Application of Depth Sensor in the Design of Hybrid Robotic Gaming Environment

by

Shiou- Min Shen

B.A.Sc., Simon Fraser University, 2010

Project Report Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering

in the
School of Engineering Science
Faculty of Applied Science

© Shiou-Min Shen 2015

SIMON FRASER UNIVERSITY

Summer 2015

All rights reserved.
However, in accordance with the *Copyright Act of Canada*, this work may be reproduced, without authorization, under the conditions for "Fair Dealing." Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.

Approval

Name:	**Shiou-Min Shen**
Degree:	**Master of Engineering**
Title:	***Application of Depth Sensor In The Design of Mobile Robotic Hybrid Gaming Environment***
Examining Committee:	**Chair: Ash Parameswaran**

Shahram Payandeh
Senior Supervisor
Professor

Andrew Rawicz
Supervisor
Professor

Date Defended/Approved: May 21, 2015

Abstract

This project combines depth sensor, virtual game platform, and mobile robots to create an environment where user can engage in a game (i.e. a game of air-hockey) by using simple hand gestures to control physical mobile robots against another user using the virtual counterparts of mobile robots in a virtual environment. The mobile robots move on an open field using DC motors, and each of them is equipped with a unique reflective marker. The overhead camera feeds the image of the field into the game program which utilizes image-processing algorithm to read the positions of the reflective markers and displays the results in the virtual environment. The depth sensor provides the skeleton models of the players which in turn give the hand positions and gestures to the program. Through the combination these information, players can then interface with the virtual environment. In the virtual environment, the real mobile robots which play against virtual robot models of similar design are controlled by the players to move the puck into their respective goal to score. Through this system, users can experience robot sports game in a hybrid gaming environment using real mobile robots and virtual robots.

Keywords: Hybrid gaming, Kinect, Robot, Game

Acknowledgements

I would like to thank my supervisors, Dr. Payandeh for his support throughout my journey. I would also like to thank my colleagues in Experimental Robotics Lab for their help and advice in academic work. I would like to express my gratitude to every teacher and professor that helped me shape my understanding of the world. Finally I would like to thank my parents, relatives, and friends for always being there for me for my entire life.

Table of Contents

List of Figures

List of Acronyms

API	Application Programming Interface
CMOS	Complementary metal-oxide-semiconductor
ERL	Experimental Robotics Lab
FTDI	Future Technology Devices International
LED	Light Emitting Diode
NUI	Natural User Interface
MIROHOT	MIcro RObot HOckey Tournament
MSDN	Microsoft Developer Network
PWM	Pulse Width Modulation
RGB	Red Green Blue
RF	Radio Frequency
SDK	Software Development Kit
SFU	Simon Fraser University
USB	Universal Serial Bus

Chapter 1.

Introduction

Robot is defined as a machine controlled by a computer that can do the work of a person. It usually consists of a combination of effectors, sensors, and a computing system. The term robot was coined by Czech writer Karel Čapek, and it is based on the Czech word robota which means serf labor [33]. In the late 1900's, with booming economy and rising popularity of science fiction, robotic systems have become a popular research field. As technology improves, the computational power is ever increasing while size and power consumption of electronic components are decreasing, and the robotic applications branched into our everyday lives. Mobile robots in particular have varied uses in commercial, industrial, and military settings, and robotic competitions like RoboCup help pushing for advancement in academic research. RoboCup, an annual international robot competition founded in 1997, has a primary focus of using teams of mobile robots to play soccer games [2]. These types of competitions challenge innovation in multiple disciplines including but not limited to electronics, mechanics, computer vision and artificial intelligence.

In Experimental Robotics Lab (ERL) at Simon Fraser University (SFU), there has been a continuous and dedicated effort in developing fist-sized mobile robots to play air-hockey games. The system consists of teams of robots and a puck on a field, an overhead cameras looking down at the playing field, and a processing computer that locates the robots from the camera images and send AI or user input commands to the robots [1]. The system continuously evolves and is often employed in various projects. This project is built on this system to create a platform where these robots can take part in a hybrid gaming environment and interact with virtual objects. Furthermore, a depth sensor is employed so users can manipulate objects in this hybrid game using simple hand gestures.

The report will start by explaining the background of the new concepts to be incorporated into the robotic air-hockey system. The next chapter will go over the design of each hardware component in this project. The chapter after that will explain the integration of the hardware and the software program to complete the hybrid gaming system. Then the results will be presented following by conclusion and future work.

1.1. Mixed Reality Gaming

Mix reality, also known as hybrid reality, combines the world of physical reality with computer generated data to create a visual and audio environment that blend elements of both real and virtual world in real time [19]. Reality is the world where actual physical existence resides, and virtuality is the world generated by computer simulation and defined by its own set of continuum rules including physics. Although people reside in reality, they can immerse in virtuality by transferring sensory information from virtual world through headsets. Figure 1.1 shows that blending of reality and virtuality can be roughly classified into 2 categories. When objects in the real world are enhanced with additional information from the virtual world, it is augmented reality, and when elements in the virtual world are enhanced with information from the real world, it is augmented virtuality. Mixed reality encompasses the field of both augmented reality and augmented virtuality.

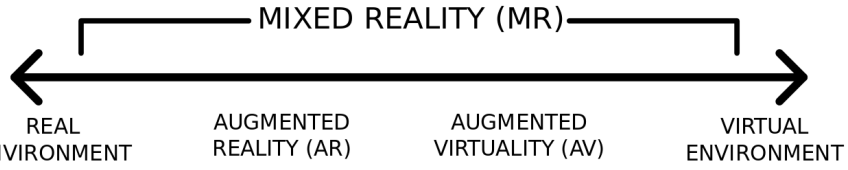
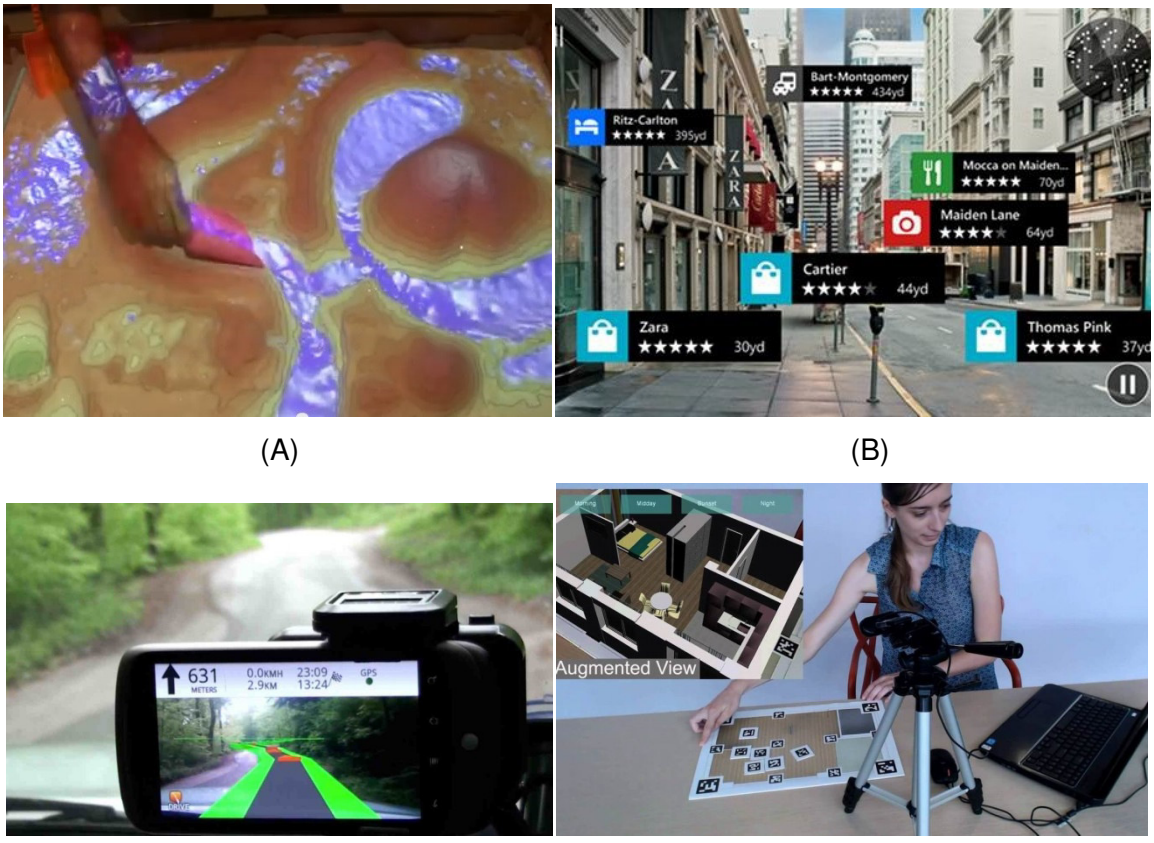


Figure 1.1 Reality-Virtuality Continuum [20]

Mixed reality systems come in many varieties depending on the purpose and the technology. The system consists of an input visual sensor which acquires image of the scene, a computer unit which processes the image, a visual output device which displays the mixed reality, and plus other sensory output devices like speakers and tactile feedbacks. The visual output device includes but not limited to a LCD display, an

optical head mounted display, or an image projector which shines the mixed reality output directly onto the scene.

Figure 1.2 (A) shows an augmented reality sandbox consists of a physical sandbox, Kinect, a projector, and a processing computer. It can generate real-time contour map based on the height of the sands. Its watershed simulation is useful for science education and military strategic planning [31]. Figure 1.2 (B) is a screenshot of Nokia HERE City Lens for Windows Phone, an augmented reality map app that pinpoints shops and restaurants and other location information to the current camera view of the street [25]. Figure 1.2 (C) shows Wikitude Drive, a navigation system which superimposes recommended driving paths on live video feed of the driver's view from smart phones so that instead of drawing driving instructions on abstract maps, the instructions are directly relevant to what the driver sees [32]. Figure 1.2 (D) shows a marker-based mixed reality for interior design. Using image processing to read the orientations and locations of the individual markers and the bottom marker board, the interior design of the virtual room can be generate at real time to help the designer visualize the result [24]. From educational, design and prototype, commercial entertainment to military applications, mix reality enhances everyday gadgets with extra information about the tasks. The potential is limitless.



(A) (B)

(C) (D)

Figure 1.2 (A) Augmented reality sandbox [31] (B) Nokia HERE City Lens [25] (C) Wikitude Drive navigation system [32] (D) Tangible augmented reality for interior design [24]

This project aims to construct a virtual robotic air-hockey environment and bring the actual physical mobile robots to the virtual world to interact with the objects in the game. In this way the virtual world and the real world can be bridged together to build a mixed reality robotic air-hockey game. The inspiration to build a hybrid gaming system came from marker-based mixed reality system in Figure 1.2 (D) where, through the use of cameras and graphic marker patterns, the system can generate extra graphic information on screen to enhance the way user experiences reality. This project applies this concept in a similar way, but with robots in the physical world interacting with virtual objects in an air-hockey game.

