering angle of robot and d is the distance between front wheels. Six frames of Figure 6.9 shows the results of the simulation.

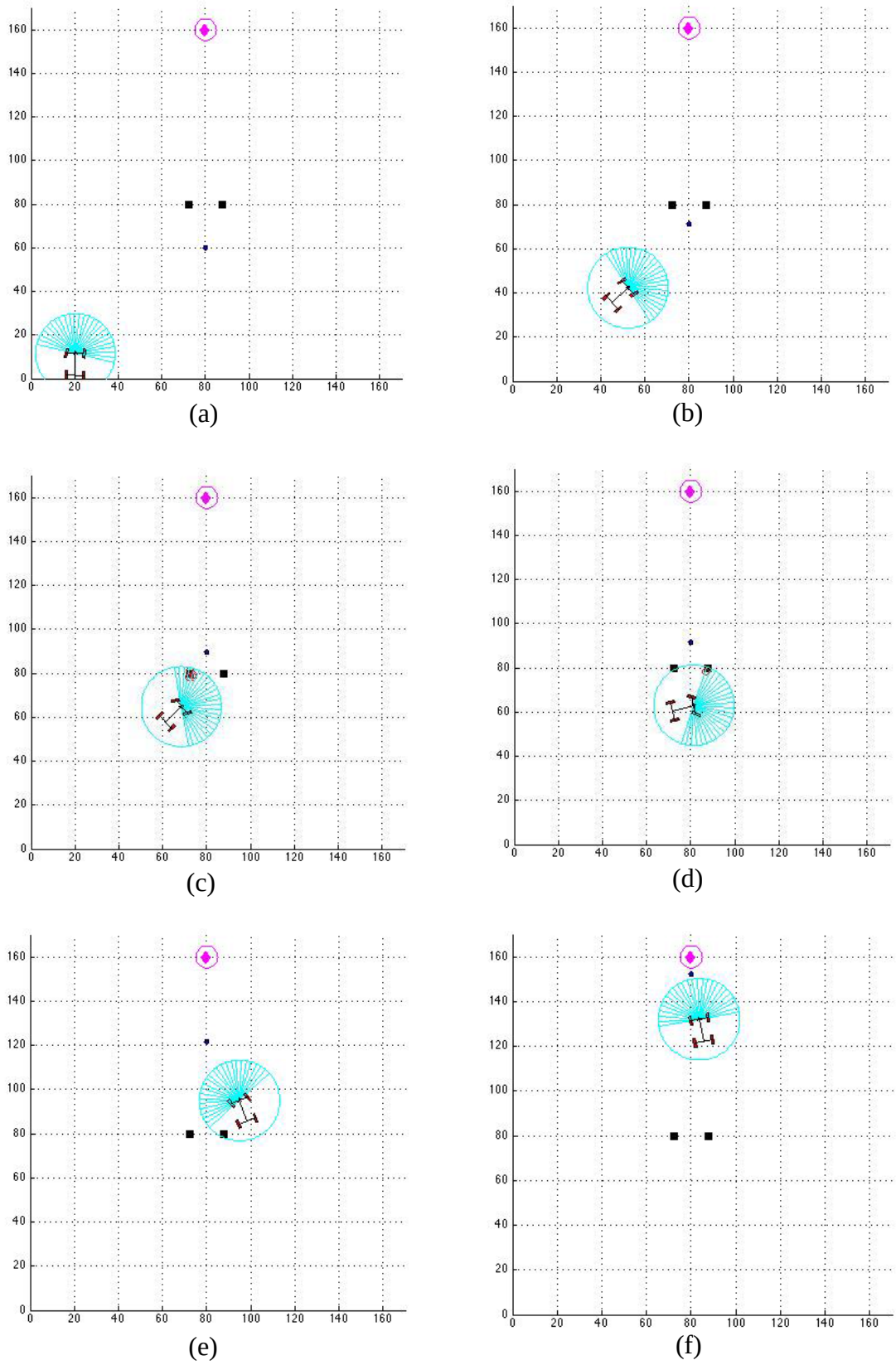


Figure 6. 10 Simulation of human-following robot

In the simulation, as in section 6.2, harmonic functions were used with respect to the obstacles and away from the goal and a non-harmonic function was used in the vicinity of the goal in an overall quasi-harmonic approach. The small dot represents the human. The obstacles by squares and the goal by a diamond. Explanations of other elements are the same as in Section 6.2. The results presented in this paper show that the quasi-harmonic approach could be used efficiently to control a non-holonomic robot to follow a human and avoid obstacles even when they form a concave block that does not permit the robot to pass in-between.

Chapter 7

Experimental Results

Experimental study has to verify the results in simulation in an actual physical environment. By doing experiments, it can be verified that the quasi-harmonic controller is feasible not only in theory but also in real world.

However, due to the limitation of the experimental equipment, not all simulations have been done can be verified as experiments. For instance, the simulation in Section 6.1, proofed that quasi-harmonic function can be used to control holonomic robot, but the related experiment cannot be prepared because there is no available holonomic robot or mechanism like Omni-wheel in our lab. Hence, only the simulations were done for the holonomic robot. We will only focus on the experiments of the non-holonomic robot. The robot we used is LEGO NXT robot. The structure and the hardware of the robot were introduced in Chapter 4.

In this chapter, 4 main experimental setup was shown. In Section 7.1, the first experiment shows the relationship of the robot's velocity and the distance between robot and obstacle based on quasi-harmonic function. In Section 7.2, the second experiment examined how a non-holonomic robot was able to avoid two obstacles and reach a fixed goal. This experiment reproduces the situation in the first simulation in Chapter 6. Section 7.3 refers to the experiment with a non-holonomic robot with five point obstacles which form a concave shape. In Section 7.4, an experiment regarding a human-following robot was made to reproduce the situation of the third simulation in Chapter 6.

The non-holonomic robot used in the experimental stage is different from the one simulated in chapter 6. This is because that one of my colleagues in the laboratory also needed to use the LEGO NXT robot, and the robot is a crawler-type robot in his experiment. For the convenience of both our experiments, I designed the robot into a

differential-wheel drive robot, shown in Figure 4.19 and Figure 4.20, which is a non-holonomic robot as well.

The differential wheel drive robot consists of two independently activated parallel wheels and two passive wheels to hold the balance of the robot. Let (x, y) be the coordinate of the midpoint of the axis between two drive wheels, and θ be the direction of drive wheels with respect to the x-axis. A possible configuration of the differential wheel drive robot can be

$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{7.1}$$

And the kinematic model of it can be

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r}{2}\cos\theta \\ \frac{r}{2}\sin\theta \\ \frac{r}{d} \end{bmatrix} \omega_r + \begin{bmatrix} \frac{r}{2}\cos\theta \\ \frac{r}{2}\sin\theta \\ -\frac{r}{d} \end{bmatrix} \omega_l \tag{7.2}$$

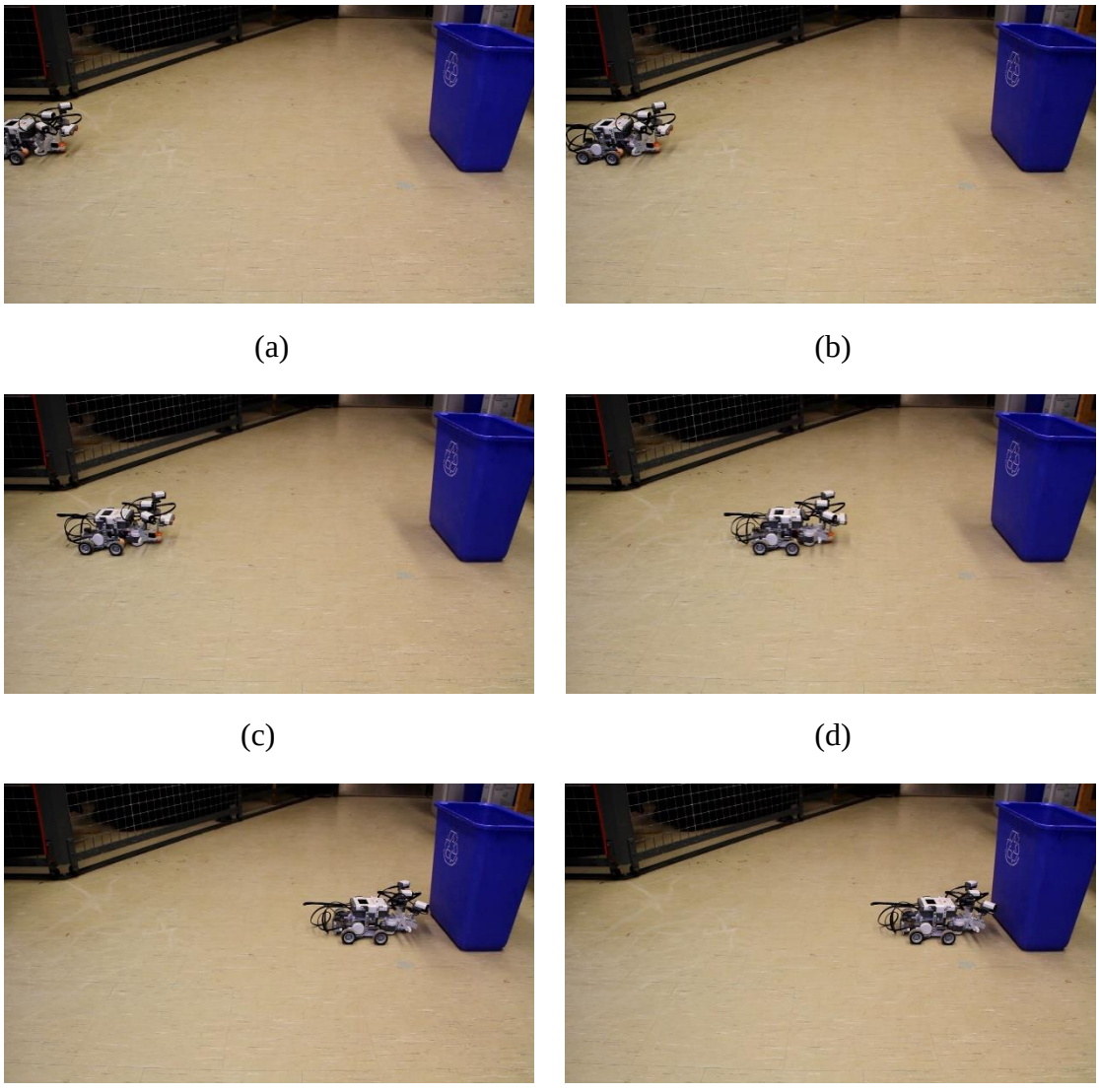
in which r is the radius of drive wheels, d is the distance between the center of two parallel drive wheels, ω_r is the angular velocity of right wheel and ω_l is the angular velocity of left wheel. The robot can be controlled by control the angular velocity of motors which drive the two drive wheels.

7.1 Experiment of Quasi-Harmonic Function

The form of quasi-harmonic function has been introduced in Chapter 3. In this section, an experiment will show how the relationship between robot's velocity and distance between robot and goal or obstacle is like. The plot of velocity-distance has been shown in Figure 3.5, and hopefully, the experiment will show the same relationship as the plot of the theoretical function.

For LEGO NXT robot, the only facility we have to input a distance value is the ultrasonic sensor. Hence this experiment will only use the ultrasonic sensor to detect

the distance between robot and an obstacle in front. And observe the robot's action with respect to the distance. The following Figure 7.1 are screenshots of the first experiment. And the time intervals between each two neighboring figures are the same, which is 3 seconds.



(a) (b)

(c) (d)

(e) (f)

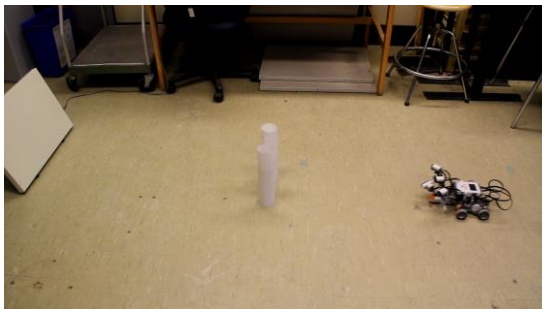
Figure 7. 1 Screenshots of experiment 1 every 3 seconds

7.2 Experiment of Non-Holonomic Robot with Two Adjacent Obstacles

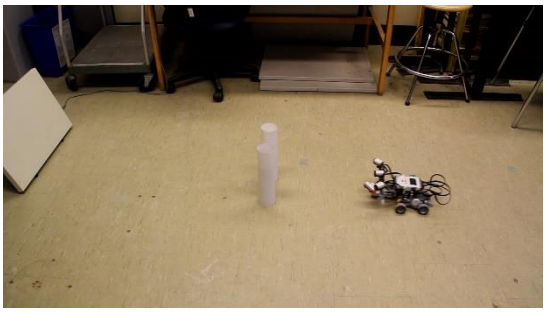
In this experiment, two thin cylinders are used as obstacles, and placed in the direct path between the robot's initial position and the goal position. The goal is a flat heater

in the far end on the other side. The robot will measure the temperature difference and confirm the position of the goal. The two obstacles are close to each other and robot cannot directly get through. An important issue is that when the robot's ultrasonic sensor reach the obstacles, it detects two obstacles, one on the left and the other on the right. Robot might be stuck by obstacles in this shape if the robot without tangential velocity.

Figure 7.2 shows snapshots of the experiment. Result of the experiment shows the robot was able to successfully navigate around the two adjacent obstacles and reach the goal with the help of PIR sensors. When all three PIR sensors and the ultrasonic sensor detect the goal, it means robot is close to the goal, then robot will stop in front of the goal. Figure 7.2 (c)-(e) show the robot move backwards when it is too close to the obstacles.

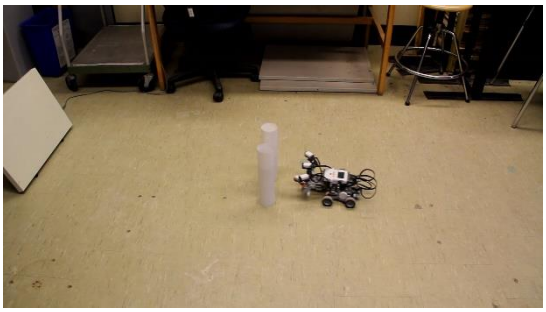


(a)



(b)

(c)



(d)

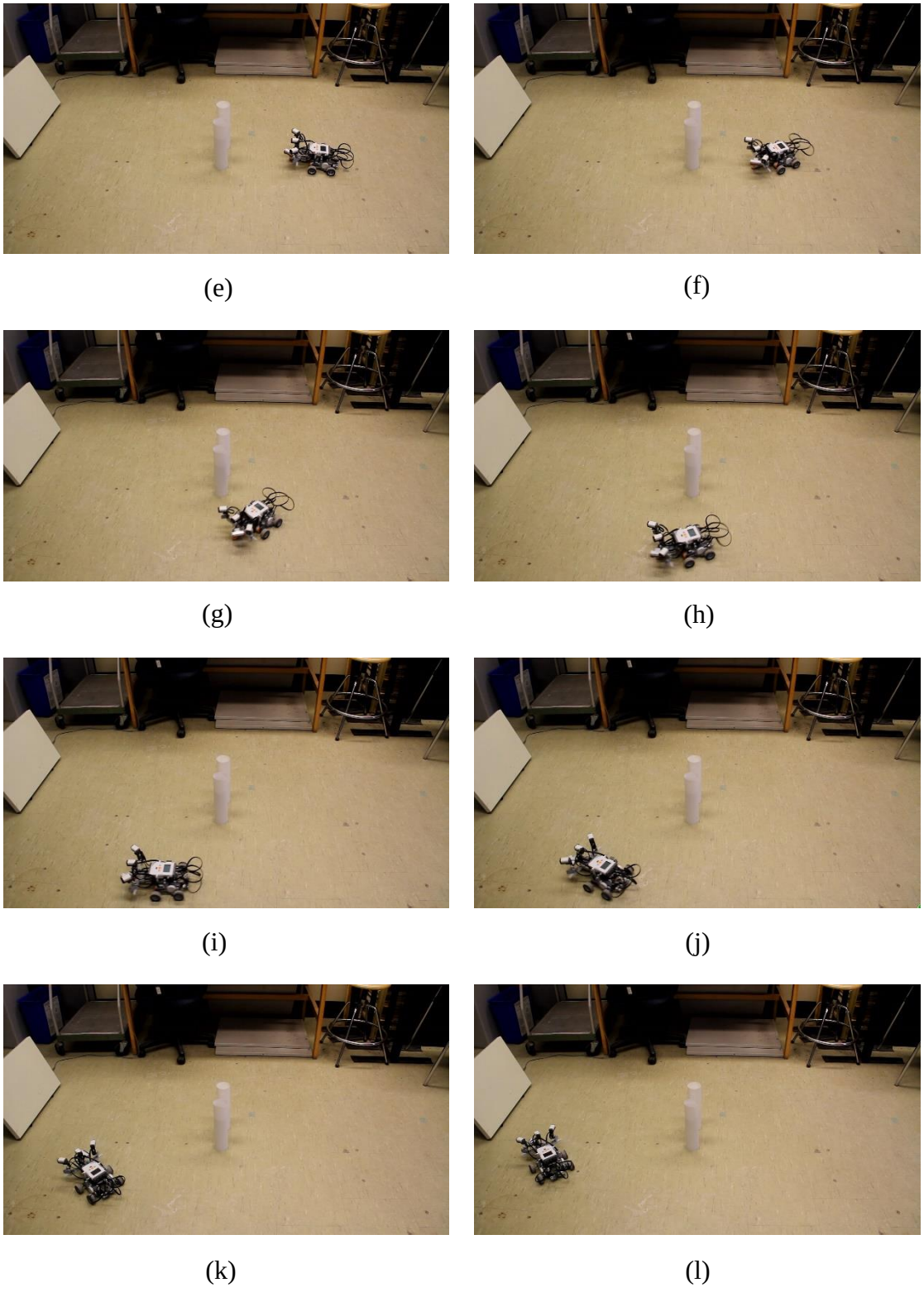
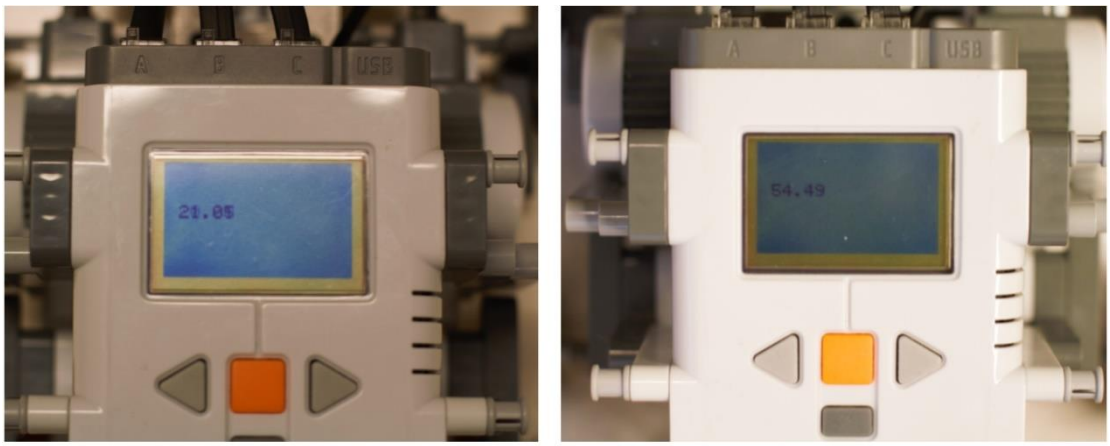


Figure 7. 2 Experiment of non-holonomic robot travelling around two adjacent obstacles and reach the goal

A series of separate experiments was made to support this experiment. The separate experiments were designed to measure the minimum temperature difference which can trigger the PIR sensor. Summarized from a series of experiments, the average value of minimum temperature difference is around 4-5 degrees centigrade.



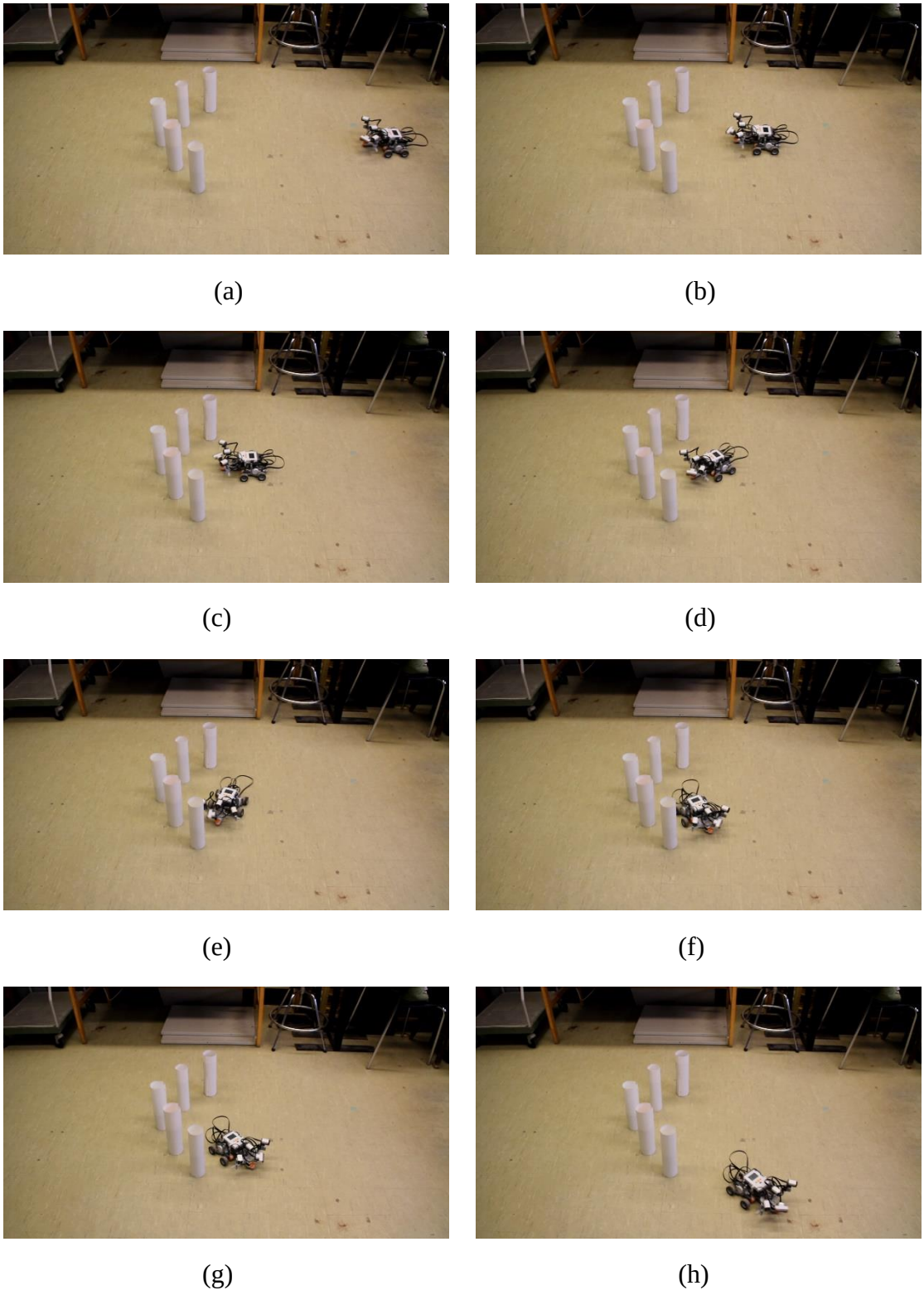
(a) (b)

Figure 7. 3 (a) Room temperature; (b) Heater surface temperature

Using the TIR sensor described in Section4.2.3, we measured that the room temperature during the experiment was 21.05 degrees, and the temperature of the flat heater's surface was 54.49 degrees as shown in Figure 7.3.

7.3 Experiments of Non-Holonomic Robot with Concave Shape Obstacle

In this section, two experiments were prepared in which the first one reappeared the simulation in Section 6.2. Five obstacles or three trash cans formed a concave shape in front of the robot. The obstacles were placed in a C-shape. Experiments in this section only used ultrasonic sensor to show the obstacle avoidance ability of the robot since the goal detection ability was tested in the last section. The process of the two experiments are shown in Figure 7.4 and Figure 7.5.



