MACHINE VISION AND DEPTH PERCEPTION USING MICROSOFT KINECT SENSOR

MUHAMMAD AZFAR BIN MD YUSOF

A thesis submitted in fulfilment of the requirements for the award of the degree of Bachelor of Engineering (Electrical-Mechatronic)

Faculty of Electrical Engineering
Universiti Teknologi Malaysia

July 2012
I declare that this thesis entitled “Machine Vision and Depth Perception Using Microsoft Kinect Sensor” is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature : ________________________________

Name : MUHAMMAD AZFAR BIN MD YUSOF

Date : JULY 2012
ACKNOWLEDGEMENT

In preparing this thesis, I was in contact with many people, researchers, and practitioners. They have contributed towards my understanding and thoughts. In particular, I wish to express my sincere appreciation to my main thesis supervisor, Dr. Zool Hilmi Ismail, for encouragement, guidance, critics and friendship. I am also very thankful to my Computer System Lecturer, Dr. Usman Ullah Sheikh for his guidance, advices and motivation. Without their continued support and interest, this thesis would not have been the same as presented here. My sincere appreciation also extends to all my colleagues and others who have provided assistance at various occasions. Their views and tips are useful indeed. Unfortunately, it is not possible to list all of them in this limited space. I am grateful to all my family members.
ABSTRACT

Robot vision had been a very common topic and interesting engineering field today. Many studies had been conducted by top university to understand the human vision inspired technology and the building of robots that mimic this human capability. This project involved the study of a robot vision, where it can estimate the position and distance of an object like human vision capability. The primary inspiration to this project is from the Kinect sensor developed by Microsoft for their game console, the X-box 360. The aim is to build a robot that can detect and avoid an object and estimate its distance. The robot will be programmed with Kinect sensor to operate without external aid from human while imitating human vision.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DECLARATION</td>
<td>ii</td>
</tr>
<tr>
<td></td>
<td>ACKNOWLEDGEMENT</td>
<td>iii</td>
</tr>
<tr>
<td></td>
<td>ABSTRACT</td>
<td>iv</td>
</tr>
<tr>
<td></td>
<td>ABSTRAK</td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>TABLE OF CONTENTS</td>
<td>vi</td>
</tr>
<tr>
<td></td>
<td>LIST OF FIGURE</td>
<td>vii</td>
</tr>
<tr>
<td></td>
<td>LIST OF TABLE</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>LIST OF APPENDICES</td>
<td>xi</td>
</tr>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.1 Background</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.2 Objectives</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1.3 Project Scope</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1.4 Problem Statement</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>RESEARCH</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2.0 Literature review</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2.0.1 Robot vision using 3D TOF Systems</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2.0.2 Adept Robot</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2.0.3 Real-Time Object Detection using Segmented and Grayscale Images</td>
<td>5</td>
</tr>
</tbody>
</table>
2.0.4 M1-Tennis ball collecting robot 6
2.0.5 B2Bot 7
2.0.6 Kinect Object Recognition, Detection and Manipulations 8
2.1 Kinect Sensor 9
2.2 Kinect main component 10
2.3 Depth Perception 11
2.4 Library Support 16
2.4.1 Laboratories Kinect Code 16
2.4.2 OpenNI 17
2.4.3 Point Cloud Library 17

3 METHODOLOGY AND ANALYSIS 19
3.0 Introduction 19
3.1 Coordinate system 20

4 RESULTS 26
4.0 Operation flow 26
4.1 Algorithm processing 27
4.2 Calculation angles on target 30
4.3 Calculation of the optimal angle to avoid objects
4.3.1 Obstacle to the left robot 32
4.3.2 Obstacle to the right robot 33
4.3.3 Obstacle in the middle of the robot 34
4.3.4 Take a Safe Path 35
4.3.5 Flag of obstacle (Obstacle_flag) 36
and blank lines (freePath_flag)
4.4 Discussions 38

5 CONCLUSIONS 39
REFERENCES 40
APPENDICES 42-93
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Original Recognition</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Real-Time Object Detection Using Segmentation and Grayscale Method</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>M1-Tennis ball collecting robot</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>B2Bot-One axis tracking robot</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>Object Aware Situated Interactive</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Kinect Sensor</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>Kinect main component</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>Kinect tilt-motor</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>Inside Kinect: RGB, IR camera and IR projector</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>depth maps</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>Pattern is projected by the projector and captured by IR camera</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>Distance to a projector from Projector Point</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>12VDC Power Adapter</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>OpenNI library coordination between hardware and end application</td>
<td>17</td>
</tr>
<tr>
<td>Page</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Point Cloud Library</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Translation</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Rotation</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Robot coordinates (observed from the top)</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Homogeneous coordinate system and Mobile Robot</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Kinect coordinates before (left) and after (right)</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Operation flow</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Multi-process handling</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>The program interface control</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Calculation of the target angle</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>obstacles on the left</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>obstacles on the right</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Obstacle in the middle of the robot</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Safe Paths</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Flag of obstacle and blank lines</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Calculations for enough space to pass</td>
<td></td>
</tr>
</tbody>
</table>
## LIST OF TABLE

<table>
<thead>
<tr>
<th>TABLE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Kinect features</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>Power consumption</td>
<td>16</td>
</tr>
</tbody>
</table>
## LIST OF APPENDICES

<table>
<thead>
<tr>
<th>APPENDICES</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Combined Code Laboratories Library (CL) and OpenNI Kinect to use the engine control Kinect</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>How to replace wiring 12V battery adapter to add up and create robot for Kinect</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>Program for Robot Vision</td>
<td>46</td>
</tr>
<tr>
<td>4</td>
<td>Program for Robot Control</td>
<td>65</td>
</tr>
<tr>
<td>5</td>
<td>Gantt Chart FYP1 and FYP2</td>
<td>78</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

1.1 Background

Vision has the most powerful robot sensory capabilities which enable a robot to have a human like sophisticated vision that allows it to respond to its environment in an intelligent and flexible manner. A service robot called robot companions is designed for personal use at home. They are expected to communicate with humans in natural way. Hence, this type of robot usually equipped with vision sensor to detect human movement.

As robot is designed to intimate the human vision, it should be perfectly sized for a living space. In this thesis, a low cost vision robot is introduced due to the design simplicity. Contrary to conventional vision robot that uses other common vision sensor to sense an object, this robot has been equipped with a new type of sensor that is Kinect sensor. This is capable in recognizing an object, where it can chase around a moving object.
1.2 Objectives

In view of the weakness stated earlier, this project is carried out with the aim of building a vision robot that can further enhance the conventional vision robot. There are three main objectives for the project:

1. To interface Kinect sensor with computer.
2. To build a small sized vision based mobile robot that can perform
3. To control the robot vision in 3-dimensional space using Kinect sensor.

1.2 Project Scope

There are three main scopes for this project:

1. Mechanical: The hardware part (mobile robot) will be constructing as the platform to testing the Kinect sensor.
2. Electronics: Main board circuit with motor control module to receive data from computer to control the robot movement using universal serial bus (USB)
3. Software: Using Microsoft Visual Studio 2010 based on C++, Point Cloud Library, OpenCV and OpenNI in as a platform to program the mobile robot
1.3 Problem Statements

Traditional vision system consists of a camera, a main board, several sensors and an associated computer to interface to the main board and execute the image processing algorithm. As a result, the whole system is limited by the cost and complexity of the hardware needed to implement them.

This has resulted in the problems associated with reducing the use of several other sensors and even power consuming.
CHAPTER 2

RESEARCH

2.0 Literature review

2.0.1 Robot vision using 3D TOF systems

Using 3D-TOF (Time of Flight) camera to fix objects segmentation to equal the value of gray level in such images captured which Stereo Vision experiencing difficulty to done it.
2.0.2 Adept robot

Adept Robot was developed by Cletus Kuhn, student majoring in Mechanical Engineering of Colorado State University. The Adept vision system provides a method to perform visual part inspection [2].

Figure 1: Original Recognition

This provides information about various functions of the vision system, explanations on how they work and in what order they should be implemented.

2.0.3 Real-Time Object Detection using Segmentation and Grayscale method

Figure 2: Real-Time Object Detection using Segmentation and Grayscale Method
This approach was developed by Manuela Veloso and Juan Fasola from University of Carnegie Mellon where it combines the color segmented processing strength with the image in grayscale from the scene to detect the AIBO robots for the ROBOCUP games [3]. They used the color segmented processing to locate the robots and images in grayscale for final classification. This method provides a high detection accuracy and fast processing time.

2.0.4 M1-Tennis ball collecting robot

M1 uses Cognachrome Vision System developed by Newton Research Labs to track the tennis ball that scatter around and collect them. Cognachrome Vision System consists of custom frame grabber and processing hardware on board [4].

However the system is too complex and bigger in physical making it not so suitable for on board vision for small mobile robots.

Figure 3: M1-Tennis ball collecting robot
2.0.5 B2Bot

![B2Bot-One axis tracking robot](image)

**Figure 4:** B2Bot-One axis tracking robot

B2Bot is a robot attached with CMUCam mount on a servo, two DC motors and a custom controller called Cerebellum [5]. This mobile robot use CMUCam as vision sensor. However, the robot is limited to do only one axis object tracking.

Kinect is hot new technology that can be applied in various ways. There have been a few projects that used Kinect sensor technology.
2.0.6 Kinect Object Recognition, Detection and Manipulations

![Figure 5: Object Aware Situated Interactive](image)

RGB Division from University of Washington has developed object recognition as part of their project Object Aware Situated Interactive or codename “OASIS”. The approach uses both depth and color information from the camera underlying Kinect to recognize different objects [6].
2.1 Kinect Sensor

Kinect sensor features a RGB camera and 3-Dimensional depth sensor that run on proprietary software which provides motion capture in 3D. The depth sensor consists of the combination of an infrared laser projector with a monochrome CMOS sensor that captures 3D video data under ambient light conditions. The Kinect software can automatically calibrate the sensor according to the human physical motion, filtering out the presence of other obstacles.

Figure 6: Kinect Sensor
2.2 Kinect Main Component

![Diagram of Kinect Main Component](image)

**Figure 7: Kinect Main Component**

The Kinect includes: RGB camera, depth sensor (3D Depth Sensors), Range microphone (Multi-array) and the elevation angle motor control (Motorized Tilt).