1.2. Gesture-based User Interface

Gesture recognition is a field in computer science where technology and mathematical algorithms are employed to interpret the body motions of human gestures [26]. Gesture recognition has wide application in sign language interpretation and consumer electronics, and it is an integral component to the development of natural human-machine interface where the user interacts with a system through intuitive everyday actions [28]. Earliest implementation of gesture recognition employs wearable gloves with arrays of analog sensors on each finger to measure the finger movements. These wearable gloves, categorized as wired gloves, are notably expensive to produce due to the cost of the sensors, and the higher end devices also include haptic feedback to simulate sense of touch [27]. An example of using a wired glove to interact with virtual reality is shown in Figure 1.3 (A). Alternatively, handheld controller-based devices like Wii Remote and PlayStation Move can also detect wrist and body motions through the use of accelerometers and gyroscopes as shown in Figure 1.3 (B). Although handheld controllers cannot detect the fine movements of the fingers, they provide the ability to operate in 3D work space at a much lower cost. Lastly, the combination of cameras and image processing techniques can offer users gesture recognition capabilities without having users carrying any peripherals. The complexity of the image processing varies depending on the type of cameras. For example, it is more difficult to detect gestures with a single 2D color camera compare to an array of 2D cameras because one single camera has limited field of vision and has difficulty measuring distance while a camera array can combine results to produce 3D stereo vision.

Figure 1.3 (A) Interaction with virtual reality using a wired glove [29] (B) Using Wii Remote to play Wii Sports [30]

In the previous implementation of robotic air-hockey system, the users control the robots using standard input devices like keyboard and gamepads, and they directly input the rotation and forward motions of the robots. This type of control scheme requires first person perspective to operate effectively, but the users can only look at the playing field through fixed camera bird's eye view. A more intuitive control scheme is for users to generate the destination locations for individual robots using drag and drop operations similar to the mouse control scheme. Furthermore, for multiple users to perform drag and drop operations on multiple robots simultaneously, the input device needs the ability to register their commands at the same time, and therefore standard computer mouse is not suitable for this task.

To perform a drag-and-drop operation with multiple users, the input device needs to register inputs from multiple users. A gesture recognition system using image processing can accomplish this task. Users' hand and arm motion in open space would translate directly to control the robots, and multiple users can share a single camera without conflict by standing side by side within the view of the camera. There are several ways to implement such a gesture control system. One method is to use webcam in combination with cursors made of color patterns, and users wear the markers using

gloves to move the cursors. However, processing the patterns will be troublesome because hands move in 3D space and will bend the marker patterns and make them difficult to recognize by the camera, and lighting condition and users' clothes can affect the image processing operation.

Depth-camera-based gesture system, on the other hand, has advantages over 2D marker-based gesture system in the area of motion capture. Depth cameras are less sensitive to color and illumination, and therefore users' clothing and lighting of the room do not affect the performance of the system. Depth camera will capture the scene directly without depending on recognition of marker patterns, and then use the depth information to determine the locations and gestures of the users.

Since 2009 when Microsoft first launched their commercial depth sensor, Microsoft Kinect, application of depth sensors have been greatly expanded with the widely available commercial product ranging from gaming to military applications. It plays a major role in many of Xbox best-selling games like Child of Eden and Dance Central series, and it also helps defend the South Korean borders with its motion detection technology [3]. In particular, it also has many applications in the field of robotics. For example, its skeleton tracking capability can be used to control a humanoid robot to mimic the motion of the user [4] as shown in Figure 1.4; this is useful for carrying out tasks in environment where humans cannot be physically preset, including handling of hazardous materials. In addition, it can also act as the vision system for the navigation of autonomous mobile robots [5] as shown in Figure 1.5. The depth sensor gives autonomous robots the ability to measure distances, avoid obstacles, and map out the area while navigating in an indoor environment.

Figure 1.4 Mapping body pose to anthropomorphic robot through the use of Kinect skeleton tracking [4]

Figure 1.5 (A) Autonomous surveillance robot (B) Depth camera view and decision window [5]

Furthermore, the Kinect for Windows SDK provides powerful API with extensive documentation. Microsoft also employs dedicated support team in Microsoft Developer Network (MSDN) forum, so questions for their products are answered in a timely fashion. For the purpose of this project, Microsoft Kinect is the best choice to build a user interface for gesture inputs.

The next chapter lays out the structure of the physical system in this project. Then it will go over the design of each hardware component of the system. The robotic air-hockey system structure will come first, and follow by the Kinect device.

Chapter 2.

System Hardware Structure

In the physical setup of the system, the game involves 2 mobile robots, a digital camera placed on top of the field to get the overhead view of the field, a Kinect motion sensor, and a computer where the virtual world takes place. The mobile robots roam on the playing field while the camera, positioned above the field, records an image of the field with robots in it and sends it to the computer. The computer then processes the image and uses the result to determine the speed the robots need to reach their destinations. At the same time, the Kinect records the motion of the player and sends it to the computer. Then the computer extracts the player gestures to update the status of the game and the destinations of the robots. The relationship between the components of the system is shown in Figure 2.1, and the system in action is shown in Figure 2.2.

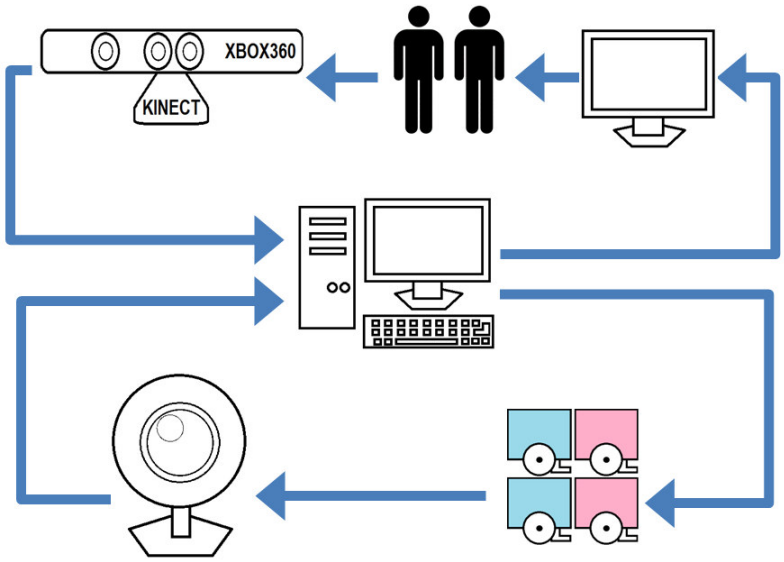


Figure 2.1 System flow

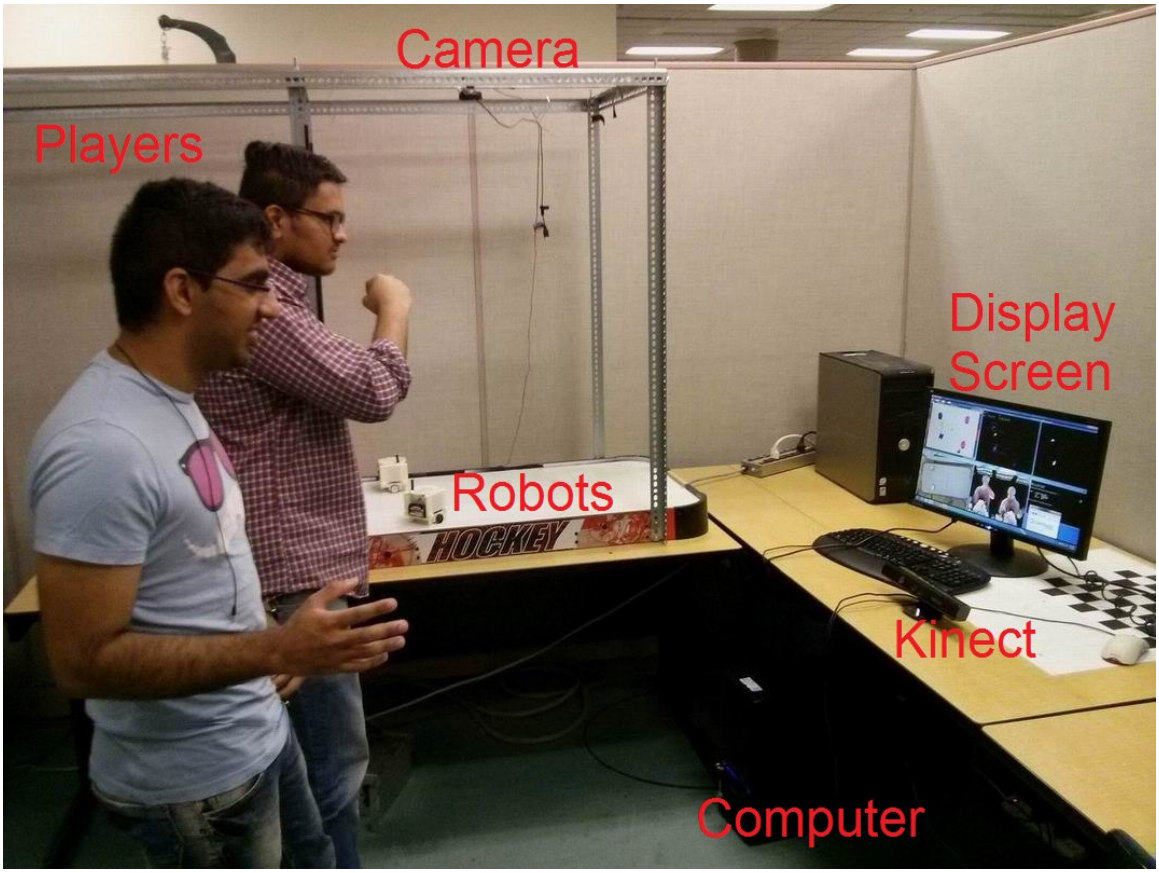


Figure 2.2 Hybrid robotic air-hockey game in action

2.1. Robot Control Process

The computer interacts with the robots by first looking at where the robots are and then tell them which direction to go. The positions of the mobile robots are monitored through the digital camera which is mounted on a steel frame above the air-hockey field, and then the system acquires images from the digital camera using OpenCV computer vision library. The camera is required to capture at 640x480 resolutions at 30 frames per second, and most modern webcams are capable of this operation. The current system for distinguishing robots employs polygon shapes made of reflective tapes in combination with bright white LED light source placed beside the overhead digital camera (See Figure 2.3 for the set up of the camera). Each robot has a unique pattern, and they are differentiated by the number of sides in the reflective

pattern as shown in Figure 2.4. The combination of LED lights and reflective patterns produces highly visible markers which are very useful for image segmentation, and detail explanation for this is in Section 3.2.