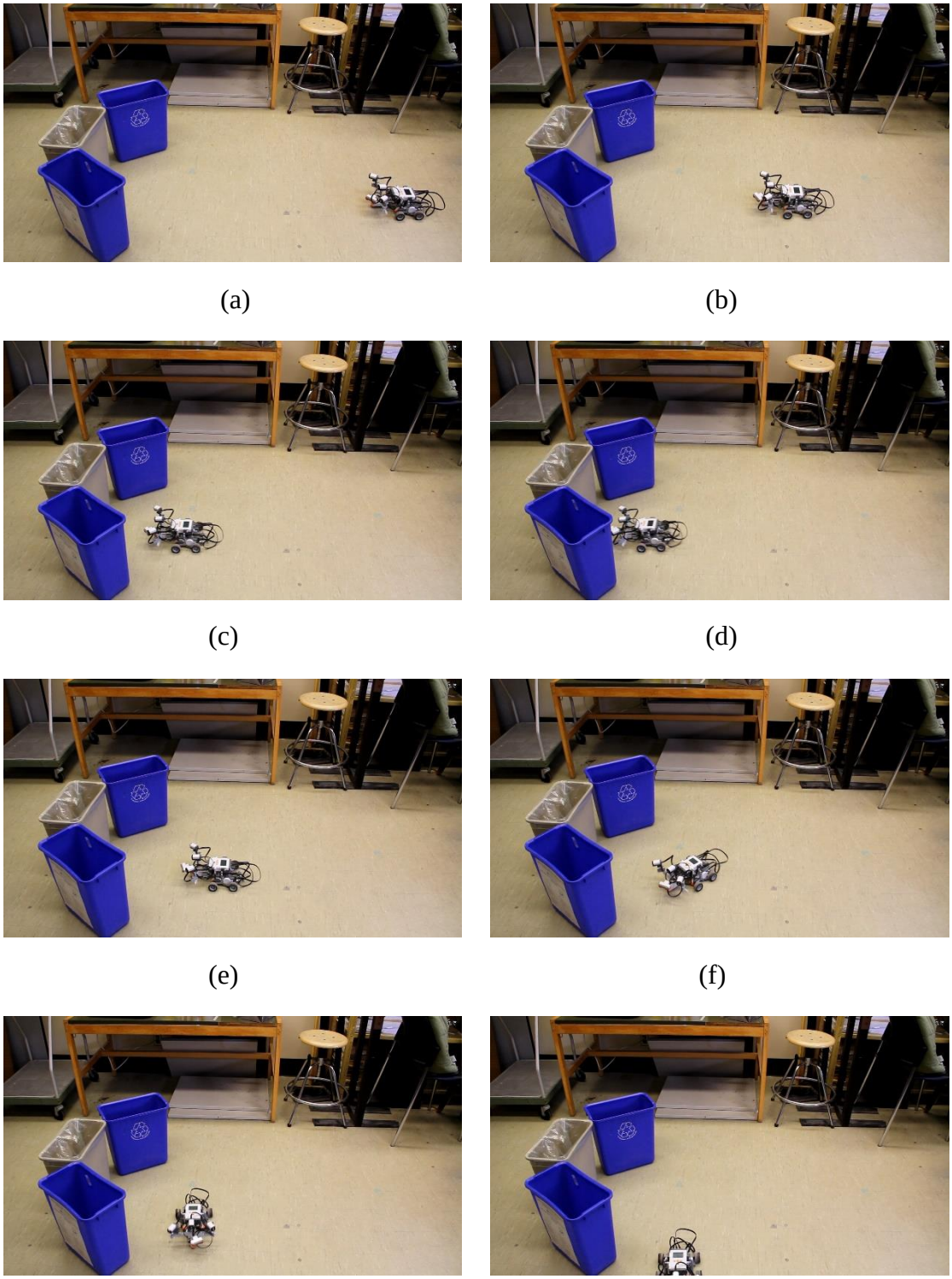
(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 7. 4 Experiment of non-holonomic robot with 5 obstacles



(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 7. 5 Experiment of non-holonomic robot with 3 trash can obstacles

7.4 Experiments of Non-Holonomic Human-Following Robot with Obstacles

This experiment reappeared the third simulation in Chapter 6. And different from all the other experiments above, this experiment was done outdoors while other experiments were processed in the laboratory. It is because the minimum temperature difference that PIR sensor can measure is around 4-5 degrees centigrade. The temperature difference between room temperature in the lab and the surface temperature of a dressed human body is less than 1 degree centigrade when the room temperature is around 20 degrees, see Figure 7.6. The outdoor temperature was -7 degrees and surface temperature of dressed human body was -1 degree, as in Figure 7.7 (experiment was processed at 5:47pm February 9th 2014 in Ottawa, Canada).

(a) (b)

Figure 7. 6 (a) Room temperature; (b) Dressed human body temperature indoor

(a) (b)

Figure 7. 7 (a) Outside temperature; (b) Dressed human body temperature outdoor

The temperature difference was only 6 degrees even outdoors, it was just 1 degree higher than the minimum value PIR sensor can detect. Hence the human following process was not that accurate and the robot lost the human once and rotated anticlockwise to search and sense the human, in Figure 7.8 (l)-(n), when the human went behind the obstacles. After it found the human, it went towards the human. Figure 7.8 (j) and (k) show the robot move backwards when it is too close to the obstacles. During the production of the experiment video, two ladies passed by and seemed interested in the LEGO robot, they walked into the angular field of the camera. The process of the experiment is shown in Figure 7.8.

(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

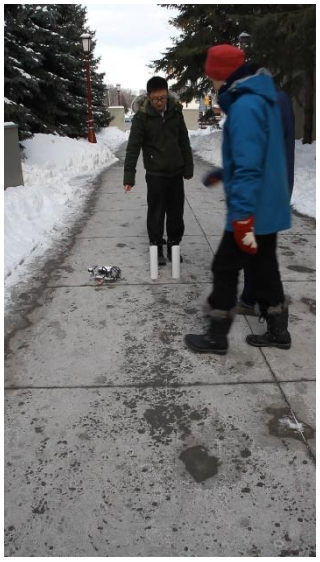
(i)

(j)

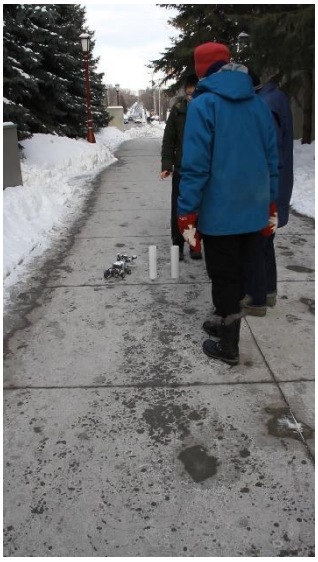
(k)

(l)

(m)



(n)



(o)

(p) (q)

Figure 7. 8 Experiment of human-following robot

Chapter 8

Conclusions

8.1 Summary

In this thesis, we proposed a navigation strategy for human-human following robot in a two-dimensional terrain to track a human without colliding with obstacles in the environment.

For the motion control problem, we focus on the issue of mobile robot motion control with collision avoidance for the case of an unknown environment. The navigation controller was developed using velocity potential fields approach in a modified, quasi-harmonic solution. A quasi-harmonic function based controller uses harmonic solutions for collision avoidance and smoothly changes toward a non-harmonic solution which arrives toward a zero velocity command only when approaching the goal. The velocity potential fields and the resulting velocity vector commands are investigated analytically using symbolic math solutions of MAPLE^{TM} as well as in simulations using MATLAB^{TM}. Motion simulations of both holonomic and non-holonomic robot motion illustrate how the proposed approach operates. Experiments were also made to test the algorithm in environment with simple and complex obstacles. The results show that quasi-harmonic function approach can be used to control a robot to move around obstacles. The robot smoothly slows down when it gets close to the goal and final velocity tends to zero and stop accurately at the goal.

After the motion controller was developed and tested, a human-following strategy was designed to let a non-holonomic robot have the ability to follow a human in an unknown environment with obstacles. The approach is based on velocity potential fields that permit to generate velocity vector commands that drive the robot at a safe

distance with regard to the human while avoiding obstacles. To follow a human, Passive Infrared Sensor (PIR Sensor) was selected in this thesis. To keep a safety distance from the human and also have a stronger and more accurate ability to detect and follow the human, a PIR sensor array was used. The PIR sensor array divides the detection range into several parts. According to these parts, the robot will get the pose information of the human and decide what kind of motion it will take next. Hence the human motion localization will be addressed in the robot frame. Then based on the relative position between human target and robot, the robot gets the capability to follow a human safely. Simulation for the human-tracking process was carried out in MATLAB™, and experiment with LEGO MINDSTORMS NXT robot reproduced the conditions from simulation.

8.2 Main Research Contributions

A quasi-harmonic function based navigation controller was proposed and improved using a velocity potential field navigation controller for robot navigation problem. [60] Velocity potential field method needs to adapt its navigation controller according to obstacles or goals. To avoid obstacles, harmonic functions are used because they do not lead to local minima as in the case of artificial potential fields. However, to reach and stop at a goal area, harmonic function is not suitable because it cannot make the robot stop at the goal point. Proposed solution is a particular non-harmonic function, but such function will lose of the advantages of saddle point of velocity potential fields and may cause local minimum. To make things worse, such switch is always abrupt which is harmful to robot. A quasi-harmonic function based controller overcame the above problem. It uses harmonic solutions for collision avoidance and smoothly changes toward a non-harmonic solution which tends toward a zero velocity command only when approaching the goal. Quasi-harmonic function approach even avoids the saddle point difficulties from the harmonic function approach and includes tangent velocity commands which help the robot move away from unstable equilibrium points.

A human-tracking approach based on PIR sensor array was investigated. [61] Unlike human-tracking robot based with acoustic sensor or optical imaging sensors, the approach proposed in this thesis is a simple, economical and effective way to track a human. The approach does not need sophisticated computer vision algorithms or sufficient illumination conditions and can be realized with simple experiment equipment. The more PIR sensors are used, the more accurate the calculation of human position will be. A high accuracy of human location determination can be achieved if enough PIR sensors were used with this human-tracking approach.

The human-tracking approach along with the quasi-harmonic controller showed that the quasi-harmonic function can be realized for a human-following robot in changing and dynamic environment.

8.3 Future Work

In the next step of the research, a more accurate human position detection algorithm can be developed with a larger PIR sensor array and a robot with stronger processor and more I/O ports. In our current studies, the results of human localization are less precise due to the limitation of LEGO NXT robot and because an important input data for the quasi-harmonic controller is the relative distance. The human-following process based on quasi-harmonic function has some flaws more or less due to this approximate human localization approach.

Furthermore, since motion of human can be very complicated, the robot will be designed to adjust more motion condition of human in the future. For instance, when the human turns back and goes towards the robot, the robot should have ability to move backward to avoid to be found by the human.

Also a human interaction robot with a signal source attached on the human can be developed. The signal source provides robot with the accurate position of the human so that the robot can be better controlled with quasi-harmonic controller to track and assistant the human. The quasi-harmonic controller can also be applied to an human

interaction robot in a intelligent space in which relative position between target human and robot is also easy to be obtained.

References

[1] Flash, T., Meirovitch, Y., & Barliya, A. (2013). Models of human movement: trajectory planning and inverse kinematics studies. Robotics and Autonomous Systems, 61(4), 330-339.

[2] Laumond, J. P., Sekhavat, S., & Lamiraux, F. (1998). Guidelines in nonholonomic motion planning for mobile robots (pp. 1-53). Springer Berlin Heidelberg.

[3] Sekmen, A. S., Wilkes, M., & Kawamura, K. (2002). An application of passive human-robot interaction: human tracking based on attention distraction.Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 32(2), 248-259.

[4] Feng, G., Guo, X., & Wang, G. (2012). Infrared motion sensing system for human-following robots. Sensors and Actuators A: Physical, 185, 1-7.

[5] Hao, Q., Brady, D. J., Guenther, B. D., Burchett, J. B., Shankar, M., & Feller, S. (2006). Human tracking with wireless distributed pyroelectric sensors.Sensors Journal, IEEE, 6(6), 1683-1696.

[6] Shankar, M., Burchett, J. B., Hao, Q., Guenther, B. D., & Brady, D. J. (2006). Human-tracking systems using pyroelectric infrared detectors. Optical Engineering, 45(10), 106401-106401.

[7] Gopinathan, U., Brady, D., & Pitsianis, N. (2003). Coded apertures for efficient pyroelectric motion tracking. Optics express, 11(18), 2142-2152.

[8] Sakurai, A., Nakamura, M., & Nakamura, J. (2011). Self-organizing map analysis of sensor networks for human movement tracking. Sensors and Actuators A: Physical, 166(1), 141-148.

[9] Connolly, C. I., Burns, J. B., & Weiss, R. (1990, May). Path planning using Laplace's equation. In Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on (pp. 2102-2106). IEEE.

[10] Akishita, S., Kawamura, S., & Hayashi K. (1990). New navigation function utilizing hydrodynamic potential for mobile robot. In Proceedings of the IEEE International Workshop on Intelligent Motion Control. 1990, vol. 2, pp. 413–417.

[11] Jo, D., & Didier, K. (1991). A reactive robot navigation system based on a fluid dynamics metaphor. In Parallel Problem Solving from Nature (pp. 355-362). Springer Berlin Heidelberg.

[12] Kim, J. O., & Khosla, P. K. (1992). Real-time obstacle avoidance using harmonic potential functions. Robotics and Automation, IEEE Transactions on, 8(3), 338-349.

[13] Masoud, A. A. (2007). Decentralized self-organizing potential field-based control for individually motivated mobile agents in a cluttered environment: A vector-harmonic potential field approach. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 37(3), 372-390.

[14] Chanson, H. (2009). Applied hydrodynamics: an introduction to ideal and real fluid flows. CRC Press.

[15] Sasiadek, J. Z., & Necsulescu, D. (2012, October). Decentralized Control of Autonomous Mobile Robots Formations Using Velocity Potentials. InMultivehicle Systems (Vol. 2, No. 1, pp. 31-36).

[16] Luo, R. C., Huang, C. H., & Lin, T. T. (2010, November). Human tracking and following using sound source localization for multisensor based mobile assistive companion robot. In IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society (pp. 1552-1557). IEEE.

[17] Murray, J. C., Erwin, H., & Wermter, S. (2004, September). Robotics sound-source localization and tracking using interaural time difference and cross-correlation. In Proceedings of NeuroBotics Workshop (pp. 89-97).

[18] Burion, S. (2004). Human detection for robotic urban search and rescue (No. LSRO2-REPORT-2004-002).

[19] Granata, C., Bidaud, P., Salini, J., & Ady, R. (2013). Human activity analysis: a personal robot integrating a framework for robust person detection and tracking and physical based motion analysis. Paladyn, Journal of Behavioral Robotics,4(2), 131-146.

[20] Huang, J., Supaongprapa, T., Terakura, I., Wang, F., Ohnishi, N., & Sugie, N. (1999). A model-based sound localization system and its application to robot navigation. Robotics and Autonomous Systems, 27(4), 199-209.

[21] Sonoura, T., Yoshimi, T., Nishiyama, M., Nakamoto, H., Tokura, S., & Matsuhira, N. (2008). Person following robot with vision-based and sensor fusion tracking algorithm. Computer vision, 519-538.

[22] Lee, J., Knox, B., & Stone, P. (2008). Inter-classifier feedback for human-robot interaction in a domestic setting. Journal of Physical Agents, 2(2), 41-50.

[23] Burke, M. G. (2011). Visual servo control for a human-following robot (Doctoral dissertation, Stellenbosch: University of Stellenbosch).

[24] Warszawska, P., Automatyki, I., & Stosowanej, I. I. (2013). Paweł Rabiński Vision based gesture-driven human-robot interface.

[25] Muñoz-Salinas, R., Aguirre, E., García-Silvente, M., & Gonzalez, A. (2005). People detection and tracking through stereo vision for human-robot interaction. In MICAI 2005: Advances in Artificial Intelligence (pp. 337-346). Springer Berlin Heidelberg.

[26] Shah, H. N. M., Ab Rashid, M. Z., & Tam, T. Y. (2013). Develop and Implementation of Autonomous Vision Based Mobile Robot Following Human.International Journal of Advanced Science and Technology, 51.

[27] Braun, T., Szentpetery, K., & Berns, K. (2005, June). Detecting and following humans with a mobile robot. In Proceedings of the EOS Conference On Industrial Imaging and Machine Vision.

[28] Cai, Q., & Aggarwal, J. K. (1999). Tracking human motion in structured environments using a distributed-camera system. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 21(11), 1241-1247.

[29] Lee, S. J., Shah, G., Bhattacharya, A. A., & Motai, Y. (2012). Human tracking with an infrared camera using a curve matching framework. EURASIP Journal on Advances in Signal Processing, 2012(1), 1-15.

[30] Naik, N. (2004). Infrared Imaging Sensor Brick for Modular Robotics.

[31] Fernández-Caballero, A., Castillo, J. C., Martínez-Cantos, J., & Martínez-Tomás, R. (2010). Optical flow or image subtraction in human detection from infrared camera on mobile robot. Robotics and Autonomous Systems, 58(12), 1273-1281.

[32] Yang, Y., Feng, G., Wang, S., Guo, X., & Wang, G. (2013, June). Using bearing-sensitive infrared sensor arrays in Motion localization for human-following robots. In Control Conference (ASCC), 2013 9th Asian (pp. 1-5). IEEE.

[33] Fang, J. S., Hao, Q., Brady, D. J., Guenther, B. D., & Hsu, K. Y. (2006). Real-time human identification using a pyroelectric infrared detector array and hidden Markov models. Optics express, 14(15), 6643-6658.

[34] Jin, T., Lee, J., & Hashimoto, H. (2006). Position control of mobile robot for human-following in intelligent space with distributed sensors. INTERNATIONAL JOURNAL OF CONTROL AUTOMATION AND SYSTEMS, 4(2), 204.

[35] Morioka, K., Lee, J. H., & Hashimoto, H. (2008). Intelligent Space for Human Centered Robotics. Advances in Service Robotics, 181-192.

[36] Morioka, K., Lee, J. H., & Hashimoto, H. (2002). Human centered robotics in intelligent space. In Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on (Vol. 2, pp. 2010-2015). IEEE.

[37] Morioka, K., Oinaga, Y., & Nakamura, Y. (2011). Control of Human-Following Robot Based on Cooperative Positioning with an Intelligent Space. IEEJ Transactions on Electronics, Information and Systems, 131, 1050-1058.

[38] Hashimoto, H., & Brscic, D. (2009). Mobile robot as physical agent of intelligent space. CIT. Journal of Computing and Information Technology, 17(1), 81-94.

[39] Brscic, D., & Hashimoto, H. (2007, November). Tracking of humans inside intelligent space using static and mobile sensors. In Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE (pp. 10-15). IEEE.

[40] Sasaki, T., & Hashimoto, H. (2006, October). Human observation based mobile robot navigation in intelligent space. In Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on (pp. 1044-1049). IEEE.

[41] Dang, Q. K., & Suh, Y. S. (2011, October). Human-following robot using infrared camera. In Control, Automation and Systems (ICCAS), 2011 11th International Conference on (pp. 1054-1058). IEEE.

[42] Nagumo, Y., & Ohya, A. (2001). Human following behavior of an autonomous mobile robot using light-emitting device. In Robot and Human Interactive Communication, 2001. Proceedings. 10th IEEE International Workshop on (pp. 225-230). IEEE.

[43] Chanson, H. (2009). Applied hydrodynamics: an introduction to ideal and real fluid flows. CRC Press.

[44] Bertin, J. J. & Smith, M. L. (1998). Aerodynamics for engineers. Prentice Hall. p. 668.

[45] Necsulescu, D., Nie, G. & Sasiadek, J. (2014). Mobile Robot Motion Control in Crowded Environments using Quasi-Harmonic Method. In Method and Models in Automation and Robotics, 2014. Proceedings. 19th International Conference. IEEE.

[46] Zeldovitch, I. & Mychkis, A. (1974). Elements de Mathematiques Applique. Editions M. Moscou.

[47] Greenberg, M. D. (1998). Advanced engineering mathematics (pp. 1177-1178). Prentice-Hall.

[48] Lego Robotics FAQ. (2009). Available: http://en.wikipedia.org/wiki/Lego_Mindstorms#RCX. [Accessed: 12-Feb-2014].

[49] http://en.wikipedia.org/wiki/Lego_Mindstorms#RCX.[Accessed: 15-Feb-2014]

[50] LEGO MINDSTORMS User Guide .

[51] http://en.wikipedia.org/wiki/Lego_Mindstorms_NXT.[Accessed: 15-Feb-2014]

[52] http://en.wikipedia.org/wiki/Speed_of_sound. [Accessed: 24-Mar-2014].

[53] http://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NIS1070. [Accessed: 17-Feb-2014]

[54] http://dexterindustries.com/manual/thermal-infrared-sensor/.[Accessed: 12-Mar-2014]

[55] http://en.wikipedia.org/wiki/Lego_Mindstorms_NXT_2.0#Sensors. [Accessed: 12-Mar-2014]

[56] http://www.philohome.com/sensors/legorot.htm. [Accessed: 12-Mar-2014]

[57] http://www.philohome.com/nxtmotor/nxtmotor.htm. [Accessed: 14-Mar-2014]

[58] Sasiadek, J. Z., Lu, Y., & Polotski, V. (2006, September). Navigation of autonomous mobile robot with gate recognition and crossing. In Robot Control(Vol. 8, No. 1, pp. 115-120).

[59] Bemporad, A., De Luca, A., & Oriolo, G. (1996, April). Local incremental planning for a car-like robot navigating among obstacles. In Robotics and Automation, 1996. Proceedings. 1996 IEEE International Conference on (Vol. 2, pp. 1205-1211). IEEE.

[60] Necsulescu, D. & Nie, G. (2014). Quasi-harmonic Approach to Non-holonomic Robot Motion Control with Concave Obstacles Avoidance. In Chinese Control and Decision Conference (CCDC), 2014 26th Internationcal Conference. IEEE.

[61] Nie, G. & Necsulescu, D. (2014). Quasi - Harmonic Functions Approach to Human - Following Robots. In International Conference of Control, Dynamic Systems, and Robotics (CDSR), 2014 1st Internatical Conference. Avestia.

[62]http://www.generationrobots.com/en/content/61-nxt-g-programmation-lego-mindstorms-nxt [Accessed: 14-Mar-2014]

Appendix A

Development Environment of LEGO NXT Robot

The LEGO NXT robot is equipped with CPU, actuators and sensors to realize autonomous motion in an unknown and dynamic environment. Its autonomy depends on the successful implementation of reliable control algorithms. How to combine those controllers with the hardware of the robot to drive and organize the hardware instead of just icy components stacked together? The answer is the software. If hardware is like a body of a human, program represents the mind or consciousness of the human, then software is a method to let the human accept the consciousness or to teach the human how to think. Software is like an education system and different programs gives different educational models. For instance, both western education and traditional eastern education teach human to learn, but we all know there are differences between these two education patterns.

This section will describe the software required to build controllers for the LEGO NXT robot. As introduced at the beginning of Chapter 4, LEGO offers a friendly programming environment by supporting many kinks of programming languages. Hence there are various kinds of software based on different programming languages to build controller for LEGO NXT robot. Table A-1 gives some software which can program for LEGO NXT, but the list does not include all features and all kinds of software.

Table A- 1 NXT programming software

Software	Language type	Control type	NXT Required Firmware	Link Type	Link Source	Read Sensors
NXT-G Retail	Graphic	Standard	Standard	USB/BT	Desktop	Yes

LEGO NXT Mobile Application	Simple RC	Remote Control	Standard (# 2)	Bluetooth	Phone or PDA	No
FunkNXT	Simple RC	Remote Control	Standard	Bluetooth	Phone	Yes
BT RC	NXT-G	NXT to NXT Remote	Program running on NXT	Bluetooth	Another NXT	User Programmable
Simple BT Remote	Simple RC	Remote Control	Standard	Bluetooth	Desktop	Yes
RobotC	Simple RC	Remote Control	Standard (# 1)	USB/BT	Desktop	Yes
BricxCC	Simple RC	Remote Control	Standard	USB/BT	Desktop	Yes
OnBrick PDA	Graphic	Programmable RC	Standard	Bluetooth	PDA	Yes
OnBrick PC	Graphic	Programmable RC	Standard	Bluetooth	Desktop	Yes
NXT Director	Simple RC	Customizable Remote Control	Standard	Bluetooth	Palm PDA	No?
RoboDNA	Simple RC	Remote Control	Standard	Bluetooth	Desktop	Yes
MS Robotics Studio	.NET	User Program running on PC	Standard	Bluetooth	Desktop	Yes
NI LabVIEW Toolkit	Graphic (LabVIEW G)	User Program running on PC	Standard	USB/BT	Desktop	Yes
RoboLab	Graphic	User Program running on PC	Standard	USB	Desktop	Yes
iCommand	Java	User Program running on PC	Standard	Bluetooth	Desktop or PDA	Yes
LEGO::NXT	Perl	User Program running on PC	Standard	USB/BT	Desktop	Yes
nxt-Ruby	Ruby	User Program running on PC	Standard	Bluetooth	Desktop	Yes
NXT#	C#	User Program	Standard	Bluetoot	Desktop	Yes?

		running on PC		h		
Mindsqualls	C#	User Program running on PC	Standard	Bluetoot h	Desktop	Yes
NXT Python	Python	User Program running on PC	Standard	USB/BT	Desktop	Yes?
My Robot Me	Graphic	User Program running on PC	Standard	USB/BT	Desktop	Yes

Two popular NXT programming software will be described in detail in this Chapter. Section A.1 introduces the official LEGO MINDSTORMS NXT-G 2.0 programming software. Section A.2 presents another popular programming software, NI LabVIEW Toolkit, which is more general than NXT-G and also can be used for LEGO NXT.

A.1 LEGO MINDSTORMS NXT-G 2.0

NXT-G is a graphical programming environment which LEGO created it specifically for LEGO MINDSTORMS NXT robot based on NI LabView. NXT-G is also called G language. Since it can make complicated programs with simple drag-and-drop, graphical programming language, NXT-G has become one of the top 20 popular program languages according to TIBOE programming community index 2012. NXT-G is compatible with Mac or PC and the minimum technical requirements are listed in the Table A-2 [62].

Table A- 2 Minimum technical requirements of NXT-G

Microsoft Windows	**Apple Macintosh**
1.5 GHz processor	**Power PC G3, G4, G5 to 1.3 GHz processor**
512 MB RAM	**512 MB RAM**
300 MB of disk space	**300 MB of disk space**
1 USB port	**1 USB port**
CD-ROM drive (for installation)	**CD-ROM drive (for installation)**
Bluetooth adaptor (optional, required for Bluetooth communication)	**Bluetooth adaptor (optional, required for Bluetooth communication)**

Figure A.1 [62] shows the NXT-G development environment. The palette is the area where the function blocks that can be dragged and dropped into the work area.