- **Camera RGB**: like a regular camera, resolution 640 × 480 with speed of 30 fps.
  - Sensor depth: the depth is obtained through a combination of two sensors: light projector infrared (IR Projector) and infrared camera (IR camera).
  - Multi-microphone array comprises four microphones are arranged along the Kinect as above Figure 7, is used to control applications by voice.
  - Elevation angle motor control: DC motor is relatively small, makes the adjustment to the camera up and down.
2.3 Depth Perception

Pairs of IR sensors and IR camera projector will work together to produce high value Light picture with deep technology Coding from PrimeSense.
Different techniques used Stereo Camera with the same camera pairs to build created depth map, or techniques. TOF defines the distance with estimated travel time of light goes on and on in space; technical Light Coding Using an infrared light projector coupled with a continuous camera IR to calculate the distance. The calculation is done inside the Kinect with System on Chip or SoC in the PrimeSense PS1080 chip.

This new technology is said to meet more accurate, cheaper prices for use in indoor environments. Projector will project a beam of infrared light, creating bright spots, gathering spot of light emitted is fixed. This light spot is generated by a light source transmitted through a diffraction grating. A set of bright spots are captured by IR camera, through algorithm particularly in the PS1080 integrated SoC for the depth map. The nature of the solution the computational technique is based on the geometric relationship between the camera and IR sensor. Figure 11 shows clearly the pattern set of bright spots from the Projector and photographed by the IR camera.
Figure 12: Distance to a projector from Projector Point

It is assumed that the Projector transmits a light green along the way; it will be taken as a blip by the IR camera by touching objects on the surface time. The three planes at three different distances: the plane near the Kinect (Close plane), flat surface away from the Kinect (Distant plane) and the reference plane (Reference plane) between the two planes. In particular, the ground reference plane is known front inside the Kinect with full information about the distance. In addition, the IR camera image plane was added and the projection of the plane points collection in space.

Consider the three cases when the ray of light green touch three points on three planes respectively A, B, C and three-point is projected onto the image plane, respectively A', B', C'. Observe the position A', B' and C', remark: as close to point A the Kinect (or close as close the Kinect plane), then A 'as far B 'on the right side; vice versa, as far the Kinect point C (or as far away Distant plane Kinect), then C 'as far as B' on the left. Note that the direction, the starting point of light from the projector and position B 'is the projection of point B in the plane reference to the image plane, thus it is possible to calculate the depth of images or distance to an object.
The Kinect executes the same thing to the set of the remaining bright spots broadcast from projector. It finds the center point of a light and IR camera capture the similarities of the bright spots on the reference plane (for example: Figure 12, A and B, C and B are similar pairs, to find the gap between these two points horizontally as shown on the image plane, and note that the value this difference is calculated in pixel units. Gather all the difference value from a set of bright spots, will make the difference map (disparity map), this value has greater distance or depth images value from which build the depth map to the value actually measured in meters. However, amount of light spots from projector smaller than the total pixels number on the image plane part of the IR camera to images depth values will be interpolated due to aggregation.

According to calculations by Nicolas Burrus, the person discovers the way for learning about the Kinect. His mathematical formula between the true value $Z$ distance in meters and the difference value $d$:

$$z = \frac{1}{-0.00307111016 d + 3.3309495161}$$  \hspace{1cm} (1)$$

where $d$ is the integer represented as 11 bits, or about to change from 0 to 2047. With experimental measurements on OpenNI library, the value of variable $Z$ of about 0.5 to 6.0 meters and a depth of stable maps of about 0.5 to 5.0 meters. Therefore, the $d$ value actually varies in the range from 434 to 1030. Thus, in the space between 0 to 0.5 meters in front of Kinect.

Some other features of interest Kinect: camera focal length and aperture (field of view), power supply and power consumption, operating environment. Microsoft’s Kinect commercial specifications should not be published. The parameters presented below are the experimental measurements:
a. Focus, aperture and RGB camera IR camera:

i. Both RGB and IR camera are spaced 2.5 cm should be slightly different in collection of frames from two cameras. The parameters in Table 1 were measured by experiments:

<table>
<thead>
<tr>
<th>Feature</th>
<th>RGB Camera</th>
<th>IR Camera</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal</td>
<td>~62°</td>
<td>~58°</td>
</tr>
<tr>
<td>Vertical</td>
<td>~48°</td>
<td>~44°</td>
</tr>
<tr>
<td>Diagonal</td>
<td>~72°</td>
<td>~69°</td>
</tr>
<tr>
<td>Pixels</td>
<td>525</td>
<td>580</td>
</tr>
</tbody>
</table>

b. Power supply and power consumption:

i. Since the Kinect requires more power, an external power supply is obtained from 12V Direct Current adapter. Xbox-360 version will not need adapter because it has the AUX port to give a special connector. With USB connectivity it is possible to communicate the Kinect with the computer. The modification of power adapter 12V battery for use on mobile robot is added in Appendix 2.

Figure 13: 12VDC power adapter
Table 2: Power Consumption

<table>
<thead>
<tr>
<th>Power consumption (idle)</th>
<th>~3.3W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power consumption (active)</td>
<td>~4.7W</td>
</tr>
</tbody>
</table>

c. Operating Environment:

Kinect is a device designed for use in indoor environments (indoor). In outdoor environment, test results for the depth map is not accurate at time of strong light, but acceptable results when the light (on evening time).

2.4 Library support

Kinect has been developed by many software developers, not only on developing games for Xbox array but also the array of image processing applications in medicine, robotics, mapping and many more. There were many Kinect libraries written for Kinect. Below are the lists of libraries use for developing image processing from Kinect sensor:

2.4.1 Laboratories Kinect Code

Code Laboratories (CL) is software specializing in home support development; developers exploit the features of the image processing device. CL provides users to control the Kinect basic features that are camera, audio
and motor.

2.4.2 OpenNI

OpenNI library is considered the biggest library in the presence of Kinect Beta SDK; this library supports multiple languages on many platforms, allowing programmers to write codings for Kinect applications. The main purpose of OpenNI is to build the API standard, allowing library combined with the ability to increase strength middleware Kinect.

![OpenNI library coordination between hardware and end application](image14)

**Figure 14**: OpenNI library coordination between hardware and end application

2.4.3 Point Cloud Library

![Point Cloud Library](image15)

**Figure 15**: Point Cloud Library
PCL is a library provides support for Point Cloud for image processing in 3D space. This library are built with many algorithms such as filtering (filtering), restore for surface (surface Reconstruction), partition (segmentation), feature estimates object (feature Estimation). PCL can be used on multiple platforms such as Linux, Mac, Windows and Android. To simplify development, PCL is divided into several small libraries and can be compiled separately. It can be said PCL is a combination of many small modules. These modules do substance libraries also perform individual functions before closing PCL package. The basic library is:

a. Eigen: open a support library for linear operations is used in almost all the mathematical calculations for PCL.
b. FLANN: (Fast Library for approximate Nearest Neighbors) find support for the neighboring points in 3D space.
c. Boost: help to share pointers on all the modules and algorithms in PCL to avoid copying duplicate data were obtained on the system.
d. VTK: (Visualization Toolkit) support for multiple platforms in the collection of data 3D display support, to estimate the volume of objects.
e. CMinPack: a library open help for the settlement the computational linear and not linear.
CHAPTER 3

METHODOLOGY AND ANALYSIS

3.0 Introduction

The detection and avoid obstacles is always referred to the self-propelled robot. Along with the development of technology, more and more built-in sensor was developed for robot to do this simple and accurate application. Previously, the super-sensors for sound are widely used; recently, people interested in applying image processing technology to the robot.
3.1 Coordinate system

In homogeneous coordinates, each point \((x, y, z)\) in Cartesian coordinates is represented by a set of four values in the compact four-dimensional space\((h_x, h_y, h_z, h)\). It is convenient for calculation, people often choose \(h = 1\). Thus, a point \((x, y, z)\) in Cartesian coordinates will be transformed into the point \((x, y, z, 1)\) in homogeneous coordinates, while the \((x, y, z, w)\) in homogeneous coordinates \((with \ w \neq 0)\) will correspond to the point \((x/w, y/w, z/w)\) in the Cartesian coordinate. Three-dimensional transformation of \(P\) into the variable \(Q\) has the form: \(Q = PM\) of which: \(Q = (x', y', z', 1)\), \(P = (x, y, z, 1)\), \(d = (dx, dy, dz)\) is the translation vector and \(M\) is the matrix 4x4 transformation in homogeneous coordinates:

\[
M = \begin{bmatrix}
    a & b & c & 0 \\
    d & e & f & 0 \\
    g & h & i & 0 \\
    dx & dy & dz & 1
\end{bmatrix}
\]  

(2)

Figure 16: Translation
Translation vector in 3-dimensional transformation has a very intuitive effect: each translated to a point about the $dx, dy, dz$ in 3axis. Matrix $M$ allows translation the following form:

\[
M = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
dx & dy & dz & 1
\end{bmatrix}, Q = [x', y', z', 1]
\] (3)

Unlike in two-dimensional rotations around any point, the three-dimensional rotations around a coordinate axis have the matrix representation of rotations around the $x, y, z$ axis with counterclockwise lake corresponding $\beta, \varphi, \theta$ angle respectively to $R_x (\beta), R_y (\varphi), R_z (\theta)$:

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{rotation.png}
\caption{Rotation}
\end{figure}

Observed elevation of the robot model in Figure 18, the original coordinates of the robot was chosen as the center O (0, 0). Corresponding position of the center two left and right wheels are R (20.5, 0), L (-20.5, 0) (in units of centimeters). During the move, the coordinates of robot will be updated with new location. The updated values include the current coordinates $x$ (x_cur), $y$ coordinates of the current (y_cur) and the angle is at (axis O’Y 'compared to the axis OX, angle_cur). Thus, starting at boot time: $x$ _cur = 0, angle_cur $y$ _cur = 0 and = 90 °.
As the robot move, it will have two main movements: go straight (up or down) an L alpha angle and rotate around the center a robot. The problem simply applies to basic translation and rotation in 2D space. The equation of the new coordinates robot for each action:

- **Go straight a segment L:**

  \[
  \begin{align*}
  x_{\text{new}} &= x_{\text{cur}} \pm L \cos(\text{angle}_{\text{cur}}) \\
  y_{\text{new}} &= y_{\text{cur}} \pm L \sin(\text{angle}_{\text{cur}}) \\
  \text{angle}_{\text{new}} &= \text{angle}_{\text{cur}}
  \end{align*}
  \]

  The "+" for forward cases and "-" the backward cases.