(A) (B)

Figure 2.3 Overhead digital camera setup on air-hockey field

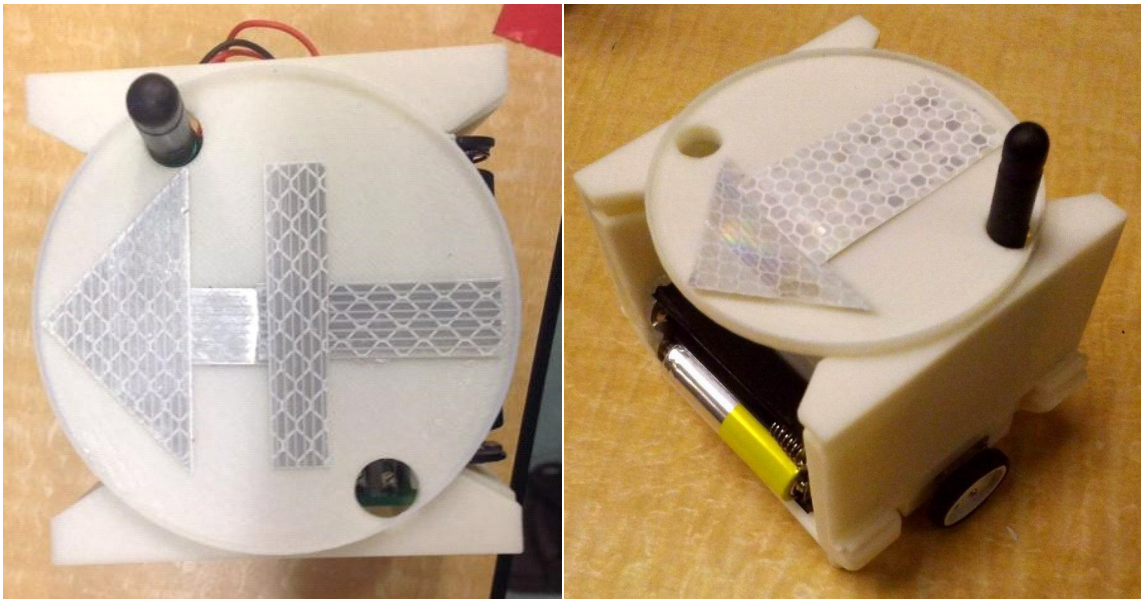
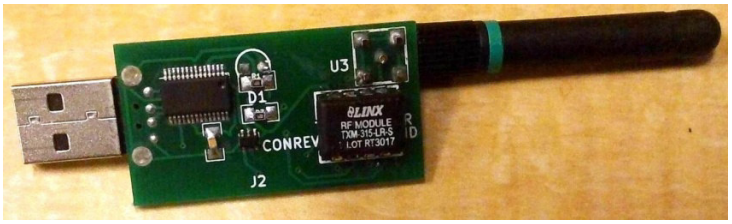


Figure 2.4 Reflective patterns for mobile robots

After the computer uses an image-processing algorithm to determine the positions of the robots and calculated their paths, it sends the robot motor control commands through the radio transmitter. The transmitter board, shown in Figure 2.5 (A), uses TXM-315-LR serial RF transmitter which transmits at 315MHz, and because the transmitter sends data using serial protocol, FT232RL is employed for serial-to-USB signal conversion. The circuit diagram of the transmitter board is shown in Figure 2.6. The computer talks to the transmitter board through the USB socket using FTDI driver and library by Future Technology Devices International (FTDI)[6]. The communication between the computer and the robots is one-way only with no feedback, so the transmission package includes a checksum byte as shown in Figure 2.5 (B). The **start** byte indicates the beginning of a new data package. The **ID** byte designates which robot to execute the command. The **cmd** byte is used to activate shooting command. **V** and **w** are linear and angular velocities respectively. And the last part of the package, the **checksum**, is calculated from these parameters. The computer constantly transmits new commands, and the robots continuously execute the last correct commands until they receive the new commands with the correct checksums[6]. The design specification of the robots is mentioned in the following sub-section.



data package
+start : unsigned char +ID : unsigned char +cmd : unsigned char +v : signed short +w : signed short +checksum : unsigned char

Figure 2.5 (A) USB Transmitter Board (B) Data package [6]

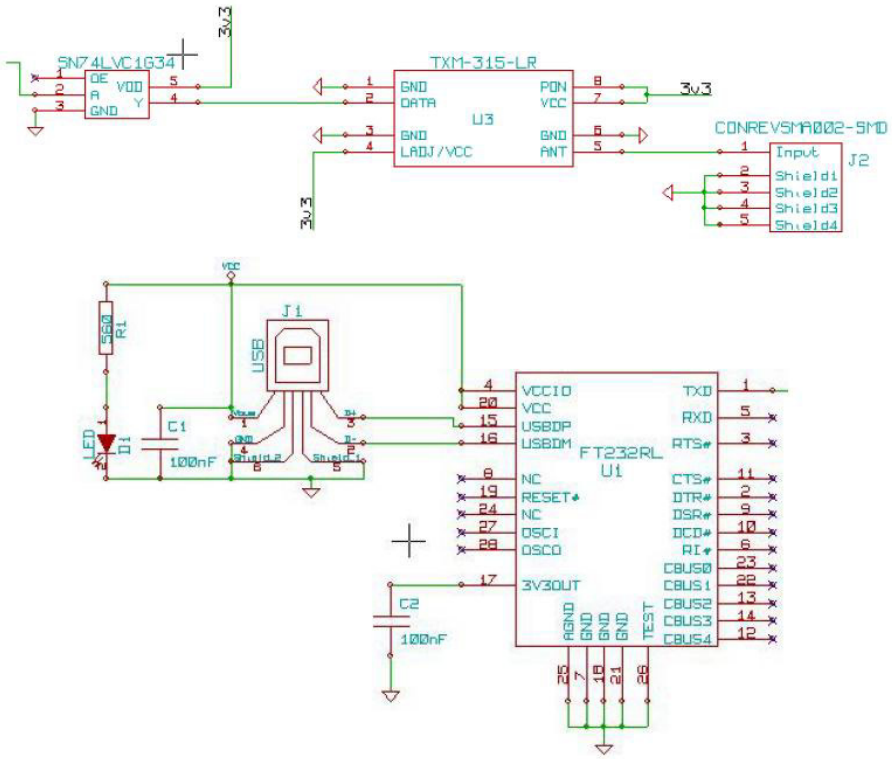


Figure 2.6 Transmitter circuit [6]

2.1.1. Mobile Robot

The mobile robots, shown in Figure 2.7, are designed to perform 3 tasks, to receive and interpret commands from users, to move from an initial position to a final destination, and to shoot the puck. The robots receive the commands using a one way RXM-315-LR receiver chip at 315 MHz located on a board separate from the processing unit (see Figure 2.8). The received message which includes the angular and linear

velocity and the shoot command is deciphered by the dsPIC30F2010 microcontroller, and the microcontroller (see Figure 2.9) generates the Pulse Width Modulation (PWM) signals for the motors. There are 2 DC motors to drive the 2 wheels of the robot, and they are actuated by an H-bridge (see Figure 2.10) when the H-bridge receives the PWM signals from the microcontroller. The shooting mechanism consists of a single shooting motor, and the microcontroller generates the PWM signals for it when a shooting command is received, but the shooting function is not used in this project. Figure 2.11 shows interconnection of the major building blocks of the mobile robot components.

The mobile robots have the dimension of 9x9x9cm and run on 4 AA batteries. Using a set of 2 DC motors to drive the wheels, each robot can achieve a maximum speed of 20cm/s and angular speed of 114 degrees/s. The frames of the robots are designed using Solidworks and then built using rapid prototype machine. The frames are made of plastic and can be assembled with super glue. The full specifications of the robots are listed in [6].

