UNIVERSITY OF SCIENCE ADVANCED
PROGRAM IN COMPUTER SCIENCE

PHAN HONG DUC - DO HOANG TUNG

CONTROLLING ROBOT USING PATH FOLLOWING AND KINECT

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

HO CHI MINH CITY, 2014

UNIVERSITY OF SCIENCE ADVANCED
PROGRAM IN COMPUTER SCIENCE

PHAN HONG DUC 0712732
DO HOANG TUNG 0712723

CONTROLLING ROBOT USING PATH FOLLOWING AND KINECT

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

THESIS ADVISOR
DR. NGUYEN TUAN NAM

HO CHI MINH CITY, 2014

ACKNOWLEDGMENTS

We want to express our gratitude to **Dr. Nguyễn Tuấn Nam**, who has helped and taught us many things since the first day we began to do this thesis.

Next, we want to thank **Sơn**, our special friend in University of Technology, for his technical supports in robot field.

We are also very grateful that we are the students of the APCS program. We have learned a lot of new knowledge and technologies, which is very necessary to do this thesis.

Finally, we want to thanks our families. They are always the great source of energy, they give us the strength to go on whenever we feel weak, encourage us to go on until we finish this thesis.

University of Science
Advanced Program in Computer Science

Thesis Proposal

Thesis's title: Controlling robot using path following and Kinect

Advisor: Dr. Nguyen Tuan Nam

Duration: From 1/6/2013 to 31/12/2013

Students: Do Hoang Tung – 0712723 and Phan Hong Duc – 0712732

Type: Technology with demo application

Contents:

1. **Main objectives:**
   - Build a robot that can follow line, stop when it meets the junction, the end of line or receive command "stop" from human.
   - The robot will receive orders from human through Kinect, based on gesture recognition.

2. **Details:**
   - Research and analyze some algorithms and models about path following robot.
   - Research and study gesture recognition, including its concept, history, terminologies, algorithms and real-world applications.
   - Study Artificial Intelligence
   - Study some mechanism of the path following robot, to have enough knowledge build an application.

3. **Demo application:**
   - A robot that can follow line, using P.I.D algorithm. A proportional-integral-derivative controller (PID Controller) is a common technique used to control a wide variety of machinery including vehicles, robots and even rockets.
   - Kinect will track human actions and give orders to the robot, using skeletal algorithm. Skeleton tracking is the processing of depth image data to establish the positions of various skeleton joints on a human form. For example, skeleton tracking determines where a user's head, hands, and center of mass are. When the positions of these joints change to some location, they can give orders to the robot.

Research timeline:

From June 1st 2013 to June 30th 2013: review some information about path

following and Kinect in order to write thesis proposal.

From July 1st 2013 to August 31st 2013: Study Artificial intelligence, gesture recognition and PID controller.

From September 1st 2013 to October 31st 2013: Begin to write program, using C and C#.

From November 1st 2013 to December 2013: Begin to experiment, take results.

From December 1st 2013 to January 2014: write thesis, prepared to present.

Approved by the advisor(s)	**Ho Chi Minh city, 25/12/2013**

TABLE OF CONTENTS

LIST OF FIGURE

LIST OF TABLES

ABSTRACT

This thesis concerns the problem of controlling robot. Robots have replaced humans in the assistance of performing many repetitive and dangerous tasks which humans prefer not to do, or are unable to do due to size limitations, or even those such as in outer space or at the bottom of the sea where humans could not survive the extreme environments. With the concerns about increasing use of robots and their role in society, it is very important to research how to control robots efficiently. For that purpose, we propose a hybrid approach that combines path following and Kinect to control robot. We use Kinect to give orders to the robot such as move forward, backward, turn left, turn right. The robot is forced to move on the lines that we set up at the beginning. The content consists of the following:

Chapter 1

Problem statement

1.1 Introduction

The task of path following system consists of getting the robot to follow a specific path, which is defined as line follower. The line may have difference appearances, which can be a black line on a white surface or vice-versa. Let's start with the basic layout of the robot that would be suitable for path following. At the bellow is the simplified drawing from the top view of a robot with all the details we need.

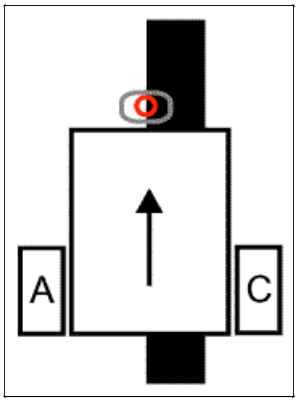

Figure 1.1. Basic layout of the path following robot.

Robot is a robot direction with two different engines, each connected to one of the wheels A and C. The robot has a light sensor mounted on the front straight down just to see what it is, but the carpet (floor, ground, anything on the robot). Red circles represent quite a small place on the mat that light sensor can actually "see". The rest of the robot is a large rectangle with an arrow, which shows the normal direction of travel.

Our goal is to get the robot to follow the black line fat until it meets the junction or get command "stop". At the fork, the robot will stop and wait for

commands from humans. Path following is a basic robot behavior and is often one of the first things people learn. A mobile device can follow a line showing all the characteristics of a real robot. It uses sensors to gather information about the world around it and change its behavior depending on that information.

The robot will continue to follow the road if no new commands from humans or do not meet the junction. Kinect plays an important role in making the robot commands. It will track and recognize human gestures, and ordered directly to the robot based on human actions.

1.2 Project scope

There are three main scopes for this project:

1. Mechanical: the hardware part (mobile robot) will be constructed as mentioned in the previous section to test Kinect and path following system.
2. Electronics: Main board circuit with motor control module to receive data from computer to control robot movement.
3. Software: Using Mircrosoft Visual Studio 2010 and Keil uVision4 based on C and C++.

1.3 Proposed method

There are two tasks need to be done, each of them is coded based on an algorithm.

For path following, we use a PID controller algorithm. A proportional-integral-derivative controller (PID Controller) is a common technique used to control a wide variety of machinery including vehicles, robots and even rockets. A PID controller calculates an "error" value as the difference between a measured process variable and a desired setpoint. The controller attempts to minimize the error by adjusting the process control inputs.

Kinect tracks human actions and gives orders to the robot, so basically, the algorithm is all about gesture recognition. There are several approaches for interpreting a gesture, but in this thesis, we will focus on skeletal-based tracking algorithm. Skeleton tracking is the processing of depth image data to establish the positions of various skeleton joints on a human form. For example, skeleton tracking determines where a user's head, hands, and center of mass are. When the positions of these joints change to some location, they can give orders to the robot.

Chapter 2

Gesture Recognition

2.1 Definition

Gesture recognition is a topic in computer science and technology language.The goal is to interpret human gestures via mathematical algorithms . Gestures can originate from any moving body or the State but typically stem from the face or hand . Focus in this field include emotion recognition from the recognition of facial gestures and hand . Many methods have been made using cameras and computer vision algorithms to interpret sign language . However, the identification and recognition of posture , gait , proxemics , human behavior , and is also the subject of gesture recognition techniques . Gesture recognition can be seen as a way for computers to begin to understand human body language , thus building a bridge between the rich and the more people use plain text interface or even press GUI (graphical user interface) , which is still limited to keyboard and mouse input .

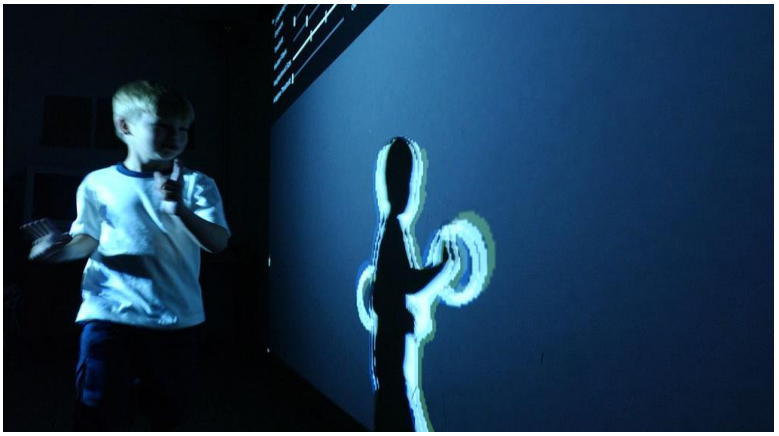

Figure 2.1. A child being sensed by a simple gesture recognition algorithm detecting hand location and movement.

Gesture recognition enables humans to interface with the machine (HMI) and interact naturally without any mechanical device. Using the concept of gesture recognition, perhaps just a finger across the screen to move the cursor. This would potentially make conventional input devices such as mouse, keyboard and touch screen is not necessary.

The ability to track a person's movements and determine what gestures they perform can be achieved through various tools. Although there are a large number of studies carried out in gesture recognition based on image / video, there are some changes in the tools and environments used between implementations. In this thesis, we will explain the Kinect sensor, which is one type of depth perception cameras.

2.2 Kinect sensor

Figure 2.2. Kinect Sensor

Kinect Sensor RGB camera and a depth sensor 3D runs on proprietary software provides motion capture in 3D space. Depth sensor consists of a combination of an infrared laser projector with a monochrome CMOS sensor that collects data 3D video in low-light conditions surrounding environment. Kinect software can automatically calibrate the sensor according

to the physical movement of people, filtering out the presence of other obstacles.