The work area is where program is made. The configuration panel includes settings of a selected block in the work area and the selected block can be adjusted in this area. The controller is used to transfer program from PC to the robot using the USB connection or Bluetooth. The Robot Center is the place where the tutorials can be found. There are 4 main categories of robots that can be constructed using Lego MINDSTORMS NXT, including vehicles, robot arms, animals and humanoids. A second tab is available in this space, giving access to the Lego portal where you will be able to visit forums, download additional sounds for the robot, etc. The online help contains all the support that might be needed to use the interface.

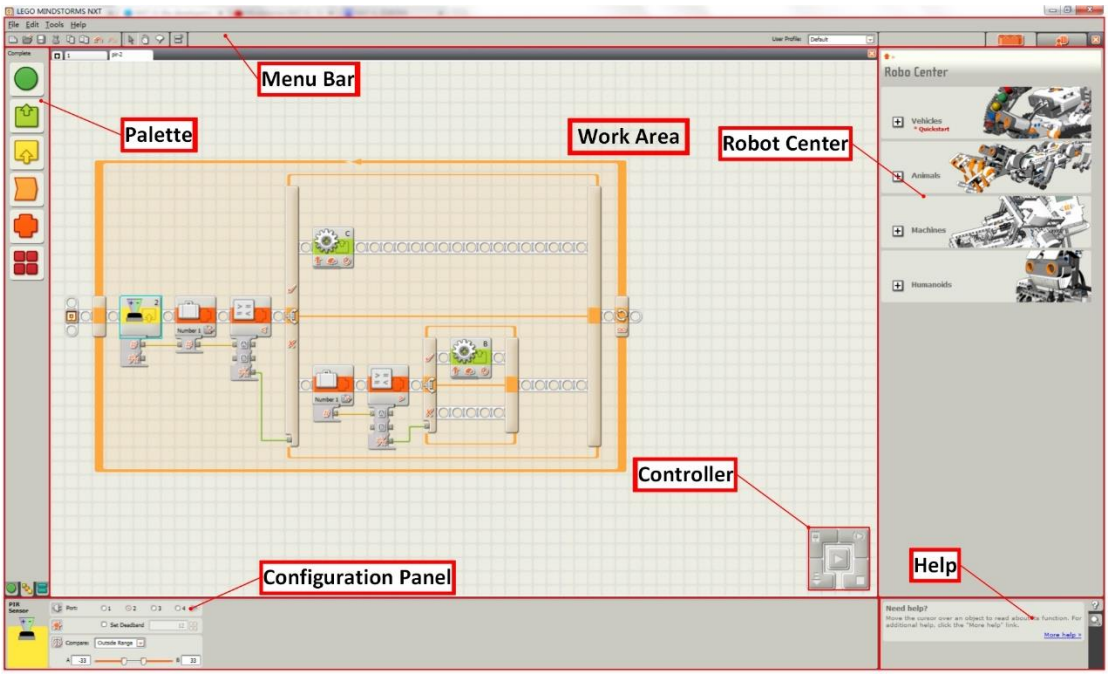


Figure A. 1 NXT-G development environment

A.1.1 Programming Principles of NXT-G

Programming using NXT-G is entirely graphical. It is easy to learn and produce advanced programs using NXT-G even you have no programming knowledge before. No code will be using in NXT-G environment. By default, only a starting point will be shown in the work area at the very beginning of a new program. From this starting point, boxes from the palette can be dragged and dropped into the Work Area to

design a program. Only one block can be dragged at a time, hence the blocks are added one after the other on an axis termed the sequence beam. Figure A.2 [62] shows the programming model and thread of NXT-G.

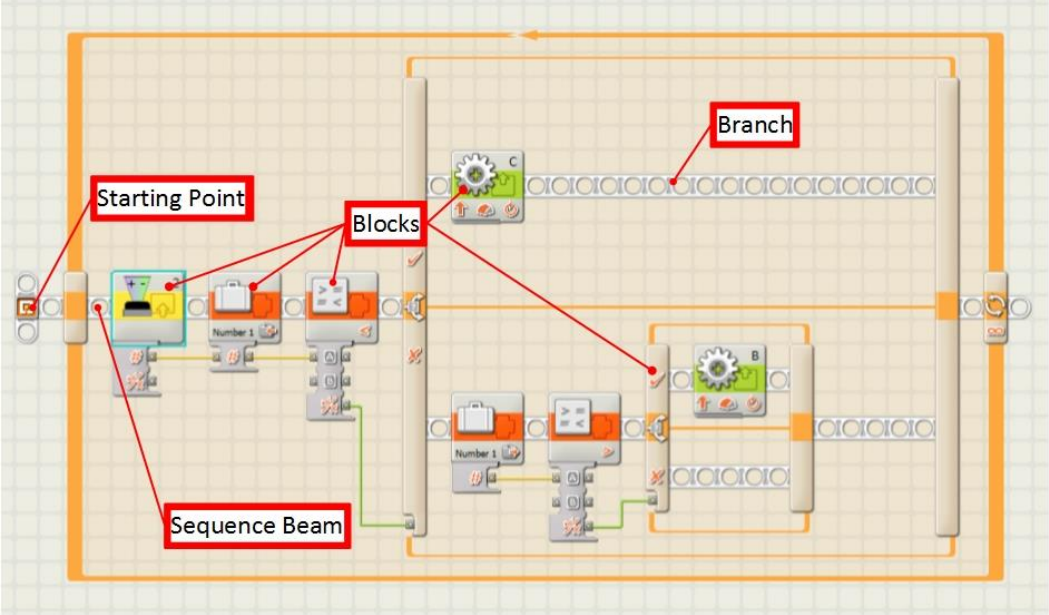


Figure A. 2 Programming model and thread of NXT-G

Blocks that are not connected to a sequence beam are not designed into the program. A no connection block appears fuzzily in the work area as the label shows in Figure A.3.

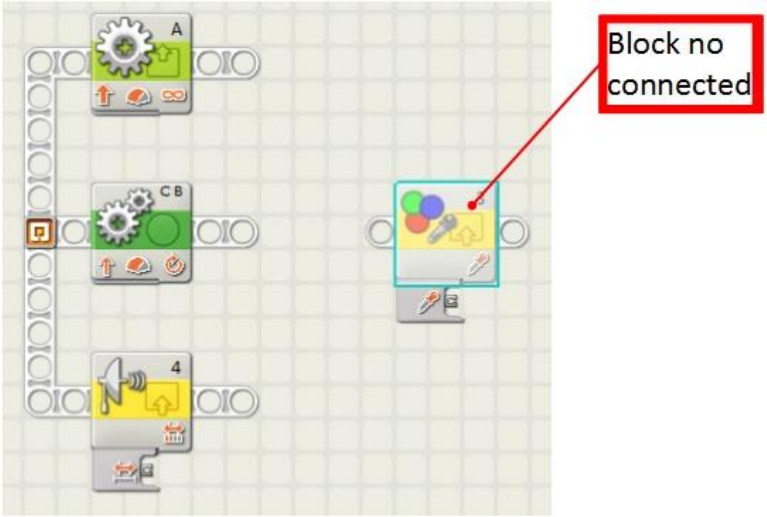


Figure A. 3 Parallel program and no connection block

It is possible to run more than one program sequence in parallel in NXT-G by creating additional sequence beams from the starting point, as shown in the following Figure A.3.

A.1.2 The Blocks Available in NXT-G

Figure A.4 shows the initial blocks provided by NXT-G. There are 35 blocks (right side of Figure A.4) totally divided into 7 main families (left side of Figure A.4).

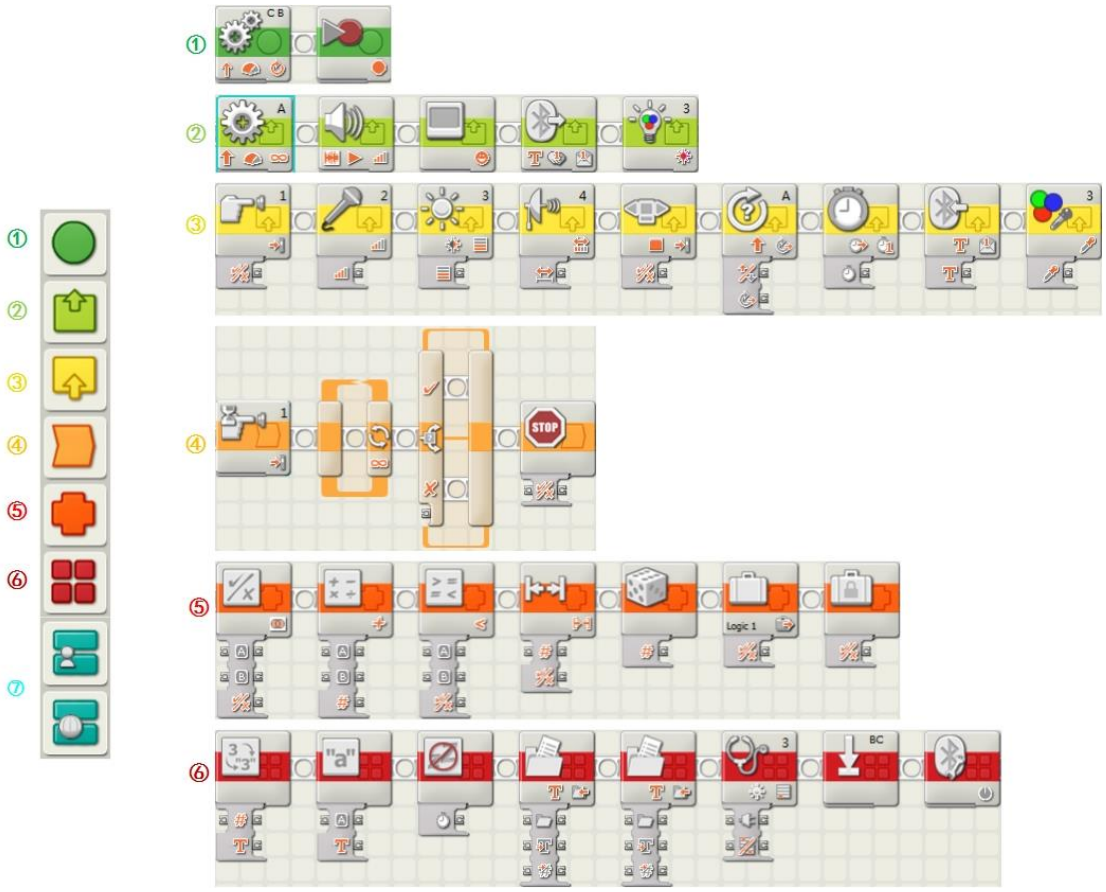


Figure A. 4 Blocks available in NXT-G

1) Common Blocks.

Common Blocks are colored in dark green in Figure A.4. There are two blocks: Move Block and Record/Play Block. The Move Block is used to set a robot to go forwards or backwards in a straight line or to turn by following a curve. To define how far the robot will go is done using the Duration property.

2) Action Blocks

Action Blocks are used to make control the robot results in an action. In other words, these blocks will be connected with output ports on NXT Brick. The color of this family is grass green and in Figure A.4, for the family members are Motor Block,

Sound Block, Display Block, Send Message Block, and Color Lamp Block from left to right. One point to mention is the difference between Motor Block and Move Block. The Move Block allows for precise control of one motor's speed, while the Move Block controls all motors together.

3) Sensor Blocks

Sensor Blocks are designated with yellow. There are 9 blocks offered given initially by NXT-G which are Touch Sensor Block, Sound Sensor Block, Light Sensor Block, Ultrasonic Sensor Block, NXT Buttons Block, (Built-in) Rotation Sensor Block, Timer Block, Receive Message Block, Color Sensor Block. Sensor Blocks are connected to the input ports of the NXT Brick.

4) Flow Blocks

Flow Blocks are designated in Orange. This family always presents the logic of the controller in a program. Wait Block lets the robot sense its environment for a certain condition before it continues. Loop Block used to repeat sequences of code. Switch Block used to choose between two sequences of code. Stop Block used to stop the program and any running motors, lamps or sounds.) are included in this Family.

5) Data Blocks

Designated in red color, Data Blocks include Logic Block, Math Block, Compare Block, Range Block, Random Block, Constant Block, Variable Block.

6) Advanced Blocks

Advanced Blocks are designated in dark red as shown in Figure A.4 shows. This family contains Number to Text Block, Text Block, Keep Alive Block, File Access Block, Calibration Block, Reset Motor block, Bluetooth Connection Block.

7) Custom Blocks

Custom Blocks showed in cyan, are the two blocks: My Blocks and Internet Blocks. Custom Blocks, allow programmers to create their own blocks or download blocks deigned by others, making the programming more flexible. When a non-standard sensor is purchased (e.g. TIR sensor introduced in Section 4.2.3 and PIR sensor introduced in Section 4.2.2), the sensor should be supplied with an additional block on

the producer's website making it possible to configure and interact with the sensor, as shown in Figure A.5.

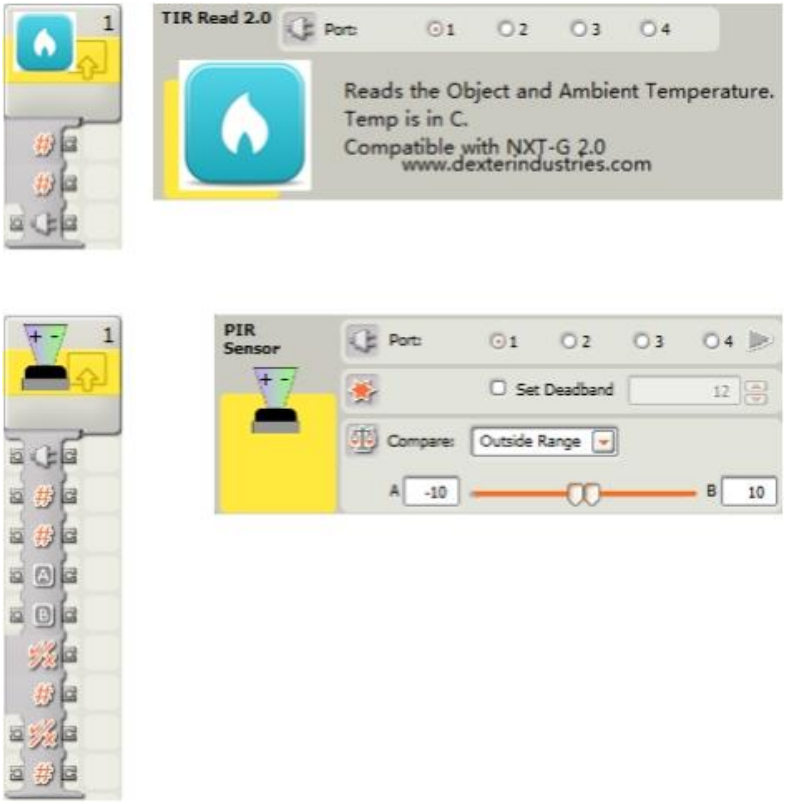


Figure A. 5 TIR block (upper) and PIR block (below)

A.2 National Instrument LabVIEW 2012

Laboratory Virtual Instrument Engineering Workbench (LabVIEW) is a development environment and system-design platform for a virtual programming language researched and developed by National Instrument. The programming language used in LabVIEW is a dataflow programming language. Execution is determined by the structure of a graphical block diagram (the LabVIEW-source code) on which the programmer connects different function-nodes by drawing wires. As with C and Basic, LabVIEW is also a general-purpose programming language, which has a function library large enough to complete any programming task. The function library of LabVIEW includes data acquisition, GPIB, serial port control data analysis, data display, data storage, etc. There are also traditional program debugging tools

included in LabVIEW to facilitate program debugging, for instance, setting breakpoints, displaying data and sub VIs for animation way and so on. Other than C and Basic development environment, LabVIEW utilizes Graphical Language (G language) as the programming language and the form of its program is block diagram. Graphical Language has become one of the mainstream program language nowadays.

A.2.1 Introduction to VI and Graphical Programming

LabVIEW brings the creation of user interface into the development cycle. The program of LabVIEW is called a virtual instrument (VI). VI consists of three parts which are front panel, back panel and connector panel.

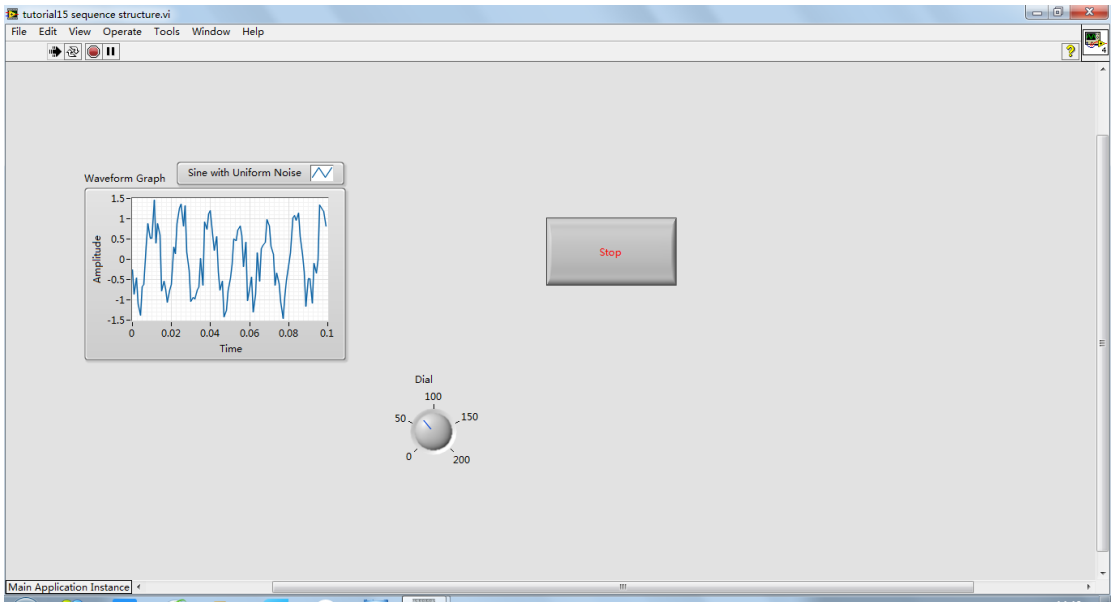


Figure A. 6 Front panel of a single VI

On the front panel, customs can build using controls and indicators. Figure A.6 shows a VI's front panel. Controls, which allow customs to give information to the VI, can be seen as inputs of a program. Figure A.7 (a) shows a part of frequently-used control blocks. Corresponding to controls, indicators, shown in Figure A.7(b), are outputs of data. They display the results of a program based on the input information from controls. Connector panel is used to call block diagrams of other VIs.

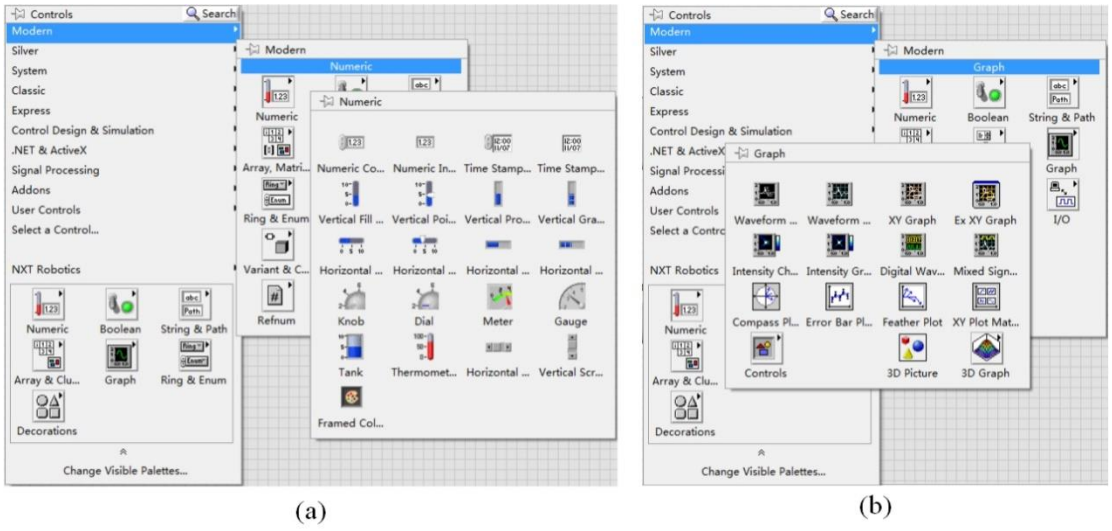


Figure A. 7 (a) Frequently-used controls; (b) Frequently-used indicators

The back panel, on which a block diagram is designed, contains the graphical source code. All of the controls and indicators placed on the front panel will appear on the back panel as terminals as illustrated in Figure A.8. The back panel also contains structures and functions which determine the way controls supply data to indicators. The structures and functions, as Figure A.9 (a) and (b) show, can be found on the Functions palette and placed on the back panel.

There are nodes for structures and functions, through which structures and functions are connected to one another using wires. And at the same time, on the front panel, corresponding controls and indicators will also be connected. For example, in Figure A.6 and Figure A.8, a rotary panel controller was wired to a simulation signal function so that the scope can be wired to the simulation signal function so that the scope displays the sinusoidal wave controlled by the rotary panel. Thus a VI can be easily tested before being embedded as a subroutine into a larger program because program on the back panel can be run and debug in a VI interface when controls (inputs) and indicators (outputs) on the front panel serve as a user interface. Compared with classical instruments, virtual instruments can greatly reduce the research and development efforts.

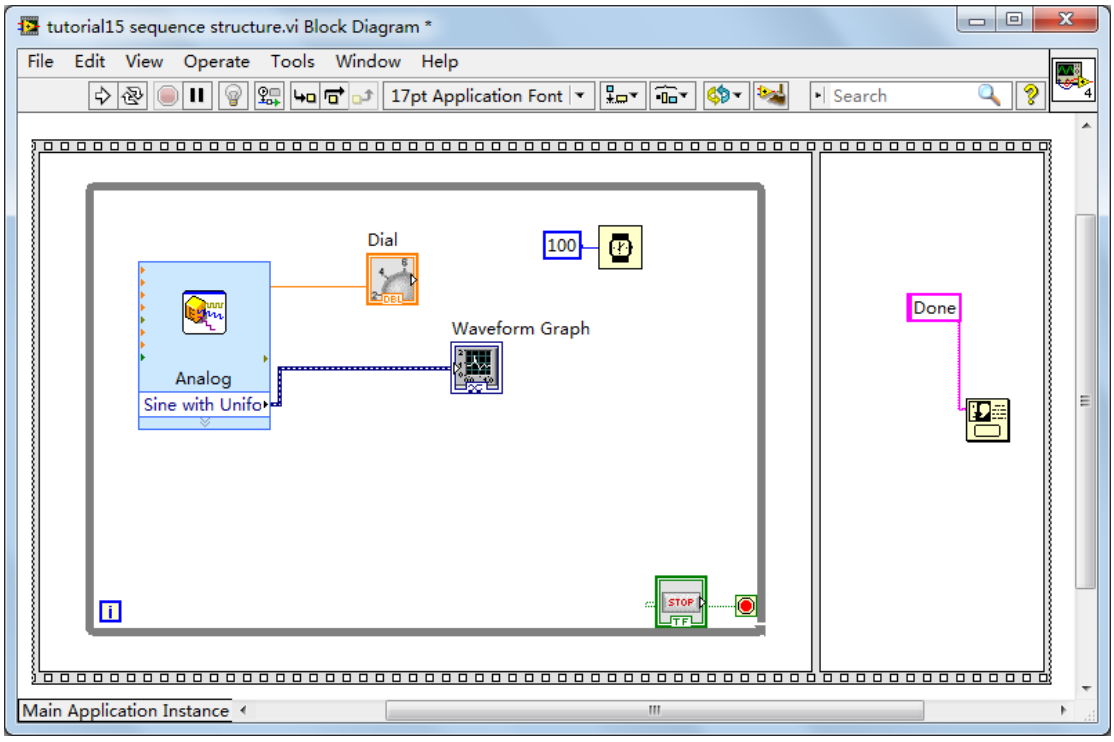


Figure A. 8 Back panel of a single VI

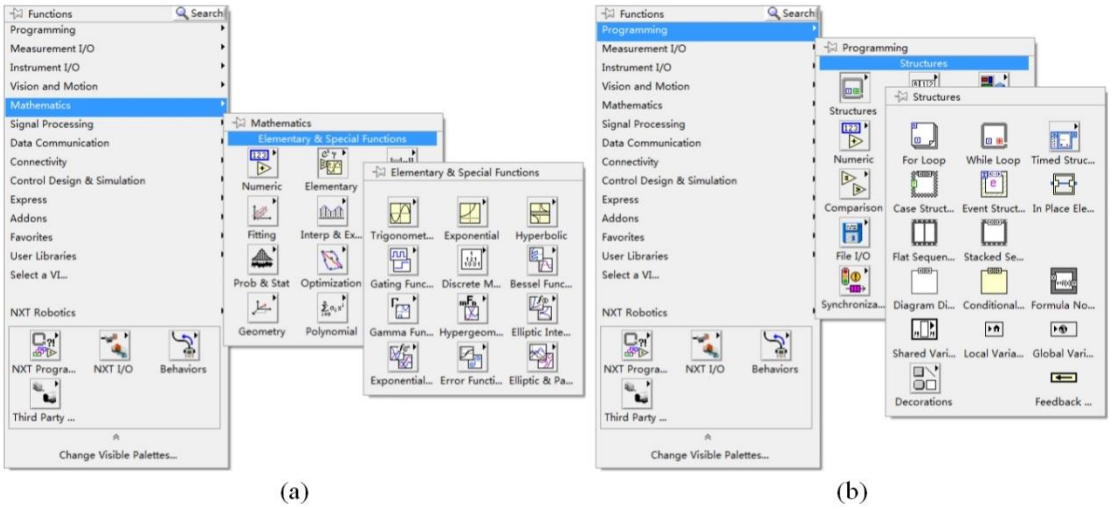


Figure A. 9 (a) Examples of functions; (b) structures

A.2.2 LabVIEW for LEGO MINDSTORMS

As has been introduced in Section A.1, the LEGO NXT-G was based on and powered by NI LabVIEW. Hence NI LabVIEW can be directly used to program for

LEGO NXT robot. To do this, LabVIEW 2012 SP1 LEGO MINDSTORMS NXT module patch needs to be installed. This patch gives LabVIEW users a series of LEGO NXT controllers and indicators, with which LEGO programs can be made. After the patch was installed, some basic LEGO NXT blocks appeared on the functions palette of LabVIEW as Figure A.10 shows.