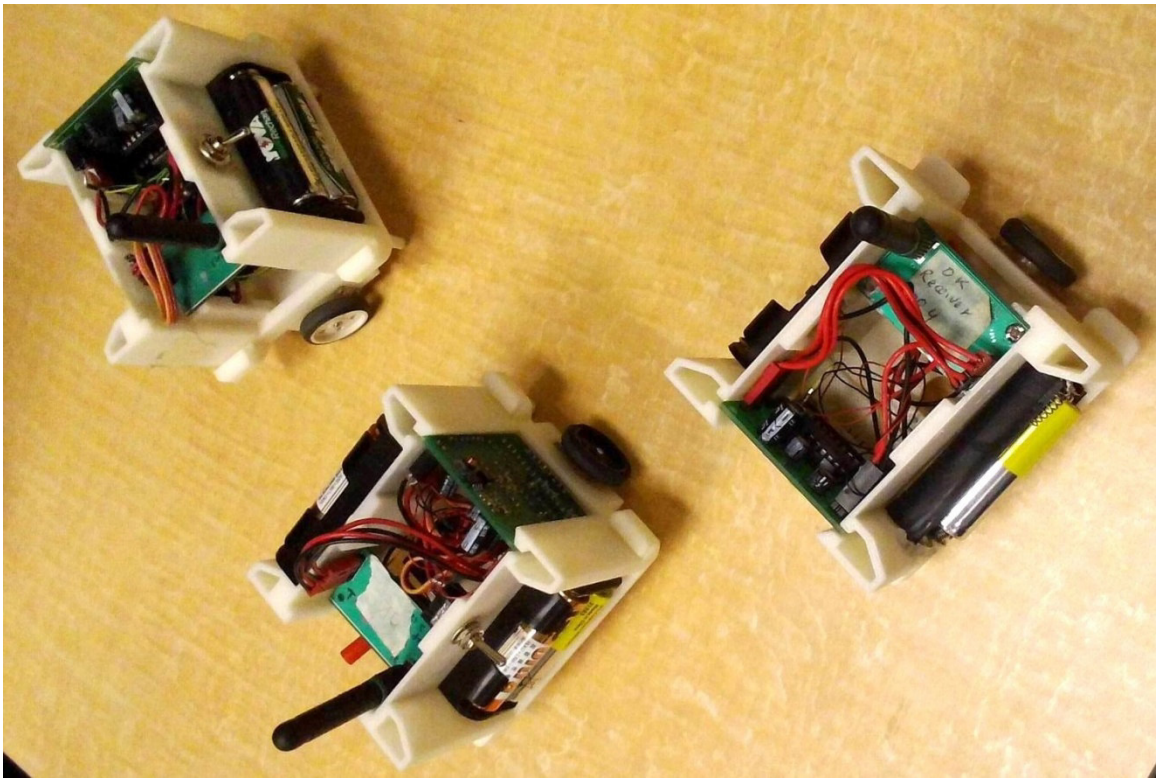


Figure 2.7 Mobile robots

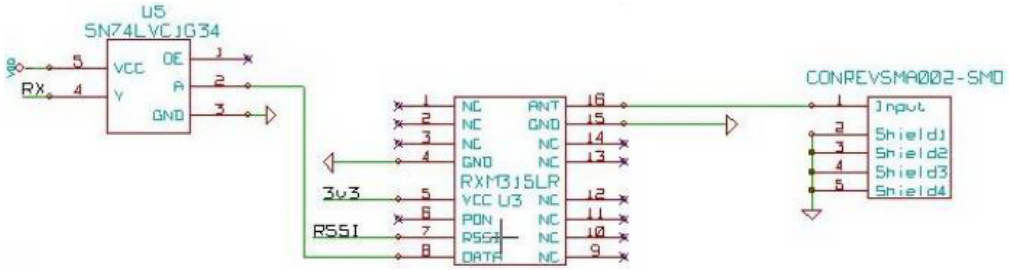


Figure 2.8 Receiver circuit

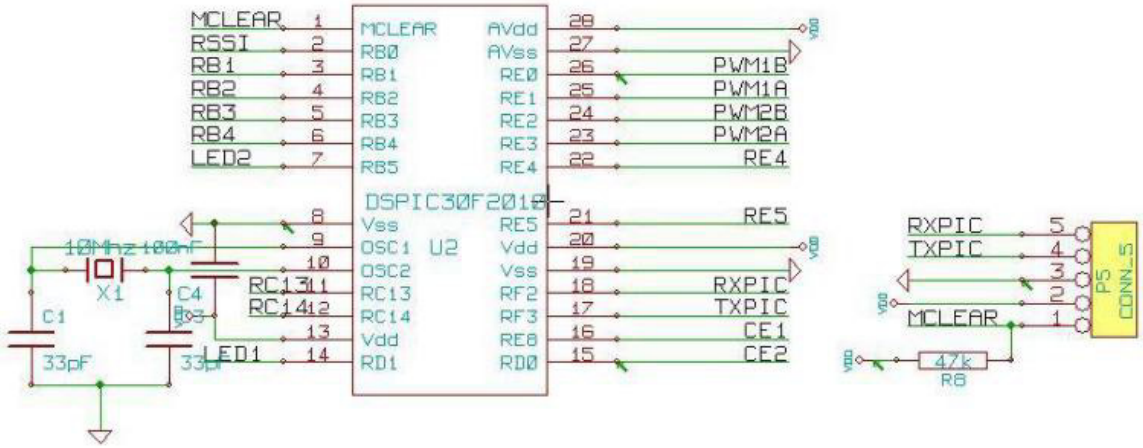


Figure 2.9 Microcontroller circuit

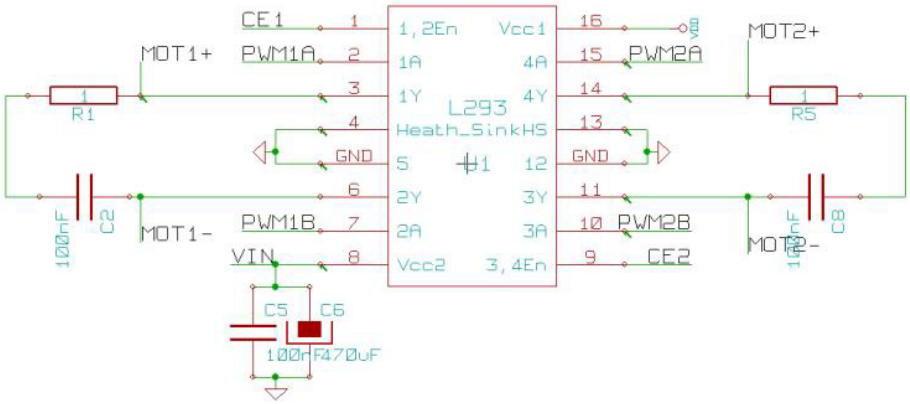


Figure 2.10 H-bridge circuit

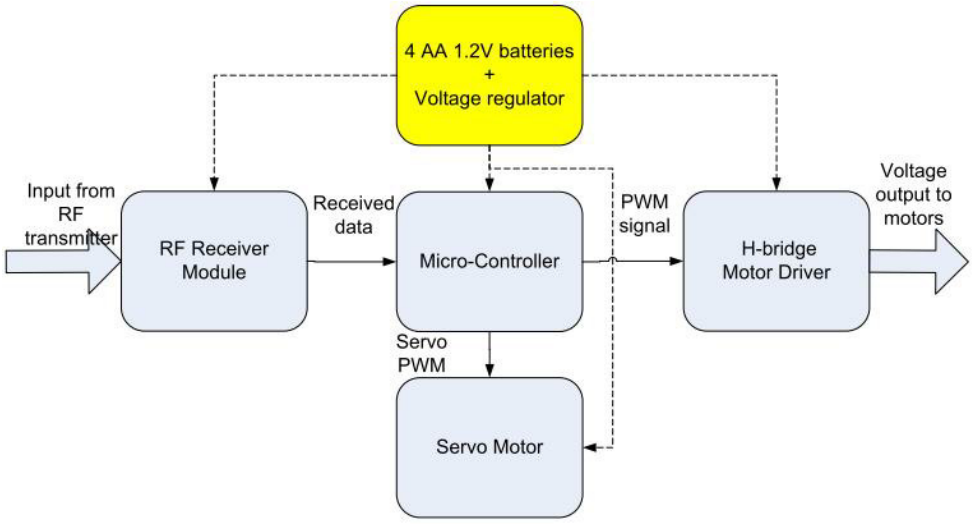


Figure 2.11 Building blocks mobile robots

2.2. User Input Process

The users determine the paths the robots travel by analyzing the current progress of the game, and then input their decisions to the game through the input device. Since the game takes place in the virtual space, the computer needs to display the positions of the robots and the ball and game control options to the users, and then wait for user inputs. In this particular setup, the computer relays the game information to the users through a user interface program on the monitor, and the user interface program is built using OpenCV and OpenGL libraries (see Chapter 3). The users input commands in the form of hand gestures, and they are detected using Microsoft Kinect motion sensor which acts as the user input device. The computer communicates with Kinect through the drivers in Microsoft Kinect SDK. The structure of Kinect is explained in the following subsection.

2.2.1. Kinect Motion Sensor

Since Kinect's debut in 2009, several versions of Kinect optimized for different purposes have been introduced in the market, but in general, Kinect devices usually

consist of 5 components; a RGB camera, an infrared emitter, a monochrome CMOS infrared sensor, an array of 4 microphones, and a tile motor as shown in Figure 2.12. The RGB camera operates similar to a regular webcam and can stream videos at a resolution of 640 by 480 at 30 frames per second. However, the main feature of Kinect lies in the infrared emitter and sensor pair. The infrared emitter, placed on one end of Kinect, sends out infrared lights of dotted encoding pattern (see Figure 2.13 (A)). Objects at various distances are covered with the light pattern, and the brightness of the dots increases as distance decreases. Then the infrared sensor, located near the center of Kinect, detects the unique light patterns on the objects and triangulates them using parallax, so the onboard processor can determine the depth of each object [7][8], as shown in Figure 2.13 (B).

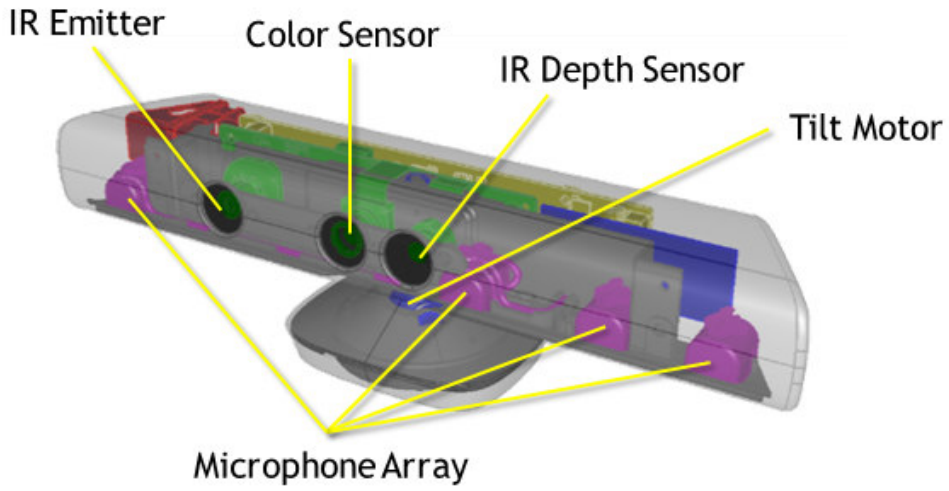
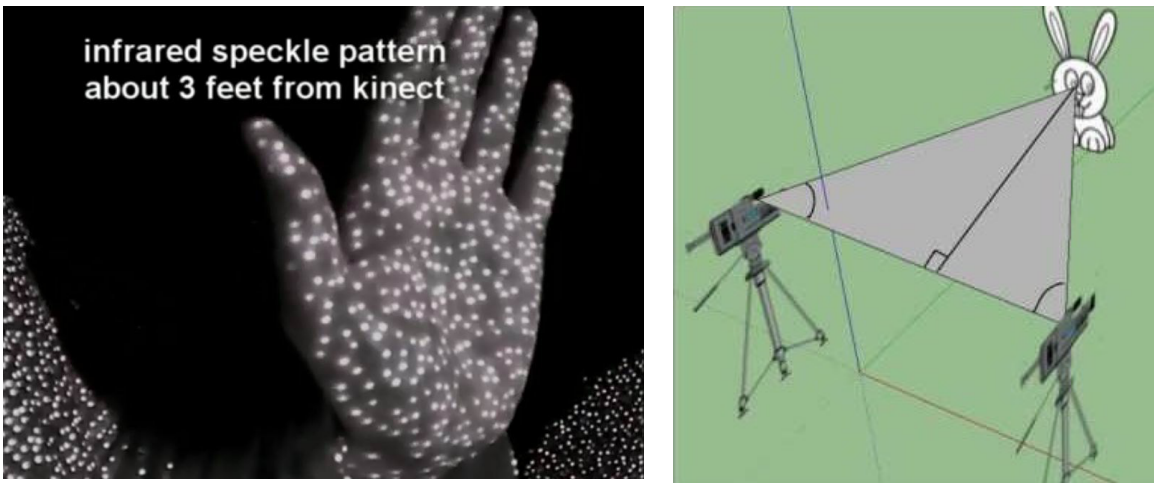


Figure 2.12 Kinect hardware components [9], used with permission from Microsoft

Kinect is capable of tracking a maximum of 6 people within its view but only provides full body skeleton models for the 2 primary players. The generated skeleton models consist of 20 joints each. It can capture depth images at 320x240 and 640x4880 resolutions, and its optimum operating range is between 1.2m and 3.5m [10].

In the next chapter, the role of each piece of hardware in the program will be explained. Each device is a crucial component to this game. The combination of hardware and software is what makes hybrid robotic air-hockey game possible.



(A) (B)

Figure 2.13 (A) Pattern output from Kinect infrared projector (B) Depth distance calculation using triangulation [7]

Chapter 3.

System Program Structure

In the program that facilitates the robotic air-hockey game in the virtual world, OpenCV is used to acquire images from the RGB webcam, and Kinect for Windows SDK is used to communicate with the Kinect depth camera. Using the functionalities of the API in the libraries, the positions of the robots and the users can be obtained from the acquired images. With the information of the robots and users known, the game can decipher player's action with respect to the game interface and update the robot's positions accordingly. Figure 3.1 describes this process, and the specifics of the program are described in the subsections. Since the software development of the MIROHOT project uses exclusively C and C++, naturally C++ is the best choice to develop this project, and Visual Studio 2010 is used to build it.

The program takes 2 inputs, the users' gestures captured by the depth sensor and positions and orientations of the robots captured by the digital camera. The processing of the depth image and digital camera image extracts the current user inputs and robot position data. Finally the program outputs the data on the virtual environment and transmits the motor control data to the robots. The program interface screens are shown in Figure 3.2.