2.2.1 Kinect main components

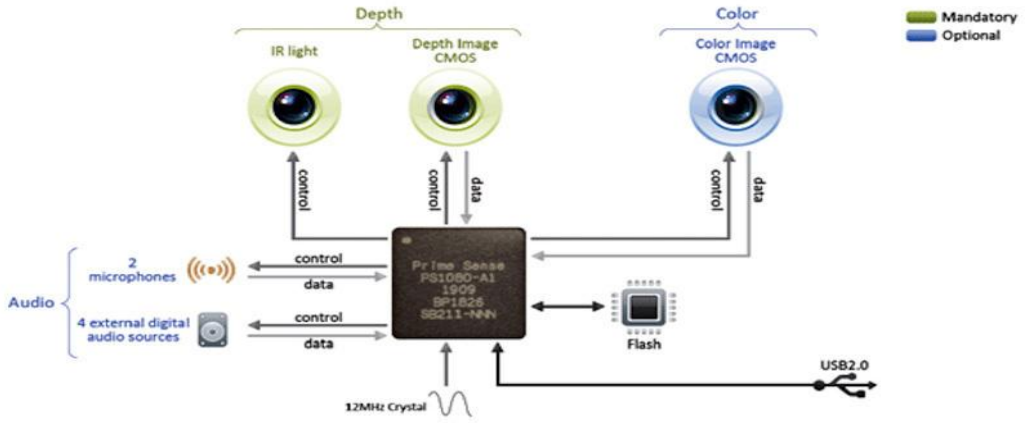

Figure 2.3. Kinect Main Component

Kinect include RGB camera, depth sensor (3D sensor depth), the microphone (Multi-array) and elevation angle motor control (motor 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 2.3, 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.

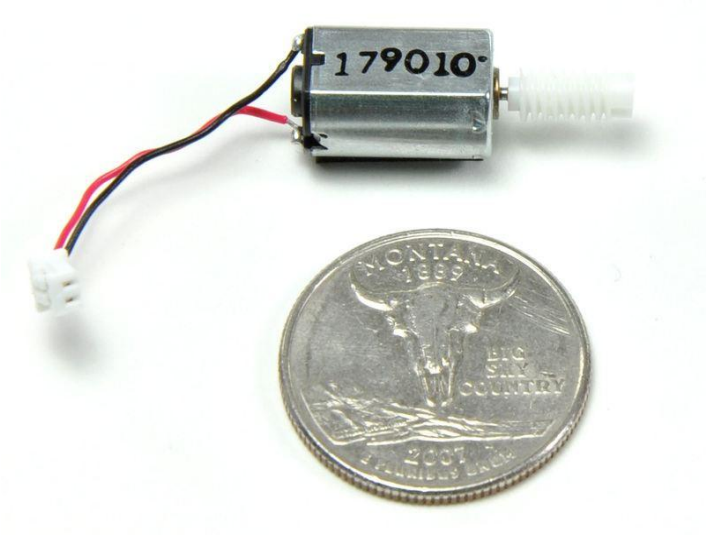

Figure 2.4: Kinect Tilt-Motor

2.2.2 Depth perception

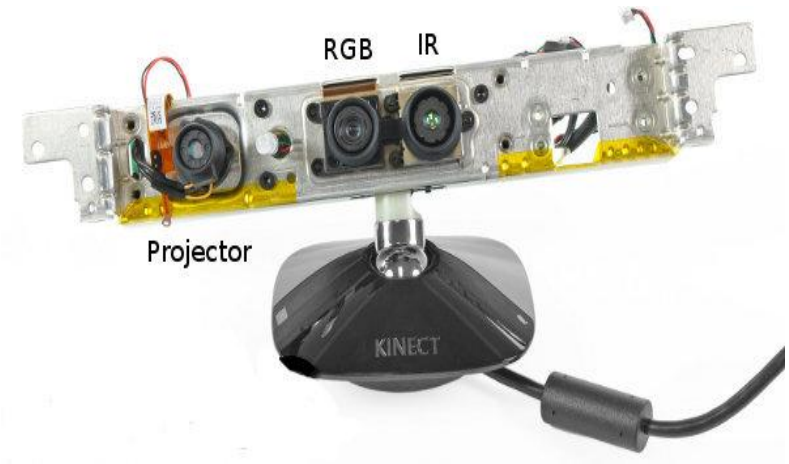

Figure 2.5. Inside Kinect: RGB, IR camera and IR projector

Pair of infrared sensors and infrared camera projector will work together to create an image of the light high-value encryption technology from PrimeSense depth .

Different techniques are used alongside Stereo Camera with a built camera to create depth maps , or engineering . TOF to determine the distance to estimate the travel time of light off and on in space , light coding techniques use an infrared projector combined with a continuous infrared cameras to calculate approximately way . The calculation is done inside the Kinect system on chip SoC or the PrimeSense PS1080 .

This new technology was supposed to meet a more accurate , cheaper to use in indoor environments . The projector will project a beam of infrared light , creating bright point , gathering place of the emitted light is fixed . This spot light generated by a light source is passed through a diffraction grating . A set of bright spots were taken by infrared cameras , through special algorithms in the PS1080 SoC integration for depth map . The nature of the technical solution is calculated based on the geometric relationship between the camera and infrared sensors

2.2.3 Library support

Kinect has been developed by many software developers not only develop games for Xbox array, but the array of image processing applications in medicine, robotics, mapping and more. There are many libraries written for Kinect Kinect. Below is a list of libraries used to develop image processing from Kinect sensor.

- **Laboratories Kinect Code**

Code Laboratories (CL) is a software developer specializing in the support, development of mining features of the image processing device. CL provides users control the basic features Kinect camera, and engine sound.

- **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 codes 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.

- **Point Cloud Library**

PCL is a library providing support for Point Cloud for image processing in 3D space. This library was built with many filtering algorithm (filters), surface restoration (at the surface), partition (segment), estimates the object features (feature estimate). PCL can be used on multiple platforms like Linux, Mac, Windows and Android. To simplify development, PCL library is divided into many small and can be compiled separately. PCL can say is a combination of many small modules. These modules do library also performs the function of individual packets before closing the PCL.

- **Microsoft Kinect SDK beta**

Kinect SDK beta, which was first introduced by Microsoft on June 16, 2011, is a powerful tool for developers. It allows developers to access all the capabilities of Kinect. The only downside is that it's Kinect SKD beta only supports Microsoft visual studio, especially C, C + and Visual Basic. However, one of the many highlights of its functions is to monitor bone, which makes Kinect SKD beta best choice for this thesis.

2.3 Algorithms

Gesture recognition interface (HMI) and interact naturally without device. Perhaps just a finger across the screen to move the cursor. This would potentially make conventional input devices such as mouse, keyboard and touch screen is not necessary.

2.3.1 3D model-based algorithm

Approach a 3D model can be built using models or bone, or even a combination of the two. Integrated approach widely used in industry and computer animation for the purposes of computer vision. These models are often created complex 3D surfaces, such as NURBS or polygon meshes.

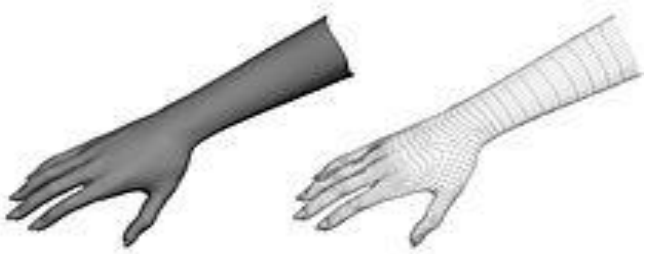

Figure 2.6. A real hand (left) is interpreted as a collection of vertices and lines in the 3D mesh version (right), and the software uses their relative position and interaction in order to infer the gesture.

The limitation of this method is that it is very computationally intensive, and the direct analysis system is still being developed. For the moment, a more interesting approach would be the object map primitive simple organ of the most important (example, cylinders for the arms and neck, demand for the head) and analyze how these interact with each other. Moreover, a number of abstract structures like super quadrics and generalized cylinders may be better suited for approximate body parts. Very interesting about this approach is that the parameters for these objects is

quite simple. In order to better model the relationship between these, we use the limited and decentralized among our subjects.

2.3.2 Appearance-based algorithm

These models do not use a spatial representation of the body, because they derive parameters directly from the image or video using a sample database. Some are based on the 2D pattern distortion of parts of the human body, especially the hands. Sample deformation is the set of points on the outline of an object, such as the use of interpolation nodes to approximate object outline. One of the simplest interpolation function is linear, which implements an average shape from the point, change parameters and deformators outside perspective. The template-based models are used primarily for monitoring hand, but can also be used to classify simple gestures.

Figure 2.7. These binary silhouette (left) or contour(right) images represent typical input for appearance-based algorithms. They are compared with different hand templates and if they match, the correspondent gesture is inferred.

A second approach gesture detection using appearance-based models using image sequences as gesture templates. The parameters for this method is one of two self-image, or certain features are derived from. Most of the time, only one (monoscopic) or two (stereoscopic) views are used.

2.3.3 Skeletal-based algorithm

Instead of using deep processing of 3D models and deal with lots of parameters, one can only use a simplified version of the joint angle parameters along with segment lengths. This is known as a representative body of fthe bone, where a virtual human skeleton is computed and parts of the body are mapped to certain segments. The analysis here is done by using the position and orientation of these segments and the relationship between each one of them (for example that is the angle between the joint and the relative positions or orientations).