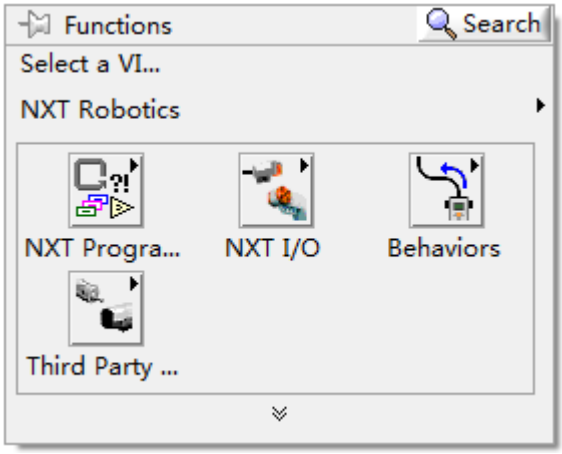


Figure A. 10 Functions included in LEGO NXT module patch for LabVIEW

The reason why LabVIEW 2012 was chosen rather than the latest version LabVIEW 2013 is that besides LEGO MINDSTROMS NXT module patch, there is a LabVIEW MINDSTORMS competition toolkit only for LabVIEW 2012. The official description about this toolkit is "The MINDSTORMS Competition Toolkit (MCT) that contains VIs that are specifically designed to help you program a robot in competitions using the LEGO MINDSTORMS NXT brick, like the FIRST Tech Competition (FTC). This software is meant to be installed only after installing LabVIEW for LEGO MINDSTORMS 2012. " This toolkit might give users more choices when programming. After installed the toolkit, the start interface became as shown in Figure A.11, and an additional function series appeared on NXT Robotics function palette as shown in Figure A.12.

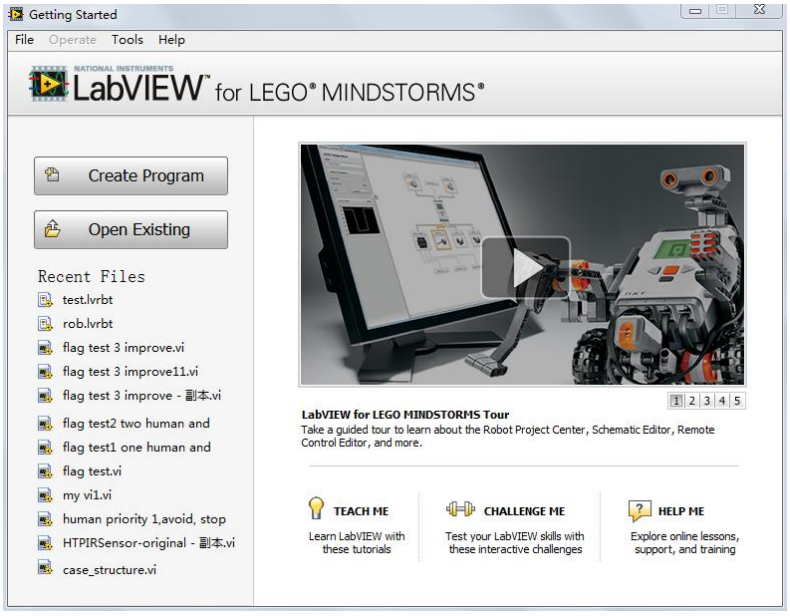


Figure A. 11 Start interface of LabVIEW for LEGO MINDSTORMS

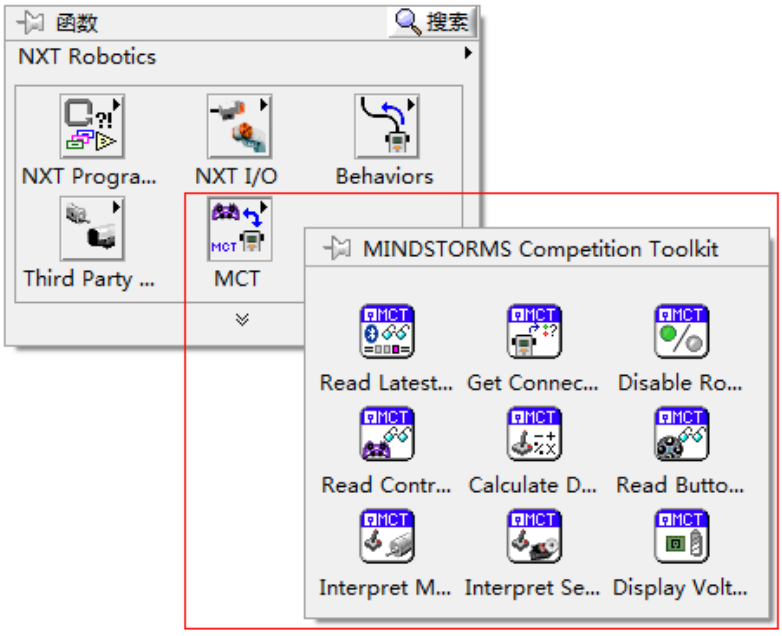


Figure A. 12 MINDSTORMS competition toolkit

LabVIEW for LEGO MINDSTORMS provides unique features for programming the NXT including:

- NXT Schematic Editor visualized configuration and testing of motor and sensor connections
- NXT Remote Control Editor to control the NXT using a joystick or the keyboard

- Additional NXT tools:
- Data Viewer
- Sensor Viewer to view real-time sensor data from a PC
- Remote Display to display NXT screen and buttons on a PC
- Piano Player
- Picture Editor
- NXT Terminal helps users to manage programs and memory on the NXT
- Instructional videos, tutorials, and teaching resources, are built in the software including: Getting Started videos and tutorials. LabVIEW programming challenges integrated in the software environment. A library of program templates. Beginner and advanced user-interface modes. Online community.

The NXT functions are classified into five categories in LabVIEW for LEGO MINDSTORMS as illustrated in Figure A.12 and A.13.

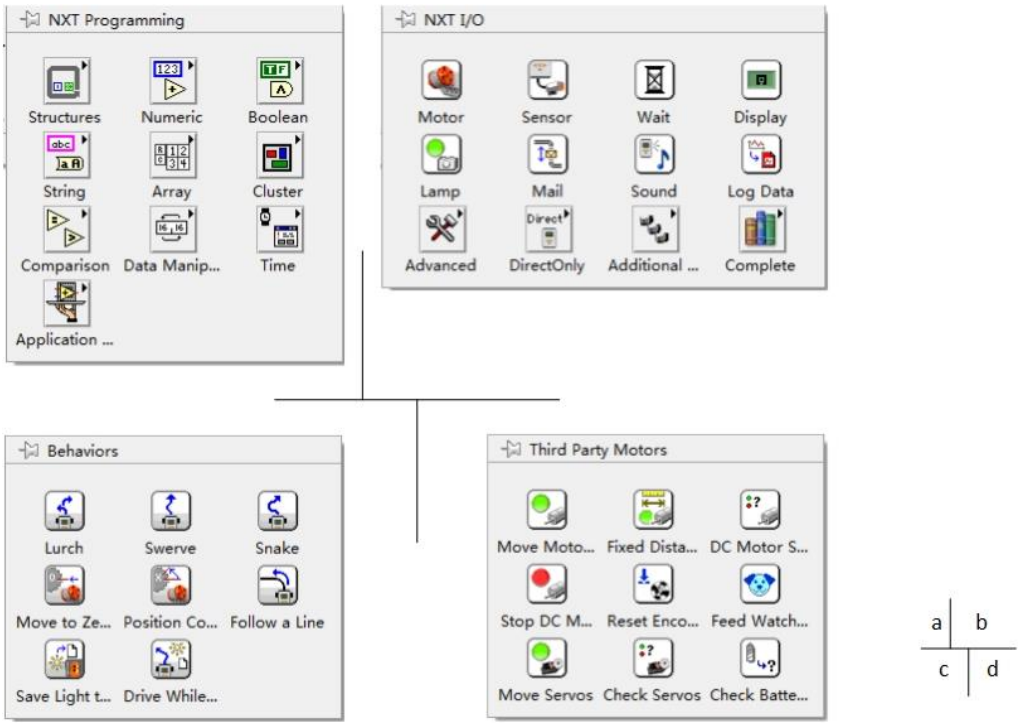


Figure A. 13 (a) NXT programming; (b) NXT I/O; (c) Behaviors; (d) Third party motors

NXT programming functions, shown in Figure A.13 (a), are used to build the structure and realized the logic part of the control program. Typical functions are structure functions, numeric functions, comparison functions, etc.

NXT I/O functions, in Figure A.13 (b), include functions related to controllers, indicators, and actuators. For instance, to control sensors, display, and motors, I/O functions have to be utilized.

Behaviors functions are some typical motion programs that are ready-made. As in Figure A.13 (c), these functions actually can be understood as some pre-programmed sub functions. Users can directly put them into their programs if needed to save time and energy.

MINDSTORMS Competition Toolkit functions are shown in Figure A.12. It gives some fixed model used for LEGO companions.

There are also functions for nine popular third party motors in the 4th series of functions shown in Figure A.13 (d). In addition, some frequently used third party sensors and motors are categorized into the NXT I/O functions as Figure A.14 shows.

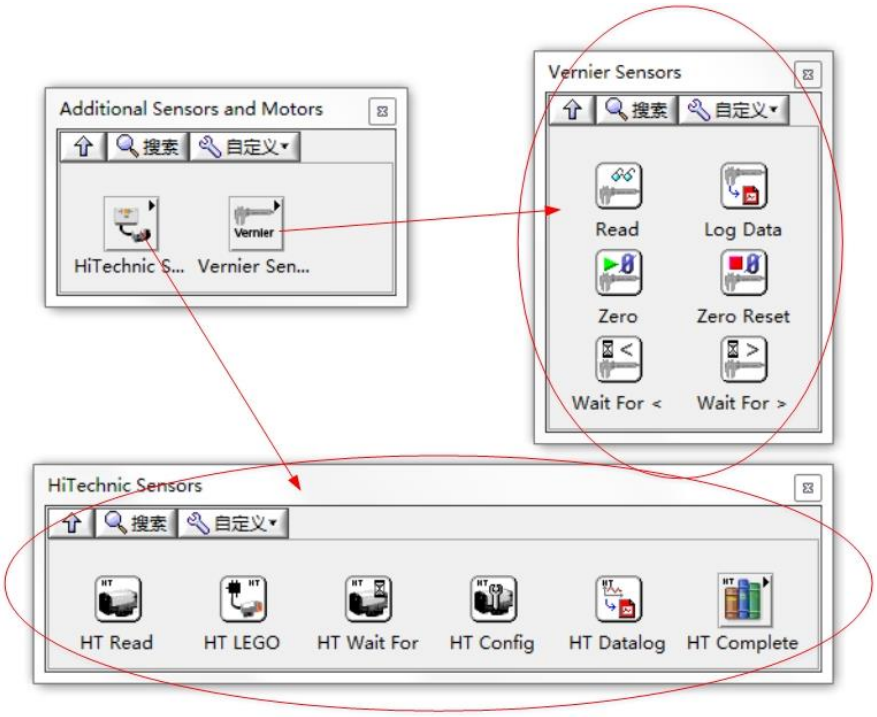


Figure A. 14 Additional third party sensors and motors

However, not every third party sensors or motors have a LabVIEW NXT function. For instance, PIR sensor and TIR sensor used in our experiments are not included. Whether a sensor's block is included in LabVIEW depends on whether the producer of the sensor designs the corresponding functions for LabVIEW environment.

Unfortunately, for PIR sensor, only NXT-G block was made. Hence function of PIR sensor needs to be programed customized programmed. A customized PIR sensor program can be as following Figure A.15 illustrateds.

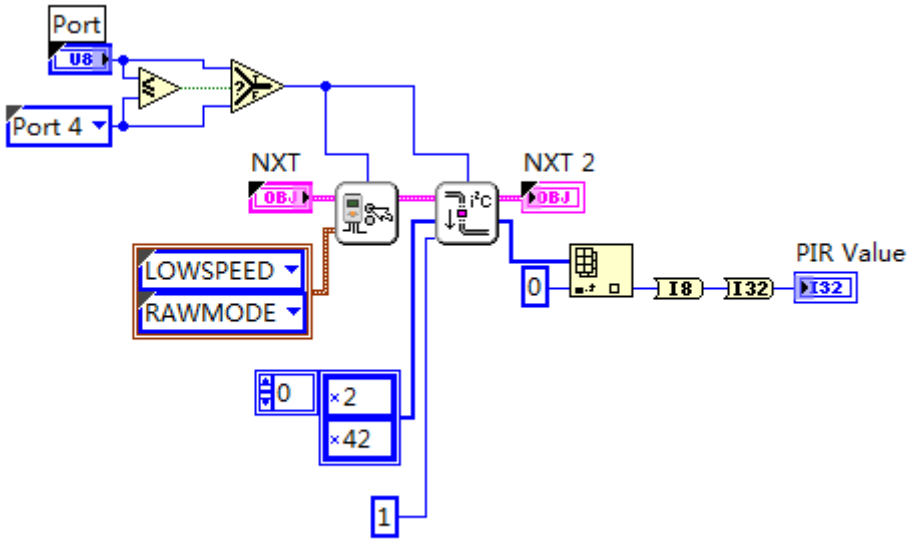


Figure A. 15 A customized PIR sensor function

With all the controllers, indicators and functions in the front and back panel, programs can be designed no matter how complicated they are.

Appendix B

MATLAB Simulation: Holonomic Robot Collision Avoidance Controller Code

B.1 Main Function

```
% Clear all windows
clc
clf

% Initial robot values
robot_pose=[30,5,pi/2];
sensor_radius=4;
max_vel=4;
max_ang_vel=1;

% Map dimensions
map_max=60;
map_min=0;

% Goal position on the map
goal=[map_max/2,map_max];
goal_radius=sensor_radius;

% Time settings
t_sim=100;
T=0.05;
t_steps=floor(t_sim/T);


% move
j=0;
for t=1:t_steps

    % Axis properties
    clf;
```

```
    xmax=map_max;
    xmin=map_min;
    ymax=map_max;
    ymin=0;

axis([xmin,xmax+2*goal_radius,ymin,ymax+2*goal_radius]);
    axis equal;
    axis manual;
    grid on;
    hold on;

    % Plot goal and the circle of interest around the goal
    scatter(goal(1),goal(2),100,'d','m','filled');

rectangle('position',[goal(1)-goal_radius,goal(2)-goal_ra
dius,2*goal_radius,2*goal_radius],'curvature',[1,1],'edge
color','m');

    % Plot the narrow passage
    % Check if sensor readings intersect with an obstacle
    % If an obstacle is in range
      % Set obst_dist and obst_angle values
    % If no obstacle is in view
      % Set obst_dist=0

[narrow_opening]=narrow_passage(robot_pose,sensor_radius)
;
    obst_dist=narrow_opening(1);
    obst_angle=narrow_opening(2);

    % Calculate the new robot pose
    robot=[robot_pose(1),robot_pose(2),robot_pose(3)];
    obst_radius=sensor_radius;

[new_pose]=obstacle_controller(robot,goal,max_vel,max_ang
_vel,goal_radius,obst_radius,obst_dist,obst_angle,T);
    robot_pose=[new_pose(1),new_pose(2),new_pose(3)];
    m(t,:)=robot_pose;
    plot(m(1:t,1),m(1:t,2),'r');

    % Plot the robot and its sensors
    [circle]=sensor_view(robot_pose,sensor_radius);
    [XL,YL]=moving_arrow(robot_pose);
    fill(XL,YL,'r');
```

```
    % Save a picture every 10 seconds
    if(mod(t,10)==0)
        filename=strcat('pics/',int2str(t),'.png');
        saveas(gcf,'filename');
    end
    hold off;
    j=j+1;
    M(j)=getframe;
end

% Save the simulation as an AVI file
movie2avi(M,'video/simulation.avi');
```

B.2 Robot Sensors Detect Obstacles

```
function [ narrow_opening ] =
narrow_passage( robot_pose,sensor_radius)

% Obstacle parameters
obst_radius=sensor_radius;
%ob_width=1;
%ob_x=map_max/2;
%x_dist=0;
%y_dist=0.6*map_max;

% Set the dimmensions of the obstacle
xlimit=[27.5,28.5,31.5,32.5];
ylimit=[29.5,30.5];
ob_leftX=xlimit([1 2 2 1 1]);
ob_leftY=ylimit([1 1 2 2 1]);
ob_rightX=xlimit([3 4 4 3 3]);
ob_rightY=ob_leftY;

% Draw obstacles in the simulation
% Obstacles are colored black 'k'
fill(ob_leftX,ob_leftY,'k');
fill(ob_rightX,ob_rightY,'k');

% Check to see if the robot sensors intersect with an obstacle
obst_dist=0;
obst_angle=0;
for
beam_angle=(robot_pose(3)-pi/2):pi/12:(robot_pose(3)+pi/2
);

x_ob=[robot_pose(1),robot_pose(1)+obst_radius*cos(beam_an
gle)];

y_ob=[robot_pose(2),robot_pose(2)+obst_radius*sin(beam_an
gle)];
    [xT,yT]=polyxpoly(x_ob,y_ob,ob_rightX,ob_rightY);
    [xB,yB]=polyxpoly(x_ob,y_ob,ob_leftX,ob_leftY);
    mapshow(xT,yT,'displaytype','point','marker','o');
    mapshow(xB,yB,'displaytype','point','marker','o');

    obst_distT=sqrt(xT.^2+yT.^2);
```

```
    obst_distB=sqrt(xB.^2+yB.^2);
    if(size(obst_distT,1)==2)
        obst_distT=obst_distT(1);
    elseif(size(obst_distB,1)==2)
        obst_distB=obst_distB(1);
    end


if(isempty(obst_distT)==false&&isempty(obst_distB)==true)
        obst_dist=obst_distT;

elseif(isempty(obst_distT)==true&&isempty(obst_distB)==fa
lse)
        obst_dist=obst_distB;
    end


if(isempty(obst_distT)==false||isempty(obst_distB)==false
);
        obst_angle=beam_angle;
        if(obst_angle>=2*pi)
            obst_angle=obst_angle-2*pi;
        elseif(obst_angle<0)
            obst_angle=obst_angle+2*pi;
        end
    end
end
% Return obst_dist and obst_angle values to the main
narrow_opening=[obst_dist,obst_angle];
end
```

B.3 Collision Avoidance Controller

```
function [ new_position ] =
obstacle_controller( robot,goal,max_vel,max_ang_vel,goal_
radius,obst_radius,obst_dist,obst_angle,T )

%  Constants
k_a=2;
k_n=0.5;
k_t=0.5;

% Calculate the x and y components of the distance to goal
delta_x=goal(1)-robot(1);
delta_y=goal(2)-robot(2);

% Calculate the distance to the goal
dist_goal=sqrt(delta_x^2+delta_y^2);

% Calculate angle to goal
theta=atan(delta_y/delta_x);

% Calculate the change in the angle between the current and
goal position
delta_theta=theta-robot(3);

% Calculate the distance relation
dist_relation=exp(-dist_goal/goal_radius);

% Calculate attractive velocity
u_a=k_a*max_vel*(1-dist_relation);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%

% Find new theta value
close_to_zero=0.2;
if(abs(delta_theta)<close_to_zero)
    theta=theta;                                %ÕâÒ»¶ÎÄÚÈÝ
elseif(delta_theta>=0&&abs(delta_theta)<pi)    %ÊÇµôÍ·È¥Ä¿
±ê£¬Ò»µãµãµôÍ·
    theta=robot(3)+max_ang_vel*T;
elseif(delta_theta>=0&&abs(delta_theta)>=pi)
    theta=robot(3)-max_ang_vel*T;
elseif(delta_theta<=0&&abs(delta_theta)<pi)
```

```
    theta=robot(3)-max_ang_vel*T;
elseif(delta_theta<=0&&abs(delta_theta)>=pi)
    theta=robot(3)+max_ang_vel*T;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%

% Calculate repulsive velocity
% If no obstacle is in view
  % Set normal and tangent values to zero
% If an obstacle is in view
  % Calculate the normal and tangent values
if(obst_dist==0)
    normal=0;
    tangent=0;
    obst_relation=0;
    normal_angle=0;
    tangent_angle=0;
    theta_obst=0;
else
    normal=1/obst_dist;
    tangent=1/obst_dist;
    obst_relation=exp(-obst_dist/obst_radius);

% Calculate normal angle
normal_angle=obst_angle+pi;
if(normal_angle>3*pi/2)
    normal_angle=normal_angle-2*pi;
end

% Calculate tangent angle
tangent_angle=normal_angle-pi/2;
if(tangent_angle>3*pi/2)
    tangent_angle=tangent_angle-2*pi;
end

% Calculate obstacle angle
theta_obst=(normal_angle+tangent_angle)/2;
if(theta_obst<0)
    theta_obst=theta_obst+2*pi;
end

% Calculate change in angle between the robot and obstacle
delta_theta_obst=theta_obst-robot(3);
```

```
if(abs(delta_theta_obst)<close_to_zero)
    theta_obst=theta_obst;
elseif(delta_theta_obst>=0&&abs(delta_theta_obst)<pi)
    theta_obst=robot(3)+max_ang_vel*T;
elseif(delta_theta_obst>=0&&abs(delta_theta_obst)>=pi)
     theta_obst=robot(3)-max_ang_vel*T;
elseif(delta_theta_obst<0&&abs(delta_theta_obst)<pi)
     theta_obst=robot(3)-max_ang_vel*T;
elseif(delta_theta_obst<0&&abs(delta_theta_obst)>=pi)
    theta_obst=robot(3)+max_ang_vel*T;
end

% Ignore goal while obstacle is in view
u_a=0;
theta=0;
end

% Calculate normal and tagential velocity components
u_n=k_n*normal*obst_relation;
u_t=k_t*tangent*obst_relation;

% Calculate the sum of all velocity components
u_totX=u_a*cos(theta)+u_n*cos(normal_angle)+u_t*cos(tange
nt_angle);
u_totY=u_a*sin(theta)+u_n*sin(normal_angle)+u_t*cos(tange
nt_angle);

% Calculate the changle in x and y distance for the current
time step
delta_d=[u_totX*T,u_totY*T];
delta_theta=theta+theta_obst;

% Send the robot's next position to the main function
new_position=[robot(1)+delta_d(1),robot(2)+delta_d(2),del
ta_theta,u_totX,u_totY];
end
```

B.4 Direction Arrow

```
function [ X,Y ] = moving_arrow( Q )
x=Q(1);
y=Q(2);
theta=Q(3);
l=1;
X=[x,x+l*cos(theta-2*pi/3),x+l*cos(theta),x+l*cos(theta+2
*pi/3),x];
Y=[y,y+l*sin(theta-2*pi/3),y+l*sin(theta),y+l*sin(theta+2
*pi/3),y];

end
```

B.5 Sensor View

```
function [ circle ] = sensor_view( Q,radius )
x=Q(1);
y=Q(2);
theta=Q(3);

% Plot beam
for beam_angle=(theta-pi/2):pi/12:(theta+pi/2);
    beamX=[x,x+radius*cos(beam_angle)];
    beamY=[y,y+radius*sin(beam_angle)];
    plot(beamX,beamY,'c');
end

% Plot circle
circle=rectangle('Position',[x-radius,y-radius,2*radius,2
*radius],'curvature',[1,1],'edgecolor','c');
end
```