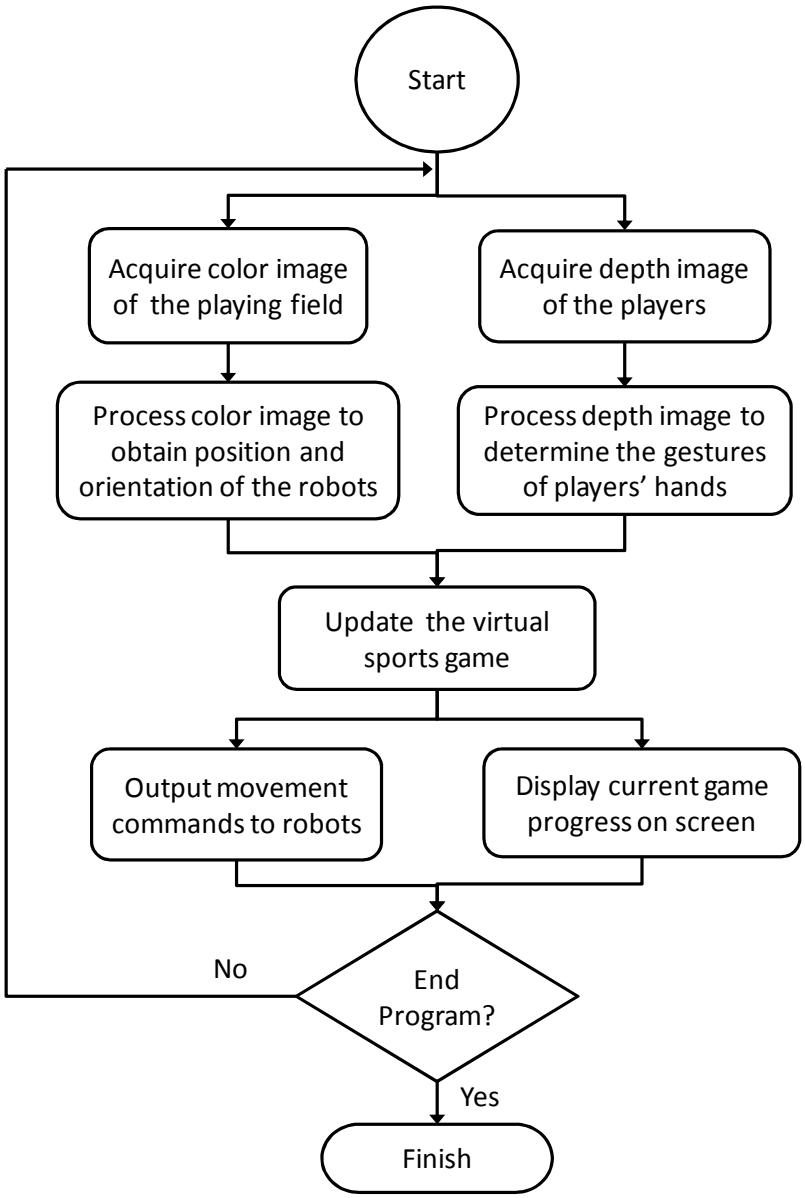


Figure 3.1 Program flow chart

Figure 3.2 Program interface screens

3.1. Kinect for Windows SDK

There are several options that a user can employ to access a Kinect device on a PC. There is a fully open source library called OpenKinect, but it cannot make use of the full capabilities of the Kinect device. There is a private library called OpenNI maintained by PrimeSense, the company that developed the technology inside Kinect, but the library is now defunct after Apple's acquisition of PrimeSense. Finally, Microsoft released the official all-inclusive software development kit (SDK) called Kinect for Windows SDK. It offers full access to Kinect's functionalities in its application programming interface (API), and it includes samples in various languages and extensive documentations. Most important of all, active user developers and technical support teams on the MSDN

forums provide support and discussion in a timely fashion. For the aforementioned reasons, Kinect for Windows SDK version 1.8 is used in this project.

For the purpose of interacting with the robots using gestures, the program needs to be able to track the hand positions and gestures from multiple users. The SDK provides 2 sets of API to accomplish this task, Natural User Interface (NUI), and KinectInteraction. NUI provides color and depth video streaming, real time body tracking, and skeleton generation as shown in Figure 3.3. KinectInteraction processes the information from NUI and outputs the status of users' hands.

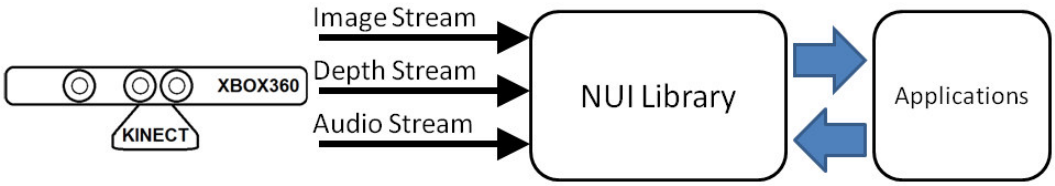


Figure 3.3 Kinect Hardware and Software Interaction with an Application [21]

3.1.1. NUI API

NUI, the core of the Kinect SDK, allows access to the basic functions of the Kinect device. Other Kinect API's require this component to run before they can execute their functions. In this project, NUI is used to facilitate depth video stream and skeleton generation.

The Kinect SDK is multi-threaded and event-driven. To access Kinect device with NUI, an object for the Kinect device is created and initialized. Then a handle for each stream and its event is created and initialized like the following.

```
INuiSensor* sensor;
HANDLE depthEvent, depthHandle, skeletonEvent
NuiCreateSensorByIndex( 0, &sensor ); //Open default sensor
sensor->NuiInitialize();
depthEvent = CreateEvent();
```

```
sensor-> NuiImageStreamOpen ( y,y,y,y, depthHandle ); //y's are the setting parameters
skeletonEvent = CreateEvent();
sensor->NuiSkeletonTrackingEnable( skeletonEvent,0);
```

After the initialization step is done, the main loop of the program requires updates from the streams. In every cycle of the loop, the program checks if the streams invoke an event flag. If an event does occur, the program will execute the processing function of the respective stream. The depth event is always occurring because depth stream is constantly acquiring video.

```
If (WAIT_OBJECT_0 == WaitForSingleObject( (x)Event, 0))
{ Process(x)Function (); }
```
Where Process(x)Function is the user specified function to process data for a specific event (x)

The main tasks of the processing functions for depth stream and skeleton generation are to capture the current frame and visualize the depth data and user skeletons for the players. The captured frame is sent into interaction stream instance so it can analyze the depth frame and skeleton data and identify an interaction.

When Kinect streams the depth video of the scene, the stream format consists of 2 parts, the depth data and the player index number. The depth data is the detected depth values from the depth sensor in millimeters. Kinect internally segments the depth data to identify up to 6 players in the scene, and assign a player index number of 1 to 6 for each depth bit belonging to the player, and 0 if there is no one in the scene [11].

The incoming depth stream needs to be processed to generate the skeleton models for the players. To build the skeletons, first the body poses must be recognized so the limbs, joints, and body parts are labeled correctly, the SDK accomplishes this take using a built-in training set of 1 million samples. It uses a tree-like decision model to match the depth data with the training set [12][13].

Figure 3.4 shows the skeleton generated from the depth data. The red lines are the visible body bones, and the blue lines are the inferred body bones. Because the SDK

creates a skeleton for whole body, it also tries to infer the location of the limbs that are not detected by the camera. This capability allows it to guess the posture the player is currently in.

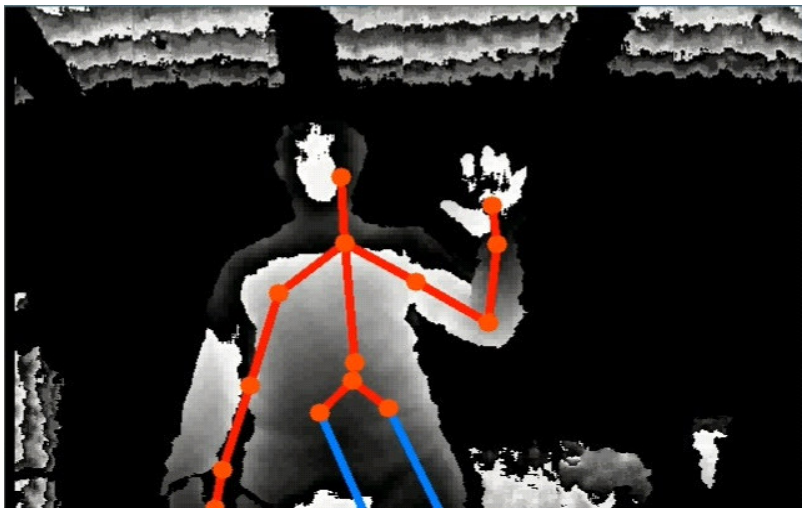


Figure 3.4 Depth video and player skeleton

3.1.2. KinectInteraction API

From depth video streaming to the skeleton generation are all accomplished using the NUI API. However, the core of the SDK does not process the gestures of the hands, and KinectInteraction API is used to obtain the gesture information. Because C++ is the language used to write this program, Kinect SDK does not provide higher level libraries for gestures and interactions in this language. KinectInteraction library is the only library available in C++ for gesture recognition (see Figure 3.5), and there is also no sample code for this API. However, the API is used in a similar fashion as NUI. It is event driven and is used in conjunction with NUI.

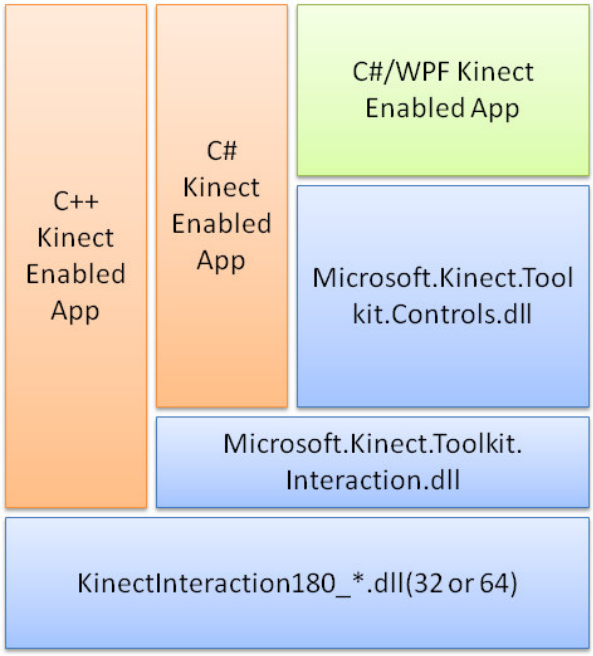


Figure 3.5 KinectInteraction Architecture [22]

The library requires its own instance and initiation separate from NUI, and it also requires an event handler to handle events like in the following.

```
INuiInteractionStream* interactionStream;
CIneractionClient interactionClient;
HANDLE interactionEvent;
interactionClient = new CIneractionClient();
NuiCreateInteractionStream();
interactionEvent = CreateEvent();
interactionStream->Enable(interactionEvent);
```

The INuiInteractionStream object is required by the processing function for depth stream and skeleton generation, so these functions can pass the latest frame to the interaction stream and let it analyze the data and identify an interaction event.

During each loop, the program checks to see if the interaction stream invokes an event flag. If an event does occur, the program will execute the function to process

interaction. Interaction event only occurs when the hands change positions or change gestures.

```
if(WAIT_OBJECT_0 == WaitForSingleObject(interactionEvent, 0))
{ ProcessInteractionFunction(); }
```

The processing function for interaction uses the interaction data to update the positions and gestures of players' left and right hands.

KinectInteraction analyzes the depth data and the skeleton models of the depth data to track the hands of the users. The API can detect 2 types of gestures, pressing and gripping motion, for both hands from the first 2 players in the scene. It also rescales the positions of the hands relative to the shoulder joint, and then normalizes the position values to a range of 0 to 1 [14].

The information on the inner working of the KinectInteraction API is not available, but it seems to analyze the size of the hand to detect open and closed palm. The sample code for this API is also not present in the SDK, and this part is built based on the combination of SDK sample programs and code snippets from the MSDN forum [15][16].

3.2. Image-processing algorithm

As robots roam the playing field, the overhead digital camera delivers the stream of 640 x 480 RGB color video to the computer, and then image-processing techniques are employed to obtain the positions and orientations of the robots in RGB color images. The general outline of the image processing steps is shown in Figure 3.6.

