May 2013

Shanghai Service Robot

Jeremy Lester Lowrey
Worcester Polytechnic Institute

Timothy John Sharood
Worcester Polytechnic Institute

SHANGHAI SERVICE ROBOT

A Major Qualifying Project Report:

Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science By:

Jeremy Lowrey Timothy Sharood

Date: May 27, 2013

In partnership with
Shanghai University
Partners: Weiyong Mao, Si Li

Approved:

Professor Yiming (Kevin) Rong, Major Advisor, Mechanical Engineering

Professor Sheng Bao, Co Advisor, SHU

Abstract

A robotic base and control program capable of following a user was created that linked TwinCAT automation software, and Visual Studios C++. Microsoft's Kinect sensor and Xbox Controller were used for communication between the robot and the user. The robot used skeletal gestures, speech recognition, and remote commands to accomplish tasks.

Table of Contents

Table of Figures

Acknowledgments

The project team would like to extend our sincerest appreciation to the students, faculty, and administration of Shanghai University (SHU) and Worcester Polytechnic Institute (WPI) that made it possible for this joint project to take place. The project members from WPI both strongly believe that the China MQP project Center made for one of the best experiences in their time at WPI. The team would like to extend a special recognition to Professor Yiming (Kevin) Rong for his efforts in organizing the project team.

Chapter 1: Introduction to the Project Team

The Shanghai Service Robot Project team consisted of four engineering students, two students from Shanghai University (SHU) and two Robotic Engineering students from Worcester Polytechnic Institute (WPI). The project team was based in Shanghai, China at the Baoshan campus. Communication was established 7 weeks prior to the project start date of August 19, 2012 in order to establish a plan of action, and become familiar with the conditions of the project.

Project members included Weiyong Mao a Mechanical engineering student from SHU with a minor in Computer Science. Xi Li is also a Mechanical engineering undergraduate from SHU. The team members from WPI consisted of two Robotics Engineering Students, Jeremy Lowrey, and Timothy Sharood.

Mao and Li's primary language is Mandarin Chinese, however they both studied English in university and were able to communicate in English. Jeremy and Tim did not have any experience with Mandarin. At times vocal communication was difficult due to the language barrier. To help overcome this issue the team found written communication was useful to express many ideas, problems, and technical terminologies.

The Shanghai Service Robot stems from a previous joint project established in 2010 between SHU and WPI known as the Elderly Assist Robot. The project team consisted of nine undergraduate students from SHU, and four students from WPI. The project was sponsored by Beckhoff and the goal was to:

> "Create a robot capable of assisting elderly people with tasks in their everyday lives. The project focused on the design, simulation, and the implementation of a mobile robotic base with an attached robotic arm. The project culminated in a prototype robot capable of performing basic chassis and arm control which can be used as a platform for future development."

To see the full project report from the Elderly Assist Robot visit:

http://www.wpi.edu/Pubs/E-project/Available/E-project-101910-140332/unrestricted/Elderly_Assist_Robot_Final_Report.pdf

During preliminary communications the project expectations and the foundation for the project were described as:

> "Preliminary foundations: A mobile robot car, which can be autonomous navigation indoor based on laser scanning system.

> Project expectations: This project seeks to design a wheeled service robot for patrol indoor autonomous or remote controlled by voice, and which can provide voice inquiry services based on network. Other functions may be decided by the panel discussion."

The Shanghai Service Robot project team was given the robotic base designed by the Elderly Assist Robot project team as a starting point. The Team was given the opportunity to choose the Specific project goals and desired functionalities. After careful deliberation and assessment of the state of the robot, its hardware, software, and the current resources of the Shanghai Service Project team, the team established project goals and objectives.

The scope of this project was limited by the time allotted to the team for the project. The W.P.I. project members had only 8 weeks in Shanghai. The first week of the project was classified as the Preliminary qualifying Project (PQP), where the team became acquainted with one another, and discussed the project. After brainstorming ideas the team put together a list of goals, tasks, and objectives in which the project would follow. Given that the project team only had 8 weeks, it was critical that time was utilized effectively.

After receiving the project description, the students from WPI realized the benefits that a Kinect sensor might offer. After discussing this option with the members from SHU the team chose to bring a Kinect from the United States because the Kinect was not yet available in China.

Chapter 2: Introduction to the Project

During industrial and medical tasks such as dialysis, welding, and patrolling; heavy objects and equipment need to be moved along with a person as they move throughout a workspace. Often times it is inconvenient or impossible to move such objects as you travel through the workspace without assistance.

The circumstances of these various industrial and medical tasks vary widely from task to task. The methods of communication available in each of these situations are also very different, and can change through out the duration of a single task. It may be appropriate to communicate ideas through speech in a warehouse or during patrolling but it may be difficult to communicate ideas with speech on a manufacturing floor with many welders running. An assistant in these situations would need to be able to communicate in a method that is appropriate for the circumstances.

To have humans dedicated to assisting in all of these situations may not be reasonable, or even possible in some of the situations. A feasible solution to this problem is a device able to follow a person with the capability of carrying large loads, and responding to various sorts of verbal or physical commands given by a human.

Project Goal

The goal for this project is to create a robotic base that can identify and follow a human. Ideally the robotic base would be able to respond to commands coming from a wide variety of input sources, enabling it to be useful in a variety of applications. The robot would be able to recognize commands given by a human and keep a safe predefined distance when following.

This goal was broken down into various objectives. First the robot must be able to drive in a fashion that allows for the robot to follow a human in a work environment. Interaction with the human is very important; to accomplish satisfactory human interaction the robot must be able to identify a human, whether it is by voice control, gesture recognition, or physical control. After these two objectives are accomplished the resulting functions can be merged to aid in the next objective; which is to have the robot follow a human.

Chapter 3: Background Research

Industry Robots

Robotic applications are on the rise, with the desire to eliminate Dirty Dull or Dangerous tasks jobs for robots are opening in all kinds of industries. From healthcare, leisure, warehouses, to automated assembly plants, robots can be found everywhere. There are many different forms of robotics but they all rely heavily on Artificial Intelligence or a main computer and program for processing data. It is because robots can make intelligent decisions, or easily re-orient their tasks that make them so appealing to corporations looking to increase productivity and efficiency.

Robotic arms were first introduced in car manufacturing plants and immediately made an impact in the industry. At first robotic arms were massive, they did not have very many degrees of freedom and they were primarily used only to assist the user whether it be by lifting or rotating parts. Now robotic arms come in all sizes, they do everything from perform medical surgeries such as the Da Vinci Surgical Machine, or take the place of users on the assembly line with a humanoid like robot capable of learning on the job such as Rethink Robotics: Baxter Bot.

Another highly popular robotics field is that of mobile robotic navigation systems and robots. Kiva systems is a company that uses mobile robots for warehouse automation. Kiva who was recently purchased by the internet giant Amazon for $775 Million dollars, uses multiple robotic bases that lift and move crates throughout a warehouse which would normally be done by a forklift while avoiding objects and each other to increase productivity.

Medical robots like the Swisslog shown below assist nurses by providing routine care or deliver goods to others (Swisslog). Robotic bases in hospitals have almost limitless capabilities, a base capable of storage is able to deliver medications to patient’s rooms, act as supply closets and deliver lab samples (Kiplinger). Robots can also provide doctors halfway across the world with a way of interacting with patients in the form of a telepresence machine or video feed that lets the doctors and patients interact with one another from various locations.

Figure 1: Maserati Plant

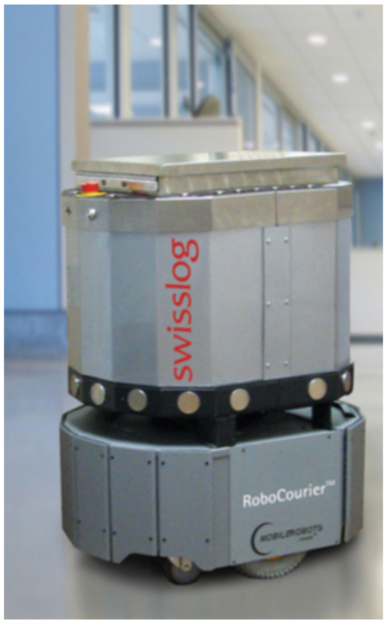


Figure 2: Swisslog Robotic Platform

Hands Free Control Methods (Kinect)

Humans are receptive to using voice and gesture communication; it has been our natural form of communication since the beginning of time. The project team found that voice and gesture control would be very valuable to the average user. Consider a situation where a user is on one side of a room or in the kitchen where they can't reach or see the robot, with voice commands a user could just call it over. Consider the situation where a user working on a car and needs a tool from a toolbox but doesn't want to move, typically they would just ask help from an assistant. Humans are innately receptive to voice and gesture commands commonly use such commands to communicate to those around them. If robots are going to work intuitively with humans they must become receptive to voice and gesture commands.

Until recently, humans were the only ones capable of interpreting and responding to this type of communication. Unlike humans, a computers natural language is not English, Spanish, Chinese, Sign Language, or some other verbal or gestural language. Computers operate by processing numbers.

The Kinect is a new type of sensor array developed by Microsoft. The Kinect uses a series of sensors including an RGB camera, an array of microphones, and an IR emitter and IR depth sensor. The kinects array of sensors can be used to track the movemnts and sounds made by humans and convert this information into numbers a computer can understand. With this device programs used for games and robots can now take the movements and words of the human users as input. The key feature to this device is the depth sensor or threedimensional sensor that allows for the tracking of movement.