- **Rotate by alpha angle:**

  \[
  \begin{align*}
  x_{\text{new}} &= x_{\text{cur}} \\
  y_{\text{new}} &= y_{\text{cur}} \\
  \text{angle}_{\text{new}} &= \text{angle}_{\text{cur}} + \alpha
  \end{align*}
  \]
Negative alpha when robots turn right (upon clockwise) and positive when to the left (counter clockwise). Angle_cur value in the range 0 - 2 * PI should be keeping in mind when this angle is outside this range:

\[
\begin{align*}
\text{angle} & \_\text{cur} < 0 : \text{angle} & \_\text{cur} = \text{angle} & \_\text{cur} + 2*\text{PI} \\
\text{angle} & \_\text{cur} > 2*\text{PI} : \text{angle} & \_\text{cur} = \text{angle} & \_\text{cur} - 2*\text{PI}
\end{align*}
\]

Calculate coordinates Kinect

Information about the environment in the process of moving the robot is put Kinect the 3D processor in order to give a command to the robot accurately. With the strong support of the library Point Cloud, Kinect gives the right information determine the coordinates of the point or object in the real environment than in the original coordinates Kinect. A major drawback is the Kinect blind spot in the range 0 to 50 meters, is a large gap to avoid material functional. That is the main reason why the Kinect have to be placed high with it heads down with a certain angle, and is set back behind the robot to see an overview of the most forward robot environment. The problem of Kinect displacement coordinate system on the robot coordinate system has to be synchronized. There is little difference between the two axes: for Kinect coordinate system, Z represents the distance to the object, Y represents the height of the object respectively corresponding to the Y and Z coordinate system the robot. The remaining X will not change. The goal is to bring the plane of Kinect OXZ on OXY plane coincides with the robot. Observing Figure 19, it can be find out at an angle of 30 °, Kinect also differences in the direction Z a Y axis 9 cm and a piece of 60 cm.
Figure 19: Homogeneous coordinate system and mobile robot

- Turn plane backward OYZ X axis clockwise an angle 30°.

\[
\begin{bmatrix}
X_1 \\
Y_1 \\
Z_1
\end{bmatrix} =
\begin{bmatrix}
X & Y & X
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \beta & \sin \beta \\
0 & -\sin \beta & \cos \beta
\end{bmatrix}
\begin{bmatrix}
X \\
Y_1 \\
Z_1
\end{bmatrix} =
\begin{bmatrix}
X \\
Y \cos \beta - Z \sin \beta \\
Y \sin \beta + Z \cos \beta
\end{bmatrix}
\]

- Y-axis translation stage to a 60 cm and Z axis translation stage to 9 cm.

\[
\begin{bmatrix}
X_2 \\
Y_2 \\
Z_2
\end{bmatrix} =
\begin{bmatrix}
X_1 \\
Y_1 \\
Z_1
\end{bmatrix} +
\begin{bmatrix}
0 \\
-60 \\
9
\end{bmatrix} =
\begin{bmatrix}
X \\
Y \cos \beta - Z \sin \beta - 60 \\
Y \sin \beta + Z \cos \beta + 9
\end{bmatrix}
\]

- Reversing Y.

\[
\begin{bmatrix}
X_3 \\
Y_3 \\
Z_3
\end{bmatrix} =
\begin{bmatrix}
X_2 \\
-Y_2 \\
Z_2
\end{bmatrix} =
\begin{bmatrix}
X \\
- Y \cos \beta + Z \sin \beta + 60 \\
Y \sin \beta + Z \cos \beta + 9
\end{bmatrix}
\]
Figure 20 shows the axis system Kinect after making the transformation. System shaft dimensions X, Y, Z correspond to the red, green, blue.

**Figure 20**: Kinect coordinates before (left) and after (right)
CHAPTER 4

RESULTS

4.0 Operation flow

An overview of the main block and the blocks task is given in Figure 21. Computer links Kinect along with programming tools to create blocks of a circuit and microcontroller circuit’s capacity to create Block 2, Block 3 is a mechanical model of the mobile robot. Block 1 acted as the eyes and brain of the robot, robot analysis does not ahead of time and passes appropriate orders to Block 2. Block 2 will do robot controller (Block 3) in accordance with the requirements of Block 1, the movements such as turning left, right or run forward, backward with a specified value. After calculating the coordinates from the feedback signal encoder on the robot, the current position value will be passed back to Block 2 and Block 1 update then displayed to the monitor screen.
Figure 21: Operation Flow
4.1 Algorithm processing

Computer was used in three flow control for parallel processing: processing Kinect, move handling of robot planning and handling communication with the microcontroller.

![Diagram of Multi-process handling]

**Figure 22: Multi-process handling**

In Multi-process Handling, Kinect flow (No. 1) task analysis environment provides information on size, location if obstructions occur before the robot. Flow communications with the microcontroller (No. 3) by UART standard communication tasks commands to and receives location information from the microcontroller. Processing flow to move the robot using information from flow No. 1 and Flow No. 3, the robot moves to a desired target.
Figure 23: The program interface control

The program interface control is written in MFC in Visual Studio 2010 Microsoft. Target position to move to will be included into the Target Coordinate; current coordinates of the robot will automatically appear in the Current Coordinate block; options communication parameters for UART communication port status is set in block Serial Control; of mass Robot Status display information about detected obstacles flags, flags detecting blank lines, number of obstacles and robot status in target position or not. Tilt Control Kinect elevation angle control.

- The format control via UART communication
  - Control character: R (turn right), L (turn left), F (go straight), T (go straight on destination).
  - Control value: the value angle or distance moved the types of double data type, for three decimal places in units of centimeters.
  - ‘#’, ‘*’: Is the special character for character start and end of string.

An order received from the microcontroller has the form: X [value 1] Y [value 2] A [value 3] N, of which:

- X, Y, A: The turn is the beginning for the character value represents the coordinates X, Y, angle current robot. N is the character string end.
- Value of 1 - 3 is the value corresponding to the coordinates, the current angle of the robot.
4.2 Calculations of angles on target

Figure 24 shows a case the right of the target robot; this robot will turn an angle alpha to the right toward the target. We have calculated for the case follows (assume y-cur always positive, the robot always move forward):

\[ \gamma = \arctan \left( \frac{y_{\text{target}} - y_{\text{cur}}}{x_{\text{target}} - x_{\text{cur}}} \right) \] \hspace{1cm} (8)

- gamma < 0 (the left landing robots):

  \[ 0 \leq \text{angle}_\text{cur} < \pi + \gamma \]
  \[ \text{Alpha} = \pi - \text{angle}_\text{cur} + \gamma + \text{Go to the left robot} \]

  \[ \pi + \gamma \leq \text{angle}_\text{cur} < 2\pi + \gamma \]
  \[ \text{Alpha} = \text{angle}_\text{cur} - (\pi + \gamma) + \text{Go to the right robot} \]


- 2*PI + \( \gamma \) \( \leq \) angle\_cur \( \leq \) 2*PI:
  Alpha = 3*PI - angle\_cur + \( \gamma \) + Go to the left robot

- \( \gamma \) \( \geq \) 0 (the right of the target robot):
  - 0 \( \leq \) angle\_cur \( < \) \( \gamma \):
    Alpha = \( \gamma \) - angle\_cur + Go to the left robot
  - \( \gamma \) \( \leq \) angle\_cur \( < \) PI + \( \gamma \):
    Alpha = angle\_cur - \( \gamma \) + Go to the right robot
  - PI + \( \gamma \) \( \leq \) angle\_cur \( \leq \) 2*PI:
    Alpha = 2 * PI - angle\_cur + \( \gamma \) + Go to the left robot
4.3 Calculations of the optimal angle to avoid objects

The aim is to control a robot in the direction of smaller angles while avoiding obstacles. Cases can occur are:

### 4.3.1 Obstacle to the left robot

![Diagram of obstacle on the left]

Figure 25: obstacle on the left

Observing Figure 25, the angle of rotation needed to get rid of obstacles as alpha (α) the robot in a position to the obstacle. Angle alpha is altogether when we know max_point coordinates, angles and magnitudes O’Lφ.

\[
\alpha = \phi' - \phi = \cos^{-1} \left( \frac{-x_{\text{max}}}{O'L} \right) - \phi,
\]

turned toward the right direction robot.
4.3.2 Obstacle to the right robot

As with obstacles on the left, the angle alpha is calculated as follows:

\[ \alpha = \phi' - \phi = -1 \cos \left( \frac{x_{\text{min}}}{O'L} \right) - \phi \], turned left towards the robot.
4.3.3 Obstacle in the middle of the robot

In this case, the robot will find angles of alpha dodge objects too small, if magnitude more obstacles to the left, the robot to the right (as shown in Figure 8.8) and vice versa. The formula in each case (in which the angle $\phi$ and $O'L = O'L \ 'know$)

- Back left:
  \[
  \alpha = \phi + \phi = \sin^{-1}\left(\frac{x_{\text{min}}}{O'L}\right) + \phi
  \] (9)

- Go right:
  \[
  \alpha = \phi + \phi = \sin^{-1}\left(\frac{-x_{\text{max}}}{O'L}\right) + \phi
  \]
4.3.4 Take a Safe Path

![Diagram of a robot navigating around obstacles with coordinates and equations.]

Figure 28: Safe Path

Safe distance move to get rid of the obstacle plus O'H a fixed value deltaMove enough (here choose deltaMove = 15 cm based on the test).