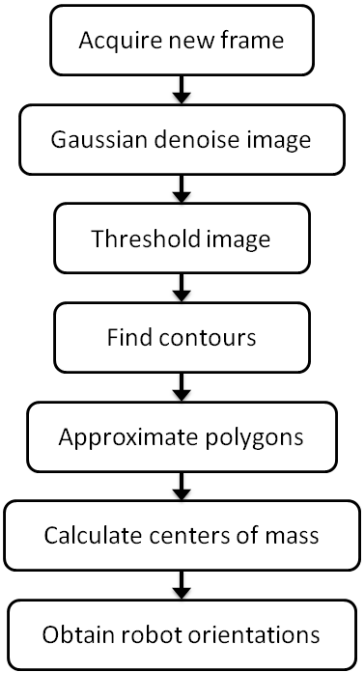
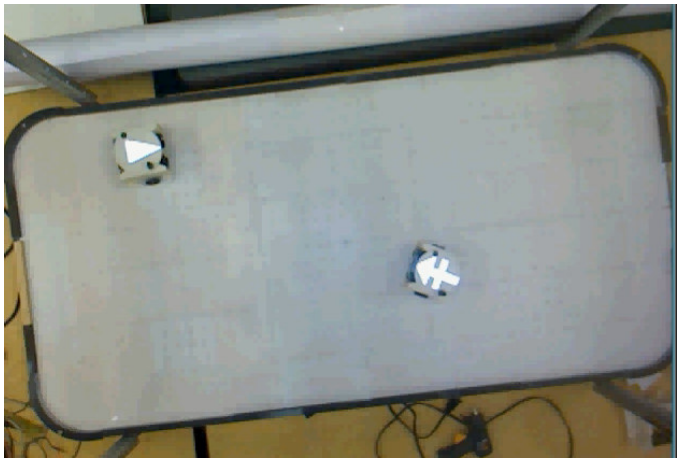


Figure 3.6 Image processing flowchart

The program first establishes a connection to the default camera using the following constructor from OpenCV

```
cv::VideoCapture cap(0);
```

Recall the method for distinguishing robots as mentioned in Chapter 2 where the camera has LEG lights installed beside them, and the robots have reflective patterns on top of them. When LED light shines on the reflective tapes, the camera will read the colors of the tapes as the largest number that the camera could perceive. This simplifies image segmentation to basic thresholding which makes it much easier to remove background image and output a binary image. Figure 3.7 (B) shows the result of thresholding the camera image in Figure 3.7 (A), and this is done using the **threshold**() function in OpenCV.



(A)

(B) (C)

Figure 3.7 (A) Camera image of robots on the air-hockey field (B) Threshold image (C) Contour search result

From the segmented image, the locations of the pixel blobs are the locations of the robots. The patterns of the reflector tapes differentiate the robots from one another, and the angles at which the patterns are oriented in the image correspond to the orientation of the robots. To determine the location of the robots, connected component analysis, **findContours**(), is applied on the segmented image to find the locations of the contours in the binary image, and then polygon-approximation algorithm, **approxPolyDP**(), is applied to simplify the shape of these contours into a smaller set of vertices. The result of connected component analysis is shown in Figure 3.7 (C), and an example output of polygon approximation is shown in Figure 3.8.

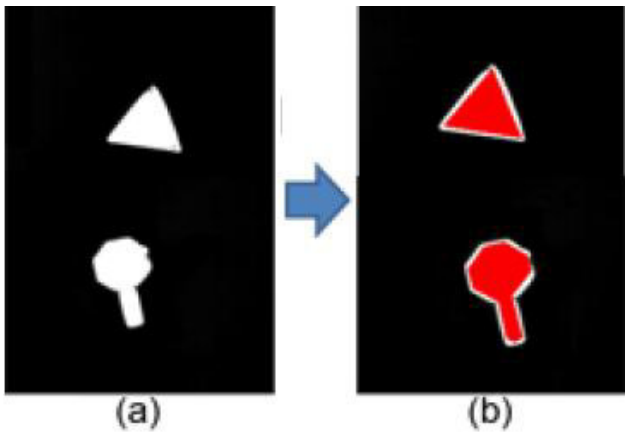


Figure 3.8 (A) Camera image (B) Red polygons are the result of polygon approximation [8]

From the approximated polygons of these pixel blobs, image central moments can be calculated with **moments**(). The equation of image moments is

$$M_{pq} = \sum_X \sum_Y x^p y^q f(x,y),$$

but the moments are calculated with translated centers for the pixel blobs. Central moments normalize the translations by using image centroids, so the moments become translation-invariant. The equation for central moments becomes

$$\mu_{pq} = \sum_X \sum_Y (x-\bar{x})^p (y-\bar{y})^q f(x,y), \tag{3-1}$$

and the center of mass can be calculated with the following,

$$(\bar{x},\bar{y}) = \left(\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right). \tag{3-2}$$

To obtain the angles of robot orientations, recall that the reflective patterns are designed so that the longest lines away from the centers of the patterns point to the front of the robots, and **approxPolyDP**() produces the vertices of the reflective polygons. Comparing the distances of the polygon vertices and the centers of masses will yield the direction of the longest line and therefore the direction that the robot is facing as shown

in Figure 3.9. The algorithm is based on Appendix C of [8] and is implemented using the OpenCV library.

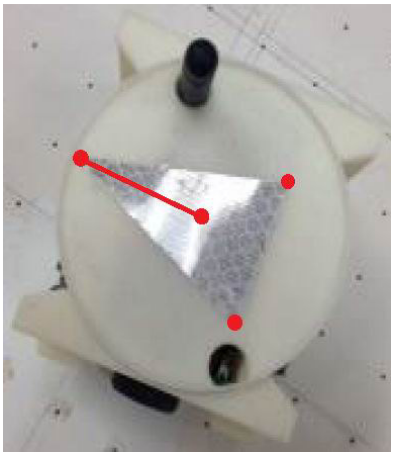


Figure 3.9 Robot orientation calculation

3.3. Virtual World Setup

The goal of this project is to allow players and physical robots to interact with virtual objects in the virtual world. Kinect depth sensor and gesture recognition achieve the former, while digital camera and image processing achieve the latter. With the information of players and robots available to the system, the bridge between the real world and the virtual world can be built.

Recall the flowchart of the program in Figure 3.1, in the main loop of the program, first the image acquisition and processing for the air-hockey field camera are performed so the robots can be located. At the same time, the program checks the event handlers of the Kinect SDK and process the data to locate the players. The update of the virtual world comes at the last step after the 2 inputs from the real world are processed. In this final step, the interactions between the users, the robots, and the game are sorted out so the objects in the game can update from time t to t+1.

There are 3 basic objects in the game, the hands, the robots, and the puck. The hand objects store the status of the hands from KinectInteraction and track the ID of

robots that are being controlled by the hands. The robot objects store the positions, orientations, and kinematic profiles of the robots. This class is used to store the information of both real and virtual robots. The puck object stores the position and its kinematic profile, and its singleton instance exists only on the virtual plane. On top of these 3 types, there are 2 classes that are composed of these basic elements. A user is defined by its 2 hands, and a robot team consists of 2 robots. By assigning real robots to 1 team and virtual robots to the other, one user would take control of the real team and the other would control the virtual team respectively. Recall that Kinect for Windows SDK supports both gripping and pressing gestures, but only gripping gesture is employed in this project because it's intuitive to the user and is less ambiguous to the program. It is difficult for the user to reach the maximum work space while maintaining pressing gesture at the same time. The class diagram showing the relationship of the classes is shown in Figure 3.10.

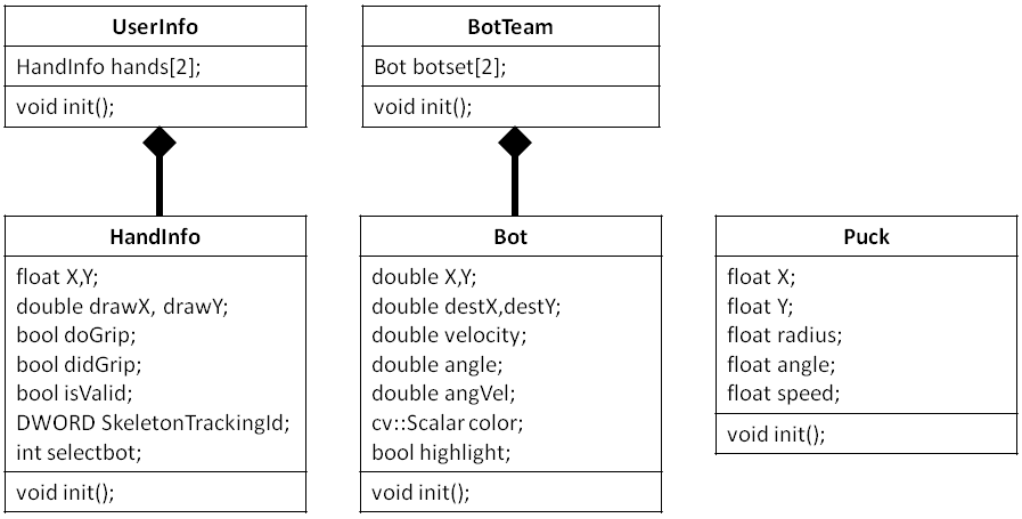


Figure 3.10 Class diagram for virtual world

The virtual world has 4 interactive elements, the virtual robots, the puck, the environment, and the game menu. The virtual robots mimic the real robots by using the same tank travel patterns. When a new destination is set for both the real and virtual robots, they rotate until they line up with the destination angle and then travel in a straight line to the designated location. Because they use the same parameters, virtual and real robots share the same data type. The virtual puck interacts with the objects in

the virtual world through collision. The collision response is assumed to be elastic so the puck bounces off the wall at the same speed with the same reflected angle (see Figure 3.11, and a constant friction force is applied on the puck when it slides on the virtual field. The puck is assumed to have no rotational component in its motion so that collision response does not involve torque. Furthermore, even though the virtual world is 3 dimensional, the objects only traverse on the XY plane which effectively makes the air-hockey game 2D. These further simplify the collision calculation when the puck collides with the 4 walls of the air-hockey field, which act as the absolute boundaries for both robots and the puck.

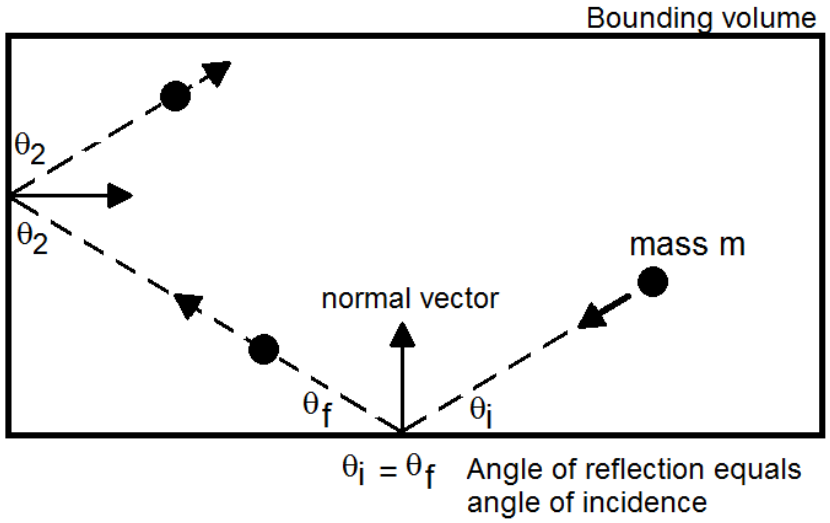


Figure 3.11 Elastic collision in 2D

The puck also collides with the robots, and the collision can be detected by checking the penetration distance. When the robots are stationary, the collision calculation is similar to the collision with the walls. However, when the puck collides with a moving robot, the result is modeled as the vector addition of two vectors. The first vector is the speed and direction the puck would end up with if it had hit a stationary robot. The second vector simulates the extra impact force caused by the robot's motion, and is calculated by multiplying the robot's motion vector by a constant. An example of this collision type is shown in Figure 3.12.