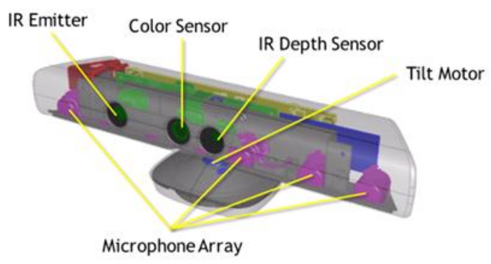


Figure 3: Microsoft Kinect sensor components

The availability of relatively cheap three-dimensional sensors is a new development. PrimeSense, founded 2005, is partly responsible for this development. PrimeSense developed and licenses a chip that receives information from an infrared CMOS (complementary metal-oxide semiconductor) sensor and in turn calculates a depth image for a given input scene.

A depth image is an image similar to that of a standard digital camera. In a standard digital image, a representation of a scene is created by dividing the viewing area of the camera into a grid. A sensor at each point along this grid records the intensity of red green and blue wavelengths of light. It is the red green and blue values that a computer uses to recreate an image on the screen. Each point in this grid is called a pixel and as the number of pixels in an image increases so does the quality of the representation of the scene depicted in the image. In a depth image the viewing area of the camera is similarly separated into pixels. However, instead of holding values for red green and blue each pixel holds a distance value. These distance values represent the distance from the camera to the nearest object within the section of viewing area of each pixel. Similarly as the number of pixels is increased, the quality of the representation of the scene is improved.

The depth image is created by representing the distance at each picture as a color. The depth images used in this project are monochromatic. The depth image is created by representing closer distances with a more intense or brighter shade of the color used in the image and distance that are far away or beyond the measurable distance of the sensor are represented by darker shades or black.

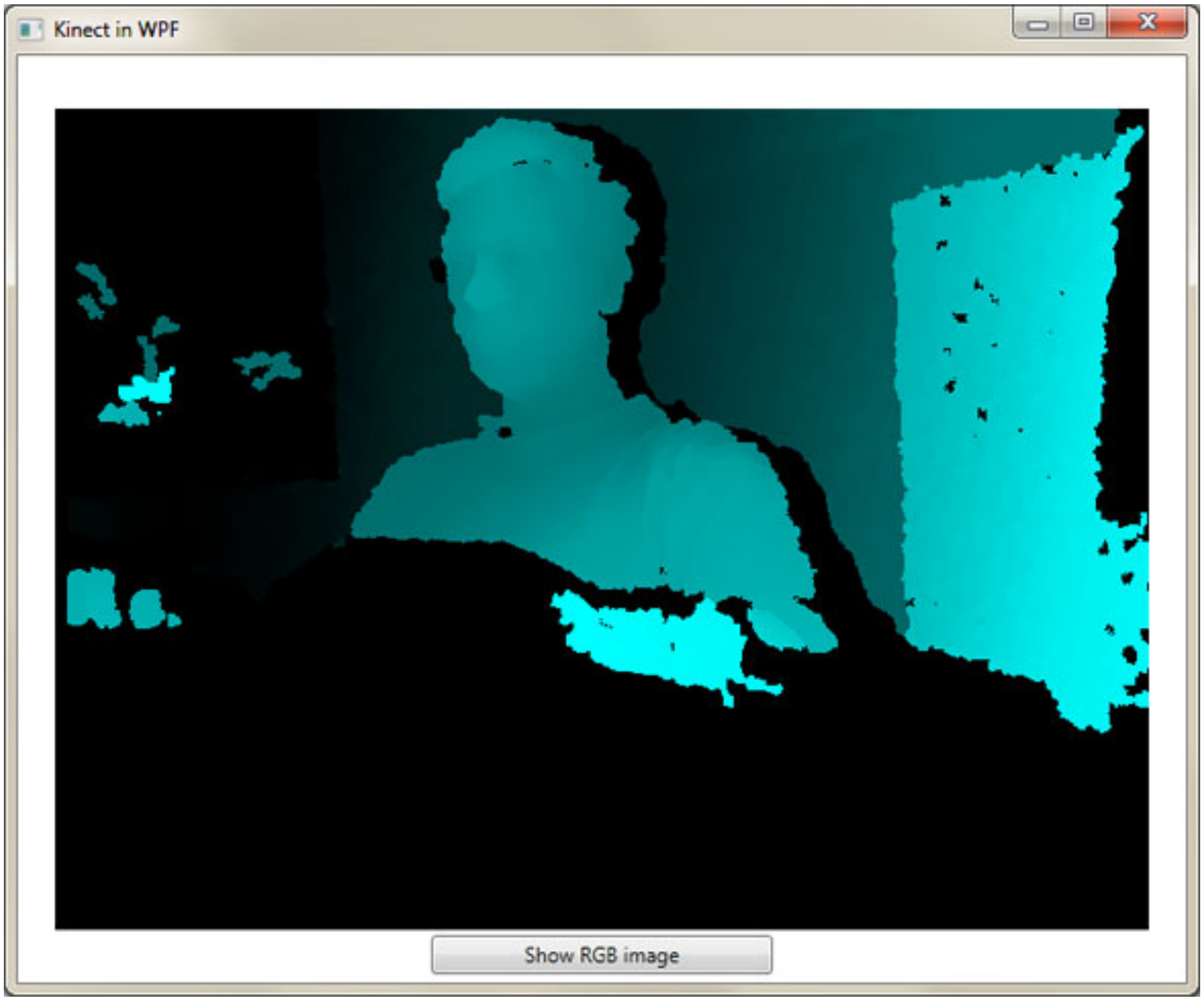


Figure 4: Monochrome depth Image

Unlike a RGB camera sensor, which directly senses the intensity of red, green, and blue light for each pixel, the sensor used for a depth image cannot directly measure distance. Instead, a series of sensors is used to calculate the distances. The sensor used to capture the image records the intensity of infrared (IR) light at each pixel. An IR emitter placed on the same device baths the scene in IR light. The IR light is reflected off objects in the room and is measured by the IR sensor. It takes time for the IR light to travel to objects in the scene and back to the sensor in the device. By varying the intensity of the IR light with time, it is possible to calculate the time taken for the light to travel to and from the device.

The chip developed by PrimeSense controls the IR emitter and reads the IR intensity data from the IR sensor. Calculations are completed within the chip using this information the chip compiles the depth image, which is then sent to the host device via USB.

PrimeSense licenses their technology to various manufacturers and as such there are a variety of three-dimensional sensors available on the market. The first available hardware was the Microsoft Kinect for Xbox; this device was designed for the Microsoft Xbox gaming system. There are two distinct versions of the Microsoft Kinect; there is the Microsoft Kinect for Xbox and the Microsoft Kinect for windows. The difference between these is that the Microsoft Kinect for windows has a shorter minimum distance for detection, it can detect objects 20 cm away, the Kinect for Xbox can only detect objects that are greater than 40 cm away. The Kinect for Xbox is designed for gaming and has a cheaper lens and hardware. The Microsoft Kinect for windows is designed for use in robotics and research.

The Kinect alone produces two video streams a regular RGB stream taken with a standard webcam and a depth image stream that is created using the IR emitter, the IR detector, and the PrimeSense chip. A set of software then uses the depth image to determine if a human being is in each frame.

Microsoft in partnership with Natural user interface (a PrimeSense Company) was the first to create software with algorithms capable identifying human joints, but since then others have created similar software. OpenNI is an open source set of software that has drivers for the Kinect and other devices using the PrimeSense depth-imaging chip. A Microsoft SDK is used in this project for skeletal recognition.

The software within the Microsoft SDK receives each depth image frame and runs it through a set of algorithms that analyze the image for humans. The algorithms in the Microsoft SDK can identify up to six individuals and can track up to two skeletons with individual joint information. Each joint for each skeleton is represented by an XYZ distance from the camera in meters. +Z represents the distance away from the camera, +Y represents the distance above the height of the camera, and +X represents the distance to the left of the camera.

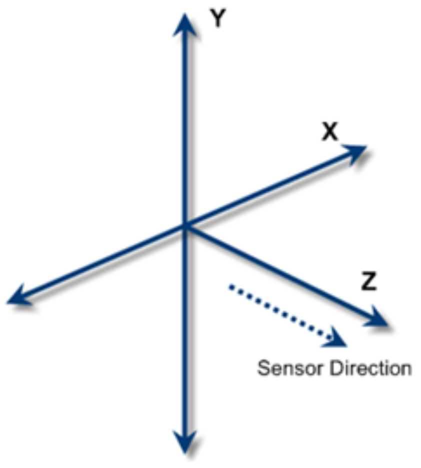


Figure 5: Kinect coordinate system

The SDK then produces a frame representing the positions of each joint within the view of the Kinect. The skeletal frame will appear black if no skeletons are detected. If the Kinect does detect a human it will plot the joints and connect the corresponding joints with a line as seen in the image below.

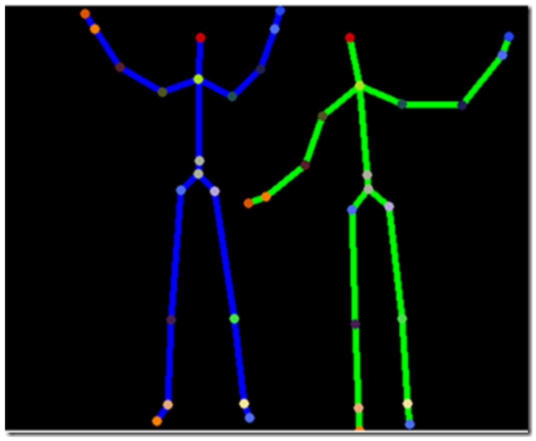


Figure 6: Skeletal tracking

Microsoft's Kinect is most known for its ability to recognize, track, and turn your movements into a control but it also has incredible speech recognition technology. IPhone's Siri application or Google's search functions found on most phones seem to surpass the Kinect's voice recognition, but in all reality, the two are completely different and incomparable.