- Go to the right obstacle:

\[
OH' = OM_1 \times \cos(\alpha - \theta) = \cos(\alpha - \tan^{-1}\left(\frac{x_{max}}{y_{max}}\right))\sqrt{(x_{max} + x_{cur})^2 - (y_{max} + y_{cur})^2}
\]

(10)

- Go to the left obstacles:

\[
OH' = OM_1 \times \cos(\alpha - \theta) = \cos\left(\alpha - \tan^{-1}\left(\frac{|x_{min}|}{y_{max}}\right)\right)\sqrt{(x_{min} + x_{cur})^2 - (y_{max} + y_{cur})^2}
\]

(11)
4.3.5 Flag of obstacle (Obstacle_flag) and blank lines (freePath_flag):

Figure 29: Flag of obstacle and blank lines

Figure 29 of space appeared in front of robot obstacle, and then the flag respectively will pop up in the defined area. Robot vision limited distance of 140 cm.
Flag of robot for safepath:

![Figure 30: Calculation for enough space to pass](image)

Flag for enough space for the robot will pop up a bore through the H1H2 distance (Figure 30) for the robot passes through a large enough (enough space for the robot passes through the H1 and H2 > 50 cm). This is calculated as the function O’H1 traveled, O’H2 similar to:

\[
O'H_2 = O'M_4 \times \cos(\chi)
\]

\[
= \cos\left(\alpha - \tan^{-1}\left(\frac{|x_4|}{y_4}\right)\right) \sqrt{(x_4 + x_{\text{cur}})^2 - (y_4 + y_{\text{cur}})^2}
\]

\[\Rightarrow H_1H_2 = O'H_2 - O'H_1 \quad (12)\]
4.4 Discussions

Several problems need to overcome:

- The mechanism of the mobile robot should not affect the perfect error the move. At this point, an omni-directional robot will meet the expectation for the application robot move indoor with the flexibility, compact structure and high precision.
- The only visual observation at robot front space should have obstructions can be touching side or behind when moving backward. This problem can be solved by equipping the sensor to detect obstacles around the robot.
- Integrated sensors needed to increase positioning accuracy.
- The program is written in a Windows environment with support from Point Cloud Library is still limited, especially in processing speed. It will be optimal if the writing on the Linux and if developed into commercial products will lower the price software down, increase competition.

Kinect devices that can help us build the map (mapping) in 3D. An integration between Kinect and certain program is use to help us build robots will be the map in areas where people cannot access.
CHAPTER 5

CONCLUSION

The concept of robot vision is based on the few theories that inspired from human vision. A mobile vision robot that can manipulate its surrounding and environment have been constructed according to schedule.

For the purpose of model building low cost mobile robots capable of moving to the exact point predetermined, avoiding obstacles and safe destination. This project has applied treatment technology 3D imaging for the sight of the robot with Kinect gaming devices, meeting the reliable than conventional sensors. Implementation process has been completed through the following tasks:

- Image processing program to recover space in front of the robot in the form of 3D, providing full information about the robot environment.
- The speed image processing on the computer about 13-15 fps, enough to meet real-time robot.
- Design and construction of a complete model of mobile robot.
REFERENCES


APPENDIX 1: Combined Code Laboratories Library (CL) and OpenNI Kinect to use the engine control Kinect.

OpenNI library does not support motor control for Kinect. The following is the installation procedure by adding CL. Both OpenNI and CL should be installed simultaneously to act as drivers to access Kinect.

- Step 1:

CL normal installation and ensure OpenNI have not installed and the program comes before (if so, to remove).
• Step 2:

Open the Computer Management window, and select Uninstall the items under the CL devices: NUI audio, camera NUI, NUI Motor.

![Computer Management Window](image)

• Step 3:

Install OpenNI and accompanying programs.

• Step 4:

As in step 1, right-click on kinect motor and update driver software
And browse to the folder of CL:

Now we can use the Kinect on the motor control library OpenNI associated with CL.
APPENDIX 2: How to modify wiring 12V battery adapter to add up and create robot for Kinect

Then use the DC plug to the battery and the wires on:
APPENDIX 3: Program for Robot Vision

// RobotVisionDlg.cpp : implementation file
#include "stdafx.h"
#include "RobotVision.h"
#include "RobotVisionDlg.h"
#include "serialCtl.hpp"
#include "serialThread.hpp"
#include "afxdialogex.h"
#include "nuiMotor.h"
#include "highgui.h"
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <sstream>
#ifdef _DEBUG
    //#define new DEBUG_NEW
#endif
#define FPS_CALC(_WHAT_)
    do {
        static unsigned count = 0;
        static double last = pcl::getTime ();
        double now = pcl::getTime ();
        ++count;
        if (now - last >= 1.0) {
            std::cout << "Average framerate("" << _WHAT_ << ")": " << double(count)/double(now - last) << " Hz" << std::endl;
            count = 0;
            last = now;
        }
    }while(false)
#ifdef _DEBUG
    //#define new DEBUG_NEW
#endif

//Point Cloud declare
boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new pcl::visualization::PCLVisualizer ("3D Viewer");
pcl::PassThrough<CloudType> pass_;
pcl::VoxelGrid<CloudType> grid_;
pcl::SACSegmentation<CloudType> seg_;
pcl::ExtractIndices<CloudType> extract_;
pcl::search::KdTree<CloudType>::Ptr tree;
pcl::EuclideanClusterExtraction<CloudType> ec;

CloudType min_pt, max_pt, minOnPath(0, 0, 0), maxOnPath(0, 0, 0); //Range of objects' position
double theta = -0.513f; //29.4 degree
const float cosTheta = cos(theta);
const float sinTheta = sin(theta);

//Controller process declare

#define RobotWidth 0.44
#define RobotLength 0.365
#define RobotCross 0.422
#define RANGEOBSTACLE 0.90
#define PI 3.1415926535897932

CWinThread *ContrThread;
double anpha, anpha_, anpha_cmd;
CString direction, direction_old;
double safe_distance, safe_distance2, distance_to_target;
unsigned long len;
CloudType maxpointTemp(0,0,0), minpointTemp(0,0,0);
bool obstacle_flag, bFirst = true, freePath;
double m_dbXcurrent_Temp, m_dbYcurrent_Temp, m_dbAngleCurrent_Temp, angleTemp;
int numObs;

//Test openCV

CvCapture* capture;
IplImage* frame;
std::vector<CloudType> matObsInfo[2];

//Point Cloud local structure

struct
{
    CloudConstPtr cloud_
    boost::mutex cloud_mutex_

    void cloud_callback(const CloudConstPtr& cloud)
    {
        boost::mutex::scoped_lock lock (cloud_mutex_);
        cloud_ = cloud;
    }

    CloudConstPtr getLatestCloud ()
    {
        boost::mutex::scoped_lock lock (cloud_mutex_);
        CloudPtr temp_cloud (new Cloud);
        CloudPtr temp_cloud2 (new Cloud);
        CloudPtr temp_cloud3 (new Cloud);
        CloudPtr cloud_rotated (new Cloud);
        CloudPtr cloud_pass_ (new Cloud);
        CloudPtr cloud_cluster (new Cloud);
        CloudPtr cloud_cluster_closest (new Cloud);
        CloudPtr cloud_plane (new Cloud);
        CloudPtr cloud_floor (new Cloud);
        CloudPtr cloud_wall (new Cloud);
        std::vector<pcl::PointIndices> cluster_indices;
        bool planeWall(false);
pass_.setInputCloud (cloud_);
pass_.filter (*cloud_pass_);

grid_.setInputCloud (cloud_pass_);
grid_.filter (*temp_cloud);

for(Cloud::const_iterator it = temp_cloud->begin(); it != temp_cloud->end(); ++it)
{
    float x = it->x;
    float y = (it->y)*cosTheta - (it->z)*sinTheta;
    float z = (it->y)*sinTheta + (it->z)*cosTheta + 0.09f;
    CloudType point;
    point.x = x;
    point.y = -y;
    point.z = z;

    cloud_rotated->push_back(point);
}

pcl::ModelCoefficients::Ptr coefficients (new pcl::ModelCoefficients ());
pcl::PointIndices::Ptr inliers (new pcl::PointIndices ());

seg_.setInputCloud (cloud_rotated);
seg_.segment (*inliers, *coefficients);

extract_.setInputCloud (cloud_rotated);
extract_.setIndices (inliers);
extract_.setNegative (planeWall);
extract_.filter (*cloud_plane);
CloudType minPlane(0, 0, 0);
CloudType maxPlane(0, 0, 0);
pcl::getMinMax3D(*cloud_plane, minPlane, maxPlane);
if(maxPlane.y > 0.15)
{
    planeWall = true; //Plane is wall
    *cloud_wall = *cloud_plane;
} else
{
    *cloud_floor = *cloud_plane; //Plane is floor
}

if(planeWall)
{
    extract_.setNegative (planeWall);
    extract_.filter (*temp_cloud3);
    seg_.setInputCloud (temp_cloud3);
    seg_.segment (*inliers, *coefficients);

    extract_.setInputCloud (temp_cloud3);
    extract_.setIndices (inliers);
    extract_.setNegative (false);
    extract_.filter (*cloud_floor);
extract_.setNegative (true);
extract_.filter (*temp_cloud2);
ec.setInputCloud(temp_cloud2);
ec.extract (cluster_indices);

else
{
    extract_.setNegative (!planeWall);
    extract_.filter (*temp_cloud2);
    ec.setInputCloud(temp_cloud2);
    ec.extract (cluster_indices);
}

float closest_z = 2, closestOnPath = 2, disTemp, disOnPath;
min_pt.x = 0;
min_pt.y = 0;
min_pt.z = 0;
max_pt.x = 0;
max_pt.y = 0;
max_pt.z = 0;

int countNum = 0, sizeMat = 0;

for (std::vector<pcl::PointIndices>::const_iterator it =
cluster_indices.begin (); it != cluster_indices.end (); ++it)
{
    CloudPtr cloud_cluster_temp (new Cloud);
    CloudType min_pt_temp(0,0,0);
    CloudType max_pt_temp(0,0,0);
    for (std::vector<int>::const_iterator pit = it-
>indices.begin (); pit != it->indices.end (); ++pit)
    {
        cloud_cluster_temp->points.push_back (temp_cloud2-
>points[*pit]);
        cloud_cluster_temp->points.push_back (temp_cloud2-
>points[*pit]);
    }
    pcl::getMinMax3D (*cloud_cluster_temp,  min_pt_temp,
max_pt_temp);
    disTemp = min_pt_temp.z;
    sizeMat = matObsInfo->size();
    matObsInfo[0].resize(sizeMat + 1, CloudType(0, 0, 0));
    matObsInfo[1].resize(sizeMat + 1, CloudType(0, 0, 0));
    matObsInfo[0][countNum] = min_pt_temp;
    matObsInfo[1][countNum] = max_pt_temp;

    if(disTemp < closest_z )
    {
        closest_z = disTemp;
        *cloud_cluster_closest = *cloud_cluster_temp;
    }
    countNum ++;
}

numObs = countNum;
viewer->setBackgroundColor (0, 0, 0);
viewer->removePointCloud("cloud 1");
viewer->removePointCloud("cloud 2");
viewer->removePointCloud("cloud 3");
viewer->removePointCloud("cloud 4");
viewer->removeText3D("text 3d");
viewer->removeText3D("text 3d2");
viewer->removeShape("Line");