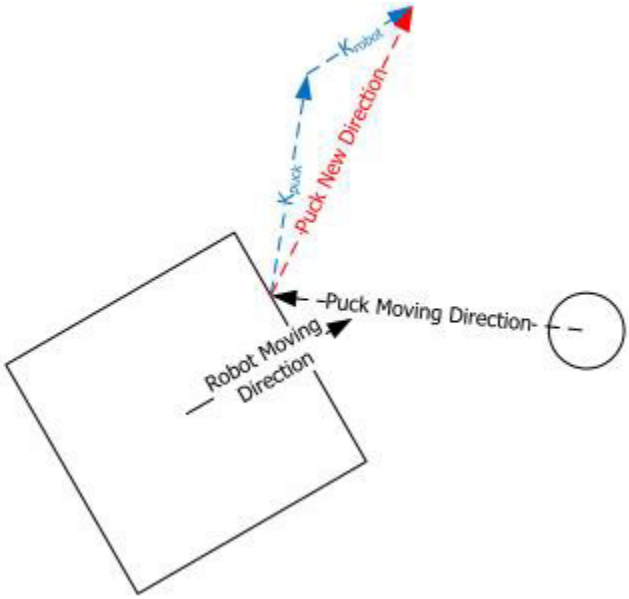


Figure 3.12 Puck Colliding with a Moving Robot

The collision response between robots is implemented as full stop for both robots. Following a collision, the robots must rotate away from the robot it collided with so that the angle Θ (see Figure 3.13) is large enough for it to move around the collision. This implementation is reasonable for collisions between real and virtual robots because the max travel speed of the robots is not fast.

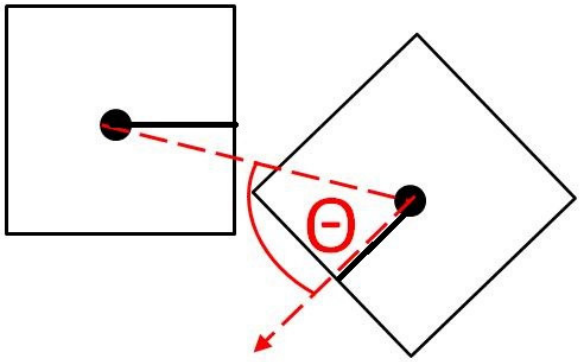
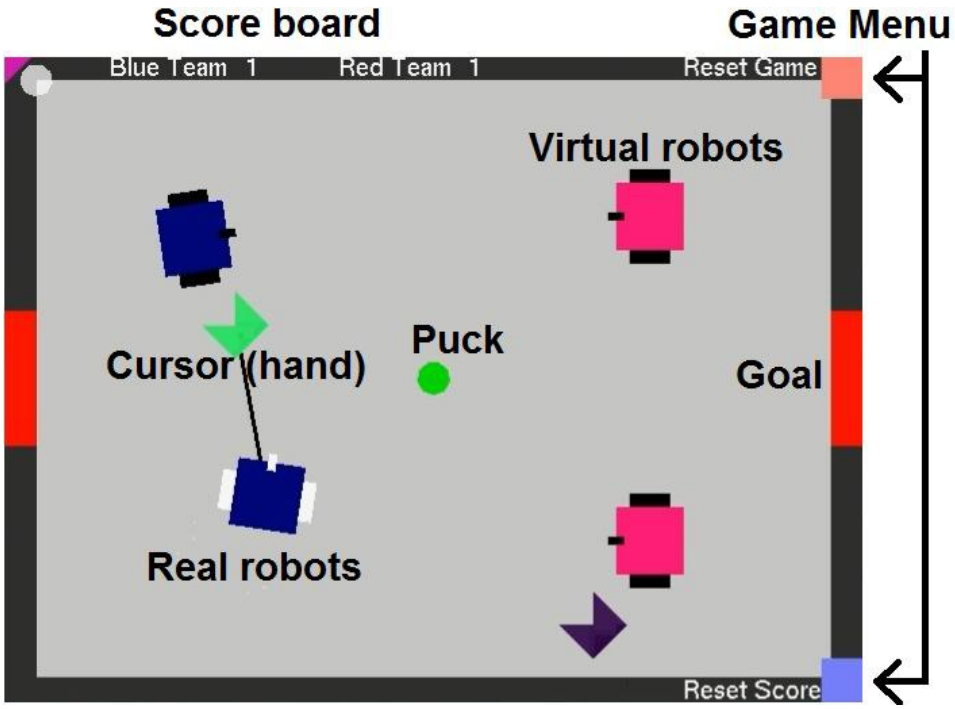


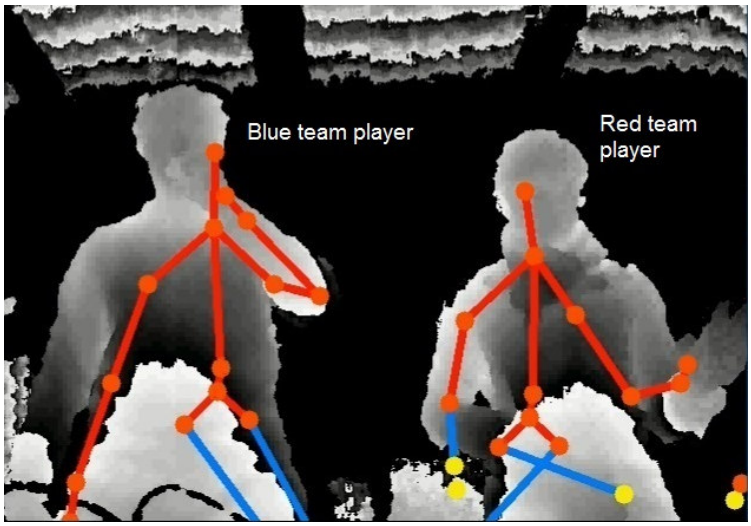
Figure 3.13 Angle between colliding robots

Finally, the last part of the virtual world is the game menu. The game menu tracks the scores of the teams and includes buttons to reset the positions of all virtual

objects and the scores. The buttons are activated by moving the hand cursors on top of the buttons and grip them. The user interface is drawn using OpenGL, and it is shown in Figure 3.14.



(A)



(B)

Figure 3.14 (A) Game display screen (B) Player position screen

The players interact with the virtual objects by using gestures with their hands. The person standing on the left controls the blue team, and the person on the right controls the red team. If there are more than 3 people in the scene, the first two visible persons from the left become the players. The game is set up to be controlled using only the players' right hands so the interface would not be cluttered with cursors. When the right hand of a player is open, it does not interact with anything in the scene, and if the hand was interacting with anything previously, the interaction will stop. If a player overlaps right hand on a robot of his team and change the hand to closed fist, a drag and drop operation is engaged, and the robot's destination will change to the current position of the player's right hand. The destination will keep updating until player opens the hand, and then the robot will continue to move toward its last recorded destination.

In the same manner as the regular air-hockey game, the players bring the puck to the goal on the opponent's side to score. The goals act as special boundaries so that when the puck reaches the goal areas, the positions of all virtual objects reset and the score counter for the scoring team is increased by 1. The process is shown in Figure 3.15. Because there are situations where the robots and the puck could get stuck in a corner, the position reset button is necessary.

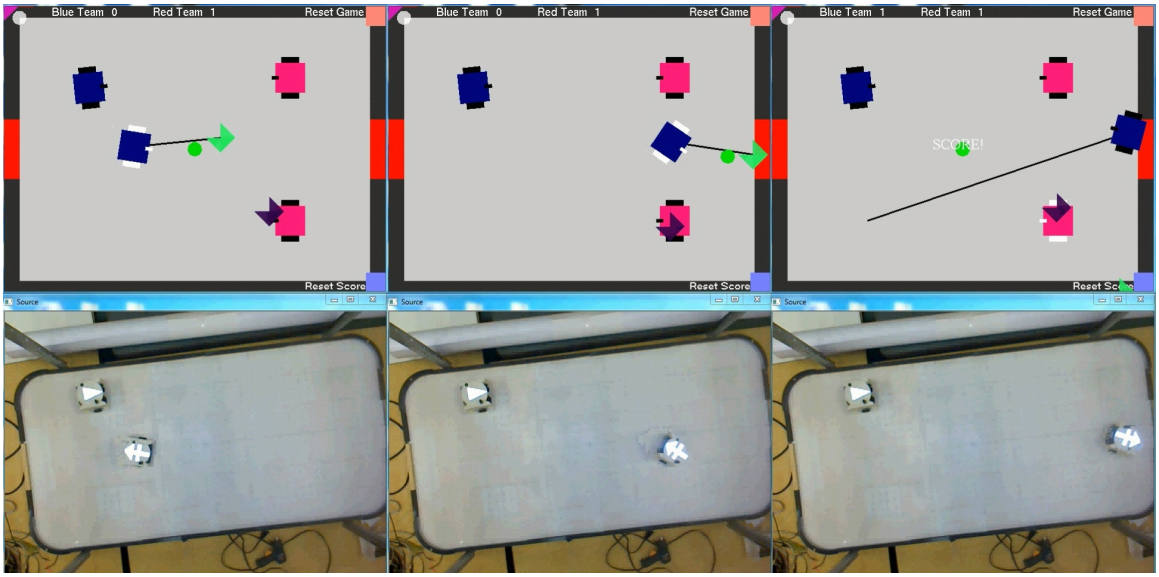


Figure 3.15 Real robot carrying the ball and scoring

Chapter 4.

Results and Discussion

Building on previous works and incorporating new technology, a peripheral-free hybrid robotic air-hockey game is built. A player can use simple gestures to control a team of 2 robots physically constructed in reality and play against another player who controls 2 robots that reside in the virtual space. The virtual robots serve as the ideal case for the real robots to compare with. The objects in real space can affect objects in the virtual space, and objects in the virtual space can affect objects in the real space. Within the boundaries of this robotic air-hockey game, reality and virtuality are successfully bridged together.

The cameras operate at 30 frames per second which means they take 33 milliseconds to capture one frame. The program runs at an average of 27 milliseconds per loop, which is less than the refresh rate of the cameras. Because the program can produce its results before the next frame arrives, it does not lose track of the robots in any given frame.

However, playing through the game reveals many limitations of this setup. The user movements and the game occur at real time, but the robots do not respond as freely because of its motion pattern and physical limitations. The robot motion algorithm is designed to move using the basic tank control scheme, so it cannot turn and advance at the same time. Secondly, due to the limited gesture recognition capabilities in the Kinect for Windows SDK, only pressing and gripping gestures are supported. Any gesture that involves complex finger orientations like a rotation gesture or pinpoint gesture is not recognizable by the SDK, so users only have limited options to control the robots.

The depth sensor accomplishes its task without losing track of the upper body skeleton of the players. However, because the detection of the hand-gripping motion is based on size, the detection could become unstable when the hands are oriented at particular angles that confuse the algorithm. The player recognition could also be disturbed when a bystander walks behind the players.