Appendix C

MATLAB Simulation: Holonomic Robot Pass Through Two Obstacles

C.1 Main Function

```
% Clear all windows
clc
clf
clear

% Initial robot values
robot_pose=[33,20,pi/2];
sensor_radius=6;
sensor_detect=6;
max_vel=4;
max_ang_vel=1;
m_2 = [0, 0];

% Map dimensions
map_max=60;
map_min=0;

% Goal position on the map
goal=[map_max/2,map_max];
goal_radius=sensor_detect;

% Time settings
t_sim=20;
T=0.05;
t_steps=floor(t_sim/T);


% move
j=0;
for t=1:t_steps;
    % Axis properties
    clf;
    xmax=map_max;
```

```
    xmin=map_min;
    ymax=map_max;
    ymin=0;

axis([xmin,xmax+2*goal_radius,ymin,ymax+2*goal_radius]);
    axis equal;
    axis manual;
    grid on;
    hold on;

    % Plot goal and the circle of interest around the goal
    scatter(goal(1),goal(2),100,'d','m','filled');

rectangle('position',[goal(1)-goal_radius,goal(2)-goal_ra
dius,2*goal_radius,2*goal_radius],'curvature',[1,1],'edge
color','m');



    % Plot the narrow passage
    % Check if sensor readings intersect with an obstacle
    % If an obstacle is in range
      % Set obst_dist and obst_angle values
    % If no obstacle is in view
      % Set obst_dist=0

[narrow_opening]=narrow_passage(robot_pose,sensor_radius)
;
    obst_dist=narrow_opening(1);
    obst_angle=narrow_opening(2);
    p=narrow_opening(3);
    b=narrow_opening(4);
    m_1(t,:)=[p,b];
    if (m_1(t,1)~=0 && m_2(1,1) == 0)
        m_2(1,1) = m_1(t,1);
    end
    if (m_1(t,2)~=0 && m_2(1,2) == 0)
        m_2(1,2)=m_1(t,2);
         [ detection_radius ] = detection(m_2(1,:));
        sensor_radius=detection_radius;

    end
```

```
    % Calculate the new robot pose
    robot=[robot_pose(1),robot_pose(2),robot_pose(3)];
    obst_radius=sensor_radius;

[new_pose]=obstacle_controller(robot,goal,max_vel,max_ang
_vel,goal_radius,obst_radius,obst_dist,obst_angle,T);
    robot_pose=[new_pose(1),new_pose(2),new_pose(3)];
    m(t,:)=robot_pose;
    plot(m(1:t,1),m(1:t,2),'r');

    % Plot the robot and its sensors
    [circle]=sensor_view(robot_pose,sensor_detect);
    [XL,YL]=moving_arrow(robot_pose);
    fill(XL,YL,'r');

    % Save a picture every 10 seconds
    if(mod(t,10)==0)
        filename=strcat('pics/',int2str(t),'.png');
        saveas(gcf,'filename');
    end
    hold off;
    j=j+1;
    M(j)=getframe;
end

% Save the simulation as an AVI file
movie2avi(M,'video.avi');
```

C.2 Robot Sensors Detect Obstacles

```
% Clear all windows
clc
clf
clear

% Initial robot values
robot_pose=[33,20,pi/2];
sensor_radius=6;
sensor_detect=6;
max_vel=4;
max_ang_vel=1;
m_2 = [0, 0];

% Map dimensions
map_max=60;
map_min=0;

% Goal position on the map
goal=[map_max/2,map_max];
goal_radius=sensor_detect;

% Time settings
t_sim=20;
T=0.05;
t_steps=floor(t_sim/T);


% move
j=0;
for t=1:t_steps;
    % Axis properties
    clf;
    xmax=map_max;
    xmin=map_min;
    ymax=map_max;
    ymin=0;

axis([xmin,xmax+2*goal_radius,ymin,ymax+2*goal_radius]);
    axis equal;
    axis manual;
    grid on;
```

```
    hold on;

    % Plot goal and the circle of interest around the goal
    scatter(goal(1),goal(2),100,'d','m','filled');

rectangle('position',[goal(1)-goal_radius,goal(2)-goal_ra
dius,2*goal_radius,2*goal_radius],'curvature',[1,1],'edge
color','m');



    % Plot the narrow passage
    % Check if sensor readings intersect with an obstacle
    % If an obstacle is in range
      % Set obst_dist and obst_angle values
    % If no obstacle is in view
      % Set obst_dist=0

[narrow_opening]=narrow_passage(robot_pose,sensor_radius)
;
    obst_dist=narrow_opening(1);
    obst_angle=narrow_opening(2);
    p=narrow_opening(3);
    b=narrow_opening(4);
    m_1(t,:)=[p,b];
    if (m_1(t,1)~=0 && m_2(1,1) == 0)
       m_2(1,1) = m_1(t,1);
    end
    if (m_1(t,2)~=0 && m_2(1,2) == 0)
       m_2(1,2)=m_1(t,2);
        [ detection_radius ] = detection(m_2(1,:));
       sensor_radius=detection_radius;

    end




    % Calculate the new robot pose
    robot=[robot_pose(1),robot_pose(2),robot_pose(3)];
    obst_radius=sensor_radius;

[new_pose]=obstacle_controller(robot,goal,max_vel,max_ang
_vel,goal_radius,obst_radius,obst_dist,obst_angle,T);
```

```
    robot_pose=[new_pose(1),new_pose(2),new_pose(3)];
    m(t,:)=robot_pose;
    plot(m(1:t,1),m(1:t,2),'r');

    % Plot the robot and its sensors
    [circle]=sensor_view(robot_pose,sensor_detect);
    [XL,YL]=moving_arrow(robot_pose);
    fill(XL,YL,'r');

    % Save a picture every 10 seconds
    if(mod(t,10)==0)
        filename=strcat('pics/',int2str(t),'.png');
        saveas(gcf,'filename');
    end
    hold off;
    j=j+1;
    M(j)=getframe;
end

% Save the simulation as an AVI file
movie2avi(M,'video.avi');
```

C.3 Collision Avoidance Controller

```
function [ new_position ] =
obstacle_controller( robot,goal,max_vel,max_ang_vel,goal_
radius,obst_radius,obst_dist,obst_angle,T )

%  Constants
k_a=2;
k_n=0.5;
k_t=0.5;

% Calculate the x and y components of the distance to goal
delta_x=goal(1)-robot(1);
delta_y=goal(2)-robot(2);

% Calculate the distance to the goal
dist_goal=sqrt(delta_x^2+delta_y^2);

% Calculate angle to goal
if(delta_x<0&&delta_y>0)
theta=atan(delta_y/delta_x)+pi;
else
    theta=atan(delta_y/delta_x);
end

% Calculate the change in the angle between the current and
goal position
delta_theta=theta-robot(3);

% Calculate the distance relation
% Calculate the distance relation
dist_relation=exp(-dist_goal^2/(goal_radius));

% Calculate attractive velocity
u_a=k_a*max_vel*15*(1-dist_relation)/dist_goal;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%

% Find new theta value
close_to_zero=0.2;
if(abs(delta_theta)<close_to_zero)
    theta=theta;                                %ÕâÒ»¶ÎÄÚÈÝ
```

```
elseif(delta_theta>=0&&abs(delta_theta)<pi)    %ÊÇµôÍ·È¥Ä¿
±ê£¬Ò»µãµãµôÍ·
    theta=robot(3)+max_ang_vel*T;
elseif(delta_theta>=0&&abs(delta_theta)>=pi)
    theta=robot(3)-max_ang_vel*T;
elseif(delta_theta<=0&&abs(delta_theta)<pi)
    theta=robot(3)-max_ang_vel*T;
elseif(delta_theta<=0&&abs(delta_theta)>=pi)
    theta=robot(3)+max_ang_vel*T;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%

% Calculate repulsive velocity
% If no obstacle is in view
  % Set normal and tangent values to zero
% If an obstacle is in view
  % Calculate the normal and tangent values
if(obst_dist==0)
    normal=0;
    tangent=0;
    obst_relation=0;
    normal_angle=0;
    tangent_angle=0;
    theta_obst=0;
else
    normal=1/obst_dist;
    tangent=1/obst_dist;
    obst_relation=exp(-obst_dist/obst_radius);

% Calculate normal angle
normal_angle=obst_angle+pi;
if(normal_angle>3*pi/2)
    normal_angle=normal_angle-2*pi;
end

% Calculate tangent angle
tangent_angle=normal_angle-pi/2;
if(tangent_angle>3*pi/2)
    tangent_angle=tangent_angle-2*pi;
end

% Calculate obstacle angle
theta_obst=(normal_angle+tangent_angle)/2;
```

```
if(theta_obst<0)
    theta_obst=theta_obst+2*pi;
end

% Calculate change in angle between the robot and obstacle
delta_theta_obst=theta_obst-robot(3);
if(abs(delta_theta_obst)<close_to_zero)
    theta_obst=theta_obst;
elseif(delta_theta_obst>=0&&abs(delta_theta_obst)<pi)
    theta_obst=robot(3)+max_ang_vel*T;
elseif(delta_theta_obst>=0&&abs(delta_theta_obst)>=pi)
     theta_obst=robot(3)-max_ang_vel*T;
elseif(delta_theta_obst<0&&abs(delta_theta_obst)<pi)
     theta_obst=robot(3)-max_ang_vel*T;
elseif(delta_theta_obst<0&&abs(delta_theta_obst)>=pi)
    theta_obst=robot(3)+max_ang_vel*T;
end

% Ignore goal while obstacle is in view
u_a=0;
theta=0;
end

% Calculate normal and tagential velocity components
u_n=k_n*normal*obst_relation;
u_t=k_t*tangent*obst_relation;

% Calculate the sum of all velocity components
u_totX=u_a*cos(theta)+u_n*cos(normal_angle)+u_t*cos(tange
nt_angle);
u_totY=u_a*sin(theta)+u_n*sin(normal_angle)+u_t*cos(tange
nt_angle);

% Calculate the changle in x and y distance for the current
time step
delta_d=[u_totX*T,u_totY*T];
delta_theta=theta+theta_obst;

% Send the robot's next position to the main function
new_position=[robot(1)+delta_d(1),robot(2)+delta_d(2),del
ta_theta,u_totX,u_totY];
end
```

C.4 Direction Arrow

```
function [ X,Y ] = moving_arrow( Q )
x=Q(1);
y=Q(2);
theta=Q(3);
l=1;
X=[x,x+l*cos(theta-2*pi/3),x+l*cos(theta),x+l*cos(theta+2
*pi/3),x];
Y=[y,y+l*sin(theta-2*pi/3),y+l*sin(theta),y+l*sin(theta+2
*pi/3),y];

end
```

C.5 Sensor View

```
function [ circle ] = sensor_view( Q,radius )
x=Q(1);
y=Q(2);
theta=Q(3);

% Plot beam
for beam_angle=(theta-pi/2):pi/12:(theta+pi/2);
    beamX=[x,x+radius*cos(beam_angle)];
    beamY=[y,y+radius*sin(beam_angle)];
    plot(beamX,beamY,'c');
end

% Plot circle
circle=rectangle('Position',[x-radius,y-radius,2*radius,2
*radius],'curvature',[1,1],'edgecolor','c');
end
```

C.6 Estimate If the Robot Can Get Through the Two Obstalces

```
function [ detection_radius ] = detection( p )

        dx=p(1,1)-p(1,2);
       if(dx>5)
          obst_radius=2;
       else
          obst_radius=6;
       end
   detection_radius=obst_radius;
end
```

Appendix D

MATLAB Simulation: Non-Holonomic Robot Collision Avoidance Controller Code

D.1 Main Function

```
% Clear all windows
clc
clf

% Initial robot values
robot_pose=[40,10,pi/2,0];
r=1;
l=10;
d=8;
sensor_radius=18;
max_vel=4;
max_ang_vel=1;

% Map dimensions
map_max=80;
map_min=0;

% Goal position on the map
goal=[map_max/2,map_max];
goal_radius=5;

% Time settings
t_sim=100;
T=0.05;
t_steps=floor(t_sim/T);


% move
j=0;
t=1;
```

```
while(abs(robot_pose(1)-goal(1))>0.001||abs(robot_pose(2)
-goal(2))>0.001)
    % Axis properties
    clf;
    xmax=map_max;
    xmin=map_min;
    ymax=map_max;
    ymin=0;

axis([xmin,xmax+2*goal_radius,ymin,ymax+2*goal_radius]);
    axis equal;
    axis manual;
    grid on;
    hold on;

    % Plot goal and the circle of interest around the goal
    scatter(goal(1),goal(2),100,'d','m','filled');

rectangle('position',[goal(1)-goal_radius,goal(2)-goal_ra
dius,2*goal_radius,2*goal_radius],'curvature',[1,1],'edge
color','m');

    % Plot the concave or convex obstacle
    % Check if sensor readings intersect with an obstacle
    % If an obstacle is in range
      % Set obst_dist and obst_angle values
    % If no obstacle is in view
      % Set obst_dist=0

[concave_convex]=concave(robot_pose,sensor_radius,map_max
);
    obst_dist=concave_convex(1);
    obst_angle=concave_convex(2);

    % Calculate the new front wheel position

robot=[robot_pose(1),robot_pose(2),robot_pose(3),robot_po
se(4)];
    obst_radius=sensor_radius;

[new_pose]=obstacle_controller(robot,goal,max_vel,max_ang
_vel,goal_radius,obst_radius,r,l,obst_dist,obst_angle,T);
```

```
robot_pose=[new_pose(1),new_pose(2),new_pose(3),new_pose(
4)];

   % o=new_pose(4)

    % Calculate the new rear wheel position
    [new_rear]=rear_wheel(robot_pose,l);
    rear_pose=[new_rear(1),new_rear(2),new_rear(3)];

    % Calculate the new left front wheel position
    [new_front_left]=front_left_wheel( robot_pose,d );

front_left_pose=[new_front_left(1),new_front_left(2),new_
front_left(3)];

     % Calculate the new right front wheel position
    [new_front_right]=front_right_wheel( robot_pose,d );

front_right_pose=[new_front_right(1),new_front_right(2),n
ew_front_right(3)];

     % Calculate the new left rear wheel position
    [new_rear_left]=rear_left_wheel(rear_pose,d);

rear_left_pose=[new_rear_left(1),new_rear_left(2),new_rea
r_left(3)];

     % Calculate the new right rear wheel position
    [new_rear_right]=rear_right_wheel(rear_pose,d);

rear_right_pose=[new_rear_right(1),new_rear_right(2),new_
rear_right(3)];



plot([robot_pose(1),rear_pose(1)],[robot_pose(2),rear_pos
e(2)],'k');


    % Plot the robot and its sensors
    [circle]=sensor_view(robot_pose,sensor_radius);
    [XL,YL]=moving_arrow(robot_pose);
    fill(XL,YL,'m');
```

```
    [XR,YR]=moving_front_left(front_left_pose);
    fill(XR,YR,'r');

    [XR,YR]=moving_front_right(front_right_pose);
    fill(XR,YR,'r');


plot([front_left_pose(1),front_right_pose(1)],[front_left
_pose(2),front_right_pose(2)],'k');

    [XR,YR]=moving_rear_left(rear_left_pose);
    fill(XR,YR,'r');

    [XR,YR]=moving_rear_right(rear_right_pose);
    fill(XR,YR,'r');


plot([rear_left_pose(1),rear_right_pose(1)],[rear_left_po
se(2),rear_right_pose(2)],'k');

    % Save a picture every 10 seconds
    if(mod(t,10)==0)
        filename=strcat('pics/',int2str(t),'.png');
        saveas(gcf,'filename');
    end
    hold off;
    j=j+1;
    M(j)=getframe;


 %
if(abs(robot_pose(1)-goal(1))<0.0001&&abs(robot_pose(2)-g
oal(2))<0.0001)
 %     break
 %   end

end

% Creat a moive and save it as an AVI file
movie2avi(M,'plot_animation.avi');
```

D.2 Robot Sensors Detect Obstacles

```
function [ concave_convex ] =
concave(robot_pose,sensor_radius,map_max)

%  Obstacle parameters
obst_radius=sensor_radius;
x_dist=map_max/2;
y_dist=map_max/2;

% Convex obstacle dimensions
xlimit=[x_dist-7.5,x_dist-6.5,x_dist-3.5,x_dist-2.5,x_dis
t-0.5,x_dist+0.5,x_dist+2.5,x_dist+3.5,x_dist+6.5,x_dist+
7.5];
ylimit=[y_dist-2.5,y_dist-1.5,y_dist-0.55,y_dist+0.45,y_d
ist+1.5,y_dist+2.5];
ob_1X=xlimit([1 2 2 1 1]);
ob_1Y=ylimit([1 1 2 2 1]);
ob_2X=xlimit([3 4 4 3 3]);
ob_2Y=ylimit([3 3 4 4 3]);
ob_3X=xlimit([5 6 6 5 5]);
ob_3Y=ylimit([5 5 6 6 5]);
ob_4X=xlimit([7 8 8 7 7]);
ob_4Y=ob_2Y;
ob_5X=xlimit([9 10 10 9 9]);
ob_5Y=ob_1Y;

% Draw obstacles in the simulation
% Obstacles are colored black¡úk
fill(ob_1X,ob_1Y,'k');
fill(ob_2X,ob_2Y,'k');
fill(ob_3X,ob_3Y,'k');
fill(ob_4X,ob_4Y,'k');
fill(ob_5X,ob_5Y,'k');

% Check to see if the robot sensors intersect with an obstacle
obst_dist=0;
obst_angle=0;
for
beam_angle=(robot_pose(3)+robot_pose(4)-pi/2):pi/24:(robo
t_pose(3)+robot_pose(4)+pi/2);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
```

```
x_ob=[robot_pose(1),robot_pose(1)+obst_radius*cos(beam_an
gle)];

y_ob=[robot_pose(2),robot_pose(2)+obst_radius*sin(beam_an
gle)];
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%
    [x1,y1]=polyxpoly(x_ob,y_ob,ob_1X,ob_1Y);
    [x2,y2]=polyxpoly(x_ob,y_ob,ob_2X,ob_2Y);
    [x3,y3]=polyxpoly(x_ob,y_ob,ob_3X,ob_3Y);
    [x4,y4]=polyxpoly(x_ob,y_ob,ob_4X,ob_4Y);
    [x5,y5]=polyxpoly(x_ob,y_ob,ob_5X,ob_5Y);

    % Plot a red circle where the intersection point occur
    mapshow(x1,y1,'displaytype','point','marker','o');
    mapshow(x2,y2,'displaytype','point','marker','o');
    mapshow(x3,y3,'displaytype','point','marker','o');
    mapshow(x4,y4,'displaytype','point','marker','o');
    mapshow(x5,y5,'displaytype','point','marker','o');

     %obst_dist(n)=sqrt(x(n).^2+y(n).^2);
    obst_dist1=sqrt(x1.^2+y1.^2);
    obst_dist2=sqrt(x2.^2+y2.^2);
    obst_dist3=sqrt(x3.^2+y3.^2);
    obst_dist4=sqrt(x4.^2+y4.^2);
    obst_dist5=sqrt(x5.^2+y5.^2);

    if(size(obst_dist1,1)>=2)
        obst_dist1=obst_dist1(1);
    elseif(size(obst_dist2,1)>=2)
        obst_dist2=obst_dist2(1);
    elseif(size(obst_dist3,1)>=2)
        obst_dist3=obst_dist3(1);
    elseif(size(obst_dist4,1)>=2)
        obst_dist4=obst_dist4(1);
    elseif(size(obst_dist5,1)>=2)
        obst_dist5=obst_dist5(1);
    end

    if(isempty(obst_dist1)==false)
        obst_dist=obst_dist1;
    elseif(isempty(obst_dist2)==false)
        obst_dist=obst_dist2;
```

```
    elseif(isempty(obst_dist3)==false)
        obst_dist=obst_dist3;
    elseif(isempty(obst_dist4)==false)
        obst_dist=obst_dist4;
    elseif(isempty(obst_dist5)==false)
        obst_dist=obst_dist5;
    end

if(isempty(obst_dist1)==false||isempty(obst_dist2)==false
||isempty(obst_dist3)==false||isempty(obst_dist4)==false|
|isempty(obst_dist5)==false)
       obst_angle=beam_angle;
       if(obst_angle>2*pi)
           obst_angle=obst_angle-2*pi;
       elseif(obst_angle<0)
           obst_angle=obst_angle+2*pi;
       end
    end
end

% Return obst_dist and obst_angle values to the main
concave_convex=[obst_dist,obst_angle];
end
```

D.3 Collision Avoidance Controller

```
function [ new_position ] =
obstacle_controller(robot,goal,max_vel,max_ang_vel,goal_r
adius,obst_radius,r,l,obst_dist,obst_angle,T)

%  Constants
k_a=2;
k_n=0.5;
k_t=0.5;

% Calculate the x and y components of the distance to goal
delta_x=goal(1)-robot(1);
delta_y=goal(2)-robot(2);

% Calculate the distance to the goal
dist_goal=sqrt(delta_x^2+delta_y^2);

% Calculate angle to goal
theta=atan(delta_y/delta_x);

% Calculate the change in the angle between the current and
goal position
delta_theta=theta-robot(3)-robot(4);

% Calculate the distance relation
dist_relation=exp(-dist_goal/goal_radius);

% Calculate attractive velocity
u_a=k_a*max_vel*(1-dist_relation);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%

% Find new theta value
close_to_zero=0.2;
if(abs(delta_theta)<close_to_zero)
    theta=theta;                                %ÕâÒ»¶ÎÄÚÈÝ
elseif(delta_theta>=0&&abs(delta_theta)<pi)    %ÊÇµôÍ·È¥Ä¿
±ê£¬Ò»µãµãµôÍ·  r/l*sin(robot(4))*max_ang_vel*T
    theta=robot(3)+robot(4)+max_ang_vel*T;
elseif(delta_theta>=0&&abs(delta_theta)>=pi)
    theta=robot(3)+robot(4)-max_ang_vel*T;
elseif(delta_theta<=0&&abs(delta_theta)<pi)
```

```
    theta=robot(3)+robot(4)-max_ang_vel*T;
elseif(delta_theta<=0&&abs(delta_theta)>=pi)
    theta=robot(3)+robot(4)+max_ang_vel*T;
end
  %  robot(3)=robot(3)+r/l*sin(robot(4))*max_ang_vel*T;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Calculate repulsive velocity
% If no obstacle is in view
  % Set normal and tangent values to zero
% If an obstacle is in view
  % Calculate the normal and tangent values

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if(robot(4)+max_ang_vel*T>=pi/6)
    robot(4)=pi/6-max_ang_vel*T;
end
if(robot(4)-max_ang_vel*T<=-pi/6)
    robot(4)=-pi/6+max_ang_vel*T;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if(obst_dist==0)
    normal=0;
    tangent=0;
    obst_relation=0;
    normal_angle=0;
    tangent_angle=0;
    theta_obst=0;
else
    normal=50*log(obst_dist);
    tangent=obst_radius/obst_dist;
    obst_relation=exp(-obst_dist/obst_radius);

% Calculate normal angle
normal_angle=robot(3)+robot(4); %obst_angle;
if(normal_angle>3*pi/2)
    normal_angle=normal_angle-2*pi;
end

% Calculate tangent angle
```

```
tangent_angle=normal_angle-pi/2;
if(tangent_angle>3*pi/2)
    tangent_angle=tangent_angle-2*pi;
end

% Calculate obstacle angle
theta_obst=(normal_angle+tangent_angle)/2;
if(theta_obst<0)
    theta_obst=theta_obst+2*pi;
end


% Calculate change in angle between the robot and obstacle
delta_theta_obst=theta_obst-robot(3)-robot(4);
if(abs(delta_theta_obst)<close_to_zero)
    theta_obst=theta_obst;
elseif(delta_theta_obst>=0&&abs(delta_theta_obst)<pi)
    theta_obst=robot(3)+robot(4)+max_ang_vel*T;
elseif(delta_theta_obst>=0&&abs(delta_theta_obst)>=pi)
     theta_obst=robot(3)+robot(4)+max_ang_vel*T;
elseif(delta_theta_obst<0&&abs(delta_theta_obst)<pi)
     theta_obst=robot(3)+robot(4)+max_ang_vel*T;
elseif(delta_theta_obst<0&&abs(delta_theta_obst)>=pi)
    theta_obst=robot(3)+robot(4)+max_ang_vel*T;
end



% Ignore goal while obstacle is in view
u_a=0;
theta=0;
end

% Calculate normal and tagential velocity components
u_n=k_n*normal*obst_relation;
u_t=k_t*tangent*obst_relation;

% Calculate the sum of all velocity components
u_totX=u_a*cos(theta)+u_n*cos(normal_angle)+u_t*cos(tange
nt_angle);
u_totY=u_a*sin(theta)+u_n*sin(normal_angle)+u_t*cos(tange
nt_angle);
```

```
% Calculate the changle in x and y distance for the current
time step
delta_d=[u_totX*T,u_totY*T];
delta_theta=theta+theta_obst;


% Send the robot's next position to the main function
new_position=[robot(1)+delta_d(1),robot(2)+delta_d(2),rob
ot(3)+r/l*sin(delta_theta-robot(3))*7*max_ang_vel*T,delta
_theta-robot(3),u_totX,u_totY];
end
```

D.4 Direction Arrow

```
function [ X,Y ] = moving_arrow( Q )
x=Q(1);
y=Q(2);
theta=Q(3);
l=1;
X=[x,x+l*cos(theta-2*pi/3),x+l*cos(theta),x+l*cos(theta+2
*pi/3),x];
Y=[y,y+l*sin(theta-2*pi/3),y+l*sin(theta),y+l*sin(theta+2
*pi/3),y];

end
```

D.5 Sensor View

```
function [ circle ] = sensor_view( Q,radius )
x=Q(1);
y=Q(2);
theta=Q(3);

% Plot beam
for beam_angle=(theta-pi/2):pi/12:(theta+pi/2);
    beamX=[x,x+radius*cos(beam_angle)];
    beamY=[y,y+radius*sin(beam_angle)];
    plot(beamX,beamY,'c');
end

% Plot circle
circle=rectangle('Position',[x-radius,y-radius,2*radius,2
*radius],'curvature',[1,1],'edgecolor','c');
end
```

D.6 Pose of the Four Wheels

D.6.1 Left-Front-Wheel

```
function [ front_left_wheel ] =
front_left_wheel( robot_pose,d )
front_left_wheel=[robot_pose(1)-d/2*sin(robot_pose(3)),ro
bot_pose(2)+d/2*cos(robot_pose(3)),robot_pose(3)+robot_po
se(4)];
end
```

D.6.2 Right-Front-Wheel

```
function [ front_right_wheel ] =
front_right_wheel( robot_pose,d )
front_right_wheel=[robot_pose(1)+d/2*sin(robot_pose(3)),r
obot_pose(2)-d/2*cos(robot_pose(3)),robot_pose(3)+robot_p
ose(4)];
end
```