Phones use a push-to-talk application, they pause what they are doing, go into voice recognition mode, then listen for the next few words or sentences said by the user. The Kinect is constantly scanning the entire room for vocal commands while casting out background sounds or music. The key to the Kinect's voice recognition is beamforming; it focuses on specific users or points in the room to listen to and can even localize the sounds. The Kinect has four microphone arrays that can isolate users with a reading angle from 58 to -58 Degrees relative to the front of the sensor. Utilizing the Kinects four microphone arrays the Kinect is able to create an imaginary cone stemming from the Kinect sensor. It can capture the length of audio signals and separate commands that travel across the sensor to only capture those directed towards it. (MSDN_Kinect)

Alternate Control Methods

Touch Screens & Remote Desktop

A human's role in robotics will never be fully eliminated; a human will always have the ability to override commands. It is because of this there is still a large need for tactile controls and touch devices. Control methods include simple wireless controllers, video game controllers, touch screens and many more. For robots being operated from a computer there is always the typical keyboard and mouse controls, remote desktop, or touch screens to interactive with the programs GUI.

Many robots, especially in the development and prototype phases are still controlled using a windows application. In General, when controlling a robot or any windows application, the operator would use a mouse and keyboard. To give versatility to those controlling the robot it is always nice to have a variety of controls when operating windows applications.

Touch capacitive screens are becoming very common in today's world. From smartphones to tablets to industrial computers the technology is everywhere. The technology is very intuitive and easy to operate. Screens for personal use range in price but large industrial capacitive touchscreen computers can be very expensive. Control can also be obtained by connecting to the main computer via Remote Desktop. This can be done by using a smartphone, tablet, or a desktop computer

Wireless Controllers

Wireless controllers are everywhere from videogames to toy cars. They all have the same basic design; they use a wireless signal to control their desired target using analog or digital signals.

Eight Channel Wireless Controller

The current control method used by the Elderly Assist Robot Project team is a simple 12V eight Channel wireless remote control switch. The switch was designed for a wide range of applications, It uses microprocessor intelligent control to output a voltage of 12V (High) or 0V (low). The control distance is 50-100 meters and the device can control High-volt appliances and equipment using a bulky relay which can control kilowatt appliance or electrical systems (Comway). The remote can be seen in Figure 7.

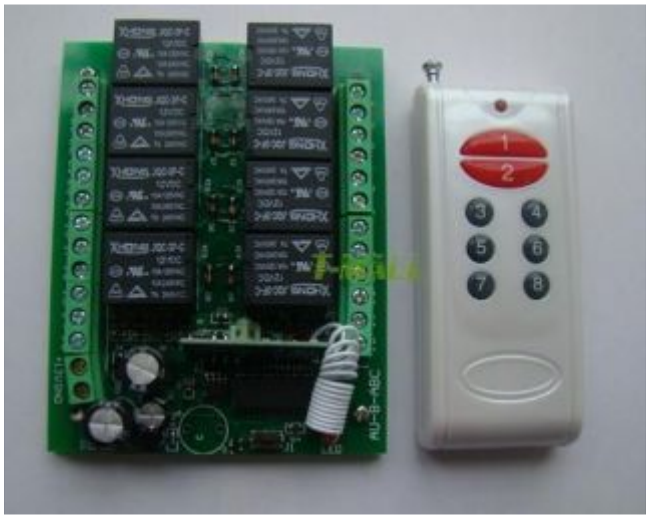


Figure 7: Eight Channel Wireless Relay and Remote

Microsoft XBOX Controller

The reason Microsoft's XBOX controller is so popular among developers is because the general public is already very familiar with it and it has a very strong development platform. This makes for an easier learning curve when transitioning between users. For example, iRobot commonly pairs a MS XBOX controller with their robots such as the 310 SUGV because they find that the majority of their users are familiar with it. These Users may include military soldiers, members of law enforcement, and many other privatized organizations.

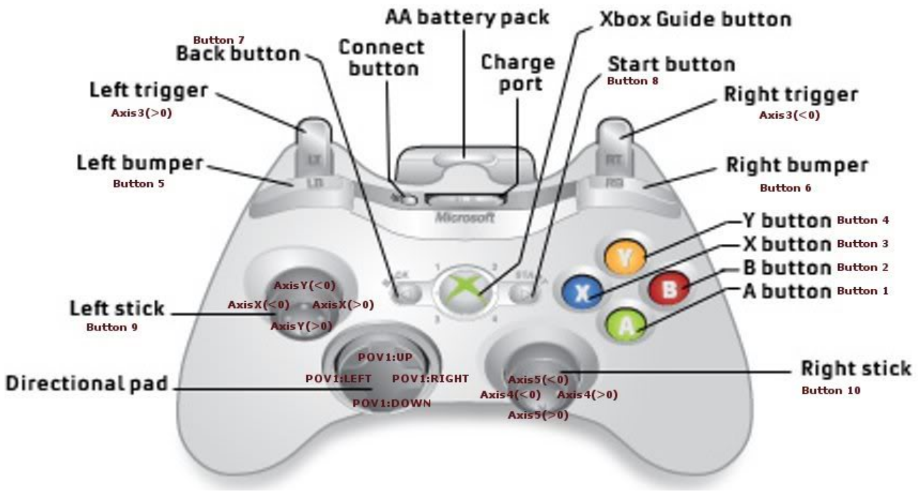


Figure 8: Microsoft Xbox 360 Controller Guide

Microsoft provides a free Application Programming Interface (API) known as XInput. XInput is provided through the DirectX software development kit (SDK) for console development provided by Microsoft. XInput gives a programmer the ability to utilize inputs acquired by the Xbox controller such as analog or Boolean data (XInput).

The data is obtained by first creating an instance of a controller in your program, for example myController. After an instance is created, data can then be read by using predefined functions such as ThumbLX or IsButtonPressed (A), for a full list of controller inputs and commands see Appendix B.

Control Hardware

Programmable Logic Controllers

Programmable logic controllers or PLCs are common in many industrial applications, and are often used to control industrial components or processes. The specific functionalities can vary from model to model but the basic function is the same, the PLC is the piece of hardware that communicates the control and sensory information between a control program and the device or sensor. The PLC hardware must be able to receive an electrical signal as input, and feed the input to a control program running on a processor, then send a control signal in response.

Modern PLCs often have a microprocessor built in so a system can stand alone without the need for a computer to run the control code. Microprocessors have become much faster and cheaper in the last few years, and as a result a competitor to the Industrial style PLC, microcontrollers like the MSP430, Arduino and the RaspberryPi, have become more common in robotics and smaller industrial machines. The functionality of a microcontroller is the same as the PLC however PLCs are designed to be more robust. PLCs are able to handle the strains of industrial environments like vibration, temperature change and electrical noise. They are designed for use with industrial systems and Industrial components. PLCs are modular and many are designed to work as a node on larger systems.

The PLC used for this project is the Beckhoff EK1100 produced in 2008 this is a controller with no onboard processor. The EK1100 controller was paired with several types of terminal blocks. Two El 1018 eight channel digital input blocks read in signals from the two limit switches of the wheel turning system and the eight channel radio controller. Two EL5101 incremental encoder interface terminal blocks received signals from the four Mitsubishi motor encoders and two blocks sent signals to the four motor controllers. The terminal blocks read in a value high +15~+30V or low -3~+5V the terminal block sends the value of each port to the EK1100 several times a second. The EK1100 controller then sends value information from all of the terminal blocks to the computer processor via an Ethernet cable connecting the two. A portion of the computer's processor is dedicated to running the PLC control code. This code is written in IEC 61131-3 standard. The IEC 61131-3 standard was created specifically for PLCs and is designed so that programs can be written so they work with PLCs from many different manufactures.

Development Software

It is important to assess a project's requirements when selecting a programming language. Like written and vocal, there are literally thousands of different programming languages. In some cases these programming languages are simply variations off more popular languages but, nonetheless they all have specific traits that make them unique. The easier the language is to program in, typically the slower the program compiles and runs. Programming languages have specific traits that coders and engineers seek when developing new projects, these include their ease of learning, ease of understanding, speed of development, community & industry support, performance, supported platforms, hardware portability, and fit-for-purpose (Britton).

Integrated Development Environment (IDE)

Developers often use Integrated Development Environments or IDE(s) for short when coding or creating a program. An IDE is a software application that provides developers a place to write, generate, and debug their code. One of the most critical features that IDEs have to offer is the class browser. A class browser allows programmers to browse and navigate the structure of their code.

An IDE is a software where developers can write their code whether it be in C++, Java, or any other language supported by the specific IDE. A programmer using an IDE can be compared to a Journalist using Microsoft word; the native environment for both programmers and journalists is a simple text document but it is often rarely used. For example in word, people often use spell check for grammar and spelling or search for new synonyms using one of the many tools in Microsoft Word, these tools are very similar to what a programmer might use in an IDE. Most IDEs have tools that allow you to see errors as you type, see highlighted syntax code, browse class structures, drag and drop utilities, auto generate code, or select templates.

Microsoft Visual Studio

Microsoft Visual Studios is an IDE developed and supported by Microsoft. It is used to develop windows applications, forms, and Graphical User Interfaces (GUI). An advantage to programming in Visual Studios is that its class browser makes it easy to communicate between C++ and Visual Basic Code. Using Visual Studios makes developing GUIs for applications much easier and allows you to communicate with other C++ classes. (MSDN_VS)

The most appealing aspect of Microsoft Visual Studios is its integration with Microsoft products such as the Kinect Sensor, and Xbox game controller through Software Development Kits (SDK). SDKs can be downloaded on Microsoft's developer's pages and come with detailed descriptions and tutorials.