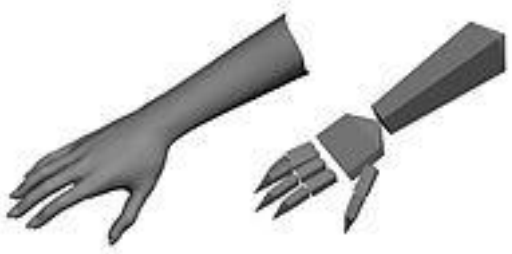

Figure 2.8. The skeletal version (right) is effectively modeling the hand (left). This has fewer parameters than the volumetric version and it's easier to compute, making it suitable for real-time gesture analysis systems.

Due to some advantages of the skeletal-based tracking, we will focus on this algorithm, which will be explained further in the next chapter.

Chapter 3

Skeletal Tracking

The raw data produced by deep Kinect has limited use . To build truly interactive experience , fun and memorable with Kinect , we need more information beyond just the depth of each pixel . This is where monitoring comes in. Skeleton skeleton tracking is processed image data to establish deep position of the various joints on a human form . For example , identifying the skeleton track where the user's head , hands , and center of mass . Subscribe skeleton provides the values X , Y , and Z for each skeleton . Subscribe skeleton and depth image analysis using complex algorithms using matrix transformation , machine learning , and other means to compute the skeleton points .

The basic principle of tracking skeletons learned here to create a platform for gesture recognition part of the thesis . After completing this chapter , we will cover all components of the Kinect for Windows SDK Kinect camera to deal with (Kinect’s cameras) .

3.1 How skeleton tracking works

Sensor Kinect depth data returns the data from which we can easily identify the pixels representing the players. Subscribe skeleton is not just about tracking joint information by reading the play, rather, it tracks the complete body movement. Real time people recognized as causing difficulties and challenges due to different body postures (consider a single body part can move in different directions and thousands of ways), size (human size change), clothing (dresses may vary from user to use), height (the height of the person can be high, low, medium), and so on..

To overcome such problems and to track the various joints regardless cause the body , Kinect uses a drawing pipelinewhere it fits to the data (raw data from the sensor depth) with data training samples . Human pose recognition algorithm using some basic character models that vary with different heights , sizes , clothes , and a number of other factors . The school has computer data collected from the base characters with different types of posture , hairstyle and clothing , and in different rotations and perspective . The data studied computer is labeled with body parts and personal fit the data to determine the depth to which part of the body it belongs to . The rendering pipeline processing data in a few steps to track human body parts from depth data.

Kinect sensor can determine the scope of a player pixels from the depth data. In the first step of the process piping drawings, the sensors identify the object the human body, which is nothing but the raw data depth similar to an object captured by the sensor. In the absence of any other logic, sensors will not know if this is a human body or something else. The following image shows what a human body just as it is represented with depth data; sensor recognizes it as a large object:

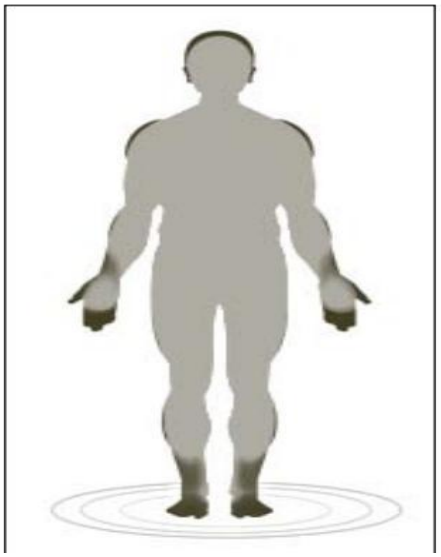

Figure 3.1. Raw depth data.

To start recognizing a human body, the sensor starts to suit each individual pixel of the depth data to the computer data learned. This game is done in the sensor with very high speed processing. The data every personal computer has been labeled learning and have some relevant value to fit the data. Consistent with this is completely based on the possibility that the data to match the computer data learned.

The next step immediately recognized position as labeled body parts by creating segments. Creating this segment is done by combining data similar could happen. Kinect uses a tree structure training (known as a decision tree) to fit the data for a specific type of human body. This tree is called a decision to Forrest.

All nodes in the tree is the data model different characters labeled with the name of a body part. Finally, all the pixels go through the tree data to match the body parts. The process of completing the combined data running over and over. Whenever there is a matched data, sensor started marking them and began creating body parts, as shown in the figure below:

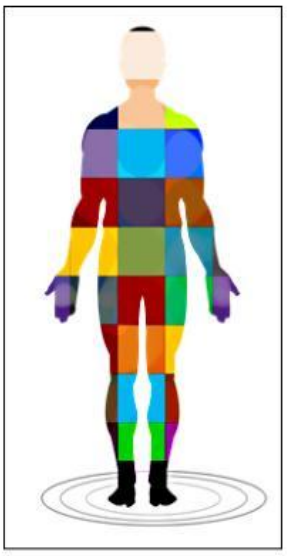

Figure 3.2. Inferred body segments.

Once the different body parts are identified, the sensor placement points with the highest data match can occur. With the general point is determined and the movement of the joints, the sensors can track the movement of the complete body. The following image shows the track joints each different body part:

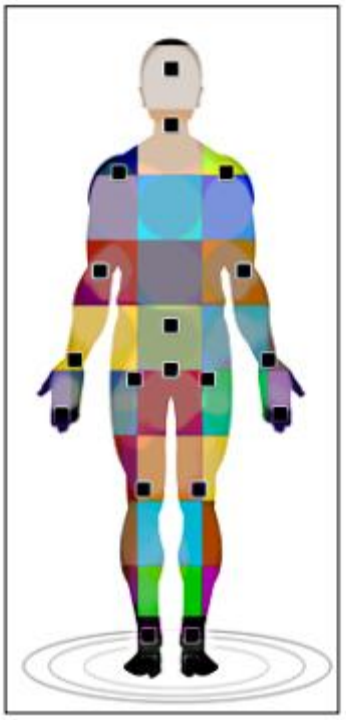

Figure 3.3. Inferred joint proposal.

The general position is measured by three coordinates (X, Y, and Z), in which X and Y determine the location of the joints, and Z represents the distance from the sensor. To get the proper coordinates, sensor features three of the same image: preview, review, and see the top, which determined that

the proposed sensor 3D body. Three views are shown in the image below:

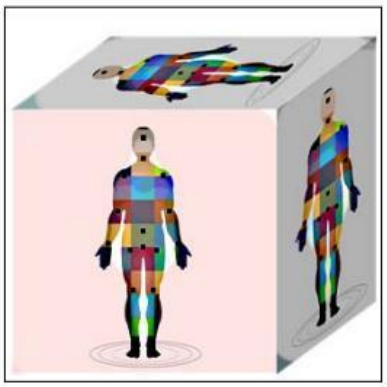

Figure 3.4. Front view, left view and top view of the same image

3.2 Skeleton tracking with the Kinect SDK

Kinect for Windows SDK provides us with a set of APIs to allow easy access to the joints. SDK supports monitoring up to 20 points overall. Each joint location is determined by its name (head, shoulders, elbows, wrists, arms, spine, hips, knees, ankles, and so on), and frame status and tracking determined by one of two hot, not hot, or just location. SDK channels used to detect the bone. Default channel tracking all 20 positions with osteoarthritis Subscribe track mode, not tracked, or inferred. The diagram below shows a complete skeleton face with Kinect sensor, with 20 points overall picture can be tracked by the Kinect for Windows SDK :

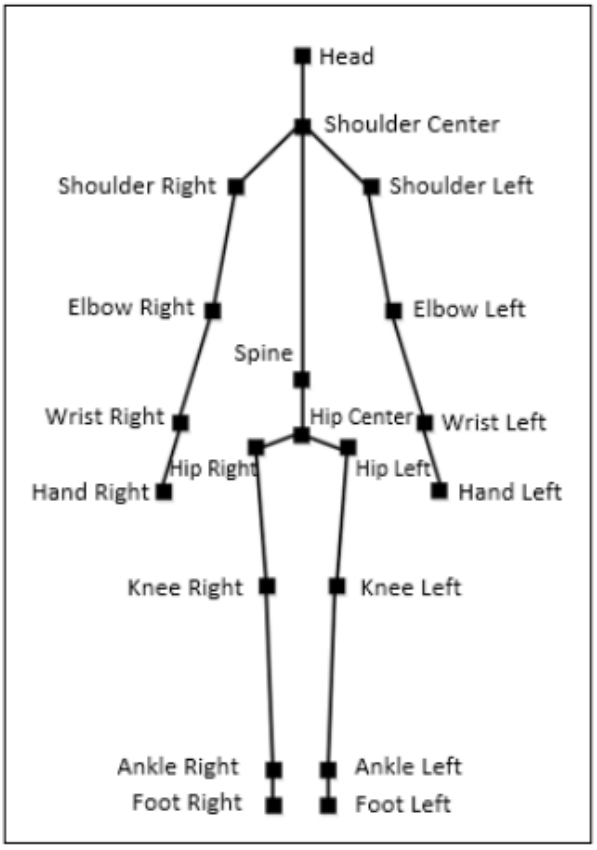

Figure 3.5. Complete human skeleton facing the Kinect sensor.