D.6.3 Left-Rear-Wheel

```
function [ rear_left_pose ] = rear_left_wheel(rear_pose,d)
rear_left_pose=[rear_pose(1)-d/2*sin(rear_pose(3)),rear_p
ose(2)+d/2*cos(rear_pose(3)),rear_pose(3)];
end
```

D.6.4 Right-Rear-Wheel

```
function [ rear_right_pose ] = rear_right_wheel(rear_pose,d)
rear_right_pose=[rear_pose(1)+d/2*sin(rear_pose(3)),rear_
pose(2)-d/2*cos(rear_pose(3)),rear_pose(3)];
end
```

D.7 Shape of the Four Wheels

D.7.1 Left-Front-Wheel

```
function [X,Y] = moving_front_left( Q )
x=Q(1);
y=Q(2);
theta=Q(3);
l=2;
X=[x+l*cos(theta-11*pi/12),x+l*cos(theta+11*pi/12),x+l*co
s(theta+pi/12),x+l*cos(theta-pi/12),x+l*cos(theta-11*pi/1
2)];
Y=[y+l*sin(theta-11*pi/12),y+l*sin(theta+11*pi/12),y+l*si
n(theta+pi/12),y+l*sin(theta-pi/12),y+l*sin(theta-11*pi/1
2)];
end
```

D.7.2 Right-Front-Wheel

```
function [X,Y] = moving_front_right( Q )
x=Q(1);
y=Q(2);
theta=Q(3);
l=2;
X=[x+l*cos(theta-11*pi/12),x+l*cos(theta+11*pi/12),x+l*co
s(theta+pi/12),x+l*cos(theta-pi/12),x+l*cos(theta-11*pi/1
2)];
Y=[y+l*sin(theta-11*pi/12),y+l*sin(theta+11*pi/12),y+l*si
n(theta+pi/12),y+l*sin(theta-pi/12),y+l*sin(theta-11*pi/1
2)];
end
```

D.7.3 Left-Rear-Wheel

```
function [X,Y] = moving_rear_left( Q )
x=Q(1);
y=Q(2);
theta=Q(3);
l=2;
```

```
X=[x+l*cos(theta-11*pi/12),x+l*cos(theta+11*pi/12),x+l*co
s(theta+pi/12),x+l*cos(theta-pi/12),x+l*cos(theta-11*pi/1
2)];
Y=[y+l*sin(theta-11*pi/12),y+l*sin(theta+11*pi/12),y+l*si
n(theta+pi/12),y+l*sin(theta-pi/12),y+l*sin(theta-11*pi/1
2)];
end
```

D.7.4 Right-Rear-Wheel

```
function [X,Y] = moving_rear_right( Q )
x=Q(1);
y=Q(2);
theta=Q(3);
l=2;
X=[x+l*cos(theta-11*pi/12),x+l*cos(theta+11*pi/12),x+l*co
s(theta+pi/12),x+l*cos(theta-pi/12),x+l*cos(theta-11*pi/1
2)];
Y=[y+l*sin(theta-11*pi/12),y+l*sin(theta+11*pi/12),y+l*si
n(theta+pi/12),y+l*sin(theta-pi/12),y+l*sin(theta-11*pi/1
2)];
end
```

Appendix E

MATLAB Simulation: Non-Holonomic Human-Following Robot

E.1 Main Function

```
% Clear all windows
clc
clf

%n=0;
% Initial robot values
human_pose=[80,60,pi/2,0];
robot_pose=[20,10,pi/2,0];
r=1;
l=10;
d=8;
sensor_radius=18.2;
human_sensor_radius=5;
max_vel=16;
max_ang_vel=4;

% Map dimensions
map_max=160;
map_min=0;

% Goal position on the map
goal=[map_max/2,map_max];
goal_radius=5;

% Time settings
t_sim=100;
T=0.05;
t_steps=floor(t_sim/T);


% move
j=0;
t=1;
```

```
while(abs(human_pose(1)-goal(1))>1||abs(human_pose(2)-goa
l(2))>1)
    % Axis properties
    clf;
    xmax=map_max;
    xmin=map_min;
    ymax=map_max;
    ymin=0;

axis([xmin,xmax+2*goal_radius,ymin,ymax+2*goal_radius]);
    axis equal;
    axis manual;
    grid on;
    hold on;

    % Plot goal and the circle of interest around the goal
    scatter(goal(1),goal(2),100,'d','m','filled');

rectangle('position',[goal(1)-goal_radius,goal(2)-goal_ra
dius,2*goal_radius,2*goal_radius],'curvature',[1,1],'edge
color','m');


[concave_h]=concave_human(human_pose,human_sensor_radius,
map_max);
    obst_dist_human=concave_h(1);
    obst_angle_human=concave_h(2);


[concave_r]=concave_robot(robot_pose,sensor_radius,map_ma
x);
    obst_dist_robot=concave_r(1);
    obst_angle_robot=concave_r(2);

    %

[human_detactive]=human_area( robot_pose,human_pose,senso
r_radius);
    agents_dist=human_detactive(1);
    agents_angle=human_detactive(2);

    % Calculate the human's position
```

```
human=[human_pose(1),human_pose(2),human_pose(3),human_po
se(4)];

[human_new_pose]=towards_target(human,goal,max_vel,max_an
g_vel,human_sensor_radius,obst_dist_human,obst_angle_huma
n,T);

human_pose=[human_new_pose(1),human_new_pose(2),human_new
_pose(3),human_new_pose(4)];


    % Calculate the new front wheel position

robot=[robot_pose(1),robot_pose(2),robot_pose(3),robot_po
se(4)];
    obst_radius=sensor_radius;

[robot_new_pose]=obstacle_controller(robot,human,max_vel,
max_ang_vel,obst_radius,agents_dist,obst_dist_robot,obst_
angle_robot,l,T);

robot_pose=[robot_new_pose(1),robot_new_pose(2),robot_new
_pose(3),robot_new_pose(4)];



    % Calculate the new rear wheel position
    [new_rear]=rear_wheel(robot_pose,l);
    rear_pose=[new_rear(1),new_rear(2),new_rear(3)];

    % Calculate the new left front wheel position
    [new_front_left]=front_left_wheel( robot_pose,d );

front_left_pose=[new_front_left(1),new_front_left(2),new_
front_left(3)];

     % Calculate the new right front wheel position
    [new_front_right]=front_right_wheel( robot_pose,d );

front_right_pose=[new_front_right(1),new_front_right(2),n
ew_front_right(3)];

     % Calculate the new left rear wheel position
```

```
    [new_rear_left]=rear_left_wheel(rear_pose,d);

rear_left_pose=[new_rear_left(1),new_rear_left(2),new_rea
r_left(3)];

    % Calculate the new right rear wheel position
    [new_rear_right]=rear_right_wheel(rear_pose,d);

rear_right_pose=[new_rear_right(1),new_rear_right(2),new_
rear_right(3)];



plot([robot_pose(1),rear_pose(1)],[robot_pose(2),rear_pos
e(2)],'k');

    %%%%%%%%%%%%%%%%%%%%%%%%%%%
   % [ circle2 ] = human_position( human_pose,r );

    % Plot the robot and its sensors
    [circle]=sensor_view(robot_pose,sensor_radius);
    [XL,YL]=moving_arrow(robot_pose);
    fill(XL,YL,'m');

    [XR,YR]=moving_front_left(front_left_pose);
    fill(XR,YR,'r');

    [XR,YR]=moving_front_right(front_right_pose);
    fill(XR,YR,'r');


plot([front_left_pose(1),front_right_pose(1)],[front_left
_pose(2),front_right_pose(2)],'k');

    [XR,YR]=moving_rear_left(rear_left_pose);
    fill(XR,YR,'r');

    [XR,YR]=moving_rear_right(rear_right_pose);
    fill(XR,YR,'r');


plot([rear_left_pose(1),rear_right_pose(1)],[rear_left_po
se(2),rear_right_pose(2)],'k');
```

```
    % Save a picture every 10 seconds
    if(mod(t,10)==0)
        filename=strcat('pics/',int2str(t),'.png');
        saveas(gcf,'filename');
    end
    hold off;
    j=j+1;
    M(j)=getframe;


 %
if(abs(robot_pose(1)-goal(1))<0.0001&&abs(robot_pose(2)-g
oal(2))<0.0001)
 %     break
 %    end

end

% Creat a moive and save it as an AVI file
movie2avi(M,'plot_animation.avi');
```

E.2 Robot Sensors Detect Obstacles

```
function [ concave_r ] =
concave_robot(robot_pose,sensor_radius,map_max)

%  Obstacle parameters
obst_radius=sensor_radius;
x_dist=map_max/2;
y_dist=map_max/2;
ob_width=3;
ob_dist=15;


xlimit=[x_dist-ob_width/2-ob_dist/2,x_dist+ob_width/2-ob_
dist/2,x_dist-ob_width/2+ob_dist/2,x_dist+ob_width/2+ob_d
ist/2];
ylimit=[y_dist-ob_width/2,y_dist+ob_width/2];
ob_leftX=xlimit([1 2 2 1 1]);
ob_leftY=ylimit([1 1 2 2 1]);
ob_rightX=xlimit([3 4 4 3 3]);
ob_rightY=ob_leftY;

% Draw obstacles in the simulation
% Obstacles are colored black 'k'
fill(ob_leftX,ob_leftY,'k');
fill(ob_rightX,ob_rightY,'k');

% Check to see if the robot sensors intersect with an obstacle
obst_dist=0;
obst_angle=0;
for
beam_angle=(robot_pose(3)-pi/2):pi/24:(robot_pose(3)+pi/2
);

x_ob=[robot_pose(1),robot_pose(1)+obst_radius*cos(beam_an
gle)];

y_ob=[robot_pose(2),robot_pose(2)+obst_radius*sin(beam_an
gle)];
    [xT,yT]=polyxpoly(x_ob,y_ob,ob_rightX,ob_rightY);
    [xB,yB]=polyxpoly(x_ob,y_ob,ob_leftX,ob_leftY);
    mapshow(xT,yT,'displaytype','point','marker','o');
    mapshow(xB,yB,'displaytype','point','marker','o');
```

```
    obst_distT=sqrt(xT.^2+yT.^2);
    obst_distB=sqrt(xB.^2+yB.^2);
    if(size(obst_distT,1)==2)
        obst_distT=obst_distT(1);
    elseif(size(obst_distB,1)==2)
        obst_distB=obst_distB(1);
    end


if(isempty(obst_distT)==false&&isempty(obst_distB)==true)
        obst_dist=obst_distT;

elseif(isempty(obst_distT)==true&&isempty(obst_distB)==fa
lse)
        obst_dist=obst_distB;
    end


if(isempty(obst_distT)==false||isempty(obst_distB)==false
);
        obst_angle=beam_angle;
    end
end

% Return obst_dist and obst_angle values to the main
concave_r=[obst_dist,obst_angle];
end
```

E.3 Collision Avoidance Controller

```
function [ new_position ] =
obstacle_controller(robot,human,max_vel,max_ang_vel,obst_
radius,agents_dist,obst_dist_robot,obst_angle_robot,l,T)

%  Constants
k_a=50;

dist_safe=2*human(4);

% Calculate the x and y components of the distance to goal
delta_x=human(1)-robot(1);
delta_y=human(2)-robot(2);

% Calculate the distance to the goal
dist_goal=sqrt(delta_x^2+delta_y^2);

dist=dist_goal-dist_safe;

% Calculate angle to goal
if(delta_x<0&&delta_y>0)
theta=atan(delta_y/delta_x)+pi;
else
    theta=atan(delta_y/delta_x);
end

alpha=robot(3)+robot(4);

% Calculate the change in the angle between the current and
goal position
delta_theta=theta-alpha;

% Calculate the distance relation
dist_relation=exp(-dist^2/200);

% Calculate attractive velocity
if(agents_dist==0)
u_a=k_a*max_vel*(1-dist_relation)/dist;
else
    u_a=human(4);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%

% Find new theta value
%close_to_zero=0.2;
%if(abs(delta_theta)<close_to_zero)
%
robot(4)=asin(max_ang_vel*l/u_a)
 %ÕâÒ»¶ÎÄÚÈÝ
if(delta_theta>=0&&abs(delta_theta)<pi)    %ÊÇµôÍ·È¥Ä¿±ê£¬
Ò»µãµãµôÍ·  r/l*sin(robot(4))*max_ang_vel*T
    robot(4)=robot(4)+max_ang_vel*T;
elseif(delta_theta>=0&&abs(delta_theta)>=pi)
    robot(4)=robot(4)-max_ang_vel*T;
elseif(delta_theta<0&&abs(delta_theta)<pi)
    robot(4)=robot(4)-max_ang_vel*T;
elseif(delta_theta<0&&abs(delta_theta)>=pi)
    robot(4)=robot(4)+max_ang_vel*T;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
% reach an obstacle
if(obst_dist_robot==0)
    u_o=0;
else
   % obst_relation=exp(-obst_dist_robot^2/obst_radius);


% Calculate change in angle between the robot and obstacle
delta_theta_obst=obst_angle_robot-robot(3)-robot(4);

if(delta_theta_obst>=-pi/2&&delta_theta_obst<=0)
    robot(4)=robot(4)+max_ang_vel*2*T;
elseif(delta_theta_obst>0&&delta_theta_obst<=pi/2)
     robot(4)=robot(4)-max_ang_vel*2*T;
end

% Ignore goal while obstacle is in view
u_a=0;
u_o=human(4);
%u_o=k_a*max_vel*(1-obst_relation);
end
```

```
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
if(robot(4)+max_ang_vel*T>=pi/4)
    robot(4)=pi/4-max_ang_vel*T;
end
if(robot(4)-max_ang_vel*T<=-pi/4)
    robot(4)=-pi/4+max_ang_vel*T;
end



% Calculate the sum of all velocity components
u_totX=u_a*cos(robot(3)+robot(4))+u_o*cos(robot(3)+robot(
4));
u_totY=u_a*sin(robot(3)+robot(4))+u_o*sin(robot(3)+robot(
4));
u_tot=sqrt(u_totX.^2+u_totY.^2);
% Calculate the changle in x and y distance for the current
time step
delta_d=[u_totX*T,u_totY*T];



% Send the robot's next position to the main function
new_position=[robot(1)+delta_d(1),robot(2)+delta_d(2),rob
ot(3)+1/l*sin(robot(4))*u_tot*T,robot(4),u_totX,u_totY];
end
```

E.4 Human Obstacle Notice

```
function [ concave_h ] =
concave_human(human_pose,human_sensor_radius,map_max)

%  Obstacle parameters
obst_radius=human_sensor_radius;
x_dist=map_max/2;
y_dist=map_max/2;
ob_width=3;
ob_dist=15;


xlimit=[x_dist-ob_width/2-ob_dist/2,x_dist+ob_width/2-ob_
dist/2,x_dist-ob_width/2+ob_dist/2,x_dist+ob_width/2+ob_d
ist/2];
ylimit=[y_dist-ob_width/2,y_dist+ob_width/2];
ob_leftX=xlimit([1 2 2 1 1]);
ob_leftY=ylimit([1 1 2 2 1]);
ob_rightX=xlimit([3 4 4 3 3]);
ob_rightY=ob_leftY;

% Draw obstacles in the simulation
% Obstacles are colored black 'k'
fill(ob_leftX,ob_leftY,'k');
fill(ob_rightX,ob_rightY,'k');

% Check to see if the robot sensors intersect with an obstacle
obst_dist=0;
obst_angle=0;
for
beam_angle=(human_pose(3)-pi/2):pi/24:(human_pose(3)+pi/2
);

x_ob=[human_pose(1),human_pose(1)+obst_radius*cos(beam_an
gle)];

y_ob=[human_pose(2),human_pose(2)+obst_radius*sin(beam_an
gle)];
    [xT,yT]=polyxpoly(x_ob,y_ob,ob_rightX,ob_rightY);
    [xB,yB]=polyxpoly(x_ob,y_ob,ob_leftX,ob_leftY);
    mapshow(xT,yT,'displaytype','point','marker','o');
    mapshow(xB,yB,'displaytype','point','marker','o');
```

```
    obst_distT=sqrt(xT.^2+yT.^2);
    obst_distB=sqrt(xB.^2+yB.^2);
    if(size(obst_distT,1)==2)
        obst_distT=obst_distT(1);
    elseif(size(obst_distB,1)==2)
        obst_distB=obst_distB(1);
    end


if(isempty(obst_distT)==false&&isempty(obst_distB)==true)
        obst_dist=obst_distT;

elseif(isempty(obst_distT)==true&&isempty(obst_distB)==fa
lse)
        obst_dist=obst_distB;
    end


if(isempty(obst_distT)==false||isempty(obst_distB)==false
);
        obst_angle=beam_angle;
    end
end

% Return obst_dist and obst_angle values to the main
concave_h=[obst_dist,obst_angle];
end
```

E.5 Human Trajectory

```
function [human_new_position] =
towards_target(human,goal,max_vel,max_ang_vel,human_senso
r_radius,obst_dist_human,obst_angle_human,T)

%  Constants
k_a=0.5;
k_n=0.5;
k_t=0.5;


% Calculate the x and y components of the distance to goal
delta_x=goal(1)-human(1);
delta_y=goal(2)-human(2);

% Calculate the distance to the goal
dist_goal=sqrt(delta_x^2+delta_y^2);

% Calculate angle to goal
if(delta_x<0&&delta_y>0)
theta=atan(delta_y/delta_x)+pi;
else
    theta=atan(delta_y/delta_x);
end

alpha=human(3);

% Calculate the change in the angle between the current and
goal position
delta_theta=theta-alpha;

% Calculate the distance relation
dist_relation=exp(-dist_goal^2/200);

% Calculate attractive velocity
u_a=k_a*max_vel*(1-dist_relation);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%

% Find new theta value
close_to_zero=0.2;
if(abs(delta_theta)<close_to_zero)
```

```
    theta=theta;                                %ÕâÒ»¶ÎÄÚÈÝ
elseif(delta_theta>=0&&abs(delta_theta)<pi)    %ÊÇµôÍ·È¥Ä¿
±ê£¬Ò»µãµãµôÍ·
    theta=human(3)+max_ang_vel*T;
elseif(delta_theta>=0&&abs(delta_theta)>=pi)
    theta=human(3)-max_ang_vel*T;
elseif(delta_theta<=0&&abs(delta_theta)<pi)
    theta=human(3)-max_ang_vel*T;
elseif(delta_theta<=0&&abs(delta_theta)>=pi)
    theta=human(3)+max_ang_vel*T;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
% reach an obstacle
if(obst_dist_human==0)
    normal=0;
    tangent=0;
    obst_relation=0;
    normal_angle=0;
    tangent_angle=0;
    theta_obst=0;
else
    normal=1/obst_dist_human;
    tangent=1/obst_dist_human;

obst_relation=exp(-obst_dist_human/human_sensor_radius);

% Calculate normal angle
normal_angle=obst_angle_human+pi;
if(normal_angle>3*pi/2)
    normal_angle=normal_angle-2*pi;
end

% Calculate tangent angle
tangent_angle=normal_angle-pi/2;
if(tangent_angle>3*pi/2)
    tangent_angle=tangent_angle-2*pi;
end

% Calculate obstacle angle
theta_obst=(normal_angle+tangent_angle)/2;
if(theta_obst<0)
    theta_obst=theta_obst+2*pi;
```

```
end

% Calculate change in angle between the robot and obstacle
delta_theta_obst=theta_obst-human(3);
if(abs(delta_theta_obst)<close_to_zero)
    theta_obst=theta_obst;
elseif(delta_theta_obst>=0&&abs(delta_theta_obst)<pi)
    theta_obst=human(3)+max_ang_vel*T;
elseif(delta_theta_obst>=0&&abs(delta_theta_obst)>=pi)
     theta_obst=human(3)-max_ang_vel*T;
elseif(delta_theta_obst<0&&abs(delta_theta_obst)<pi)
     theta_obst=human(3)-max_ang_vel*T;
elseif(delta_theta_obst<0&&abs(delta_theta_obst)>=pi)
    theta_obst=human(3)+max_ang_vel*T;
end

u_a=0;
theta=0;
end


% Calculate normal and tagential velocity components
u_n=k_n*normal*obst_relation;
u_t=k_t*tangent*obst_relation;

% Calculate the sum of all velocity components
u_totX=u_a*cos(theta)+u_n*cos(normal_angle)+u_t*cos(tange
nt_angle);
u_totY=u_a*sin(theta)+u_n*sin(normal_angle)+u_t*sin(tange
nt_angle);
u_tot=sqrt(u_totX.^2+u_totY.^2);
% Calculate the changle in x and y distance for the current
time step
delta_d=[u_totX*T,u_totY*T];
delta_theta=theta+theta_obst;



% Send the robot's next position to the main function
human_new_position=[human(1)+delta_d(1),human(2)+delta_d(
2),delta_theta,u_tot,u_totX,u_totY];
end
```

E.6 Human Detection

```
function [ human_detactive ] =
human_area( robot_pose,human_pose,sensor_radius)

%size of robot
alpha=0:pi/10:2*pi;
r=1;
x_r=r*cos(alpha)+human_pose(1);
y_r=r*sin(alpha)+human_pose(2);
fill(x_r,y_r,'b');

% Check to see if the robot sensors intersect with an obstacle
agents_dist=0;
agents_angle=0;
for
beam_angle=(robot_pose(3)+robot_pose(4)-pi/2):pi/24:(robo
t_pose(3)+robot_pose(4)+pi/2);


x_ag=[robot_pose(1),robot_pose(1)+sensor_radius*cos(beam_
angle)];

y_ag=[robot_pose(2),robot_pose(2)+sensor_radius*sin(beam_
angle)];

    [xi,yi]=polyxpoly(x_ag,y_ag,x_r,y_r);

    % Plot a red circle where the intersection point occur
    mapshow(xi,yi,'DisplayType','point','marker','o');

    dist=sqrt(xi.^2+yi.^2);
    if(size(dist,1)==2)
        dist=dist(1);
    end
    if(isempty(dist)==false)
        agents_dist=dist;
        agents_angle=beam_angle;
     %   if(agents_angle>2*pi)
      %     agents_angle=agents_angle-2*pi;
      %  elseif(agents_angle<0)
      %      agents_angle=agents_angle+2*pi;
      %  end
```

```
    end
end

% Return obst_dist and obst_angle values to the main
human_detactive=[agents_dist,agents_angle];


end
```

E.7 Human Position

```
function [ circle2 ] = human_position( Q,r )
x=Q(1);
y=Q(2);
theta=Q(3);

% Plot circle
circle2=rectangle('Position',[x-r,y-r,2*r,2*r],'curvature
',[1,1],'edgecolor','b');
end
```

E.8 Direction Arrow

```
function [ X,Y ] = moving_arrow( Q )
x=Q(1);
y=Q(2);
theta=Q(3);
l=1;
X=[x,x+l*cos(theta-2*pi/3),x+l*cos(theta),x+l*cos(theta+2
*pi/3),x];
Y=[y,y+l*sin(theta-2*pi/3),y+l*sin(theta),y+l*sin(theta+2
*pi/3),y];

end
```

E.9 Sensor View

```
function [ circle ] = sensor_view( Q,radius )
x=Q(1);
y=Q(2);
theta=Q(3);

% Plot beam
for beam_angle=(theta-pi/2):pi/12:(theta+pi/2);
    beamX=[x,x+radius*cos(beam_angle)];
    beamY=[y,y+radius*sin(beam_angle)];
    plot(beamX,beamY,'c');
end

% Plot circle
circle=rectangle('Position',[x-radius,y-radius,2*radius,2
*radius],'curvature',[1,1],'edgecolor','c');
end
```

E.10 Pose of the Four Wheels

E.10.1 Left-Front-Wheel

```
function [ front_left_wheel ] =
front_left_wheel( robot_pose,d )
front_left_wheel=[robot_pose(1)-d/2*sin(robot_pose(3)),ro
bot_pose(2)+d/2*cos(robot_pose(3)),robot_pose(3)+robot_po
se(4)];
end
```

E.10.2 Right-Front-Wheel

```
function [ front_right_wheel ] =
front_right_wheel( robot_pose,d )
front_right_wheel=[robot_pose(1)+d/2*sin(robot_pose(3)),r
obot_pose(2)-d/2*cos(robot_pose(3)),robot_pose(3)+robot_p
ose(4)];
end
```

E.10.3 Left-Rear-Wheel

```
function [ rear_left_pose ] = rear_left_wheel(rear_pose,d)
rear_left_pose=[rear_pose(1)-d/2*sin(rear_pose(3)),rear_p
ose(2)+d/2*cos(rear_pose(3)),rear_pose(3)];
end
```

E.10.4 Right-Rear-Wheel

```
function [ rear_right_pose ] = rear_right_wheel(rear_pose,d)
rear_right_pose=[rear_pose(1)+d/2*sin(rear_pose(3)),rear_
pose(2)-d/2*cos(rear_pose(3)),rear_pose(3)];
end
```

E.11 Shape of the Four Wheels

E.11.1 Left-Front-Wheel

```
function [X,Y] = moving_front_left( Q )
x=Q(1);
y=Q(2);
theta=Q(3);
l=2;
X=[x+l*cos(theta-11*pi/12),x+l*cos(theta+11*pi/12),x+l*co
s(theta+pi/12),x+l*cos(theta-pi/12),x+l*cos(theta-11*pi/1
2)];
Y=[y+l*sin(theta-11*pi/12),y+l*sin(theta+11*pi/12),y+l*si
n(theta+pi/12),y+l*sin(theta-pi/12),y+l*sin(theta-11*pi/1
2)];
end
```

E.11.2 Right-Front-Wheel

```
function [X,Y] = moving_front_right( Q )
x=Q(1);
y=Q(2);
theta=Q(3);
l=2;
X=[x+l*cos(theta-11*pi/12),x+l*cos(theta+11*pi/12),x+l*co
s(theta+pi/12),x+l*cos(theta-pi/12),x+l*cos(theta-11*pi/1
2)];
Y=[y+l*sin(theta-11*pi/12),y+l*sin(theta+11*pi/12),y+l*si
n(theta+pi/12),y+l*sin(theta-pi/12),y+l*sin(theta-11*pi/1
2)];
end
```

E.11.3 Left-Rear-Wheel

```
function [X,Y] = moving_rear_left( Q )
x=Q(1);
y=Q(2);
theta=Q(3);
l=2;
```

```
X=[x+l*cos(theta-11*pi/12),x+l*cos(theta+11*pi/12),x+l*co
s(theta+pi/12),x+l*cos(theta-pi/12),x+l*cos(theta-11*pi/1
2)];
Y=[y+l*sin(theta-11*pi/12),y+l*sin(theta+11*pi/12),y+l*si
n(theta+pi/12),y+l*sin(theta-pi/12),y+l*sin(theta-11*pi/1
2)];
end
```

E.11.4 Right-Rear-Wheel

```
function [X,Y] = moving_rear_right( Q )
x=Q(1);
y=Q(2);
theta=Q(3);
l=2;
X=[x+l*cos(theta-11*pi/12),x+l*cos(theta+11*pi/12),x+l*co
s(theta+pi/12),x+l*cos(theta-pi/12),x+l*cos(theta-11*pi/1
2)];
Y=[y+l*sin(theta-11*pi/12),y+l*sin(theta+11*pi/12),y+l*si
n(theta+pi/12),y+l*sin(theta-pi/12),y+l*sin(theta-11*pi/1
2)];
end
```

Appendix F

LabVIEW: Obstacle Avoidance

In this section, a LabVIEW program of LEGO robot obstacle avoidance is shown in Figure F.1- F.6, in which

represents the ultrasonic sensor.


represents the DC servo motor.

represents the exponential function.