// calculation for Obstacle flag
if(countNum)
{
    pcl::getMinMax3D (*cloud_cluster_closest, min_pt, max_pt);

    std::stringstream ss1, ss2;
    ss1 << "min_pt " << min_pt;
    ss2 << "max_pt " << max_pt;
}

// calculation for freePath
int checkPath = 0, checkObstacle = 0;
CloudType minOnPathTemp(0, 0, 0);
CloudType maxOnPathTemp(0, 0, 0);
for(int i = 0; i < numObs; ++i)
{
    if(((matObsInfo[1][i].x < 0.225 && matObsInfo[1][i].x > -0.225) || (matObsInfo[0][i].x > -0.225 && matObsInfo[0][i].x < 0.225)) || (matObsInfo[0][i].x < -0.225 && matObsInfo[1][i].x > 0.225))
    {
        if(matObsInfo[0][i].z < 1.2) checkPath++;
        if(matObsInfo[0][i].z < 0.9) checkObstacle++;

        disOnPath = matObsInfo[0][i].z;
        if(closestOnPath > disOnPath)
        {
            closestOnPath = disOnPath;
            minOnPathTemp = matObsInfo[0][i];
            maxOnPathTemp = matObsInfo[1][i];
        }
    }
}
if(checkObstacle) obstacle_flag = true;
else obstacle_flag = false;

if(checkPath)
{
    freePath = false;
    minOnPath = minOnPathTemp;
    maxOnPath = maxOnPathTemp;
}
else
{
    freePath = true;
    minOnPath.x = 0;
    minOnPath.y = 0;
    minOnPath.z = 0;
    maxOnPath.x = 0;
    maxOnPath.y = 0;
    maxOnPath.z = 0;
}

pcl::visualization::PointCloudColorHandlerCustom<CloudType>
single_color1(cloud_cluster, 0, 255, 0);
viewer->addPointCloud<CloudType> (cloud_cluster, single_color1, "cloud 1");

pcl::visualization::PointCloudColorHandlerCustom<CloudType>
single_color2(cloud_cluster_closest, 255, 0, 0);
viewer->addPointCloud<CloudType> (cloud_cluster_closest,
single_color2, "cloud 2");

pcl::visualization::PointCloudColorHandlerCustom<CloudType>
single_color3(cloud_plane, 0, 0, 255);
viewer->addPointCloud<CloudType> (cloud_floor, single_color3,
"cloud 3");

if(planeWall)
{
    pcl::visualization::PointCloudColorHandlerCustom<CloudType>
single_color4(cloud_plane, 255, 255, 0);
    viewer->addPointCloud<CloudType> (cloud_wall,
single_color4, "cloud 4");
}

viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 2, "cloud 1");
viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 2, "cloud 2");
viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 2, "cloud 3");
if(planeWall) viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 2, "cloud 4");
// viewer->addCoordinateSystem (1.0);
viewer->spinOnce();
viewer->initCameraParameters ();

return (cloud_cluster);
sendCommand(direction, distance_to_target);

LABEL2:
double nearTarget = distance_to_target - m_dbDisTemp;
if(nearTarget < 130) goto LABEL6;
if(obstacle_flag)
{
  LABEL3:
    minpointTemp = minOnPath;
    maxpointTemp = maxOnPath;
    if(minpointTemp.z == 0 || maxpointTemp.z == 0)
    {
      Sleep(5);
      goto LABEL3;
    }
  escapeObstacle(maxpointTemp, minpointTemp, true);
  //escape obstacle with optimal angle
    sendCommand(direction, anpha_cmd);
    waitRobot();
    if(freePath)
    {
      LABEL4:
        safe_move();
        sendCommand(direction, safe_distance);
        waitRobot();
        goto LABEL1;
    }
    else
    {
      float deltaZ = minOnPath.z -
        sqrt(maxpointTemp.z*maxpointTemp.z +
        maxpointTemp.x*maxpointTemp.x)*cos(anpha);
      if(deltaZ > 0.55) goto LABEL4;
      else
      {
        escapeObstacle(maxpointTemp, minpointTemp,
false);//rotate invert with safe angle
        double anpha_new = anpha + anpha_;
        anpha_cmd = anpha_new*180.0f/PI;
        sendCommand(direction, anpha_cmd);
        waitRobot();
        if(freePath) {goto LABEL4;}
        else
        {
          LABEL5:
          double checksafe = 100*(minOnPath.z -
          0.55);
          sendCommand("F", checksafe);
          waitRobot();
          minpointTemp = minOnPath;
          maxpointTemp = maxOnPath;
          double temp0;
          escapeObstacle2(maxpointTemp, minpointTemp);
          if(direction == "R")
          {
            temp0 =
            100*sqrt(maxpointTemp.z*maxpointTemp.z + maxpointTemp.x*maxpointTemp.x);
            temp0 = temp0*sin(anpha - atan(maxpointTemp.x/maxpointTemp.z));
else if (direction == "L")
{
    temp0 = 100*sqrt(maxpointTemp.z*maxpointTemp.z + minpointTemp.x*minpointTemp.x);
    temp0 = temp0*sin(anpha - atan(minpointTemp.x/maxpointTemp.z));
}

sendCommand(direction, anpha_cmd);
waitRobot();

sendCommand("F", safe_distance2);
waitRobot();

CString Dir(_T(""));
double temp1 = rotateBack(angleTemp, m_dbAngleCurrent, Dir);

sendCommand(Dir, temp1);
waitRobot();

if (freePath)
{
    sendCommand("F", temp0 + 15);
    waitRobot();
    goto LABEL1;
}
else goto LABEL5;

SetDlgItemText(IDC_EDIT_NUM_AT_TARGET, "Error2");

ContrThread->SuspendThread();
}

else
{
    LABEL6:
    Sleep(5);
    if (doneRobot)
    {
        // At the target, rotate robot to 90 degree & suspend program
        double finalAngle = m_dbAngleCurrent - 90.0;
        if (finalAngle < 0) sendCommand("L", -finalAngle);
        else sendCommand("R", finalAngle);
        waitRobot();
        SetDlgItemText(IDC_EDIT_NUM_AT_TARGET, "Yes");
        ContrThread->SuspendThread();
    }
    else goto LABEL2;
}

void CRobotVisionDlg::KinectThread()
{

// init grabber for MS Kinect:
local_;
std::string device_id("");
pcl::Grabber* intf = new pcl::OpenNIGrabber (device_id,
pcl::OpenNIGrabber:: OpenNI_QVGA_30Hz,pcl::OpenNIGrabber::
OpenNI_QVGA_30Hz);
boost::function<void (const CloudConstPtr&)> cloud_cb =
boost::bind(&local::cloud_callback,&local_ ,_1);
boost::signals2::connection cloud_connection = intf->registerCallback
(cloud_cb);
intf->start ();

while (!viewer->wasStopped())
{
    if (local_.cloud_)
    {
        FPS_CALC ("drawing");
        local_.getLatestCloud();

        if(obstacle_flag) SetDlgItemText(IDC_EDIT_OBS, "Yes");
        else SetDlgItemText(IDC_EDIT_OBS, "No");

        std::stringstream ss;
        ss << numObs;
        SetDlgItemText(IDC_EDIT_NUM_OBS, ss.str().c_str());

        if(freePath) SetDlgItemText(IDC_EDIT_FREE_PATH, "Yes");
        else SetDlgItemText(IDC_EDIT_FREE_PATH, "No");
    }
}

intf->stop ();
cloud_connection.disconnect();

// CAboutDlg dialog used for App About
class CAboutDlg : public CDialogEx
{
public:
CAboutDlg();

// Dialog Data
enum { IDD = IDD_ABOUTBOX };

protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Implementation
protected:
DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()

// CRobotVisionDlg dialog

CRobotVisionDlg::CRobotVisionDlg(CWnd* pParent /*=NULL*/)
    : CDialogEx(CRobotVisionDlg::IDD, pParent)
    , m_strNamePort(_T(""))
    , m_strBaudRate(_T(""))
    , m_strTextSend(_T(""))
    , m_strTextReceive(_T(""))
    , m_strPortStatus(_T(""))
    , openPortActivate(false)
    , closePortActivate(false)
    , sendActivate(false)
    , activeProcess(false)
    , m_strTest(_T(""))
    , m_strXCurrent(_T(""))
    , m_strYCurrent(_T(""))
    , m_strAngleCurrent(_T(""))
    , m_strXTarget(_T(""))
    , m_strYTarget(_T(""))
    , m_dbXCurrent(0)
    , m_dbYCurrent(0)
    , m_dbAngleCurrent(0)
    , m_dbXTarget(0)
    , m_dbYTarget(0)
    , m_angle(0)
    , doneRobot(false)
    , m_dbDisTemp(0)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    m_strNamePort = "COM2";
    m_strBaudRate = "9600";
}

void CRobotVisionDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
    DDX_CBString(pDX, IDC_COMBO_NAME_PORT, m_strNamePort);
    DDX_CBString(pDX, IDC_COMBO_BAUDRATE, m_strBaudRate);
    DDX_Text(pDX, IDC_EDIT_TEXT_SEND, m_strTextSend);
    DDX_Text(pDX, IDC_EDIT_TEXT_RECEIVE, m_strTextReceive);
    DDX_Text(pDX, IDC_EDIT_STATUS_PORT, m_strPortStatus);
    DDX_Text(pDX, IDC_EDIT_X_CURRENT, m_strXCurrent);
    DDX_Text(pDX, IDC_EDIT_Y_CURRENT, m_strYCurrent);
    DDX_Text(pDX, IDC_EDIT_ANGLE_CURRENT, m_strAngleCurrent);
    DDX_Text(pDX, IDC_EDIT_X_TARGET, m_strXTarget);
    DDX_Text(pDX, IDC_EDIT_Y_TARGET, m_strYTarget);
    DDX_Control(pDX, IDC_SLIDER_TILT, m_sliderTilt);
    DDX_Slider(pDX, IDC_SLIDER_TILT, m_angle);
}