Resources when creating and debugging programs can be very useful and important to a projects success. Microsoft Development Network (MSDN) provides online tutorials, community support, known issues and bugs, and has an active online community. These features give a programmer the resources they need to research bugs and problems they otherwise could not fix.

General Purpose Programming Languages

General purpose Programming languages are often easy to use, understand, and easy to build from. They provide an array of low level and high level programming features that satisfy a large portion of developers and students. General purpose languages are the most common languages when it comes to applications or program and they are the foundation for almost all high level program architectures. Students, professionals, and hobbyist alike all utilize the vast community support, lessons, and guides that these languages have come accustomed too. There are many different types of programming languages all with their own unique features. Some of the more popular languages include Java, C, C++, C#, Visual Basic, and python. (King)

C++

C++ Is one of the most common and popular programming language. The language was originally created as an enhancement to C adding object oriented features such as classes and many other enhancements. The addition of object oriented features made C++ a language with both high and low level programming features. Classes can communicate with each other and hierarchies can be created leading to more efficient libraries. Due to continued development, C++ became the model, and foundation for many other popular platforms such as Visual Basic, C#, and Java.

Visual Basic .NET

Microsoft's Visual Basic .NET (VB) is a programming language accessible through Microsoft's Visual Studio that creates Windows applications. Drag and Drop features allow for a programmer to drag objects such as a button onto the form (blank window) then name and title the button, list features, and select appearance; the code is then auto compiled by VB into a readable and editable C++ class. Objects can include labels (plain text), editable text boxes, buttons, radio buttons, checkboxes, and much more. See Appendix A for a screenshot from creating a VB Form.

At any time during the creation of your GUI you can switch from your GUI view to the code view. The code view shows all the auto generated code created by VB. This code can then be modified to have functions run when actions such as clicking a button are performed. These methods and functions are written in C++, they control the outcome of your program. Actions can simply switch between windows or run separate programs in other classes. This object-oriented behavior creates an architecture that makes for a great User Interface responsible for controlling an application.

Service Oriented Architectures

In order to accomplish the task of communicating cross-platform many members of the robotic world are switching to software oriented architectures or SOA for short. Service oriented architectures are ideal for a robot that needs to communicate with a number of different systems. SOA efficiently allows computers to work in parallel in order to process more data, evidently when a computer is at its maximum processing limits SOA distributes data among other machines.

Processing power is not a major concern to this project but that is not all SOAs are good for, they are very efficient in communicating with different devices. SOAs have the potential ability to communicate with a number of different devices which could be used for this project such as the PLC, Kinect, Touch Screen, and the laser range finder.

Two major competitors SOAs industry include ROS short for Robotic Operating System, and Microsoft Robotic Development Studio (MRDS). They both use the same general concepts to sparse data in order to gain maximum processing power but their user interfaces are very different.

Microsoft Robotic Development Studio

Microsoft's Robotic Development Studio (MRDS) is an official product of Microsoft research. MRDS is built from the foundation of Microsoft's .NET Framework so there is a large library of resources one can utilize. It also provides a Visual Simulation Environment (VSE) which provides the ability to simulate and test robotic applications without hardware.

MRDS is closed source and only available on Microsoft OS so for others utilizing different OS's this may pose a problem. The program is fairly simple to use for general users by utilizing a Visual Programing Language. This enables users to simple drag and drop operations and hardware onto a diagram, code can later be automatically generated to C#. VPL also gives the developer the ability to take a collection of blocks and reuse them as a single block. Since Microsoft designed the Kinect, it is well supported within MRDS and mainstreams to developers and consumers looking to develop personal and consumer robotics.

An advantage to MRDS is its DSS Manifest Editor which provides application configuration and distribution of drivetrains and sensors that allow for universal development for specific robots. Once a manifest is developed for a robot, MRDS can simulate programs using its unique Visual Simulation Environment (VSE) which allows for developers to simulate and test robotic applications in a 3D environment without hardware present.

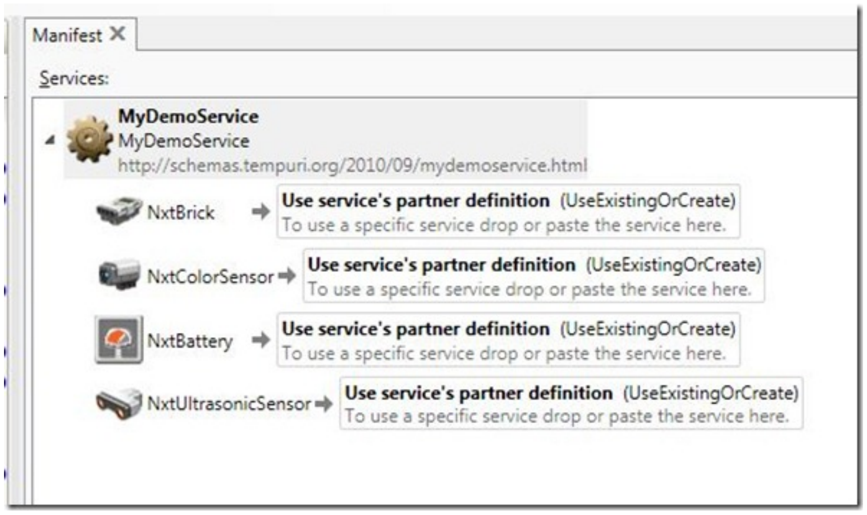

Manifest
Services:
MyDemoService
MyDemoService
http://schemas.tempuri.org/2010/09/mydemoservice.html
NxtBrick → Use service's partner definition (UseExistingOrCreate) To use a specific service drop or paste the service here.
NxtColorSensor → Use service's partner definition (UseExistingOrCreate) To use a specific service drop or paste the service here.
NxtBattery → Use service's partner definition (UseExistingOrCreate) To use a specific service drop or paste the service here.
NxtUltrasonicSensor → Use service's partner definition (UseExistingOrCreate) To use a specific service drop or paste the service here.

Figure 9: MRDS VPL Example manifest

Robotic Operating System

ROS which is short for Robotic Operating System is a free open sourced programming language which is developed by Willow Garage. ROS is primarily supported on Ubuntu but there are currently experimental versions that operate under different OS's.

ROS is highly used in industry, and given that it is open source architecture it is highly supported by community developers. For this reason ROS may have a larger variety of support for different sensors and framework code built by community developers and shared online, these codes can easily altered from program to program.

Elderly Assist Robot

The robot platform assembled by the Elderly Assist Robot project team consisted of numerous Beckhoff control components, micro sensors, a drivetrain, and aluminum framing components. The base of the robot was machined from a sheet of ½ inch aluminum into an oval shape, this base plate served as the mounting bracket for the drivetrain assembly and the aluminum framing. The drivetrain was made up of two caster wheels and two driven wheels. Two driving wheels were powered by four AC electric motors and four AC electric motor controllers. The four wheels and drive components were attached to the bottom of the aluminum base. Columns of the frame extended vertically from the base plate. The frame was built using aluminum bars cut to the appropriate sizes and held together with brackets. This worked well for prototyping given the periodic changes to the platform. Resting on the base in between the frame columns, a container held the power source. The power source consisted of two automotive batteries connected in series and linked to current regulating circuitry which could be plugged into an outlet when the robot needed charging. A power inverter was left to sit on top of the battery container. An aluminum plate was positioned between two main columns. Mounted on the aluminum plate was the Beckhoff PLC. The radio control module was taped to the side of the robot. The industrial computer was behind the plate facing the rear of the robot. The robotic arm was attached to a column in the front of the robot. The control circuitry for the arm was no longer mounted to the base. It was found that some of the components were not able to be located. These components included ultrasonic sensors, binocular camera sensors and arm encoders.

Chapter 4: Methodology

Task Specification

To ensure good use of the team's limited time together, the team developed a list of tasks that would more clearly define the problem statement and fulfill the objectives.

Objective: Robot can drive in a fashion that would allow for the robot to follow a human in a work environment.

- Evaluate and test the current condition of the robot, state of hardware, state of assembly, state of the software control program
- Fix and replace any broken hardware components of the base
- Choose appropriate locations for all hardware and sensors
- Assemble and wire the base so that it can drive
- Create a driving program that can control the hardware so that the robot can perform a range of driving movements

Objective: Robot can identify, and perform driving tasks instructed by the user by voice control, gesture recognition, and wireless controllers or touch inputs.

- Programs developed to gather command inputs must be written in a way that allows for them to communicate with the driving program
- Develop a graphical user interface for a user to select commands using a touchscreen, mouse and keyboard, or a remote desktop connection
- Develop a program that can accept button and joystick information from a wireless Microsoft Xbox controller
- Develop a program that can take skeleton and depth information from a Kinect sensor, and can infer control commands from the information
- Develop a program that can read a vocal signal from the Kinect microphone array and determine if it matches a user's commands word

Objective: Robot can follow a human.

- The previous two objectives are needed to complete this objective
- Develop a program that can receive commands from the various input programs and send commands to the driving program. These programs will be written in various languages so this program must communicate between languages.
- Develop the driving commands that determine how the robot should move

A Gantt chart was developed to help to clearly define when the tasks would need to be accomplished this can be seen in Appendix C.