The inputs are the distances detected by the ultrasonic sensor. The outputs are the velocities of motors which connected with the robot's driving wheels. Figure F.1 gives the flow chart of obstacle avoidance sub function with LabVIEW. Using case structures, the outputs are classified into six situations to realize obstacle avoidance, which can also been seen in Figure F.2-F.7.

In Figure F.2, when the ultrasonic sensor detects obstacles on both front-left and front-right side, the robot will move backward and then turn left.

In Figure F.3, the ultrasonic sensor detects an obstacle on the front-left of the robot, the robot turns right.

In Figure F.4, the ultrasonic sensor detects an obstacle on the front-right of the robot, the robot turns left.

In Figure F.5, no obstacle is detected, the robot keep moving forward.

In Figure F.6, the ultrasonic sensor detects an obstacle on the right ahead of the robot, and there is more space on the front-left of the robot than the front-right, the robot turns left.

In Figure F.7, the ultrasonic sensor detects an obstacle on the right ahead of the robot, and there is less space on the front-left of the robot than the front-right, the robot turns right.

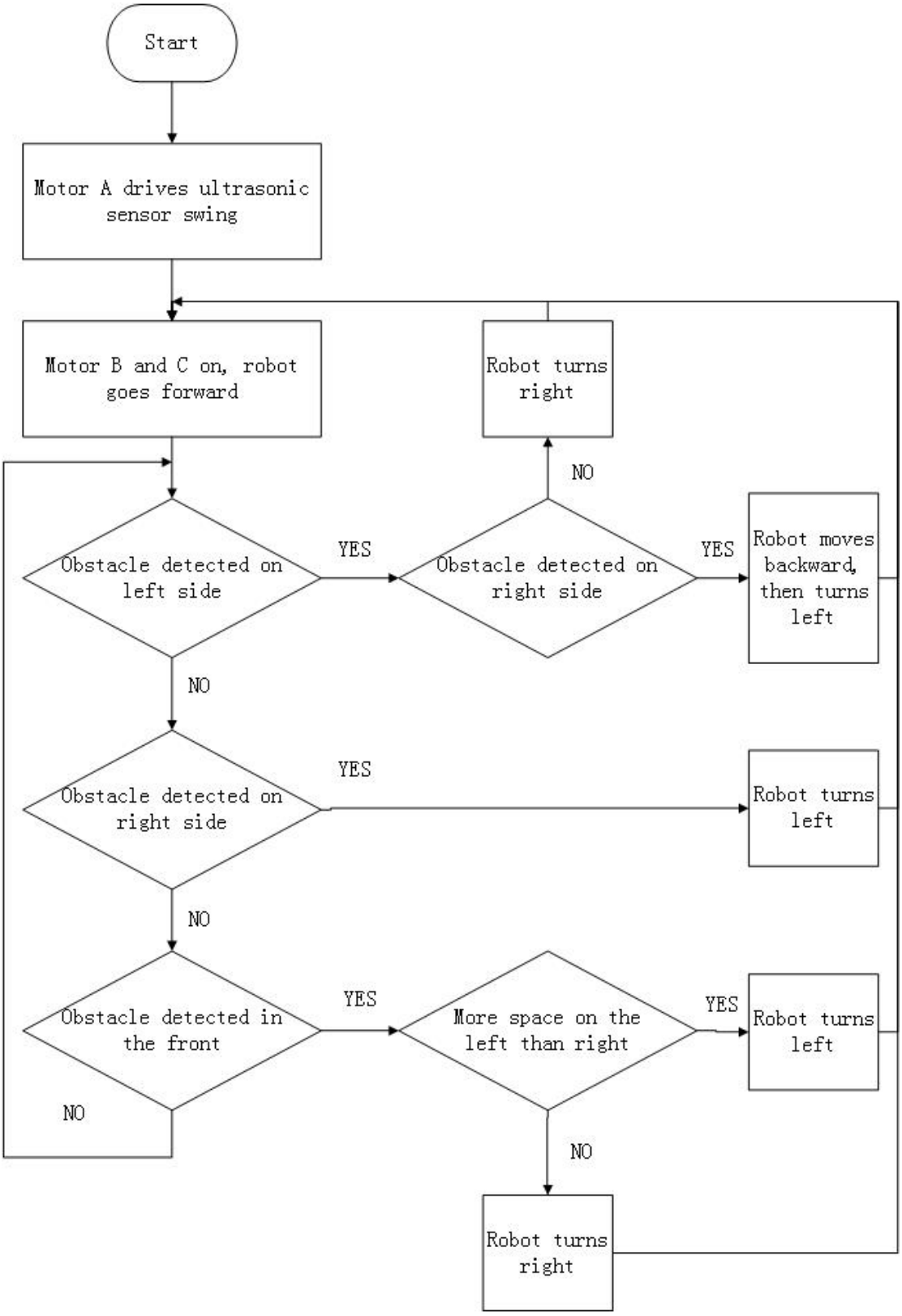


Figure F. 1 Flow chart of obstacle avoidance sub function

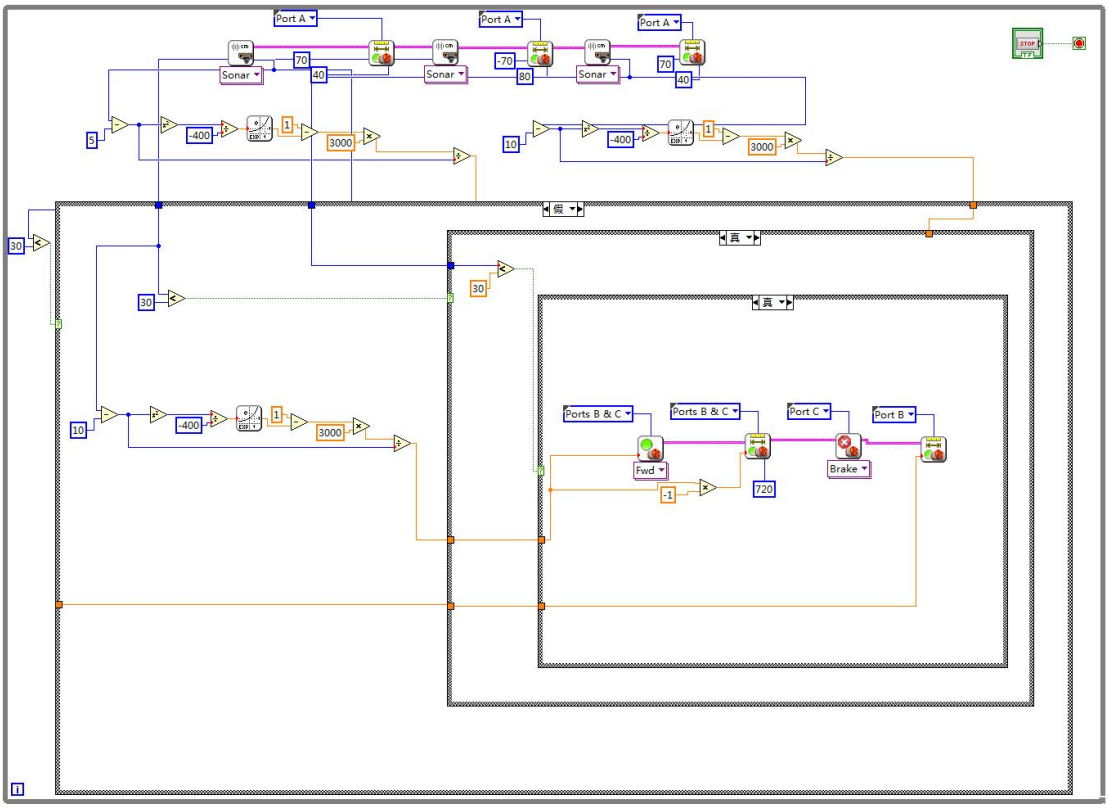


Figure F. 2 Obstacles on both left and right

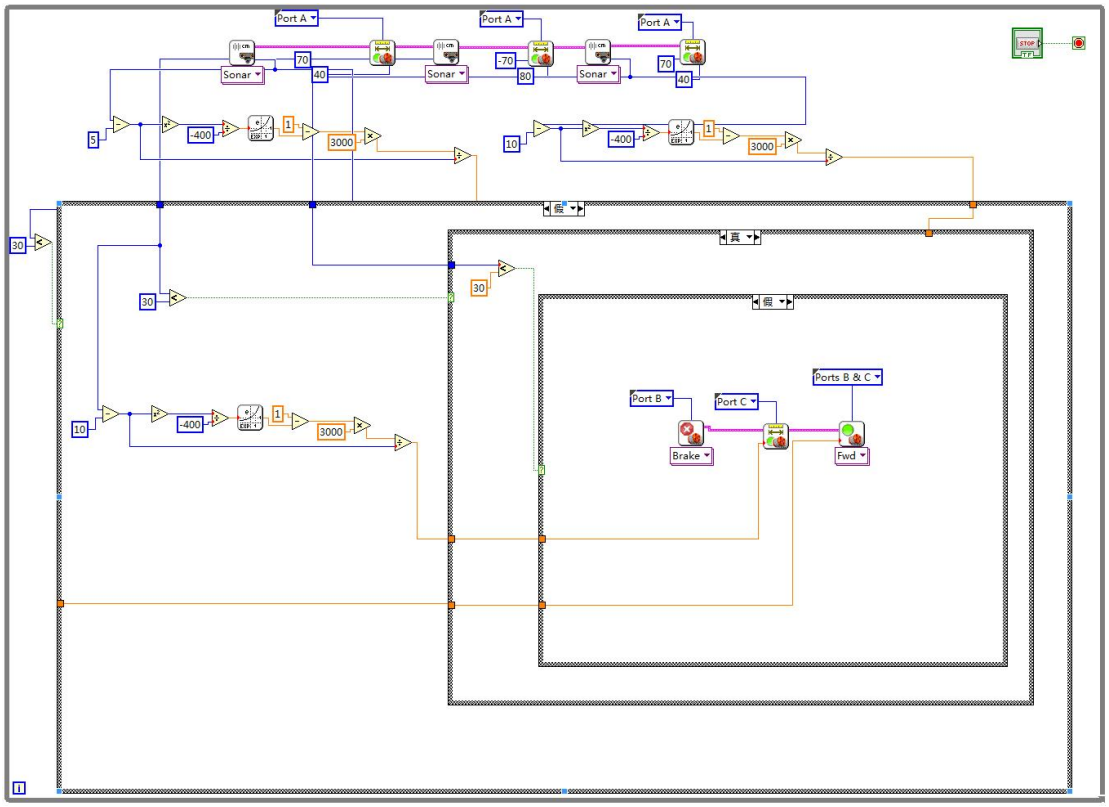


Figure F. 3 Obstacle on the left

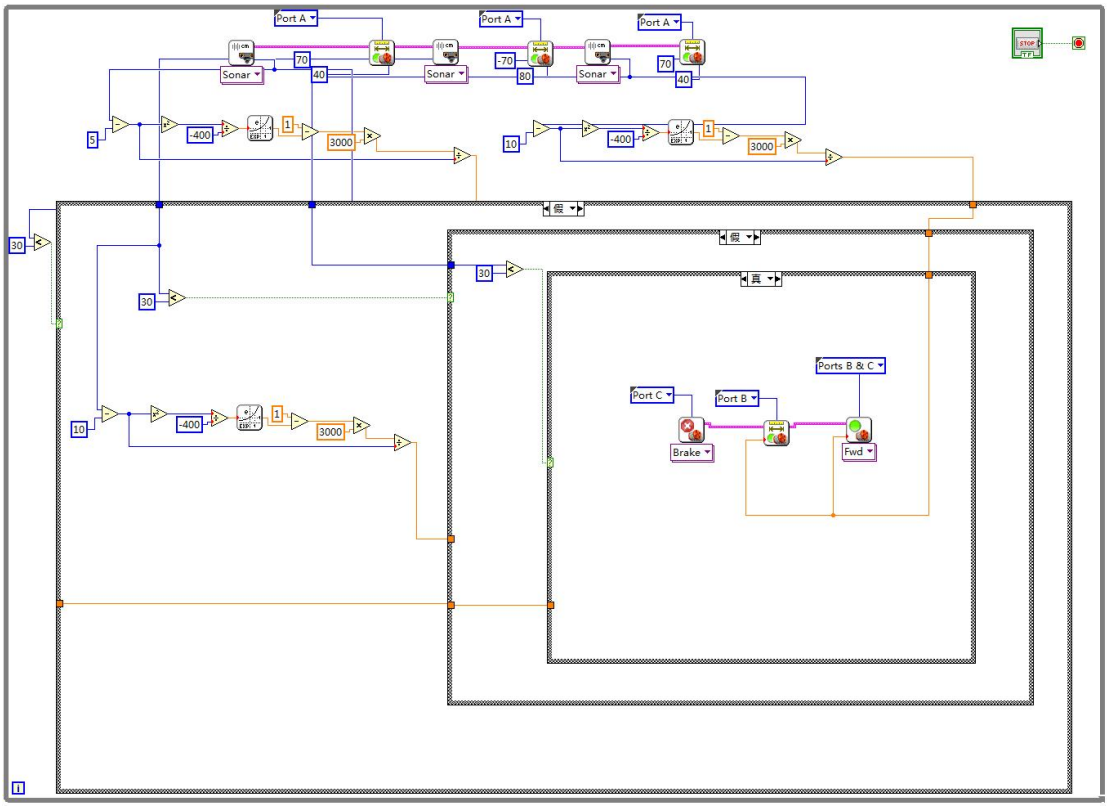


Figure F. 4 Obstacle on the right

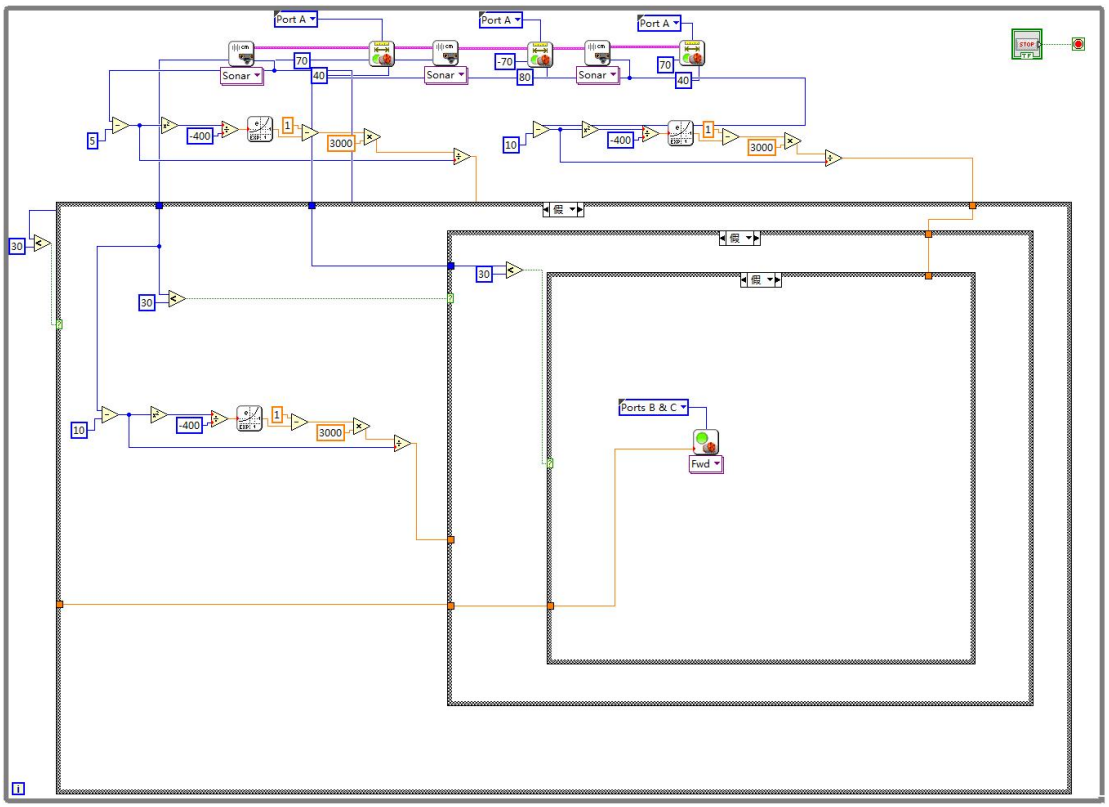


Figure F. 5 No obstalce detected

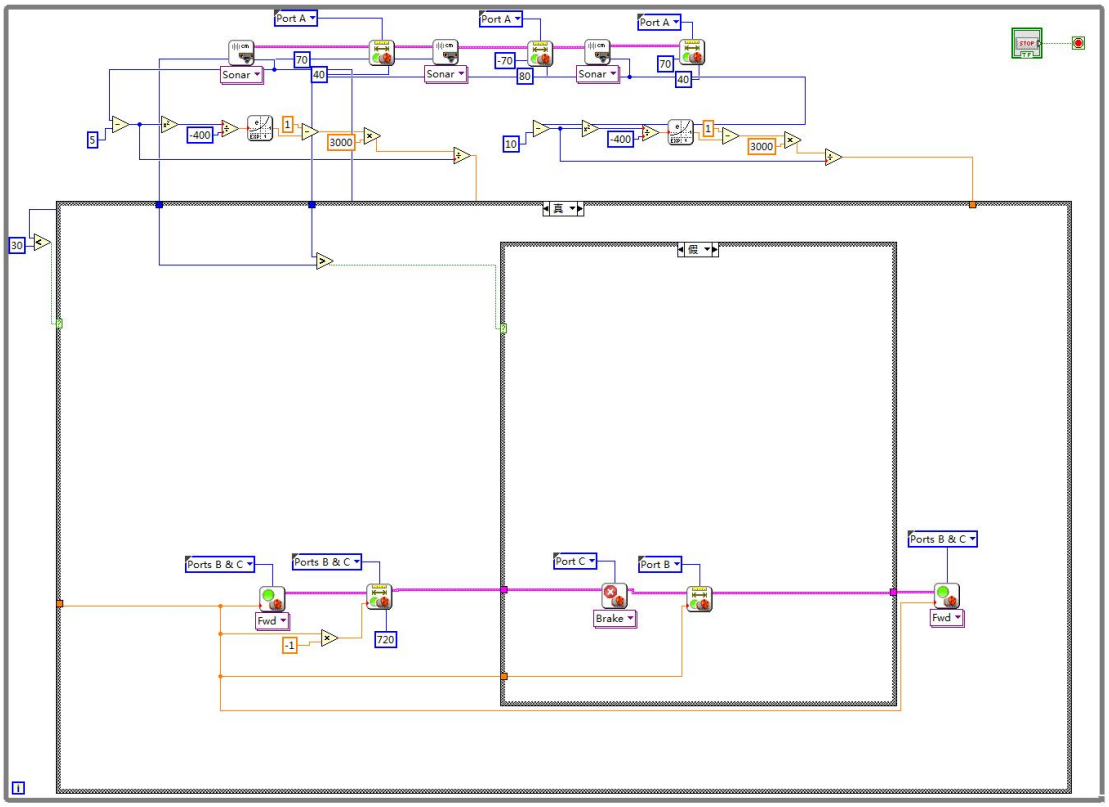


Figure F. 6 Obstacles right ahead the robot case 1

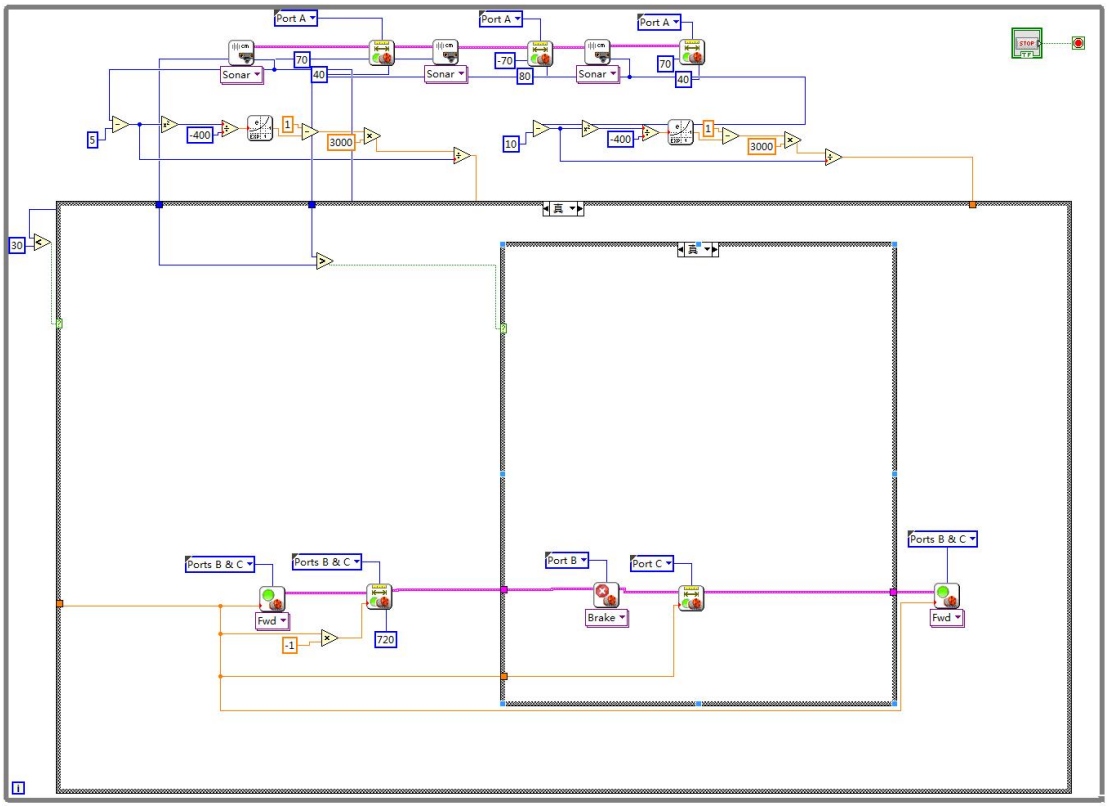


Figure F. 7 Obstacles right ahead the robot case 2

Appendix G

LabVIEW: Human Following

In this section, a LabVIEW program of LEGO robot human tracking is shown in Figure G.2- G.8, in which

represents the DC servo motor, and Figure G.1 shows a customized PIR senor control program.

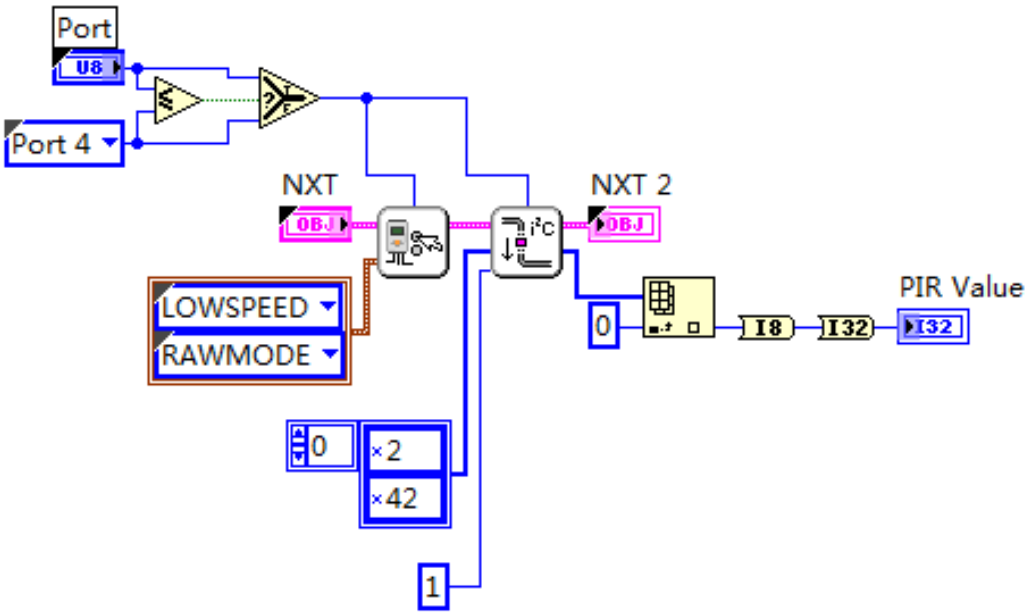


Figure G. 1 Customized PIR sensor program

The inputs are the temperature differences detected by the PIR sensor array. From Figure 4.20 in Chapter 4 it can be seen that the number of the PIR sensors are 1, 2 and 3 from right to left. The outputs are the velocities of motors which connected with the robot's driving wheels. Figure G.2 gives the flow chart of obstacle avoidance sub function with LabVIEW. Using case structures, the outputs are classified into seven situations to help the robot make decisions, which can also be seen in Figure G.3-G.9.