BEGIN_MESSAGE_MAP(CRobotVisionDlg, CDialogEx)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
END_MESSAGE_MAP()
ON_BN_CLICKED(IDC_BUTTON_OPEN_PORT, &CRobotVisionDlg::OnBnClickedButtonOpenPort)
ON_BN_CLICKED(IDC_BUTTON_CLOSE_PORT, &CRobotVisionDlg::OnBnClickedButtonClosePort)
ON_BN_CLICKED(IDC_BUTTON_EXIT, &CRobotVisionDlg::OnBnClickedButtonExit)
ON_BN_CLICKED(IDC_BUTTON_SEND_DATA, &CRobotVisionDlg::OnBnClickedButtonSendData)
ON_BN_CLICKED(IDC_BUTTON_RUN, &CRobotVisionDlg::OnBnClickedButtonRun)
ON_NOTIFY(NM_CUSTOMDRAW, IDC_SLIDER_TILT, &CRobotVisionDlg::OnNMCustomdrawSliderTilt)
ON_BN_CLICKED(IDC_BUTTON1, &CRobotVisionDlg::OnBnClickedButton1)
END_MESSAGE_MAP()

// CRobotVisionDlg message handlers

BOOL CRobotVisionDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();
    // Add "About..." menu item to system menu.
    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        BOOL bNameValid;
        CString strAboutMenu;
        bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
        ASSERT(bNameValid);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }
    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);       // Set big icon
    SetIcon(m_hIcon, FALSE);      // Set small icon
    // TODO: Add extra initialization here
    GetDlgItem(IDC_BUTTON_CLOSE_PORT)->EnableWindow(FALSE);
    serialProcess = (SerialThread*)AfxBeginThread(RUNTIME_CLASS(SerialThread), THREAD_PRIORITY_NORMAL, 0, CREATE_SUSPENDED);
    serialProcess->setOwner(this);
    m_sliderTilt.SetRange(CL_MOTOR_MIN_ANGLE, CL_MOTOR_MAX_ANGLE);
    m_sliderTilt.SetTicFreq(1000.0);
    m_sliderTilt.SetPos(CL_MOTOR_MIN_ANGLE);
    initMotorControl();
    return TRUE;  // return TRUE unless you set the focus to a control
}
void CRobotVisionDlg::OnSysCommand(UINT nID, LPARAM lParam)
{ 
    if ((nID & 0xFFF0) == IDM_ABOUTBOX) {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    } else {
        CDiaLogEx::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.
void CRobotVisionDlg::OnPaint()
{
    if (IsIconic()) {
        CPaintDC dc(this); // device context for painting
        SendMessage(WM_ICONERASEBKGND,
                    reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    } else {
        CDiaLogEx::OnPaint();
    }
}

// The system calls this function to obtain the cursor to display while the
// user drags
// the minimized window.
HCURSOR CRobotVisionDlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}

void CRobotVisionDlg::UpdateConfig(void)
{
    // constant parameter.
    configSerial_.ByteSize = 8;
    configSerial_.StopBits = ONESTOPBIT;
    configSerial_.Parity = NOPARITY;

    switch(atoi(m_strBaudRate))
    {
    case 110:
configSerial_.BaudRate = CBR_110;
break;
case 300:
    configSerial_.BaudRate = CBR_300;
    break;
case 600:
    configSerial_.BaudRate = CBR_600;
    break;
case 1200:
    configSerial_.BaudRate = CBR_1200;
    break;
case 2400:
    configSerial_.BaudRate = CBR_2400;
    break;
case 4800:
    configSerial_.BaudRate = CBR_4800;
    break;
case 9600:
    configSerial_.BaudRate = CBR_9600;
    break;
case 14400:
    configSerial_.BaudRate = CBR_14400;
    break;
case 19200:
    configSerial_.BaudRate = CBR_19200;
    break;
case 38400:
    configSerial_.BaudRate = CBR_38400;
    break;
case 56000:
    configSerial_.BaudRate = CBR_56000;
    break;
case 57600:
    configSerial_.BaudRate = CBR_57600;
    break;
case 115200:
    configSerial_.BaudRate = CBR_115200;
    break;
case 128000:
    configSerial_.BaudRate = CBR_128000;
    break;
case 256000:
    configSerial_.BaudRate = CBR_256000;
    break;
default:
    break;
}

void CRobotVisionDlg::OnBnClickedButtonOpenPort()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    UpdateConfig();
    openPortActivate = true;
    closePortActivate = false;
    activeProcess = TRUE;
    UpdateData(TRUE);
    serialProcess->ResumeThread();
    GetDlgItem(IDC_BUTTON_CLOSE_PORT)->EnableWindow(TRUE);
    GetDlgItem(IDC_BUTTON_OPEN_PORT)->EnableWindow(FALSE);
}
void CRobotVisionDlg::OnBnClickedButtonClosePort()
{
    // TODO: Add your control notification handler code here
    // Set signal of closing port serial communication.
    closePortActivate = true;
    openPortActivate = false;
    GetDlgItem(IDC_BUTTON_CLOSE_PORT)->EnableWindow(FALSE);
    GetDlgItem(IDC_BUTTON_OPEN_PORT)->EnableWindow(TRUE);
    UpdateData(FALSE);
}

void CRobotVisionDlg::OnBnClickedButtonExit()
{
    // TODO: Add your control notification handler code here
    // Set signal of closing port serial communication.
    stopMotorControl();
    serialProcess->SuspendThread();
    this->DestroyWindow();
}

void CRobotVisionDlg::OnBnClickedButtonSendData()
{
    // TODO: Add your control notification handler code here
    // Set signal to send data of serial communication.
    UpdateData(TRUE);
    sendActivate = true;
}

void CRobotVisionDlg::OnBnClickedButtonRun()
{
    UpdateData(TRUE);
    m_sliderTilt.SetPos(CL_MOTOR_MIN_ANGLE);
    Sleep(500);

    pass_.setFilterFieldName("z");
    pass_.setFilterLimits(0.5, 1.4);

    // grid_.setFilterFieldName("z");
    // grid_.setFilterLimits(0.5, 1.0);
    grid_.setLeafSize(0.04, 0.04, 0.04);

    seg_.setOptimizeCoefficients(true);
    seg_.setModelType(pcl::SACMODEL_PLANE);
    seg_.setMethodType(pcl::SAC_RANSAC);
    seg_.setMaxIterations(1000);
    seg_.setDistanceThreshold(0.035); //2cm
    // extract_.setNegative(true);

    ec.setClusterTolerance(0.05); // 3cm
    ec.setMinClusterSize(20);
    ec.setMaxClusterSize(1000);
    ec setSearchMethod(tree);
}

KinectThread();
void CRobotVisionDlg::OnNMCustomdrawSliderTilt(NMHDR *pNMHDR, LRESULT *pResult)
{
    LPNMCUSTOMDRAW pNMCD = reinterpret_cast<LPNMCUSTOMDRAW>(pNMHDR);
    // TODO: Add your control notification handler code here
    m_angle = m_sliderTilt.GetPos();
    setMotorAngle(m_angle);
    *pResult = 0;
}

void CRobotVisionDlg::OnBnClickedButton1()
{
    // TODO: Add your control notification handler code here
    //cvCreateCameraCapture(0);
    cvNamedWindow("Camera", CV_WINDOW_AUTOSIZE);
    Sleep(3000);
    //Controller thread Initiation
    m_dbXcurrent_Temp = 0;
    m_dbXCurrent = 0;
    m_dbYcurrent_Temp = 0;
    m_dbYCurrent = 0;
    m_dbAngleCurrent_Temp = PI/2;
    m_dbAngleCurrent = 90.0;
    direction_old = "";
    SetDlgItemText(IDC_EDIT_NUM_AT_TARGET, "No");
    UpdateData(TRUE);
    //updateStatus();
    m_dbXTarget = atof(m_strXTarget);
    m_dbYTarget = atof(m_strYTarget);
    if(bFirst)
    {
        ContrThread = AfxBeginThread(ControllerThread, this);
        bFirst = false;
    }
    else ContrThread->ResumeThread();
}

void CRobotVisionDlg::goTarget(void)
{
    updateStatus();
    double gamma = atan((m_dbYTarget - m_dbYcurrent_Temp)/(m_dbXTarget - m_dbXCurrent_Temp)); //assume Y is always positive
    if(gamma < 0)
    {
        anpha = PI - m_dbAngleCurrent_Temp + gamma;
        direction = "L";
    }
else if ((m_dbAngleCurrent_Temp >= (PI + gamma)) &&
(m_dbAngleCurrent_Temp < (2*PI + gamma)))
{
    anpha = m_dbAngleCurrent_Temp - (PI + gamma);
    direction = "R";
}
else if((m_dbAngleCurrent_Temp >= (2*PI + gamma)) &&
(m_dbAngleCurrent_Temp <= 2*PI))
{
    anpha = 3*PI - m_dbAngleCurrent_Temp + gamma;
    direction = "L";
}

else
{
    if((m_dbAngleCurrent_Temp >= 0) && (m_dbAngleCurrent_Temp <
    gamma))
    {
        anpha = gamma - m_dbAngleCurrent_Temp;
        direction = "L";
    }
    else if((m_dbAngleCurrent_Temp >= gamma) &&
    (m_dbAngleCurrent_Temp < (PI + gamma)))
    {
        anpha = m_dbAngleCurrent_Temp - gamma;
        direction = "R";
    }
    else if((m_dbAngleCurrent_Temp >= (PI + gamma)) &&
    (m_dbAngleCurrent_Temp <= 2*PI))
    {
        anpha = 2*PI - m_dbAngleCurrent_Temp + gamma;
        direction = "L";
    }
}

anpha_cmd = anpha*180.0f/PI;

distance_to_target = sqrt((m_dbXTarget -
m_dbXcurrent_Temp)*(m_dbXTarget - m_dbXcurrent_Temp)
+ (m_dbYTarget -
m_dbYcurrent_Temp)*(m_dbYTarget - m_dbYcurrent_Temp));