Full Kinect can track up to two users , and can detect up to six users in viewable range , the other four are called the skeleton is proposed . We can only get 20 joints complete the full skeleton track , because four other people , we will receive information only about the hip joint center . Of the two skeleton track , it will be active and the other will be considered based on how passive we are using the skeleton data . If a bone is fully tracked , the next frame after the data will return full skeleton , while the passive skeleton tracking , we would have only been proposed location . The following image shows the full skeleton tracking for two users:

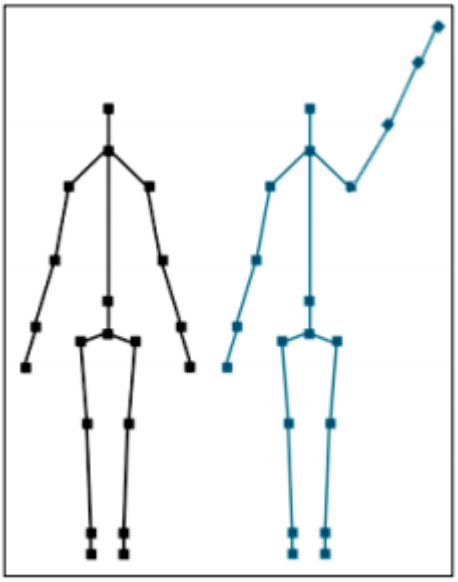

Figure 3.6. The fully tracked skeleton for two users.

The Kinect for Windows SDK also supports tracking of a seated skeleton. We can change the tracking mode to detect a seated human body that returns up to 10 joint points, as shown in the following image:

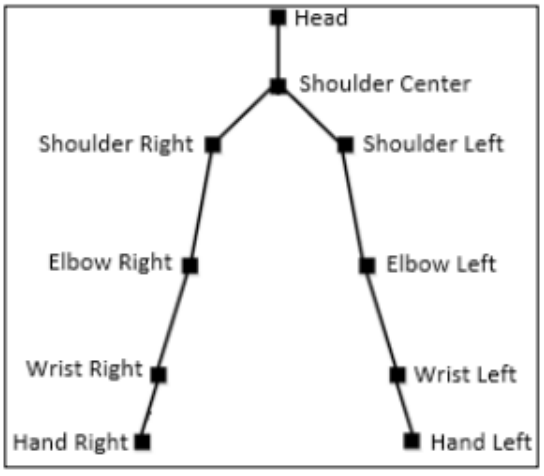

Figure 3.7. A seated human skeleton body

So far, we have covered how skeleton tracking worked and the different types of joints returned by skeleton tracking. Next, we will walk

through the basic options available for skeleton tracking and see how to get skeleton data.

3.3 Seeking Skeleton walkthrough

Before jumping into code and working with skeleton data, we should first walk through the some basic options and see how to get skeleton data.

The data comes from SkeletonStream skeleton . Data from this series can be accessed either from the events or the voting by similar colors and rivers . In this introduction , we use the event simply because it takes less code and is a more common approach and basic . KinectSensor object has a name SkeletonFrameReady event , which fires each time new data sets become available bone . Data are also available from the skeleton AllFramesReadyevent . We look at the monitoring object model skeleton in more detail shortly , but for now , we are only interested in themselves to get the data from the skeleton line . Each frame of SkeletonStream produce a collection of objects Skeleton . Skeleton Each object contains data describing the position of the bones and joints of the skeleton . Every business has an identity (head , shoulder , elbow , etc.) and a 3D vector.

There is an event handler for SkeletonFrameReadyevent. Each event handler execute it took the current frame by calling OpenSkeletonFramemethod on the event argument parameter.

Every time we process a skeleton, our first step is to determine whether we have a real skeleton. One way to do this is with the skeleton TrackingStateproperty. Only users who actively tracked by the tracking tool is drawn skeleton. We ignore any skeletons processing is not tracking a user (that is, not by the TrackingStateis SkeletonTrackingState.Tracked). While

Kinect can detect up to six users, it only tracks the general location for two people.

Processing performed on the data very simple skeleton. We chose a brush to paint stick figures based on the player's position in the collection. Next, we draw stick figures. The CreateFiguremethod draw stick figure skeleton skeletons for a single object. The GetJointPointmethod is important to draw a stick figure. This method takes the position vector of the joint and call MapSkeletonPointToDepthmethod on KinectSensor example to convert skeleton coordinate with the depth image coordinates. The method converts the business GetJointPoint skeleton from bone coordinate system to the coordinate system interface and returns a point in the user interface that businesses should be.

It is also important to point out that we are removing the value Z. Seems like a waste for the Kinect to do a bunch of work to produce a depth value for each company and then tell us not to use this data . In fact , we are using the Z value , but obviously not . It just is not used in the user interface . Coordinates transformation requires space depth value . Check this by calling the MapSkeletonPointToDepth , go through the X and Y values in general , and not set the Z value . As a result, DEPTHX and depthY always turn back to 0 . As an additional exercise , using depth values to apply a ScaleTransform bone data based on the value of Z. The scale value is inversely proportional to the depth value . This means smaller depth values , values greater scale , so close to a user 's Kinect , the greater the bone .

3.4 The skeleton object model

There are more objects, structures, and enumerations associated with skeleton tracking than any other feature of the SDK. In fact, skeleton tracking accounts for over a third of the entire SDK. Skeleton tracking is obviously a significant component. Figure 3.8 illustrates the primary elements of the

skeleton tracking. There are four major elements (*SkeletonStream*, *SkeletonFrame*, *Skeleton* and *Joint*) and several supporting parts. The subsections to follow describe in detail the major objects and structure.

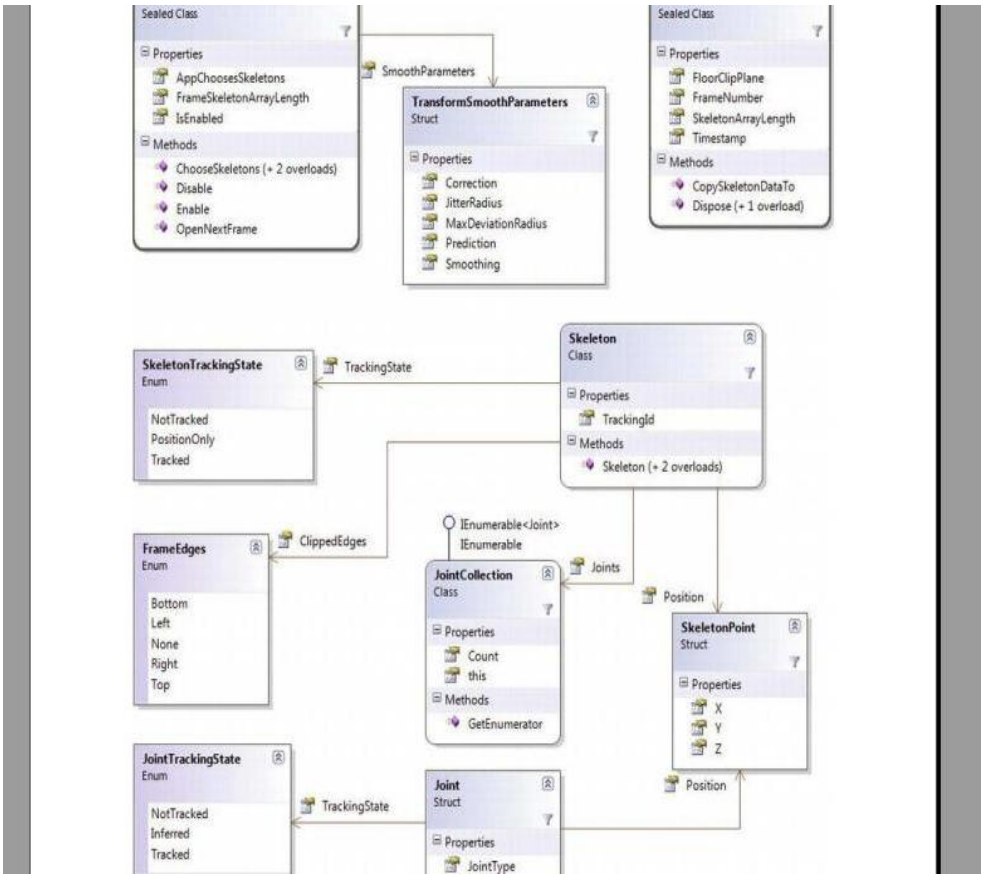

Figure 3.8 Primary elements of skeleton tracking

3.4.1 SkeletonStream

The *SkeletonStream* generates *SkeletonFrame* objects. Retrieving frame data from the *SkeletonStream* is similar to the *ColorStream* and *DepthStream*. Applications retrieve skeleton data either from the *SkeletonFrameReadyevent*, *AllFramesReadyevent* or from the *OpenNextFrame* method. Note, that calling the *OpenNextFrame* method after

subscribing to the *SkeletonFrameReady* event of the *KinectSensor* object results in an *InvalidOperationException* exception.

3.4.1.1 Enabling and Disabling

The SkeletonStream not produce any data until activated. By default , it is disabled . To activate it SkeletonStream to start generating data , called its activation method . There is also a method called Disable , which suspended production of the skeleton data . SkeletonStream object also has a property named IsEnabled , which describes the current state of the data produced bone . SkeletonFrameReady event on an object KinectSensor not fire until SkeletonStream enabled. If you choose to use a voting architecture , the SkeletonStream must be enabled before calling methods OpenNextFrame , if not , the call throws an InvalidOperationException exception . In most applications , once activated, the SkeletonStream is not disabled during the lifetime of the application . However, there are instances where it is desirable to disable the line . One example is the use of multiple Kinects in an application-an advanced topic that is not covered in this thesis. Note that only one can Kinect skeleton data reports for each process. This means that even with multiple Kinects running, applications are limited to two skeletons. Then choose which applications to enable Kinect skeleton tracking. During the application process, it is then possible to change that Kinect is actively monitored by the skeleton of a cancellation SkeletonStream Kinect and allows the other person.