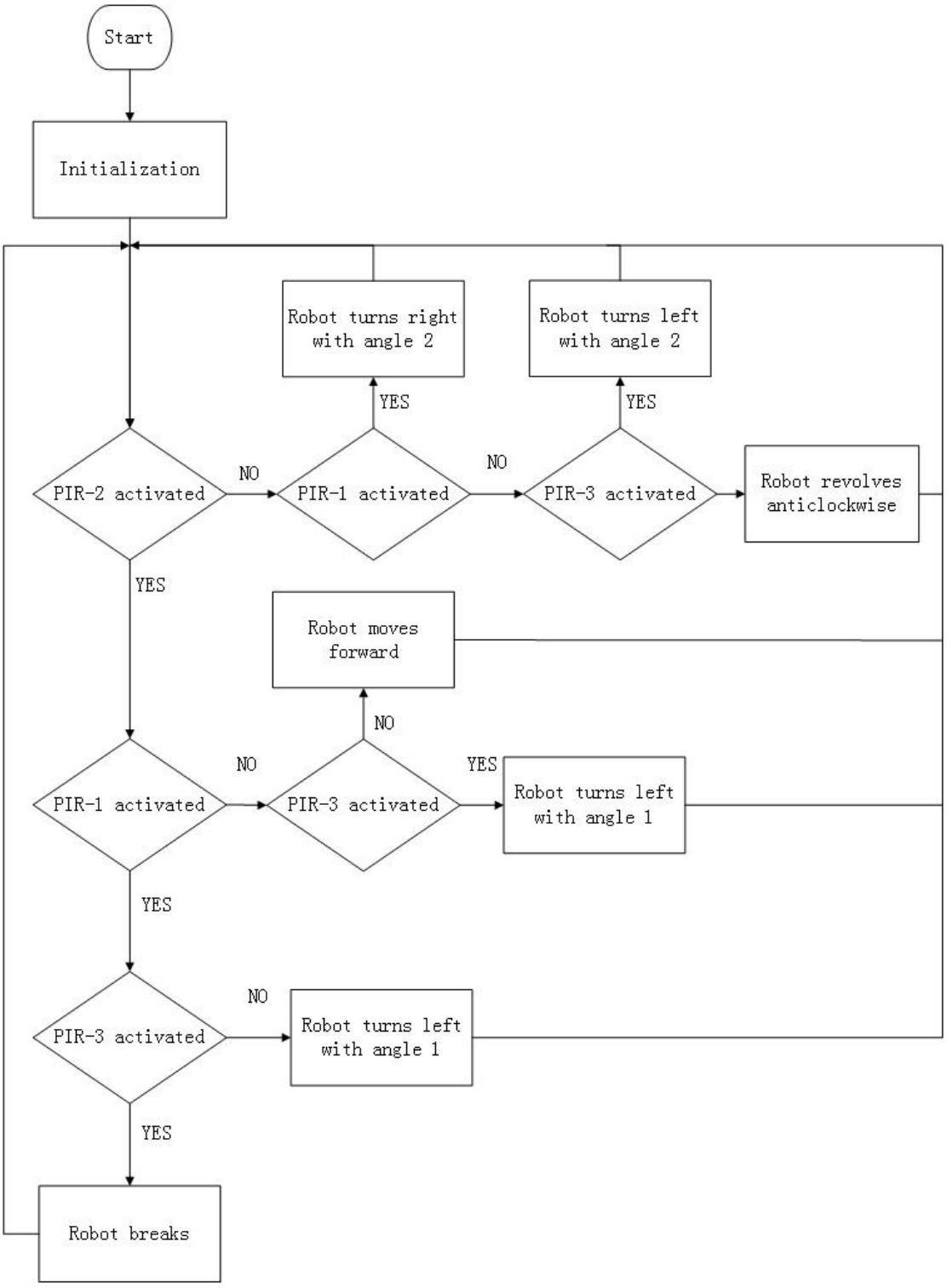


Figure G. 2 Flow chart of human following sub function

In Figure G.3, when both PIR sensor 1 and 2 detect temperature difference, robot turns right with angle 1.

In Figure G.4, when all PIR sensor 1, 2 and 3 detect temperature difference, robot takes a break because the distance between human and robot is too close.

In Figure G.5, when only PIR sensor 2 detects temperature difference, robot moves forward.

In Figure G.6, when both PIR sensor 2 and 3 detect temperature difference, robot turns left with angle 1.

In Figure G.7, when only PIR sensor 1detects temperature difference, robot turns right with angle 2.

In Figure G.8, when only PIR sensor 3 detects temperature difference, robot turns left with angle 2

In Figure G.9, when there is no PIR sensor detects temperature difference, robot revolves anticlockwise on its axis to search for a target.

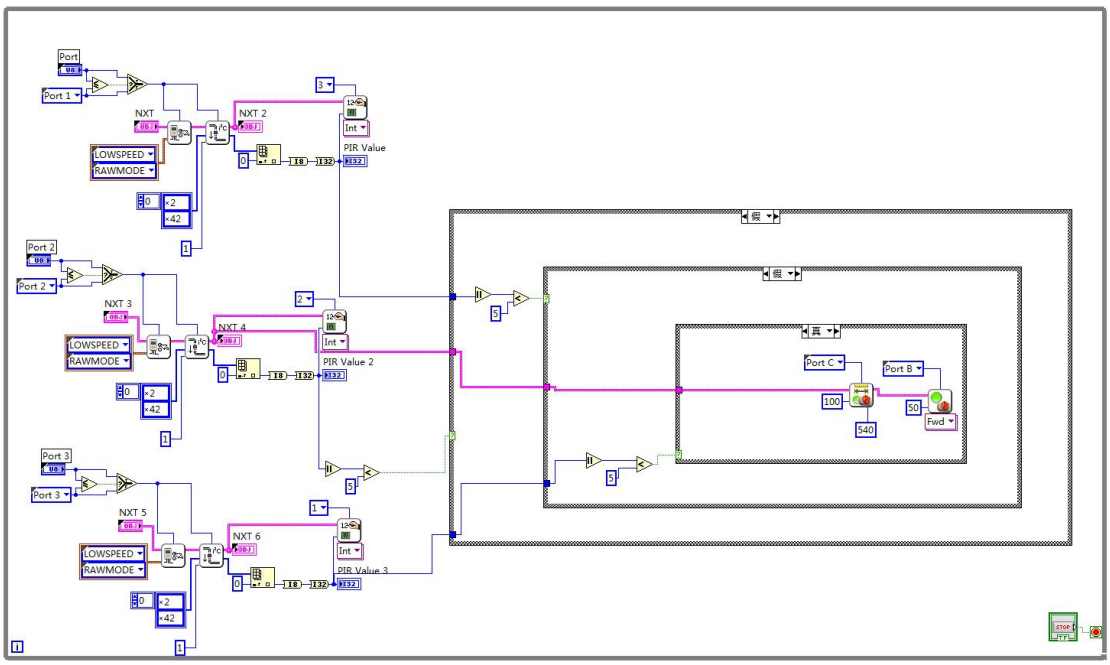


Figure G. 3 PIR sensor 1 and 2 detect tempurature difference

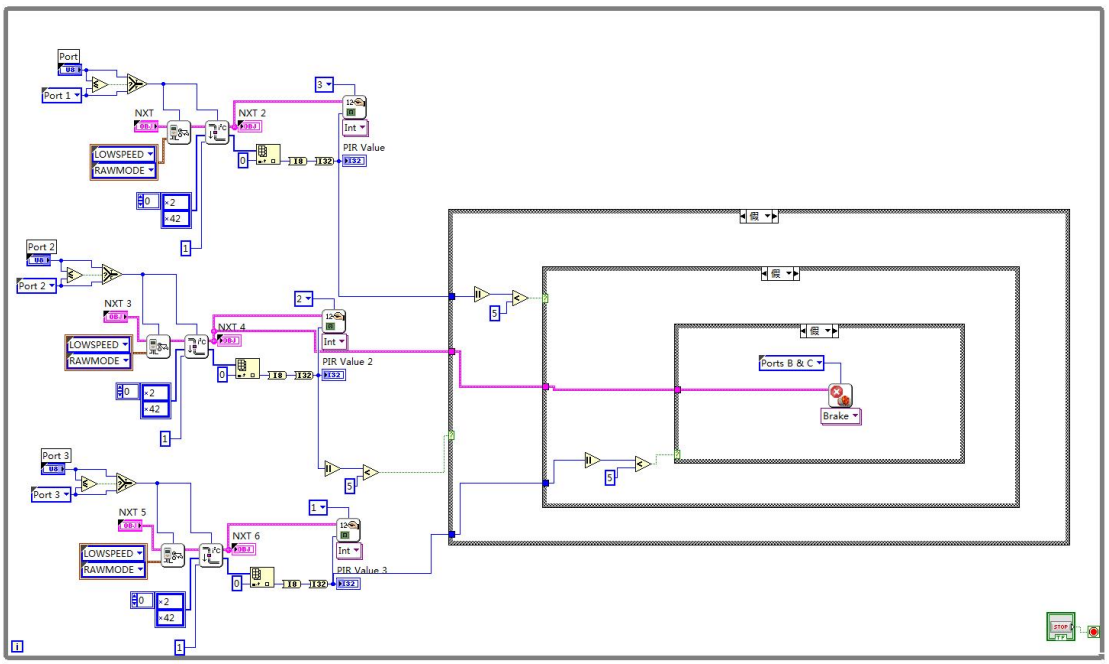


Figure G. 4 PIR sensor 1, 2 and 3 detect tempurature difference

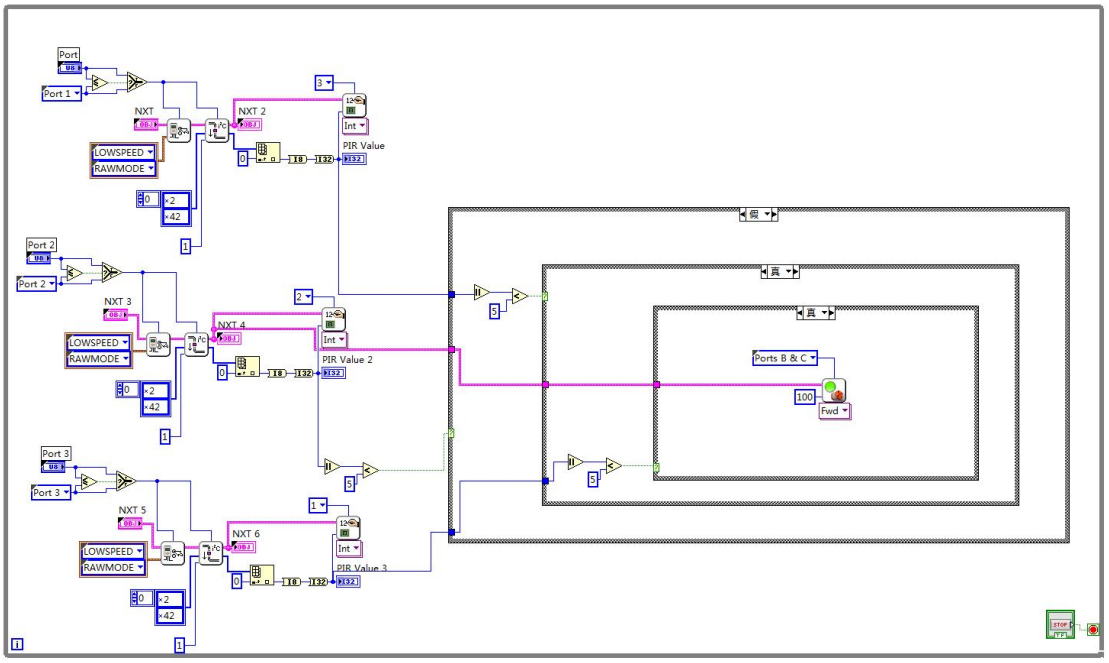


Figure G. 5 PIR sensor 2 detect tempurature difference

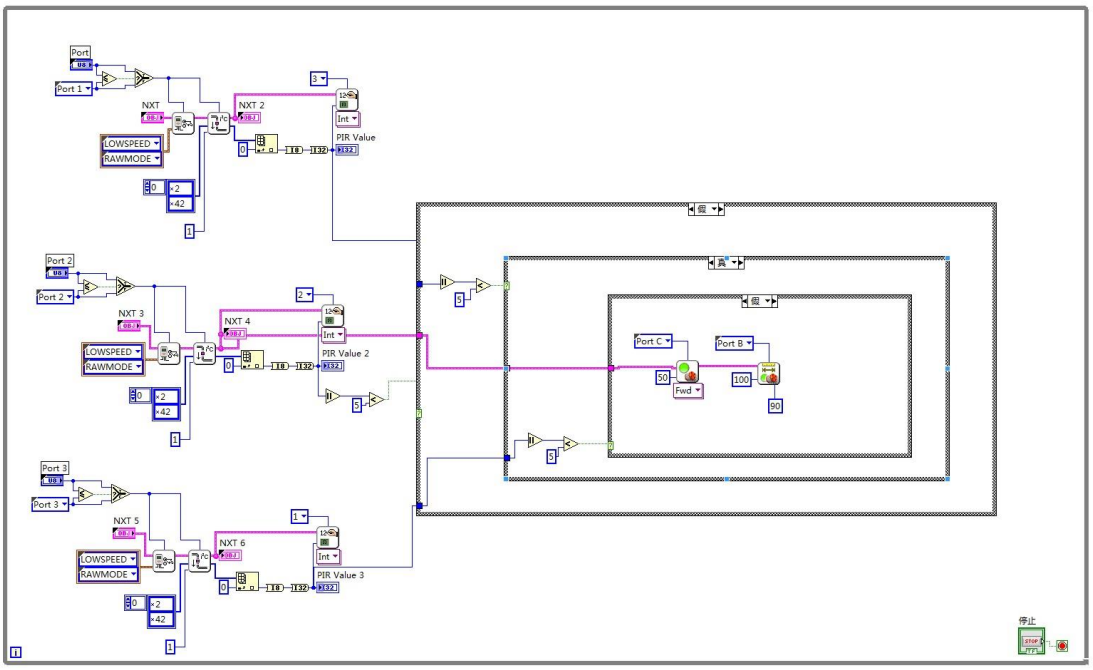


Figure G. 6 PIR sensor 2 and 3 detect tempurature difference

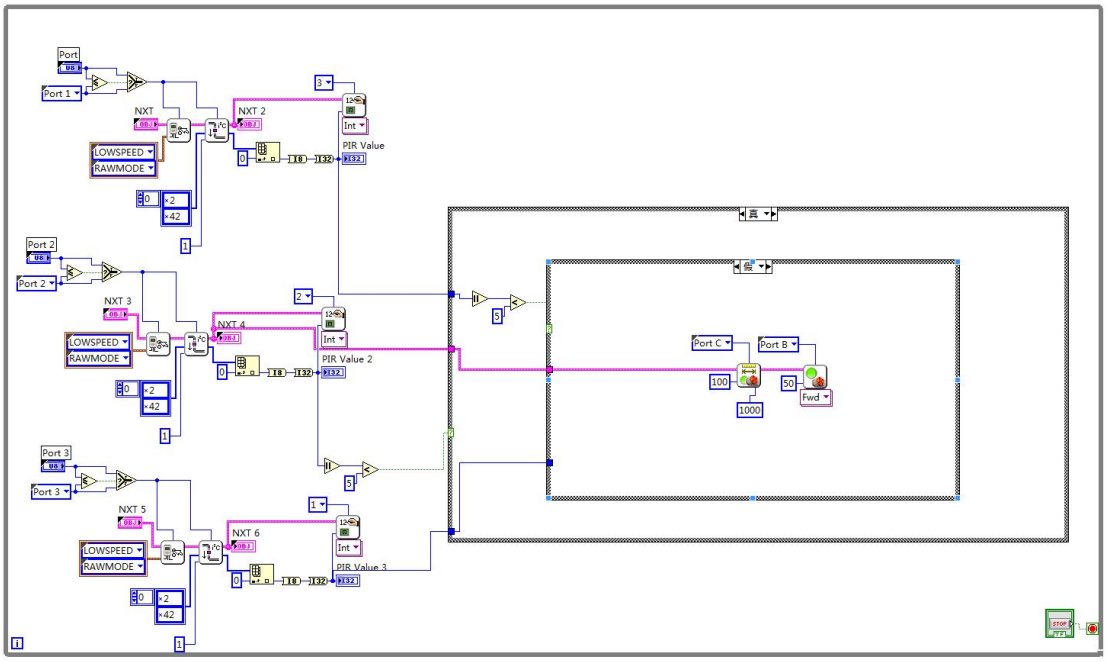


Figure G. 7 PIR sensor 1 detects tempurature difference

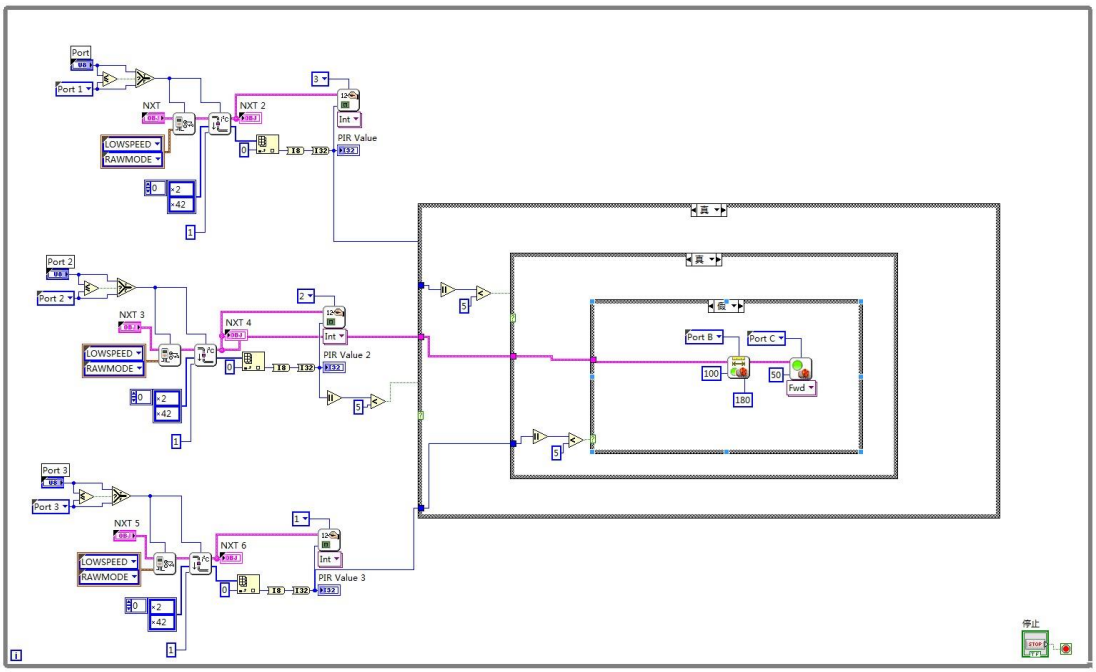


Figure G. 8 PIR sensor 3 detects tempurature difference

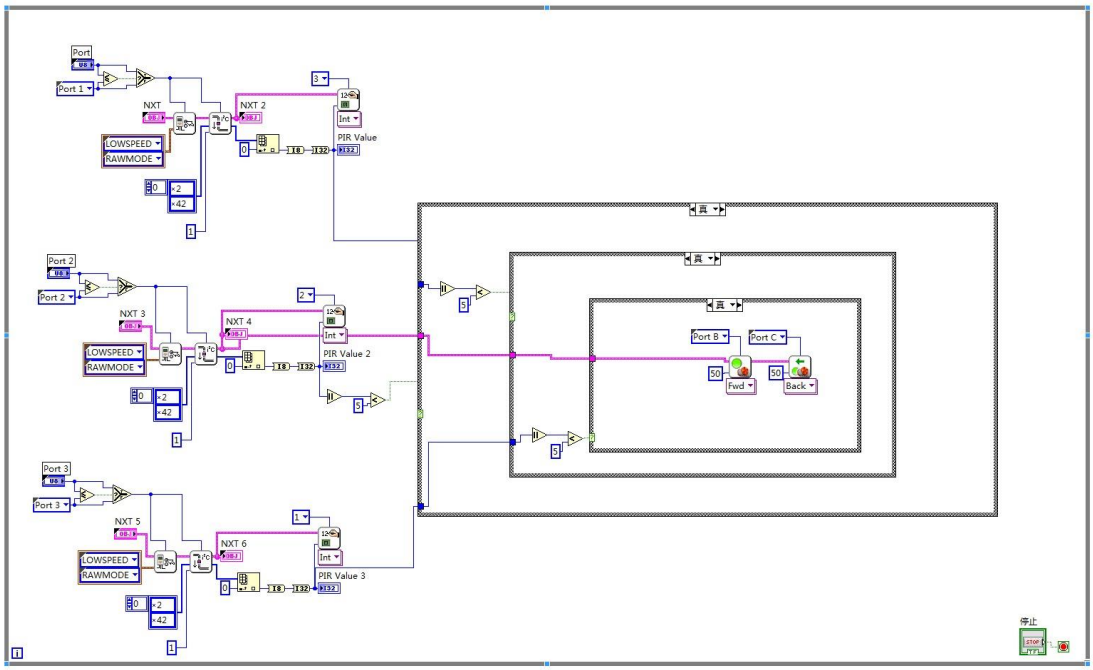


Figure G. 9 No PIR sensor detects tempurature difference

Appendix H

LabVIEW: Human Following Combined with Obstacle Avoidance

In this section, human following and obstacle avoidance functions are combined together as Figure H.1 shows.

The way of combination is that during the human tracking process, the robot keeps searching if there is any obstacle in the detection area. If so, the robot will first avoid the obstacle and then keep detecting and tracking the human. If no obstacle is detected, then the robot simply follows the human. Since obstacle avoidance function and human following function have been introduced in Appendix F and Appendix G, here only an overall flow chart of the program is shown.

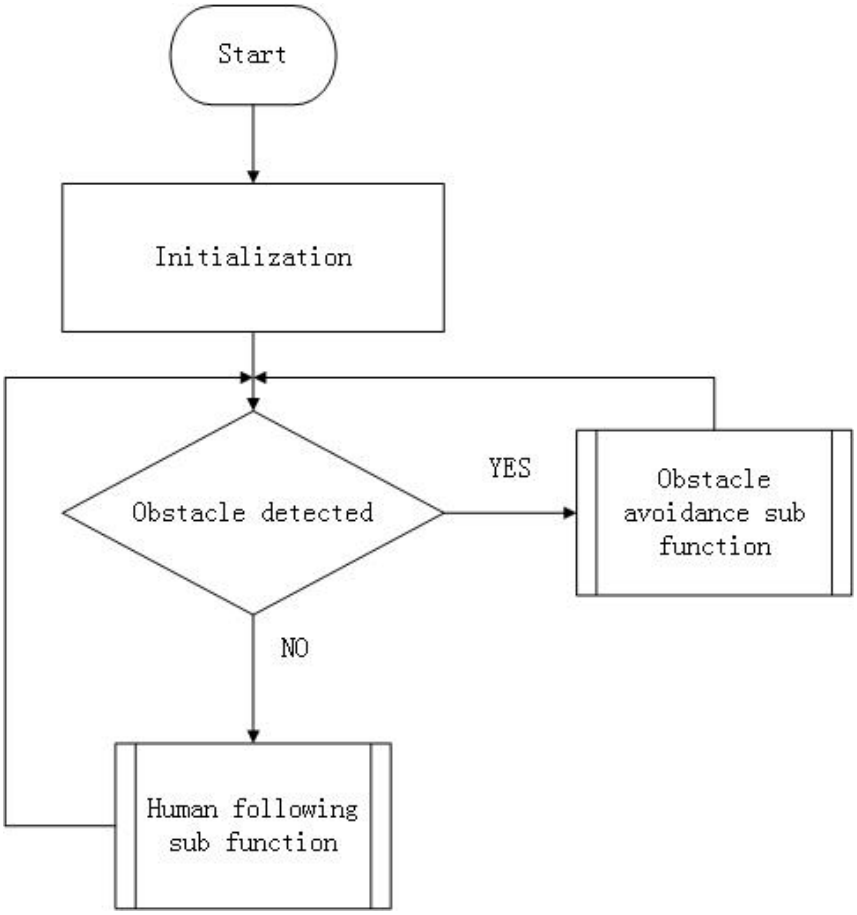


Figure H. 1 Combination of human following and obstacle avoidance functions