Base Evaluation and Redesign

August 19th, 2012 the project team arrived on site to find the starting platform in disarray. The robot had several hardware and software problems that had to be dealt with before the platform could be used for new research. The Elderly Assist Robot was inactive for two years, during this time the robot was stored in a lab along with many other supplies. The team found that during this time wires were unplugged, and the previous industrial computer attached to the robot became nonfunctional.

Hardware Replacement & Testing

The team found that the most important and immediate fix that needed to be made was the main computer used for processing and touch interface. After testing the team found that the computer would no longer boot to the Basic Input/output System menu known as the BIOS. The team recovered the necessary program files from the computer's hard drive but concluded that the motherboard was damaged and that it would need to be replaced. The time and resource constraints of the project did not provide the team with many options for replacing the computer.

An industrial style replacement computer was found that included a touch interface. The replacement could boot fully, but the team found the computer did not have the processing power for a high performance sensor like the Kinect. After careful deliberation the project team decided to use a Lenovo desktop computer with an Intel i7 processor instead. The reasoning behind this decision was that the goals of the project were more focused on visual and speech interaction rather than tactile interaction. The touch display was replaced with a standard monitor and the desktop was connected to the PLC via an Ethernet cable, with a working computer the team was able to continue testing. It was concluded that if a touch interface was desired that a replacement touch screen monitor combination could be purchased at some point in the future.

After booting the computer, the necessary software was installed to run the Beckhoff EK1100 Programmable Logic Controller (PLC). The Team began testing of the control systems by using Beckhoff's TwinCAT System Manager Application. The application showed the value of the ports on each of the terminal blocks of the PLC. This application was used to test the Limit switches and radio controller and ensure that values were being received whenever they were triggered. The Next step was to test communication between the PLC and Motor controllers.

Initially all four controllers were found to be in an error mode. Before communication between the PLC and motor controllers could be established the motor controllers had to be reset and tested for functionality. The controller specifications and user manuals were found on the Mitsubishi website. Using a special jog mode it was confirmed that two of the four were still correctly wired and functional at least enough to turn the motor over. The wiring for the other two had to be disconnected checked and connected. One of the two was continuously problematic and a complete hardware reset was performed.

Communication between each of the controllers was trouble-shooted by creating a simple TwinCAT program to run the motor forwards then backwards at a set speed. Using this simple program the team was able to get each motor controller to run at the correct speed and to report the correct encoder values.

Base Design

The robot, in the configuration designed by the previous project group, could satisfy some elements of the new objectives. The robots base plate was designed to be just smaller than the width of a common doorway that allows the robot to travel easily in an indoor environment. The center of mass was located low on the robot close to the wheels. The low center of gravity would allow for the addition of any application specific equipment in further prototypes. The drivetrain was mounted neatly under the base plate. The drivetrain could provide needed torque to move the robot and any additional equipment. The batteries could provide sufficient power to drive the robot for thirty to forty minutes. The wheel assembly allowed the robot to change orientation easily. The PLC was mounted in a convenient position on a vertical aluminum plate. The LIDAR sensor was located on the base plate facing the front directly in front of the batteries. The plate gave the expensive sensor protection from impact, and the low position above the drivetrain put it in a good location for obstacle mapping.

Some components of the base did not meet the requirements of the new objectives and needed to be redesigned. One piece of hardware that was no longer needed was the robotic arm designed by the Elderly Assist Robot project team. Originally the platform contained the small robotic arm seen in Appendix A. After hearing reviews from professors and previous students it became apparent that the arm did not work properly. When the project team arrived, parts of the arm had been disassembled the motors, and motor control circuitry of the arm, had been disconnected from the PLC, and the motor controllers were no longer mounted to the robot. The arm would not fulfill the new objectives so the team decided to remove the arm and focus efforts on the driving, sensors, and controls of the robotic platform.

The new Lenovo computer tower and separate monitor, in place of the older all in one Beckhoff, had to be integrated. In the previous configuration the screen faced the rear of the robot with the LIDAR sensor facing the front. The new robot often used the front facing Kinect and LIDAR sensors to follow users, a rear facing monitor would be inconvenient. If the monitor faced the rear, the robot would have to spin 180 degrees or a user would have to stop the robot and walk behind it in order to get information from the screen. A front facing screen would be ideal.

The 24v to 120v power inverter, new power outlet strip, radio control module, and wireless Xbox controller receiver module needed to be neatly organized and mounted to reduce the complexity of wiring and risk of damage due to impact or electrical short caused by loose wiring. The aluminum plate that held the PLC in place has just enough space to fit all these components. With some rewiring and drilling of new mounting holes the components were all organized along this plate.

The Kinect sensor also needed to be mounted on the robot and had to be placed somewhere on the robot that would allow it to easily pick up users both near and far. The Kinect sensor might also be used for room mapping at some point. A position that would allow it to pick up users and also give it a clear view of the space in front of the robot while eliminating as many blind spots as possible would be ideal. The Kinect horizontal vision range is larger than it's vertical. Users may raise their hands fully above their head or may walk too close to the Kinect. In situations like these, some of the users joints might go out of the Kinect's viewing range. To resolve this issue the Kinect has a built in motor that can adjust the angle of the camera. The range of the adjustment is also limited to 28 degrees or 14 degrees in the negative and positive directions. The best position would be one where the Kinect can stay at the zero horizontal position for most of the time and only adjust when necessary.

When the Kinect is mounted to the base plate it is too low to the ground. In this position the Kinect has a good view of obstacles close in front of the robot, but obstacles often block the view of a user. Also in this low position the Kinect needs to be mounted at an angle or constantly be at the top of its viewing range at an angle of +10 degrees.

When the Kinect is placed at head height or ~ 6ft above the ground it has a better view of users and obstacles further away, but obstacles close to the robot were out of view. When a user got close to the robot, about 2ft, most of their body is cut off from view except their head and shoulders.

When the Kinect is placed just above the screen ~ 4ft from the ground most obstacles do not block the view of users, and users both near and far from the robot are within the view. Users often crouch slightly when approaching the robot in order to better view the screen in this position they are within the view. Users 4-8 ft. away are fully in view in this position. If a user raises their hands above their head the angle only needs to be adjusted 1-2 degrees.

Once the height of the sensor was determined the position of the Kinect towards the front or the back of the robot had to be determined. The Kinect for Xbox has a 40 cm minimum range anything closer than 40 cm cannot be picked up by the sensor. If the sensor is placed in the rear, the sensor can detect anything directly in front of the robot. However, in this position the frame and computer obstruct the bottom view. If the Kinect is placed above the center of the robot and raised a few inches the view can be unobstructed, only 20cm of the min range is obstructed. In this position the Kinect the Kinect for Xbox has a range of view ~ 20cm-10ft in front of the robot. If the Kinect for Xbox was replaced with a Kinect for windows the range would be ~ 0-10ft.

Control Software

After a new computer had been installed and the wiring and communication between the PLC, Drivetrain, and radio remote had been reestablished the team was able to attempt to run the TwinCAT program from the previous project group. The program developed had four simple driving commands forward, back, rotate left and rotate right. A homing function was also created, when the button for this command was pressed. The wheels would rotate to a home position as defined in the program so that the wheels were both in the same orientation and the robot could successfully drive straight.

The program designed by the previous project group used a single eight-channel wireless radio controller that directly activated ports on the El 1018 eight channel digital input block to achieve command input. The radio controller program operated in a toggle switch fashion. When a button on the remote was depressed the remote would send a signal to the radio module. The radio module would then provide 24 volts to a port on the PLC for as long as it received the signal from the remote. The TwinCAT program would see the high value and tell the motor controllers to turn on at a set value, thus driving the robot straight, backward, or cause a zero radius turn left or right depending on the number selected. When the button was released the signal from the remote would stop and the port would return to low. The program was designed so a second click of the remote button would signal the program to stop so the button had to be released and depressed again to stop the action.

Program Development

A strong operating program that is user friendly, capable of processing sensor data, and able to output commands to the motors is required to complete a robot capable of the high level human robot interaction desired in this project. To accomplish this, the robot hardware, control peripherals, and sensors all need to communicate effectively with one another. However, not all of the hardware and sensors operate in the same programing environment.

The PLC controls and monitors all of the signals to and from the drivetrain hardware; such as the motors, motor controllers, and limit switches. The program running the PLC is written in a PLC specific language called the IEC 61131-3 standard. The Kinect sensor, LIDAR sensor, and GUI can work with programs written in many languages but they do not work with the IEC 61131-3 language In order to achieve all the desired functionalities the program that commands the motors had to be written in the IEC 61131-3 language and the programs that communicate with the sensors and GUI all need to be written in another language.

The team researched several methods to communicate between the two sets of languages. One method the team looked into was to have the user input program communicate with the PLC through hardware. This would entail hooking a microcontroller to the PLC that could send signals through the ports on the terminal blocks. The down side to this method would be an added delay between user's commands and the robots reaction. With this method the user input program would send a command to a microcontroller then the microcontroller would change the value of a pin or set of pins and this would be picked up by the PLC. Then the PLC program would send commands to the motors. There would be a delay because it takes time for a microprocessor to react to a command and for the PLC to pick up on a terminal value change. The team decided not to use this hardware method due to the delay.

Another option was to communicate through software. The only communication to other languages available using IEC 61131-3 standard is through TCP/IP Ethernet Internet protocol. Beckhoff created a library of commands that can be used to communicate between .NET framework 3 and the PLC through TCP/IP. With this method any language that can communicate through .NET framework 3 could be used because of this library. The next step was to choose a language that could communicate over .NET and could manage all of the possible input methods.