Another reason to disable the data produced is to perform bone . Handling skeleton is an expensive operation . This is evident by looking at the CPU usage of an application with skeleton tracking is enabled . To see this , open Windows Task Manager and run applications created from the beginning of this chapter . Little stick figure but it does not have a high CPU usage . Here are the results of the monitoring of bone . Disabling skeleton tracking is

useful when the application does not need skeleton data , and in some cases , disabling tracking skeleton may be necessary . For example , in a game a number of events can trigger a complex animation or video cut scenes . During the animation or video sequence bones unnecessary data . Disable tracking skeleton may also be necessary to ensure a smooth sequence of images or video .

There is a side effect to disabling the *SkeletonStream*. All stream data production stops and restarts when the *SkeletonStream* changes state. This is not the case when the color or depth streams are disabled. A change in the *SkeletonStream* state causes the sensor to reinitialize. This process resets the *TimeStamp* and *FrameNumber* of all frames to zero. There is also a slight lag when the sensor reinitializes, but it is only a few milliseconds.

3.4.1.2 Smoothing

As we gain experience working with bone data, we found that bone movements are often erratic. There are several possible causes for this, from poor application performance on the behavior of users (many factors can cause a person to shake or move smoothly), the performance of the Kinect hardware. The change of location of a business can be relatively large from frame to frame, can negatively affect an application nonchalantly. In addition to creating a user experience was embarrassing and displeasing aesthetics, it is confusing to users when their representatives or the cursor appears shaking hands seizures or worse.

The SkeletonStream there a way to solve this problem by normalizing the position value by reducing the difference in joint position from frame to frame. When allowed SkeletonStream using Activate method overload and pass in a structure TransformSmoothParameters. To refer SkeletonStream two

read-only attribute to smoothing and SmoothParameters IsSmoothingEnabled name. IsSmoothingEnabled property is set to true when the line is enabled with a default TransformSmoothParameters and wrong methods are allowed to use. Storage assets SmoothParameters smoothing parameters are determined. TransformSmoothParameters structure determines the properties:

- *Correction* – Takes a float ranging from 0 to 1.0. The lower the number, the more correction is applied.
- *JitterRadius* – Sets the radius of correction. If a joint position "jitters" outside of the set radius, it is corrected to be at the radius. The property is a float value measured in meters
- *MaxDeviationRadius* – Used this setting in conjunction with the *JitterRadius* setting to determine the outer bounds of the jitter radius. Any point that falls outside of this radius is not considered a jitter, but a valid new position. The property is a float value measured in meters.
- *Prediction* – Returns the number of frames predicted.
- *Smoothing* – Determines the amount of smoothing applied while processing skeletal frames. It is a float type with a range of 0 to 1.0. The higher the value, the more smoothing applied. A zero value does not alter the skeleton data.

3.4.1.3 Choosing Skeletons

By default , bone tools available option that actively tracked skeletons . The engine selected two skeleton bone available for the first track , which is not always desirable mainly because the selection process is unpredictable . We have the option to choose the skeleton to track using properties and methods AppChoosesSkeletons ChooseSkeletons . TheAppChoosesSkeletons property is false by default and therefore the engine bone to pick skeleton track . To manually select the skeleton to track , set the property to true and call AppChoosesSkeletons ChooseSkeletons TrackingIDs

method passing in the skeleton , we want to track . ChooseSkeletons method accepts one, two , or no TrackingIDs . The engine stops the bone all the way to the bone ChooseSkeletons passed no parameters . There are several shades to choose from skeletons .

3.4.2 SkeletonFrame

The object SkeletonStream SkeletonFrame production. When using the event model application gets an object from the arguments SkeletonFrame event by calling OpenSkeletonFrame, or method SkeletonStream OpenNextFrame on the vote. Data object containing the bone SkeletonFrame for a moment in time. The skeleton of the frame data is available by calling CopySkeletonDataTo. This method populates an array passed to it with the skeleton data. The SkeletonFrame have a property named SkeletonArrayLength, the amount of bone allowing it to data. Arrays are always paid in full even though the population has no use in the area of Kinect.

3.4.2.1 Marking time

The FrameNumber and Timestampfields mark this moment in time but the frame was recorded. FrameNumber is an integer representing the number of image frames used to create deep frame. The number of frames is not always constant, but each frame number will always be greater than the previous frame. It is possible for the engine to skip frames bone deep in the process. The reasons for this vary based on the overall application performance and frame rate. For example, a long running process in any event processing can slow down the processing line. If an application uses polling instead of event model, it depends on the application to determine how often the data generated skeletal muscle, and thereby effectively skeleton deep Derived Data.

Timestamp field is the number of milliseconds since KinectSensor be initialized. We do not need to worry about the long-running applications to

achieve maximum FrameNumber or Timestamp. The FrameNumber is a 32-bit integer while the Timestamp is a 64-bit integer. This application will be running continuously at 30 frames per second for just over two years before you reach the maximum FrameNumber, and this is way before the timestamp is nearly bare. In addition, Timestamp FrameNumber and start from zero each time KinectSensor be initialized. We can rely on the value of the Timestamp FrameNumber and unique.

At this stage in the life cycle of the Kinect SDK and overall development, the field is important because it is used to process or analyze frames, such as the smoothing of shared values. Processing gesture is another example, and most popular, used this data to a data frame in order. The current version of the SDK does not include a motor gestures. Until a future version of the SDK including gesture tracking, developers must code the gesture recognition algorithm of their own, which may depend on knowing the sequence of frame.

3.4.2.2 Frame descriptor

FloorClipPlane field is 4 sets (tuple <int, int, int, int>) with each element is a factor of the aircraft floor. General equation for the floor plane is Ax + By + Cz + D = 0, which means that the first element tuple corresponding to A, the second to B and so on. Variable D in the floor plane equation is always negative height in meters from the floor Kinect. When possible, the SDK uses image processing techniques to determine the exact value of the coefficient, however, this is not always possible, and the value must be estimated. The FloorClipPlaneis not a plane (all the elements have zero value) when the floor is not defined..

3.4.3 Skeleton

Skeleton class defines a set of fields to identify the skeleton, describing the position of the skeleton and may be the position of the joints of the skeleton. Objects of bone available by passing an array with

CopySkeletonDataTo on SkeletonFrame method. Method CopySkeletonDataTo have unusual behavior, which can affect the memory usage and references to objects Skeleton. The Skeleton object is returned only for array and not the application.

3.4.3.1 TrackingID

The skeleton tracking engine assigns each skeleton a unique identifier. This identifier is an integer, which incrementally grows with each new skeleton. Do not expect the value assigned to the next new skeleton to grow sequentially, but rather the next value will be greater than the previous. Additionally, the next assigned value is not predictable. If the skeleton engine loses the ability to track a user—for example, the user walks out of view—the tracking identifier for that skeleton is retired. When Kinect detects a new skeleton, it assigns a new tracking identifier. A tracking identifier of zero means that the Skeleton object is not representing a user, but is just a placeholder in the collection. Think of it as a null skeleton. Applications use the *TrackingID* to specify which skeletons the skeleton engine should actively track. Call the *ChooseSkeletons* method on the *SkeletonStream* object to initiate the tracking of a specific skeleton.

3.4.3.2 TrackingState

This field provides insight into what skeleton data is available if any at all. Table 3.1 lists all values of the *SkeletonTrackingState* enumeration.

SkeletonTrackingState	What it means
NotTrackedThe	Skeleton object does not represent a tracked user. The Positionfield of the Skeletonand every Jointin the joints collection is a zero point (SkeletonPointwhere the X, Y and Z values all equal zero).

PositionOnly	The skeleton is detected, but is not actively being tracked. The Positionfield has a non-zero point, but the position of each Jointin the joints collection is a zero point.
Tracked	The skeleton is actively being tracked. The Positionfield and all Joint objects in the joints collection have non-zero points.

Table 3.1. All values of the SkeletonTrackingState enumeration

3.4.3.3 Position

The *Position* field is of type *SkeletonPoint* and is the center of mass of the skeleton. The center of mass is roughly the same position as the spine joint. This field provides a fast and simple means of determining a user's position whether or not the user is actively being tracked. In some applications, this value is sufficient and the positions of specific joints are unnecessary. This value can also serve as criteria for manually selecting skeletons (*SkeletonStream.ChooseSkeletons*) to track. For example, an application may want to actively track the two skeletons closest to Kinect.

3.4.3.4 ClippedEdges

The *ClippedEdges* field describes which parts of the skeleton are out of the Kinect's view. This provides a macro insight into the skeleton's position. Use it to adjust the elevation angle programmatically or to message users to reposition themselves in the center of the view area. The property is of type *FrameEdges*, which is an enumeration decorated with the *FlagsAttributeattribute*. This means the *ClippedEdges* field could have one or more *FrameEdges* values. Table 3.2 lists the possible FrameEdges values.

FrameEdges	What it means
Bottom	The user has one or more bodyparts below Kinect's field of view.
Left	The user has one or more body parts off Kinect's left.
Right	The user has one or more body parts off Kinect's right.
Top	The user has one or more body parts above Kinect's field of view.
None	The user is completely in view of the Kinect.