An inherent user side problem with peripheral free control interface was discovered during the game. To move a robot from point A to point B using a drag and drop operation, the user moves its hand in the open air space while looking at the computer screen to see the visualization of the game. Even though the screen output acts as a visual feedback for the user to correct the hand position, the user is moving its hand in open space without a physical reference, and a small jerk could cause inaccuracy in setting the robot's path. This inherent inaccuracy exists in human muscle control, and only a peripheral interface can bypass this problem.

Chapter 5.

Conclusion and Future Work

This project set out to create a setup that combines micro mobile robots, natural user interface, and mixed reality so that users can manipulate the mobile robots to interact with objects in a different space. The result is a game that takes the real robots into the virtual space to play against their virtual counterpart while users control them using hand gestures. When the system was built as a proof of concept, many components of the system were simplified. The advanced counterparts of these components are available but require update.

There are many ways one can improve the system. The simplified base components can be replaced by advanced ones, and there are several existing works available for it. Another approach is to fundamentally enhance the systems and add extra functionality to make it more versatile. The full list of proposed changes is in the following.

To make the virtual puck's behavior in line with real physics, the puck's mass and shape should be taken into account in its collision model so that the rotational component of its kinematic profile can be determined during collision. The assumption of elastic collision may not hold in this system, and should be replaced by an inelastic collision model. In addition, the 4 corners of the virtual air-hockey ring should be round to reflect the shape of a real hockey field, so the puck would not get stuck in the corner and require reset button to reset its position.

As discussed in early sections, the robots only move using tank control at constant angular and linear velocity. Because it can only advance after it rotates and lines up with its destination, it is a 2-step process and uses up considerable time. The efficient way is to turn and move forward at the same time, and this requires different

PWM for the 2 driving motors of the robots. Previously Bezier curve was implemented to generate trajectories for this type of movement [6]. Although this algorithm improves the efficiency, there are a few minor cases where tank movement excels over trajectory, and it is best to create a decision model that can make use of both methods for best performance. Also, because the robot's built-in shooting mechanism is not used in this project, there is no distinction between the front face and back face of the robots. This opens up the option for the robot to move backward to hit the puck.

Continuing on the topic of motor speed of the robots; one of the robots' properties that was overlooked is their ability to travel at an acceleration. Even when the firmware implements tank control scheme, the implementation still allows the robot to accelerate when rotating or travelling in a straight line. When users assign new destinations using drag-and-drop, the users should be able to control the acceleration by varying the distance between the robots and their destinations.

Due to the inaccuracy of human body movement, it may be advantageous to have the game strategically generate paths for players to follow and assist the players optimize controlling the robots. The option to have the game supplying an AI to control the robots and put the players in the supervisory role can also be implemented. The work on path generation is also available in the lab, but the implementation requires heavy software update [17].

Although Kinect SDK supports two different gestures, gripping and pressing, only gripping was implemented for ease of use, and only the right hand is used to play the game in this project. The unexplored gesture inputs using the left hand and left arm could be employed to control the robots' rotation and finer motions. Also, Kinect is capable of 3D motion capture, but the user inputs were mapped to 2D to match the 2D movements of the robots. The unused dimension is another option to implement new user gesture inputs.

To improve the gesture detection and make the user interface more intuitive, it may be worth the investigation to add in finger tracking capability so more gestures could be recognized, and users could use new gestures to manipulate other parameters of the robots' movement. Because Kinect SDK does not natively support finger tracking,

it will require processing the raw depth image directly in order to implement this feature. There are existing libraries that can track fingers in an image, but several of them are either unstable, paid ware, or defunct. PrimeSense's OpenNI library for example supports finger tracking, but the library is no longer supported. The speed of the algorithm also needs to be considered for real-time usage when other components also use up time per loop.

By the time this project was completed, Kinect version 2 was publically released with improved functionality in every aspect compared to Kinect version 1 [18]. This updated hardware also comes with native finger tracking for index fingers and thumbs. Most major stores are starting to thin out their stocks for Kinect 1 since the release of Kinect 2, and it may be wise for the project to migrate to Kinect 2 in near future. However, the Kinect SDK 2 uses different API from SDK 1.8 and does not provide backward compatibility with Kinect 1, and it also doesn't install on any operating system older than Windows 8.

Finally, because the system is a multiplayer game, incorporating network capability will allow people to play with their own robot setups and compare their image processing and robot movement controls. A player will also have 1 Kinect dedicated to him/her so players can move without space limitations. There are active developers working on network transmission in the lab, with their help this should come easily.

The project has reached its initial working stage by the time of this report. However, it's still in its basic form and requires many updates to reach maturity. Changing technology and upgrading hardware will introduce unknown factors to the development of this project, but new features that come with the risks make the upgrading worthwhile.

References

[1] H. Xiao, “*Design, development and evaluation of an experimental platform for network robotics application*,” MASC thesis, School of Engineering Science, Simon Fraser Univ., B.C., 2012.

[2] RoboCup. (n.d.). *Wikipedia*. Available: http://en.wikipedia.org/wiki/RoboCup. Accessed Mar. 19, 2015.

[3] S. Anthony, “South Korea now using Kinect to monitor, track down North Koreans in the DMZ” 2014. [Online]. Available: http://www.extremetech.com/gaming/175955-south-korea-now-using-kinect-to-monitor-track-down-north-koreans-in-the-dmz. Accessed on: Mar. 15, 2014.

[4] M. Wasielica et al., "Interactive programming of a mechatronic system: A small humanoid robot example," *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on* , vol., no., pp.459,464, 9-12 July 2013

[5] D.S.O. Correa et al., "Mobile Robots Navigation in Indoor Environments Using Kinect Sensor," *Critical Embedded Systems (CBSEC), 2012 Second Brazilian Conference on* , vol., no., pp.36,41, 20-25 May 2012

[6] J. Guo, “*Integration of Real Time Mobile Hockey Robot System*,” BASC thesis, School of Engineering Science, Simon Fraser Univ., B.C., 2009.

[7] CuriousInventor. (2013, Feb. 16). “How the Kinect Depth Sensor Works in 2 Minutes.” [Video]. Available: https://www.youtube.com/watch?v=uq9SEJxZiUg. Accessed Mar. 12, 2015.

[8] C. Diaz, “*An Experimental Study of Remote Multi-Modal*,” MASC thesis, School of Engineering Science, Simon Fraser Univ., B.C., 2014.

[9] Kinect for Windows Sensor Components and Specifications. (n.d.). Microsoft Developer Network. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx. Accessed Nov. 15, 2014.

[10] Kinect for Windows Human Interface Guidelines v1.8.0. (n.d.). Microsoft Developer Network. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj663791.aspx. Accessed Nov. 13, 2014.

[11] Depth Stream. (n.d.). Microsoft Developer Network. [Online]. Available: http://msdn.microsoft.com/en-us/library/jj131028.aspx. Accessed Nov. 13, 2014.

[12] R. Knies, (2011, Sep. 26). Kinect Body Tracking Reaps Renown. *Microsoft Research*. [Online]. Available: http://research.microsoft.com/en-us/news/features/kinectskeletal-092711.aspx

[13] J. MacCormick, (2011, Sep. 11). How does the Kinect work? *Dickinson College* [Online]. Available: http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf

[14] KinectInteraction. (n.d.). Microsoft Developer Network. [Online]. Available: http://msdn.microsoft.com/en-us/library/dn188671.aspx. Accessed Nov. 13, 2014.

[15] j_nyukai. (2013, October). N/A. *Microsoft Developer Network.* [Online]. Available: http://social.msdn.microsoft.com/Forums/en-US/7d303daf-e391-43e2-8bcc-f271d8d40e3e/

[16] King Nguyen. (2013, April). N/A. *Microsoft Developer Network.* [Online]. Available: http://social.msdn.microsoft.com/Forums/en-US/e4f5a696-ed4f-4a5f-8e54-4b3706f62ad0/

[17] W.Y. Chen, S. Payandeh, "Micro Robot Hockey Simulator - Game Engine Design," *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, vol., no., pp.9,16, 1-5 April 2007

[18] R. Filkov, (2014, May. 13). Kinect v2 – What's New. *RFilkov.com - Technology, Health and More* [Online]. Available: http://rfilkov.com/2014/05/13/kinect-v2-whats-new/

[19] B. Sobota, S. Korecko, F. Hrozek, "Mobile mixed reality," *Emerging eLearning Technologies and Applications (ICETA), 2013 IEEE 11th International Conference on* , vol., no., pp.355,358, 24-25 Oct. 2013

[20] Mixed reality. (n.d.). *Wikipedia*. Available: http://en.wikipedia.org/wiki/Mixed_reality. Accessed Mar. 25, 2015.

[21] Kinect for Windows Architecture. (n.d.). Microsoft Developer Network. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131023.aspx. Accessed Nov. 13, 2014.

[22] KinectInteraction Architecture. (n.d.). Microsoft Developer Network. [Online]. Available: https://msdn.microsoft.com/en-us/library/dn188672.aspx. Accessed Nov. 13, 2014.

[23] Yaldex. (n.d.). Smarter robots partnering with soldiers. [Online]. Available: http://www.yaldex.com/games-programming/0672323699_ch13lev1sec5.html

[24] Polistina, S. (2013, Jul. 25). “Tangible Augmented Reality for interior design.” [Video]. Available: https://www.youtube.com/watch?v=jK8cUXxIag0. Accessed Mar. 12, 2015.

[25] HERE Europe B.V. (2015, Apr. 9). HERE City Lens. [Online]. Available: http://www.windowsphone.com/en-us/store/app/here-city-lens/b0a0ac22-cf9e-45ba-8120-815450e2fd71

[26] Gesture Recognition. (n.d.). *Wikipedia*. Available: http://en.wikipedia.org/wiki/Gesture_recognition. Accessed Apr. 17, 2015.

[27] Wired Glove. (n.d.). *Wikipedia*. Available: http://en.wikipedia.org/wiki/Wired_glove. Accessed Apr. 10, 2015.

[28] M. Rouse. (2011, Apr.). Natural user interface (NUI). [Online]. Available: http://whatis.techtarget.com/definition/natural-user-interface-NUI

[29] NASA. (2014, Apr. 23).Realite virtuelle, Person with a head-mounted display (HMD), wired glove and joystick. [Online]. Available: http://commons.wikimedia.org/wiki/File:Realite_virtuelle.jpg

[30] L. Jewel. (2008, Feb. 7). Hitting the Links.... Virtually. [Online]. Available: http://usarmy.vo.llnwd.net/e2/-images/2008/02/07/12658/

[31] okreylos. (2012, May 6). “Augmented Reality Sandbox with Real-Time Water Flow Simulation.” [Video]. Available: https://www.youtube.com/watch?v=j9JXtTj0mzE

[32] Gibbons Media & Research LLC. (2010, Oct. 18). Augmented Reality Navigation System Nabs Galileo Master Award in Satellite Navigation Competition. [Online]. Available: http://www.insidegnss.com/node/2344

[33] Robot. (n.d.). *Wikipedia*. Available: http://en.wikipedia.org/wiki/Robot. Accessed May 19, 2015.

Appendix

Program Source Code

The attached file, HockeyKinectVS2010.zip, is a compressed archive of the source codes for this project. The program is built using Visual Studio 2010 on a 64 bit system. It requires OpenCV 2.4x, Kinect for Windows SDK 1.7/1.8, OpenGL Utility Toolkit, and FTDI libraries to compile.