C# in MRDS was the team's first choice for managing the User Inputs. The MRDS IDE provides a large framework of code designed for use in robotics, including a virtual 3D environment where code can be tested virtually. In MRDS high-level code like analysis of sensor input and navigation are written independent of the robotic base. The navigation program sends commands similar to drive forward, back, turn left, and turn right to a generic drive function. Then each robot base has its own manifest that communicates the standard commands to the specific hardware of that base. Code written for one robot can be used without modification on another robot just by switching which manifest is associated with the drive function. With this method, code written for this project could be tested in the virtual environment provided in MRDS and could easily be used with any future prototypes with different hardware or other robotic bases entirely. There was one major issue with this method and that is that MRDS communicates on .NET 4 standard and the Beckhoff library created for the IEC 61131-3 language uses .NET 3 and these two systems are not compatible. The team attempted to create a manifest that could communicate to the IEC 61131-3 PLC driving program, but found that would not be feasible in the limited amount of time available.

The Final option was to create a custom program and custom driving function that could send values to the IEC 61131-3 PLC driving program. To accomplish this, the team created a program using dynamic list libraries (.dll) that could be imported into any C# or C++ program. Once this .dll file was imported to a program, that program could send the .dll program commands like drive forward or back at a speed of 100 rotations per min, or turn left or right at a given speed.

The .dll program uses a function in the Beckhoff .NET libraries that allows it to change variables already created in the IEC 61131-3 PLC driving program. When then the .dll receives a command of drive forward with speed 100, the .dll program communicates this to the driving program by changing two variables of the drive forward function within the IEC 61131-3 PLC driving program. The .dll changes the value of the variable that activates the function to true and it changes the value of the rotations per minute variable to 100 (or the value given to the .dll).

For this to work new driving functions had to be written in the IEC 61131-3 PLC driving program. A new function had to be written for each type of driving command with all the appropriate variables. This included a drive forward at speed x, drive back at speed x, turn left x degrees at speed y, turn right x degrees at speed y, and a new home wheels function. During testing of the initial .dll version it was found that the Beckhoff library had many bugs when working with the older version of TwinCAT that the previous project group's control program used. The PLC program was rewritten for the newest version of TwinCAT and IEC 61131-3 so the team rewrote it and the new functions were included in this newest version. With the .dll program and the new functions in the PLC program the user input programs could send driving commands. A representation of how the user input programs communicate with through the .dll can be seen below in Figure 10: Flowchart of Program Structure

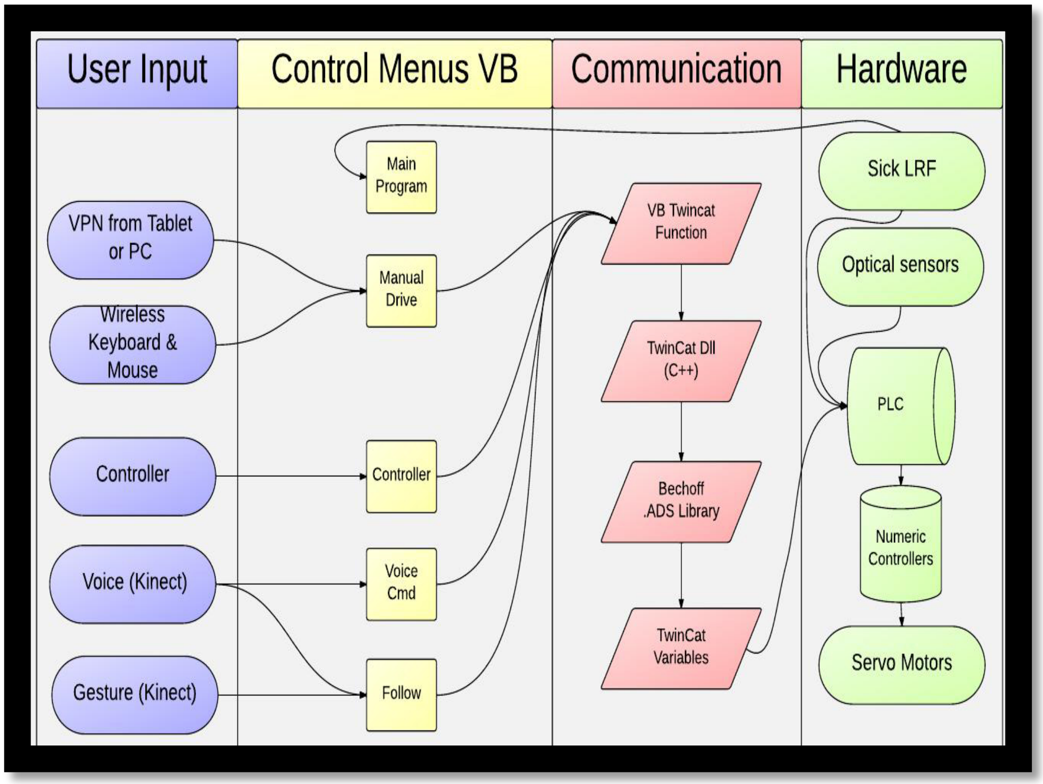
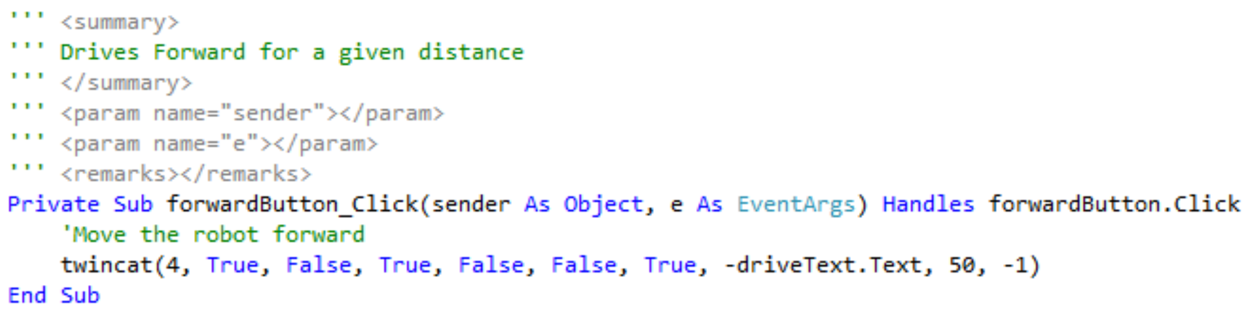


Figure 10: Flowchart of Program Structure



```
''' <summary>
''' Drives Forward for a given distance
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Private Sub forwardButton_Click(sender As Object, e As EventArgs) Handles forwardButton.Click
    'Move the robot forward
    twincat(4, True, False, True, False, False, True, -driveText.Text, 50, -1)
End Sub
```

Figure 11: VB .dll TwinCAT function

User Interface

Developing an effective and easy to use interface is extremely important when designing a product for developers and consumers. The computational power of computers far outweighs that of any user. They are excellent at receiving and processing data, however they are lacking in communication. Computers are primarily limited to a digital display to communicate and display information to the user; this is why it is extremely important to create a GUI, which is user friendly.

The GUI developed by the project team was created using Visual Basics for windows; it provides an excellent platform in which a program can be built from. Market share research shows that 84% of devices use Windows Operating Systems (OS) (NETMarketShare). If a user does not currently use a windows product, it is highly likely they have experienced it at some point in their lives. This provides old and new users alike a level of familiarity that makes them feel more comfortable with the project interface.

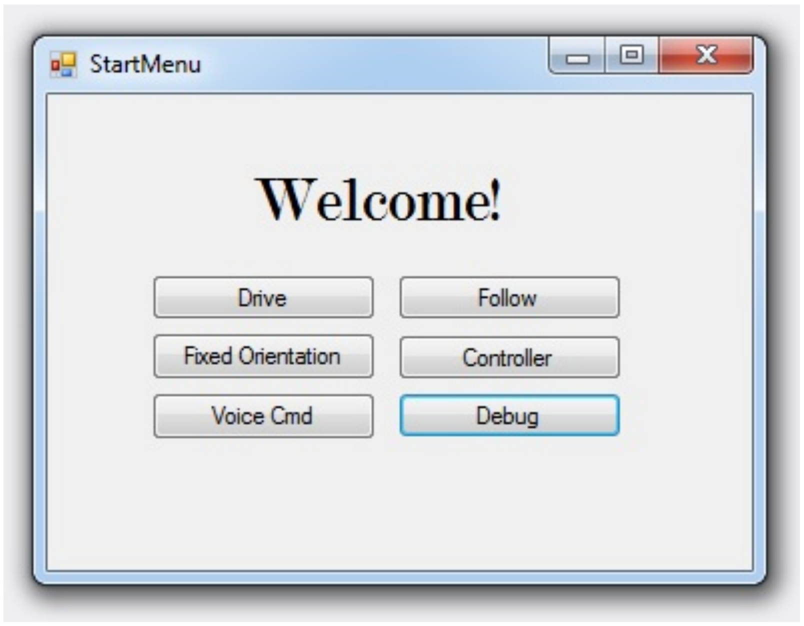


Figure 12: Program start menu (GUI)

The programs structure for user navigation is fairly simple. The start menu is comprised of 6 buttons linked to new forms, each of which can operate different control methods for the robot. While in the main menu the robot is disabled, it cannot be enabled until a selection is made in one of the alternate forms. Each form has different user control methods and many of them contain multiple methods in conjunction with one another. As a safety feature the program will not allow for multiple forms to be operated simultaneously.