Table 3.2. Possible FrameEdges values

It is possible to improve the quality of skeleton data if any part of the user's body is out of the view area. The easiest solution is to present the user with a message asking them to adjust their position until the clipping is either resolved or in an acceptable state. For example, an application may not be concerned that the user is bottom clipped, but messages the user if they become clipped on the left or right. The other solution is to physically adjust the title of the Kinect. Kinect has a built-in motor that titles the camera head up and down. The angle of the tile is adjustable changing the value of the *ElevationAngle* property on the *KinectSensor* object. If an application is more concerned with the feet joints of the skeleton, it needs to ensure the user is not bottom clipped. Adjusting the title angle of the sensor helps keep the user's bottom joints in view.

The *ElevationAngle* is measured in degrees. The *KinectSensor* object properties *MinElevationAngle* and *MaxElevationAngle* define the value range. Any attempt to set the angle value outside of these values results in an *ArgumentOutOfRangeException* exception. Microsoft warns not to change the title angle repeatedly as it may wear out the tilt motor. To help save

developers from mistakes and to help preserve the motor, the SDK limits the number of value changes to one per second. Further, it enforces a 20-second break after 15 consecutive changes, in that the SDK will not honor any value changes for 20 seconds following the 15^{th} change.

3.4.3.5 Joints

Each *Skeleton* object has a property named Joints. This property is of type *JointsCollection* and contains a set of Joint structures that describe the trackable joints (head, hands, elbow and others) of a skeleton. An application references specific joints by using the indexer on the *JointsCollection* where the identifier is a value from the *JointType* enumeration. The *JointsCollection* is always fully populated and returns a Joint structure for any *JointType* even when there are no user's in view.

3.4.4 Joint

The skeleton tracking engine follows and reports on twenty points or joints on each user. The Joint structure represents the tracking data with three properties. The *JointType* property of the Joint is a value from the *JointType* enumeration. Figure 3.9 illustrates all trackable joints.

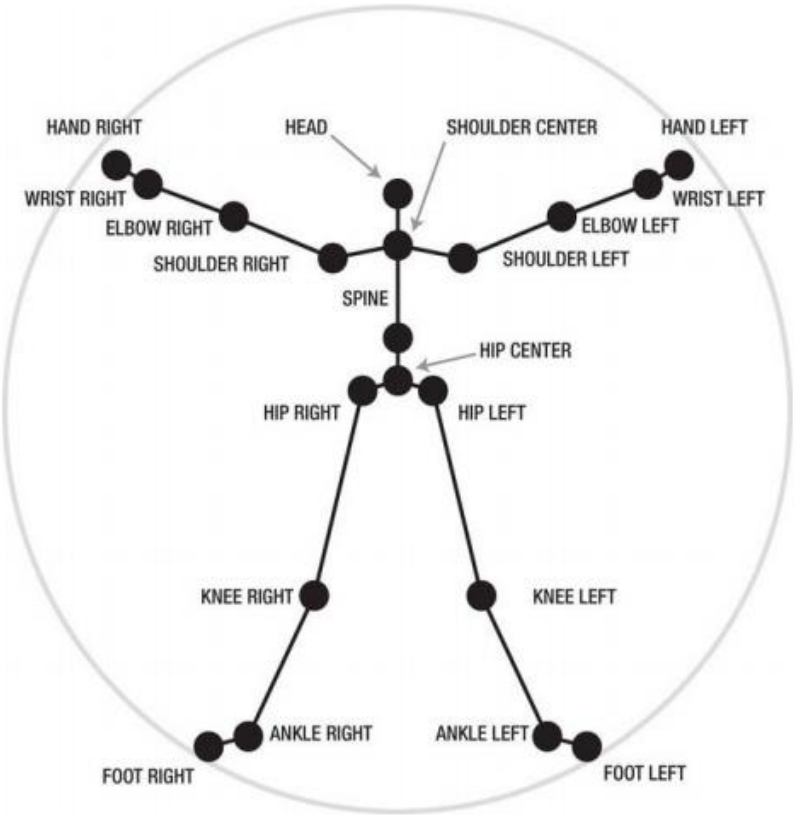

Figure 3.9. Illustrated skeleton joints

Each joint has a *Position*, which is of type *SkeletonPoint* that reports the X, Y, and Z of the joint. The X and Y values are relative to the skeleton space, which is not the same as the depth or video space. The *KinectSensor* has a set of methods that convert skeleton points to depth points. Finally, there is the *JointTrackingState* property, which describes if the joint is being tracked and how. Table 3.3 lists the different tracking states.

JointTrackingState	What it means
Inferred	The skeleton engine cannot see the joint in the depth frame pixels, but has made a calculated determination of the position of the joint.

NotTracked	The position of the joint is indeterminable. The Position value is a zero point.
Tracked	The joint is detected and actively followed.

Table 3.3. JointTrackingState values

Chapter 4

PID controller

"PID" stands for derivative , integral and propotional. As the name suggests, these terms describe three basic mathematical functions applied to the error (error = SetVal - SensorVal, where SetVal goals and values SensorVal the current input values obtained from the sensor). The main task of the PID controller is to minimize the error of whatever we are in control. It is in the input, calculate the deviation from the intended behavior and appropriate to adjust the output deviation from the intended behavior is minimized and greater accuracy is obtained.

4.1 Purpose of PID

The following line seems to be accurate when done at lower speeds. As we begin to increase the speed of the robot, it vibrate a lot and usually found out when monitoring (tracking).

Therefore some kind of robot control is required which will allow us to make it effective stream at high speed. This is where PID control shine.

To perform a series after basically can start with just three sensors on the robot to distance:

- If the sensor detects center line drive robot forward.
- If the left sensor detects the robot line drive right.
- If the sensor detects even the Robot line left calf.

This algorithm will make the robot follow the line. However, we will need to compromise with its speed according to the road efficiently. We can

increase the effectiveness of this line by increasing the number of sensors, say 5.

Here the possible combinations represent exact position like-

00100	On the central of the line
00001	To the left on the line
10000	To the right on the line

Table 4.1. Position of the sensor light decides the direction of the robot

There will be other possible combinations as 00 110 and 00 011 can provide data to us about how the far right is the robot from the center of the line (just after the left). Continue to perform better following lines we need to keep track of how long the robot does not focus on the line and how quickly it changes its position from the center. This is exactly what we can achieve by using the "PID" control. The data obtained from the sensor array will then be put to maximum use and process the following line will be much more smoother, faster and more efficient at higher speeds.

PID is all about improving our control over the robot. The idea behind our PID controller is set to a value that we want to maintain, or the speed of the engine or read from a sensor. Then we have the current readings as input and compare them with the set point. From this a value can be calculated errors (error = setpoint - actual reading). Then this error value is used to calculate how much to change the output to make the reader closer to the actual set point.

4.2 Sensors

This sensor is the first requirement in the following lines. There are many sensors can be used for the following line - infrared LED and a photodiode, LED and LDR (photoresistor), etc. When selecting sensors have three things we need to remember - response time , light sensitivity and protect the surrounding environment.

LDR (photoresistors) is great for sensitivity, they can not only feel the contrast between the two colors (one dark and one light) but also between two basic colors. Some make the competition tougher line after by a white line on a surface colors (green, red, gray). In this case LDR sensor is being used. But there is a major disadvantage in the use of LDR - slow response time of them. An LDR reaction time is about 0.1s. This may not seem much, but if we have a robot using LDR after a line with speed 1 m / s then it will receive the data from each sensor (0.1 * 1 = 0.1m) 10 cm!

IR LEDs and photodiodes for better response time. But they have two weaknesses, both of which can be dealt with - they can only feel the contrast between the two colors and they can be easily affected by ambient light.

Sensitivity can be improved in two ways . One using LEDs emit light of any color other than colors and surfaces using a photodiode suitable for light feel . Basically, if there is a white line on a red surface , we can use any other color LED , Green said , to emit light and photodiodes that respond to blue leaves . The surface is what happened , is red , absorbs all the green light emitted from LED lights and no light reaches the photodiode . Sugar , on the other hand , is white , reflecting all the green light falling on it back photodiode . Using this approach significantly increases the sensitivity of the sensor . Another way to improve sensitivity by modifying the sensor circuit .

Only by changing the resistance of a resistor in the circuit of the sensor , we can adjust it to feel between two specific colors .

Second drawback , that is , the interference of ambient light , is much more difficult to deal with . Photodiodes detect infrared light with a wavelength specific, but usually between 700-850 nanometers . Unfortunately , most of the wavelength of the infrared radiation around us fall into this range . For photodiode radiation is the same as the infrared radiation emitted from the infrared LED , and therefore they are triggered by them wrong . Even the red light (which just beyond the infrared radiation in the electromagnetic spectrum) can affect the sensor . There are many methods we can use to protect the infrared sensor interference from ambient light . One of the simplest methods of protecting the sensor from ambient light . We may also use sensor provides ambient light protection as sensor TSOP 17 .

4.3 Implementation of PID

4.3.1 Terminology

The basic terminologies that one would require to understand PID are:

- Error - Error is the amount that a device is not doing something right . For example , suppose the robot is placed at x = 4 , but it is at x = 10 , then the error is 6 .
- Proportional (P) - Time rate is proportional to the current error .
- Integral (I) - integral term depends on the cumulative error is made in a period of time (t) .
- Derivative (D) - The term derivative dependent changes in error rate .
- Constant (factor) - Each term (P , I, D) need to be tweaked in the code . Therefore, they are included in the code by multiplying the corresponding constants .