// ------------------------------------------------------------------------------------------

// Update robot's status (X_cur, Y_cur, Angle_cur, ...)
void CRobotVisionDlg::updateStatus()
{
    m_dbXcurrent_Temp = m_dbXCurrent;
    m_dbYcurrent_Temp = m_dbYCurrent;
    m_dbAngleCurrent_Temp = m_dbAngleCurrent*PI/180.0f;
}

// ------------------------------------------------------------------------------------------

// Send command to micro-controller through UART interface
void CRobotVisionDlg::sendCommand(CString dir, double val)
{
    UpdateData(TRUE);
    std::stringstream ss;
    size_t decimal_places(3); // need 3 digits decimal
ss << "*" << dir << setprecision( static_cast< int >( log10( val ) ) +
1 + decimal_places ) << val << "#";
m_strTextSend = ss.str().c_str();
sendActivate = true;
doneRobot = false;
UpdateData(FALSE);
}

//Calculate angle for the robot's safe rotation
void CRobotVisionDlg::escapeObstacle(CloudType& max_point, CloudType&
min_point, bool Optimal)
{
if(Optimal)
{
    if((max_point.x <= 0) && (max_point.x > - RobotWidth/2.0f))
    {
        anpha = acos(- max_point.x/RobotCross) -
acos(RobotWidth/RobotCross/2.0f);
        direction = "R";
    }
    else if ((min_point.x > 0) && (min_point.x < RobotWidth/2.0f))
    {
        anpha = acos(min_point.x/RobotCross) -
acos(RobotWidth/RobotCross/2.0f);
        direction = "L";
    }
    else if (max_point.x > 0 && min_point.x < 0)
    {
        if(max_point.x <= (- min_point.x))
        {
            anpha = asin(RobotWidth/RobotCross/2.0f) +
asin(max_point.x/RobotCross);
            direction = "R";
        }
        else
        {
            anpha = asin(RobotWidth/RobotCross/2.0f) + asin(-
min_point.x/RobotCross);
            direction = "L";
        }
    }
    else
    {
        anpha = 0;
        direction = "R";
    }
}
anpha_cmd = anpha*180.0f/PI;
//safe_distance = angle_current
else
{
    if((max_point.x < 0) && (max_point.x > - RobotWidth/2.0f))
    {
        anpha_ = atan(- min_point.x/min_point.z) +
asin(RobotWidth/RobotCross/2.0f);
        direction = "L";
    }
    else if ((min_point.x > 0) && (min_point.x < RobotWidth/2.0f))
    {
anpha_ = atan(max_point.x/min_point.z) +
    asin(RobotWidth/RobotCross/2.0f);
direction = "R";
}
  else if (max_point.x > 0 && min_point.x < 0)
  {
    if(max_point.x <= (- min_point.x))
    {
      anpha_ = asin(RobotWidth/RobotCross/2.0f) + atan(-
        min_point.x/min_point.z);
      direction = "L";
    }
    else
    {
      anpha_ = asin(RobotWidth/RobotCross/2.0f) +
        atan(max_point.x/min_point.z);
      direction = "R";
    }
  }
else
{
  anpha_ = 0;
}

//safe distance for robot to avoid obstacle completely
void CRobotVisionDlg::safe_move(void)
{
  double distanceTemp = 0;
  // updateStatus();
  if(direction == "R" || direction == "L")
  {
    if(direction == "R")
      distanceTemp = sqrt(maxpointTemp.z*maxpointTemp.z +
                      maxpointTemp.x*maxpointTemp.x)*cos(anpha_ -
                      atan(abs(maxpointTemp.x)/maxpointTemp.z)) + 0.15;
    else
      distanceTemp = sqrt(maxpointTemp.z*maxpointTemp.z +
                      maxpointTemp.x*maxpointTemp.x)*cos(anpha_ -
                      atan(abs(minpointTemp.x)/maxpointTemp.z)) + 0.15;
    direction = "F";
    safe_distance = 100.0f*distanceTemp;
  }
  //updateStatus();
}

//wait until robot completes command from computer
void CRobotVisionDlg::waitRobot(void)
{
  while(!doneRobot)
  {
    Sleep(5);
  }
  updateStatus();
}

//avoid obstacle 2nd time
void CRobotVisionDlg::escapeObstacle2(CloudType& max_point, CloudType& min_point)
double disTemp = 0;
if(direction == "R")
{
    if(max_point.x >= 0) anpha = asin(RobotWidth/RobotCross/2) +
        atan(max_point.x/min_point.z);
    else anpha = atan(min_point.z/abs(max_point.x)) -
        acos(RobotWidth/RobotCross/2);
    disTemp = sqrt(max_point.z*max_point.z +
        max_point.x*max_point.x)*cos(anpha + atan(abs(max_point.x)/max_point.z)) +
        0.15;
}
else if(direction == "L")
{
    if(min_point.x >= 0) anpha = atan(min_point.z/min_point.x) -
        acos(RobotWidth/RobotCross/2);
    else anpha = asin(RobotWidth/RobotCross/2) +
        atan(abs(min_point.x)/min_point.z);
    disTemp = sqrt(max_point.z*max_point.z +
        max_point.x*max_point.x)*cos(anpha - atan(abs(min_point.x)/max_point.z)) +
        0.15;
}
anpha_cmd = anpha*180/PI;
safe_distance2 = 100*disTemp;

//rotate robot back to previous angle
double CRobotVisionDlg::rotateBack(const double angleTurn, const double angleCur, CString &Dir)
{
    if(angleCur >= angleTurn && angleCur < (angleTurn + 180))
    {
        Dir = "R";
        return (angleCur - angleTurn);
    }
    else if(angleCur >= (angleTurn + 180) && angleCur <= 360)
    {
        Dir = "L";
        return (360 - angleCur + angleTurn);
    }
    else if(angleCur >= 0 && angleCur <= angleTurn)
    {
        Dir = "L";
        return (angleTurn - angleCur);
    }
}
#include <18F4550.h>
#include <STDDEF.H>
#FUSES NOWDT, HS, NOPUT, NOBROWNOUT, NOLVP, NOCPD, NOWRT, NOSLEEP
#use delay(clock=12000000)
#use rs232 (baud=9600 , parity=n , xmit=pin_C6 , rcv=pin_C7 ,bits=8)
#byte PORTA =  0xf80
#byte PORTB =  0xf81
#byte PORTC =  0xf82
#byte PORTD =  0xf83
#byte PORTE =  0xf84

#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <ctype.h>

#BIT DIR_RIGHT = PORTE.0  //0: forward, 1: backward
#BIT DIR_LEFT  = PORTE.1
#BIT LED = PORTB.7

#define time_step 50535  // 0.005 second  5535 //
#define PERIMETER_LEFT 31.95
#define PERIMETER_RIGHT 31.9
#define RADIUS 20.25
#define THRES_RIGHT 55 //Choose suitable Threshold to synchronize
#define THRES_LEFT 65  //two wheels speed
#define ERR 4
//Global variants
signed int32 num_Pulse_Right = 0, count_left = 0;
signed int32 num_Pulse_Left = 0;
float disTemp = 0;
int1 done_Timer1 = 0;

char ss;
char data_receive[50] = "";
int8 index = 0;

signed int16 pw_duty_Left = 0, pw_duty_Right = 0;
signed int32 position_Right = 0, position_Left = 0, position_set_Left = 0,
position_set_Right = 0;

float Kp_r, Kd_r, Ki_r;
float Kp_l, Kd_l, Ki_l;
signed int32 e_sum_r, e_del_r, e1_r, e2_r;
signed int32 e_sum_l, e_del_l, e1_l, e2_l;

float value_cmd, anpha = 0, angle_current = 90.0, distance_move = 0;
float x_cur = 0, y_cur = 0, x_cur_old = 0, y_cur_old = 0;
char direction_cmd, direction_cmd_old;
int1 RobotFlag = 0, RunFlag = 0, pid_Left_Flag = 0, pid_Right_Flag = 0,
donePID = 0, targetFlag = 0;

void pid_Robot();
void pid_Right();
void pid_Left();
void init_pid_Right();
void init_pid_Left();
void moveRobot(char direction, float value);
void xy_calculator(float distance);
#INT_RDA

void RDA_isr()
{
char value[20];
ss = getc();
if(ss == '*') //Beginning symbol
{
    index = 0;
    *data_receive = ""
    RunFlag = 1;
}
else if(ss == '#') //Terminal symbol
{
    data_receive[index++]="0';
    direction_cmd = data_receive[0];
    if ((direction_cmd != 'T') && direction_cmd_old == 'T' && (!donePID))
        targetFlag = 1;
    direction_cmd_old = directio
    strncpy(value, data_receive + 1, strlen(data_receive) - 1);
    value_cmd = atof(value);
    RobotFlag = 1;
}
else data_receive[index++] = ss;
}