The Drive form is primarily used for precise or predefined movements. As shown in Figure 13 the user can control the drive lengths, rotations, directions or send the robot to a local (X, Y) Coordinate. Fixed orientation is a demo mode that has a predefined "dance" that moves the robot in a very unique dance like path while maintaining a fixed orientation to the starting location. This mode demonstrates the unique wheel design that allows for simultaneous Drive and wheel axis rotation

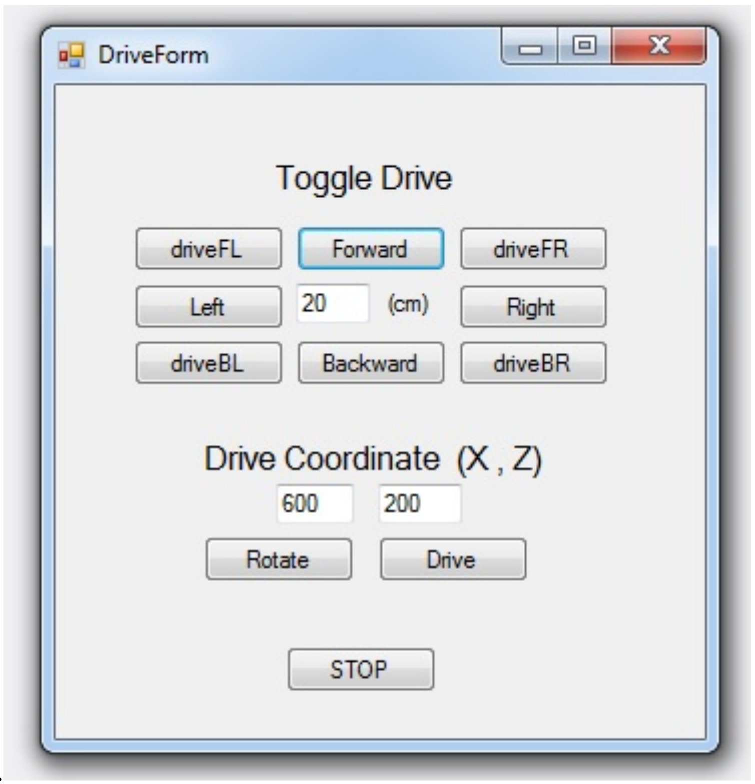
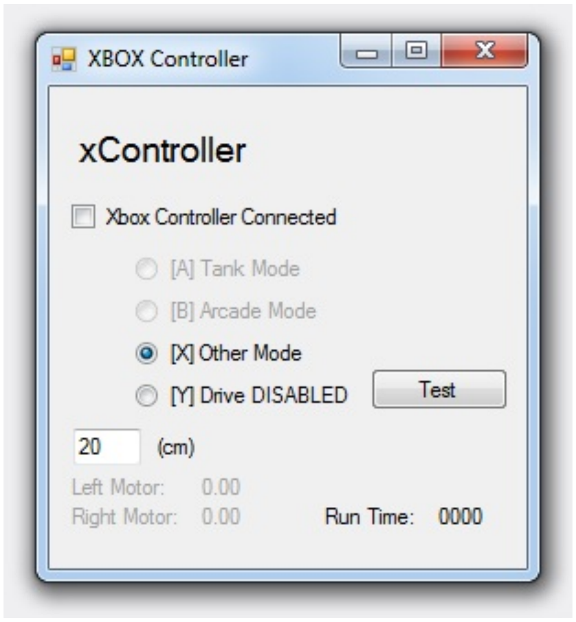


Figure 13: Drive form (GUI)

Microsoft Controller

A wireless Xbox 360 controller for windows was used for manual control, to activate this mode the user would select the Controller form shown in Figure 14. Feedback from the GUI notifies the user of the controller's connection status and displays the current drive mode. Alternate drive modes were created incase users may be more comfortable with different controller settings and buttons, a representation of the controller's layout can be found in Appendix B.

Different driving modes use data received from joysticks or buttons on the controller. Communication is made using the XInput API as described on page 17 which sends the appropriate information to the .dll drive function. Tank mode gets its name from video games but it is a highly used control method for any style 2 wheeled or tank treaded vehicle. Tank mode uses the Y coordinate value from the left joystick to control the left motors; the right joystick is used to control the right motors. A log file from tank mode is shown in Figure 15; the outputted value to the motor is a value from -1 to 1. Arcade mode controls the robot with only one joystick, the user simply pushes the left joystick in the desired direction and the (X, Y) coordinates are used to calculate the appropriate -1 to 1 value to output to the motors.



XBOX Controller

xController

Xbox Controller Connected

[A] Tank Mode

[B] Arcade Mode

[✓ [X] Other Mode]

[Y] Drive DISABLED

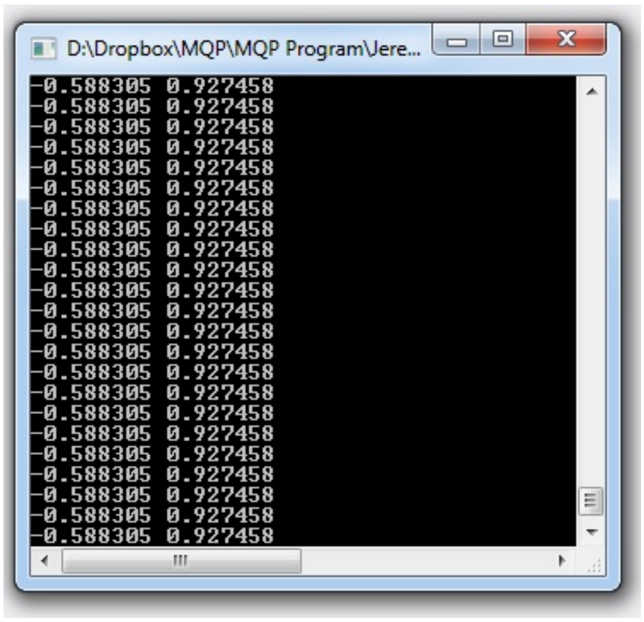
Test

20 (cm)

Left Motor: 0.00

Right Motor: 0.00

Run Time: 0000



D:\Dropbox\MQP\MQP Program\Jere...

```
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
-0.588305 0.927458
```

Figure 14: Controller Form (GUI)

Figure 15: Motor Output, Log file

Skeleton and Gesture Based Control

Follow Mode utilizes a Microsoft Kinect to track the movements of a user to determine what commands the user is giving. This mode tracks the physical position of a user and uses that information to infer commands given by the user. This mode is extremely useful for when a user dose not want to control every movement of the robot. With this mode a user can walk around and left the robot decide what movements are necessary in order to stay within a certain distance of the user.

First the follow mode program activates a session with the Kinect sensor. The Kinect then continuously sends to the program a depth image frame, a skeletal image frame, and a regular RGB image frame, thirty times each second. The follow program can then draw each frame to the screen this creates three video streams. The follow program is setup to show any or all of the video streams. This way the user can choose which video streams to use to understand exactly what the Kinect can and cannot detect.

Each time a new set of frames arrives the follow mode program checks to see if any skeletons were detected. If a skeleton has been detected the program then checks the XYZ information of each of the skeletons joints to a series of functions. Theses functions contain algorithms that compare the locations of a specific set of joints to determine if the user is in a command position. Positions where chosen so that they can be easily identified despite errors in the detection of exact joint positions. For example several functions check to see if the user has raised one or two hands above their head.

When a command position is detected a corresponding command is sent to the driving function. One command position, one hand on opposite shoulder, indicates the robot should do a predefined movement like spin. The command position one hand overhead indicates the robot should begin following a user at a predefined distance. Lowering the raised hand below ones head indicates the robot should stop following. Two hands clasped together indicate the robot should shut down. The program can be set to only react on the command of a single user or all users detected in a scene. If the program is set to respond to all users and users give different command positions the robot will do nothing.

The Follow command uses the position of the users joints to determine how the robot should move. At each frame the robot calculates the average distance to the user from the Z positions of all the known joints. During the follow command if the robot is closer or further away then the predefined following distance a corresponding command will be sent to the driving function. The follow command attempts to center the user in the view of the robot before driving forward or back.

The angle of the Kinect is adjusted to keep the user in the center to ensure as much as possible of the command user is in view. If the average Y position (vertical position) of the user falls into the bottom 1/3 of the scene or rises into the top 1/3 of the scene then a command is given to adjust the angle of the Kinect to center the scene on the users average joint height. A similar method is used to determine when the robot should rotate. The average x position of the joints of the user is used. If the x position of the user's center is in the left most 1/3 or right most 1/3 of the scene a command is sent to rotate the robot to correct number of degrees in order to bring the user back to center. If a specific command requires the position a specific joint the robot can center the scene around the position of that joint instead of the user's center.

Voice Control

In Voice Control Mode the Kinect sensor constantly scans the room looking for keywords as defined in its library. A library is created in an XML document, which correlates different keywords to processed audio waves to recognize commands. The C++ program matches speech to a list of strings in the XML document, the Speech API measures the accuracy of the sound wave to each word, and stores the confidence percentage as a variable. If a match is found, that meets the chosen percentage (ex. 80% confidence) the C++ program runs the method associated with the word.

For Example if microphone recognizes the words "robot" "drive", "forward" and a distance of "one" "meter" in sequence, the robot will complete the command. Or perhaps the robot hears "Robot follow me", the robot will use its beam forming technology to isolate the sound to a localized position from the robot. The robot will then rotate to the angle acquired from the microphones and search for the user (Brownlee). Once a user is found the program will launch the Following mode.