- P - Factor (Kp) - A constant value is used to increase or reduce the impact of rate
- I- Factor (Ki) - A constant value is used to increase or reduce the impact of integration
- D - Factor (KD) - A constant value is used to increase or reduce the impact of derivative

4.3.2 PID formulas

So what do we do with the error value to calculate how much output is changed? We'll just have to add the value to the output error to adjust the robot's motion. And this will work, and is known as proportional control (P in PID). It is often necessary to expand the scale of error values before adding it to the output using the constants (KP).

- **Proportional:**

Difference = (Target Position) - (Measured Position)

Proportional = Kp*(Difference)

This approach will work, but it was found that if we want to have a quick response time, using a large constant, or if the error is very large, the output could overshoot from the set value. Hence the change in output may turn out to be unpredictable and fluctuating. To control this, the derivative expression comes to the limelight.

- **Derivative:**

Derivative gives us the rate of change of error. This will help us know how fast the error changes over time and we can match the output settings.

Rate of Change = ((Difference) – (Previous Difference))/time interval

Derivative= Kd *(Rate of Change)

The time can be obtained by using the timer of the microcontroller. Built improve steady-state performance, ie, when the output is stable compared to how our words set point. By adding up all the previous errors can monitor if there are errors accumulate. For example, if the position is slightly to the right all the time, the error will always be positive so that the total error will be larger, the inverse is true if the position is always on the left. This can be monitored and used to further improve the accuracy of the following line.

- **Integral**

Integral = Integral + Difference

Integral = Ki*(Integral)

Term	Expression	Effect
Proportional	Kp*error	It reduces a large part of the error based on present time error.
Integral	Error dt	Reduces the final error in the system. Cumulative of a small error over time would help us further reduce the error.
Derivative	Kd*derror/dt	Counteracts the Kp and Ki terms when the output changes quickly.

Table 4.2 Summary of PID control

Therefore, Control value used to adjust the robot's motion = (Proportional) + (Integral) + (Derivative)

4.4 Tuning

Implementation of PID will prove to be useless and not more troublesome, unless the value is continuously adjusted depending on the robot's platform is designed to run on. Physical environment in which the robot is being operated and no significant changes can be modeled mathematically. It consists of ground friction, the motor inductance, center of mass, etc. Thus, the constant guessing numbers obtained by trial and error. The value of their most consistent changes from robot to robot and also the circumstances in which it is being run. The aim is to establish the constant such that the settling time is minimal and there is no overshoot.

There are some basic guidelines that will help reduce the tuning effort.

- Start with Kp, Ki and Kd equal to 0 and Kp work with first. Trying to set the value Kp 1 and observe the robot. The goal is to get the robot follow the road even if it is very wobbly. If the robot overshoots and loss of flow, reduce Kp values. If the robot can not move a turn or seems slow, increase the value of Kp.
- Once the robot can follow the line somewhat, assign a value from 1 to KD (Ki ignore for the moment). Try to increase this value until the little number rocking (wobbles) .
- Once the robot is quite stable in the following line, specifying a value of 0.5-1.0 km. If Ki value is too high, the robot will jerk left and right quickly. If it is too low, we will not see any difference felt. From the accumulation area, has a Ki value of significant impact. We can adjust it by 0.01 will increase.

- Once the robot is following suit with high accuracy, we can increase the speed and see if it can still follow the line. Affect the speed PID controller and will require retuning as the speed changes.

Chapter 5

Controlling robot using path following and Kinect

In previous chapters, we managed to explain all the knowledge which involves in building the robot that follows lines and takes orders from Kinect. In this chapter, we will explain further about the preparation to run robot, all of the troubles we meet during some run tests.

5.1 Preparation

5.1.1 Set up some programs

In order to control the robot, first we need to install the program for the chip STM32F103C8T6, using Flash Loader. Here is the interface of the program when running:

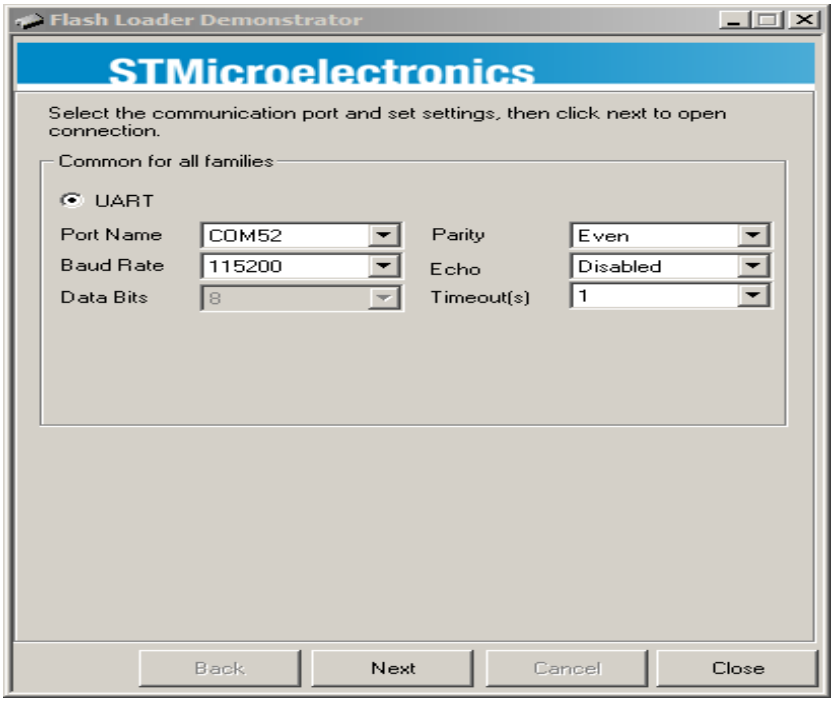

Figure 5.1. Flash Loader user interface

There's some modification needed to be done:

- In the port name, choosing the COM gate that we are using.
- Choose the compatible baud rate, here is 115200

- Parity choosing Even
- Echo : Disabled

Click next, there will be some success notifications after that

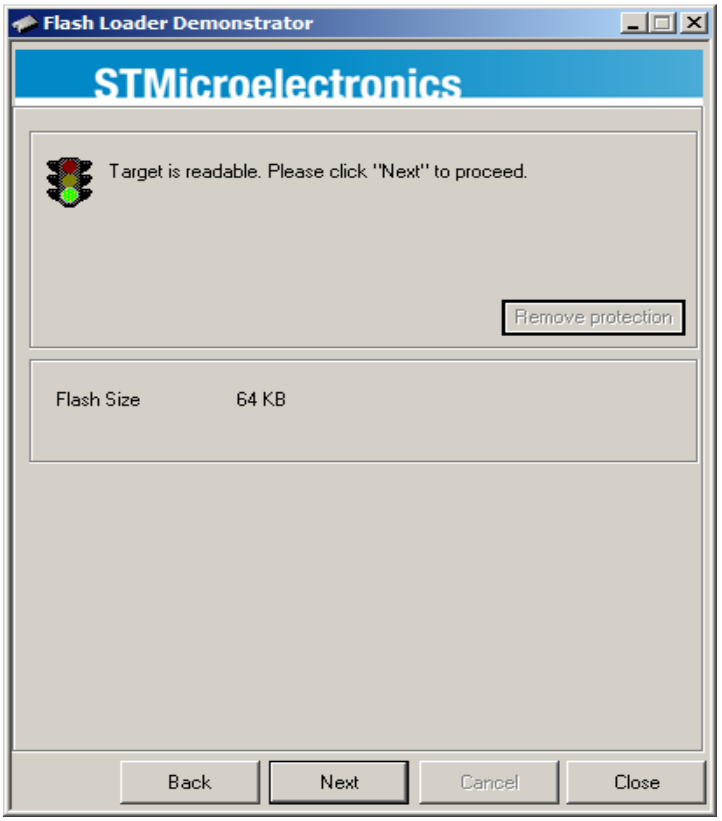

Figure 5.2. Success notification 1

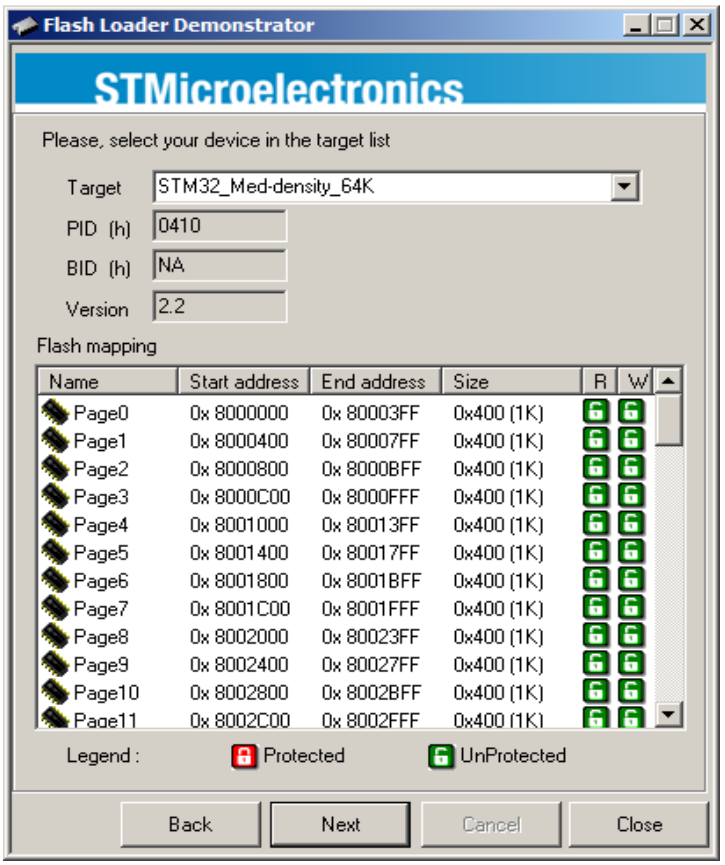

Figure 5.3. Installation step 1

After that, we choosing the hex file to add, then click next, the installation will be completed.