#INT_EXT1 //Count pulses in Right wheel
void EXT1_isr()
{
    disable_interrupts(int_ext1);
    if(input(PIN_D1) == 1)
        num_Pulse_Right++;
else num_Pulse_Right--; 
    position_Right = num_Pulse_Right;//*360/300;

enable_interrupts(int_ext1) ;
}

#INT_EXT2 //Count pulses in Right wheel
void EXT2_isr()
{
    disable_interrupts(int_ext2);
    if(input(PIN_D2) == 1)
    {
        num_Pulse_Left++;
        if(direction_cmd == 'T') count_left++;
    }
    else
    {
        num_Pulse_Left--;
        if(direction_cmd == 'T') count_left--;
    }

    position_Left = num_Pulse_Left;//*360/300;

enable_interrupts(int_ext2) ;
}

#INT_TIMER1 //Count number of pulses per time unit
void TIMER1_isr() {
    //printf(lcd_putchar "\ncount \%ld", ++count);
    // disable_interrupts(INT_TIMER1);
    done_Timer1 = 1;
```c
set_timer1(time_step);
// enable_interrupts(INT_TIMER1);

void main()
{
    set_tris_a(0);
    set_tris_b(0xFF);
    set_tris_e(0);
    set_tris_c(0x80);
    set_tris_d(0x0F);
    DIR_RIGHT = 0; // right wheel forward
    DIR_LEFT = 0;
    /*
       lcd_init();
       lcd_putchar("Test");
       lcd_gotoxy(1,2);
       lcd_putchar("Dong Co");
       delay_ms(2000); */

    setup_adc_ports(NO_ANALOGS);
    setup_timer_1(T1_INTERNAL | T1_DIV_BY_1);
    set_timer1(time_step);
    setup_timer_2(T2_DIV_BY_4,255,1); // f_pwm = 2.9 Khz, T_pwm = 4.(PR2+1).Tosc.Pre-scale
    output_low(PIN_C1); // Set CCP2 output low
    output_low(PIN_C2); // Set CCP1 output low

    setup_ccp1(CCP_PWM); // PWM1
    setup_ccp2(CCP_PWM); // PWM2
    set_pwm1_duty(0);    // PIN_C2, Right motor
    set_pwm2_duty(0);    // PIN_C1, Left motor
```
enable_interrupts(INT_RDA);
enable_interrupts(int_ext1);
enable_interrupts(int_ext2);
ext_int_edge(1, H_TO_L);
ext_int_edge(2, H_TO_L);
enable_interrupts(INT_TIMER1);
enable_interrupts(GLOBAL);

printf("X%.3fY%.3fA%.3fN", x_cur, y_cur, angle_current);
while(true)
{
  if(done_Timer1)
  {
    moveRobot(direction_cmd, value_cmd);
    pid_Robot();
    done_Timer1 = 0;
    //LED=1;
  }
}

void xy_calculator(float distance)
{
  x_cur = x_cur_old + distance*cos(angle_current*PI/180.0);
  y_cur = y_cur_old + distance*sin(angle_current*PI/180.0);
  x_cur_old = x_cur;
  y_cur_old = y_cur;
}

void pid_Robot()
{
  if(targetFlag)
  {

distance_move = PERIMETER_LEFT*count_left/300.0;
xy_calculator(distance_move);
count_left = 0;
distance_move = 0;
printf("X%.3fY%.3fA%.3fT", x_cur, y_cur, angle_current);
targetFlag = 0;
}
if(direction_cmd == 'T') // send distance continuously
{
    disTemp = PERIMETER_LEFT*count_left/300.0;
    printf("X%.3fD", disTemp);
}

pid_Right();

pid_Left();

if (!pid_Left_Flag) && (!pid_Right_Flag) && (!donePID))
{
    if(direction_cmd == 'R')
    {
        angle_current -= value_cmd;
        direction_cmd = "";
    }
    else if(direction_cmd == 'L')
    {
        angle_current += value_cmd;
        direction_cmd = "";
    }
    else if (direction_cmd == 'F' || direction_cmd == 'T')
    {
        xy_calculator(value_cmd);
        direction_cmd = "";
    }
else if (direction_cmd == 'B')
{
    xy_calculator(-value_cmd);
    direction_cmd = "";
}

if (angle_current >= 360.0) angle_current -= 360.0;
if (angle_current < 0 ) angle_current += 360.0;

printf("X%.3fY%.3fA%.3fN", x_cur, y_cur, angle_current);
donePID = 1;
}
}

void init_pid_Right()
{
    Kp_r = 2.5;
    ki_r = 0.0000001;
    kd_r = 2.0;
    e_sum_r = 0;
    e_del_r = 0;
    e1_r = 0;
    e2_r = 0;
    pw_duty_Right = 0;
}

void init_pid_Left()
{
    Kp_l = 2.5;  //0.9
    ki_l = 0.0000001;
    kd_l =2.0;      //1
    e_sum_l = 0;
    e_del_l = 0;
}
e1_l = 0;
e2_l = 0;
pw_duty_Left = 0;
}

void pid_Right()
{
    signed int16 temp_kp = 0;
signed int16 temp_ki = 0;
signed int16 temp_kd = 0;
signed int16 pw_duty_Right_Temp = 0;

e2_r = position_set_Right - position_Right;
e_sum_r += e2_r;
e_del_r = e2_r - e1_r;
e1_r = e2_r;

temp_kp = (signed int16)((float)kp_r*e2_r);
temp_ki = (signed int16)((float)ki_r*e_sum_r);
temp_kd = (signed int16)((float)kd_r*e_del_r);

pw_duty_Right_Temp = (signed int16)( temp_kp + temp_ki + temp_kd);

    if (pw_duty_Right_Temp > THRES_RIGHT) pw_duty_Right_Temp =
THRES_RIGHT;
    if (pw_duty_Right_Temp < -THRES_RIGHT) pw_duty_Right_Temp = -
THRES_RIGHT;

    if (pw_duty_Right_Temp < 0)
    {
        DIR_RIGHT = 1;
pw_duty_Right = - pw_duty_Right_Temp;
    }
else
{
    DIR_RIGHT = 0;
    pw_duty_Right = pw_duty_Right_Temp;
}
set_pwm1_duty((int16)pw_duty_Right);
if(e2_r > -ERR && e2_r < ERR )
{
    pid_Right_Flag = 0;
}
else
{
    pid_Right_Flag = 1;
}
}

void pid_Left()
{
    signed int16 temp_kp = 0;
    signed int16 temp_ki = 0;
    signed int16 temp_kd = 0;
    signed int16 pw_duty_Left_Temp = 0;
    e2_l = position_set_Left - position_Left;
    e_sum_l += e2_l;
    e_del_l = e2_l - e1_l;
    e1_l = e2_l;
    temp_kp = (signed int16)((float)kp_l*e2_l);
    temp_ki = (signed int16)((float)ki_l*e_sum_l);
    temp_kd = (signed int16)((float)kd_l*e_del_l);
pw_duty_Left_Temp = (signed int16)(temp_kp + temp_ki + temp_kd);
if (pw_duty_Left_Temp > THRES_LEFT) pw_duty_Left_Temp = THRES_LEFT;
if (pw_duty_Left_Temp < -THRES_LEFT) pw_duty_Left_Temp = -THRES_LEFT;

if (pw_duty_Left_Temp < 0)
{
    DIR_LEFT = 1;
    pw_duty_Left = -pw_duty_Left_Temp;
}
else
{
    DIR_LEFT = 0;
    pw_duty_Left = pw_duty_Left_Temp;
}
set_pwm2_duty((int16)pw_duty_Left);
if(e2_l > -ERR && e2_l < ERR)
{
    pid_Left_Flag = 0;
}
else
{
    pid_Left_Flag = 1;
}
}

void moveRobot(char direction, float value)
{
    if(RobotFlag)
    {
        init_pid_Right();
    }
init_pid_Left();
pid_Left_Flag = 1;
pid_Right_Flag = 1;
donePID = 0;

num_Pulse_Right = 0;
um_Pulse_Left = 0;
position_Right = 0;
position_Left = 0;
RobotFlag = 0;
switch(direction)
{
    case 'T':
        position_set_Left  = (unsigned int32)(300.0*value/PERIMETER_LEFT);
        position_set_Right = (unsigned int32)(300.0*value/PERIMETER_RIGHT);
        distance_move = 0;
        count_left = 0;
        break;
    case 'F':
        position_set_Left  = (unsigned int32)(300.0*value/PERIMETER_LEFT);
        position_set_Right = (unsigned int32)(300.0*value/PERIMETER_RIGHT);
        break;
    case 'B':
        position_set_Left  = (unsigned int32)(-300.0*value/PERIMETER_LEFT);
        position_set_Right = (unsigned int32)(-300.0*value/PERIMETER_RIGHT);
        break;
    case 'R':
anpha = - value*PI/180.0;
position_set_Left = (unsigned int32)(-300.0*(anpha*RADIUS)/PERIMETER_LEFT);
position_set_Right = (unsigned int32)(300.0*(anpha*RADIUS)/PERIMETER_RIGHT);
break;
case 'L':
anpha = value*PI/180.0;
position_set_Left = (unsigned int32)(-300.0*(anpha*RADIUS)/PERIMETER_LEFT);
position_set_Right = (unsigned int32)(300.0*(anpha*RADIUS)/PERIMETER_RIGHT);
break;
default: break;
APPENDIX 5: Gantt chart FYP1 and FYP2

<table>
<thead>
<tr>
<th>FYP 1 task</th>
<th>Week</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FYP 1 task</th>
<th>Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>FYP Briefing Session</td>
<td></td>
</tr>
<tr>
<td>Decide and analyze the topic that want to do</td>
<td></td>
</tr>
<tr>
<td>Literature review</td>
<td></td>
</tr>
<tr>
<td>Design the mobile robot</td>
<td></td>
</tr>
<tr>
<td>Study on C++, ROS, Openni and openCV</td>
<td></td>
</tr>
<tr>
<td>Research about the suitable components and part that use in this system</td>
<td></td>
</tr>
<tr>
<td>FYP 1 proposal submission</td>
<td></td>
</tr>
<tr>
<td>Preparation for Seminar FYP 1</td>
<td></td>
</tr>
<tr>
<td>Seminar FYP 1</td>
<td></td>
</tr>
<tr>
<td>Preparation for FYP 1 report</td>
<td></td>
</tr>
<tr>
<td>FYP 1 report submission</td>
<td></td>
</tr>
</tbody>
</table>

Gantt chart Semester 1
<table>
<thead>
<tr>
<th>FYP 2 task</th>
<th>Week</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>mobile robot construction</td>
<td>2</td>
</tr>
<tr>
<td>Programming</td>
<td>3</td>
</tr>
<tr>
<td>Testing</td>
<td>4</td>
</tr>
<tr>
<td>Debug and troubleshooting</td>
<td>5</td>
</tr>
<tr>
<td>FYP 2 Seminar Preparation</td>
<td>6</td>
</tr>
<tr>
<td>FYP 2 seminar and demonstration</td>
<td>7</td>
</tr>
<tr>
<td>Writing thesis</td>
<td>8</td>
</tr>
<tr>
<td>Final thesis draft submission</td>
<td>9</td>
</tr>
<tr>
<td>Thesis submission (hardcover and softcopy)</td>
<td>10</td>
</tr>
</tbody>
</table>