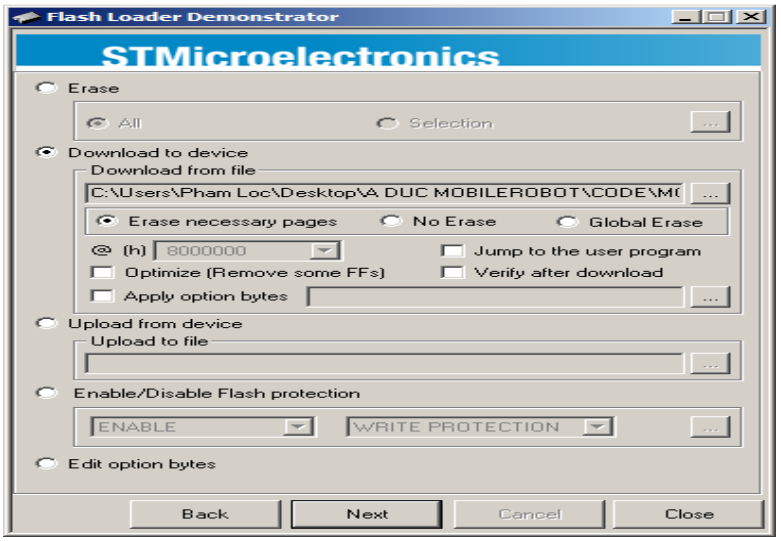

Figure 5.4 Installation step 2

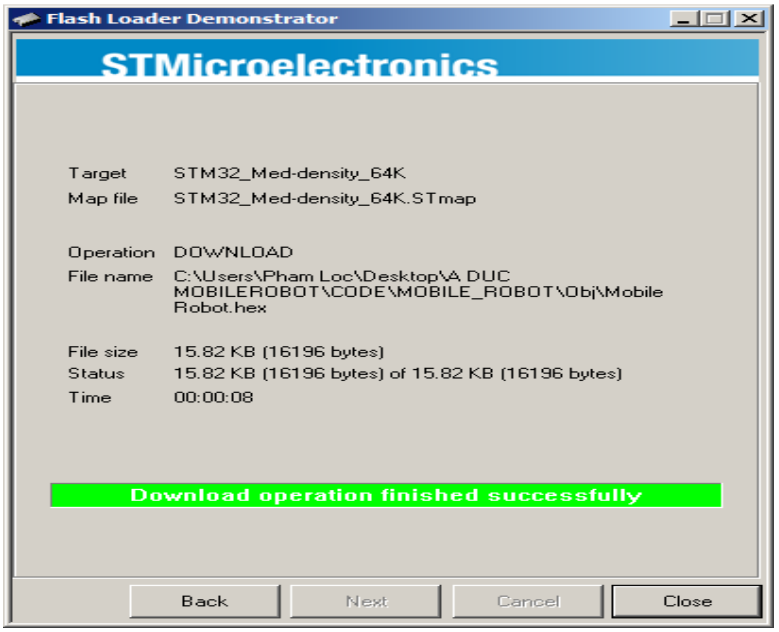

Figure 5.5 Success Notification 2

5.1.2 Set up the path

As mentioned in the first chapter, the line may have different appearances, but the most ideal is black line on white surface or vice-versa. In this thesis, we will choose to do run test with black line on white surface. The path must have at least two crossroads, must be twisted and long enough in order to prove the path following function.

5.1.3 Training the line and the environment

In order to follow the path, the robot has to detect the difference between the color of line and the color of the ground, or the environment. For that purpose, it must study both of the colors and save them in the robot's memory. We call this function "train". Before interpret how to train, let's look again at the general view of robots:

Figure 5.6. The robot pictured from above

Figure 5.7. Light sensors of robot

Figure 5.8 Sequence of control switches

The outlook of the robot is just as simple as it is. The light sensors will take the role of detecting, training the line and the environment. The sequence of control switches will let the robot train the path. We will now move on to how to train the robot.

First, let's look again at Figure 5.8. There are three buttons: red, yellow and green, but we will only use the red and yellow ones. The green button is built with the intent to save for another function in the future. The red button will train the color of the environment, and the yellow button will train the color of the path. Note that when training, we must put the color we want to train directly under the light sensors. After training, the robot will remember all of the samples, unless there's a change in the surrounding environment, such as sunlight, light of the electric bulb, etc. If we train successfully, the robot's led will be like this when moving on the path:

Figure 5.9. The robot after training

5.2 Test cases

The purpose of the thesis is that the robot can follow the line and take commands from human. If the robot can somehow move when there's no line, or when there's no orders from human, that means we must have failed. Therefore, we have thought up some test cases to check the ability of the robot

- Case 1: the color of the line is "closed" to the color of the environment. In figure 5.10, the color of the line is black, and the color of the floor is brown. The robot can still follow the line, but sometimes, it stops in the middle, and do not run even if we tell it to stop.

Figure 5.10. Straight line

- Case 2: Straight line, no junctions. The robot stops at the end of line.
- Case 3: command the robot to turn left/turn right/go straight/ when there's no line: the robot does not go straight, but it does turn right or left, but will stop after 15 seconds.
- Case 4: The Y-shaped junction. Figure 5.11, 5.12 and 5.13 shows 3 kind of y-shaped junctions. The robot in all of those cases can stop and the junction, but when turning left or right, the robot in figure 5.11 and 5.12 will go out of line. It's due to the angle between 2 roads is too small, which leads to the inability of the robot's sensors to detect the line.

Figure 5.11. Y-shaped junction 1.

Figure 5.12. Y-shaped junction 2.

Figure 5.13. Y-shaped junction 3.

- Case 5: the roundabout. This case is similar to the y-shaped junction case.
- Case 6: normal case: this is the main case we use in all of our demo videos. We will set up a circle like figure 5.14

Figure 5.14. Normal case

5.3 Evaluation

As mentioned in the previous section, we have used test case 6 as the main test case to run our demos. We have recorded a lot of videos in University of Science with the setup similar to Figure 5.14. After those experiments, we have figured some problems

- The robot sometimes stops in the middle of the race because of some dirt between the floor bricks. Robot's sensors detect the dirt as the junction, which is shown in Figure 5.15

Figure 5.15 "Dirt" environment

- When turning left or right, if there's are some environmental problems, such as the adhesion of the robot's wheels is not good enough, the robot will keep on turning until the robot runs out of battery or turns off the power.

Although we have detected those bugs, we still cannot fix those yet. However, beside of those problems, the robot is running pretty stable all of the time we test it. The robot can detect a wide range of junctions, from y-shaped junction to perpendicular crossroad. By applying the PID algorithm, we have greatly reduced the wobble of the robot when it is changing direction. The Kinect also detect the human's gesture accurately.

CONCLUSION

We have studied two new technologies for controlling robot, which are path following and Kinect. We have discussed all of their features, algorithms and application in real life through those previous chapters. However, it is a very long road when we come up with the idea of this thesis

At the beginning, we decided to use Kinect to detect obstacles, and let the robot to automatically avoid it. We have spent about 2 months to study about this problem, especially PCL library and Cmake application. However, this turn out to be fruitless because PCL is still developing. There are a lot of problems in set up those library and program. After knowing the idea of controlling power point slides though Kinect, It becomes our aspiration to do the controlling robot by human gestures.

During the time we develop this idea, we thought that it is too simple. We have done some searching, and come up with the idea of path following robot. We are so excited about that, and decide to combine both techniques into a hybrid approach. We hope that this will be a great contribution to the robot industry.

FUTURE WORKS

Due to time constraints, we have only finished the foundation of the path following robot using Kinect. There are a lot of things we would like to add to our robot, to make it more useful:

- Add some new function to the Kinect, such as speech recognition and face recognition. The Kinect will be trained and can only detect the face and voice of a certain user.
- Our path following robot at this point is pretty much dependant on human. When the roads are complicated, when there are a lot of junctions, it does not know how to go from A to B automatically. We intend to develop this function to enhance to competitiveness of our robot to the others.
- We also want to continue our old failure, which is using Kinect to let the robot avoid obstacles. This function can be developed into the system that replaces the guide dogs for the blinds.

REFERENCE

[1] Jarett Webb and James Ashley, “Beginning Kinect programming with Microsoft Kinect SDK”.

[2] Abhijit Jana, “Kinect for Windows SDK Programming Guide”.

[3] Muhammad Azfar Bin MD Yusof, “Machine vision and Depth perception using Microsoft Kinect sensor”, University of Technology, Malaysia, 2012.

[4] Abhishek Kar, “Skeletal traking using Microsoft Kinect”, 2012.

[5] J. Sluka, “PID controller for Lego Mindstorms robots”, 2009

[6] Lâm Duy Tiến, “Thiết bị Kinect và ứng dụng trong nhận dạng, đo kích thước vật thể”, University of Technology, HCM city, 2012.

[7] N H Đức, N V Đức, “Robot tự vận hành tránh vật cản sử dụng thiết bị Kinect”, University of Technology, HCM city, 2012.

[8] P T Hưng, “Nhận diện và theo dõi vật thể dung thiết bị Kinect”, University of Technology, HCM city, 2012.