Chapter 5: Results

Weekly Progress:

Week 1: Explored methods to achieve tasks
Week 2: Repair hardware, boot & debug old program
Week 3: Achieved minimal communication with TwinCAT & C++
Week 4: Individual Kinect, and Controller Code written C++.
Redesigned TwinCAT program
Week 5: Achieved prototype driving (3DOF) between controller & Robot
Week 6: Combining all logic to VB
Week 7: User Interface, Final Assembly of Robot
Week 8: Debug and Organize commands

The weekly progress varies from the original plan outlined in the Gantt chart in Appendix C Good progress was made each week. The amount of time required to create a stable communication method between the control programs and the driving program was severely underestimated. The Gantt chart shows this task being completed during the third week. However, a working version of this method was not finished until the fifth week and final version was ready in the seventh week. Several errors and bugs also slowed the design Process.

Final Results

The Shanghai Service Robot is capable of taking user input from radio controller, Xbox Wireless controller, Wireless mouse and keyboard, voice control, and gesture based control. The robot is capable of driving in any direction and controlling its orientation. These capabilities were combined to achieve following of human users.

The robot is capable of a wide variety of complex movement due to the omnidirectional wheel design. For the Xbox controller the team created several driving modes that made use of the unique drive platform and the variety of buttons and joysticks available on the controller. The users could drive the robot in either tank or arcade mode for direct driving control and could run a preprogrammed path.

In voice control mode the robot could identify a human voice, determine what command words had been given. After identifying the correct command word the robot would perform the given driving command. The robot was listening to the sounds around it and could pick out command words said by one user, even when other people are talking.

In skeletal tracking mode the robot used the Kinect sensor to track a user. The program tracks twenty user joints and can maintain orientation to center its view on a chosen joint or the users average center position. The robot can track the user even if other humans entered and left the view of the robot as long as the main user did not. The gesture program determined if the user was giving commands by comparing the positions of the users joints. With this method several command gestures were created that told the robot when to follow how close to follow and where to look.

The Visual basic Mouse and keyboard control program allows the robot to be accessed directly from the robot screen or remotely from any other PC. In this mode any of the other modes could be activated and any of the driving commands could be given. The robot could also be told to go to a specific (x, y) location from its current orientation.

The robot was able to take and interpret commands from all of those input methods and move as desired. Only simple commands were used to demonstrate the functionality of the input methods that were created. Additional commands can easily be added that are suited to any type of situation.

Errors & Bugs

The shanghai service robot demonstrated all of the objectives. Though the robot was capable of the desired functionalities it sometimes functioned undesirably during testing. The team often encountered issues caused by loose wiring, but these issues were easy to fix. Rewiring the robot with new cables and securely mounting the electrical components solved the problem. There were some bugs that were harder to diagnose and solve.

One persistent bug occurred with the Kinect driver created by Microsoft. This driver allowed the computer to recognize the Kinect as a Kinect instead of an unknown piece of hardware. The driver would become corrupt and stop working if a program using the Kinect crashed or when the computer was shut down abruptly, as would happen if the power were unplugged. The only way to solve this issue is to uninstall the Kinect driver and reinstall it.

An error with PLC began to occur in the final week of testing. The PLC hardware would not communicate with the Beckhoff software that controls the processing of information from the PLC on the desktop computer's processor. The error began sporadically. At first a restart of the computer was required for the PLC hardware to begin communicating again. After that happened for a few days that no longer solved the problem. The final solution was to reinstall TwinCAT and all Beckhoff software, to replace the PLC and all terminal blocks with identical models, and to relink the TwinCAT program was with the new PLC. The cause of the error is still unknown

Not all of the issues were caused by physical events some were caused by the software controlling the robot. Objects with four legs such as tables or chairs would sometimes be recognized as human skeletons. This issue became problematic if the Kinect lost sight of the user being followed. The Kinect would sometimes track a chair or table instead of attempting to locate the skeleton of the real user that it just lost. This issue has to do with the accuracy of the algorithms that are used to identify human skeletons. These algorithms are part of the Microsoft SDK and would not have been feasible to change.

Another software issue was due to the limitations of the TwinCAT Libraries, PLC language, and the design of the communication. Movement based on acceleration control requires a system that can quickly adjust acceleration, for this real time control is ideal. The team was unable to achieve a real time system due to the delay that occurs between the user input program and activation of the motors. Some of the earlier designs of the driving function contained an argument for acceleration. Through testing it was found communication by the TwinCAT libraries was not quick enough. The PLC program could not keep up with the driving function and would create unreal values if the driving function were used repeatedly and rapidly as was required for acceleration-based control.

Chapter 6: Conclusion & Future Works

The Shanghai Service Robot could assist a worker to increase productivity. The robot could follow a worker or be commanded in to position while carrying heavy equipment. The robot is able to communicate through various methods and perform complicated driving movements. The orientation and position can easily be controlled through any of the control methods.

The Shanghai Service Robot prototype was able to demonstrate the specific tasks developed for this project. The control commands the project team created allow the robot to move at set speeds in any direction and in specific drive patterns, but the robot would need to be configured for the specific speeds and drive patterns required for a particular situation. Some future work would have to be completed before the robot could be used in an industrial environment where it would be expected to perform flawlessly

Future Work

The following functionality is slow and can sometimes be inaccurate. In a business situation where time and inaccuracy cost money the current prototype would not be ideal. Further work on the path planning and obstacle avoidance programming would need to be done before the robot could be used commercially.

The mechanical state of the robot is in a prototype stage comprised of aluminum fixtures and brackets. Ideally the team would have liked to redesign the base to make it lighter and perhaps generate a more user friendly appeal by removing sharp corners and making the housing specific for the materials used. If we were to redesign we would have housed the computer, PLC, and other circuitry staked towards the base plate to make a base that was only approximately 3 feet tall. With a smaller base it would allow the robot to have a lower center of gravity as well as leave space on the top for additional attachments and fixtures. Fixtures could include Monitors, Storage racks, robotic arms, medical devices, or any fixture a potential buyer may need.

Financial resources limited the components used in this design. If future work were to be done with this robot it would be ideal to replace several pieces of hardware. The team found that a lot of excess materials were no longer needed or could be accomplished using a cheaper product. Using the originally purchase price of the hardware and considering the current state, the robot could cost approximately $5000. Components such as the PLC, battery system, desktop computer, and drive train should be replaced. These components could be substituted in the future with newer cheaper hardware.

A newer PLC with an embedded processor would take processing strain off of the main control computer. With smaller processing requirements a laptop could be used as a control computer. Using a laptop instead of the desktop would free up space on the robot and decrease the weight of the robot. A laptop would also lessen the electrical draw on the batteries because a laptop could have its own internal battery. A set of relatively cheap nickel-cadmium batteries would increase the battery life of the robot and decrease the size and weight. Although there are advantages to the Omni directional wheel design, equivalent movement could be achieved through a simpler drivetrain design. A simple differential drive combined with a rotating platform above the robots base where the main components were housed could also maintain orientation while following a user. This method would reduce cost and allow the use of generic parts. This type of drive train could easily be changed depending on the specific requirements of the users, which would make the product more appealing to potential customers.

The team estimates that with a few altercations to the robot, the robot could be made approximately $4,000 and sold for $8,000-$10,000. This is a fair and reasonable price for an industrial robotic base capable of holding multiple attachments that has the ability to navigate in medical and warehouse like facilities.

With the completion of this future work the Shanghai Service Robot would thrive as an industrial robot in factories, medical facilities or warehouses. The industrial design of the Shanghai Service Robot gives it the capabilities to carry large loads and be very stable with limited risk of breaking. The many diffract communication methods it is capable of would make it very versatile. The Shanghai Service robots following and positioning abilities would make it extremely useful to many types of users.

Maserati is an example customer who may find the Shanghai Service Robot useful. The Shanghai Service Robot would thrive in Maserati’s Factory in Modena Italy, which is home to several robotic systems. They produce some of the highest performing and best quality cars on the market. They also have one of the most impressive high tech factories there is. Each Maserati car is accompanied by its own trolley while traveling down the production line. The trolleys house the parts for each car to ensure that no mistakes are made and parts are readily available. The Shanghai Service Robot could assist the trolley or worker to increase productivity.

Works Cited

Britton, Chris. *MSDN*. January 2008. Skyscrapr. <http://msdn.microsoft.com/en-us/library/cc168615.aspx>.

Brownlee, John. *The technology behind the Kinect's voice-recognition is ingenious, but doesn't work very well*. Geek.com, 5 August 2010. <http://www.geek.com/games/the-technology-behind-the-kinects-voice-recognition-is-ingenious-but-doesnt-work-very-well-1275099/>.

Comway. *12V Wireless 8 Channel Way Remote Control Switch*. WG-8010-232. n.d. <http://www.goodluckbuy.com/12v-wireless-6-channel-way-remote-control-switch-motor.html>.

King, Ritchie S. "The Top 10 Programming Languages." 28 September 2011. *Spectrum.* <http://spectrum.ieee.org/at-work/tech-careers/the-top-10-programming-languages>.

Kiplinger, Washington Editors, ed. *6 Fields Where Robots Are Taking Charge*. March 2013. <http://www.kiplinger.com/slideshow/business/T057-S005-robots-taking-charge/>.

MSDN_Kinect. *KinectAudioSource.BeamAngle Property*. Prod. Microsoft. 2012. <http://msdn.microsoft.com/en-us/library/microsoft.kinect.kinectaudiosource.beamangle.aspx>.

MSDN_VS. *Introducing Visual Studio*. Prod. Microsoft. 2010. <http://msdn.microsoft.com/en-us/library/fx6bk1f4(v=vs.80).aspx>.

NETMarketShare. "Top Operating System Share Trend." 2013. <http://marketshare.hitslink.com/os-market-share.aspx?qprid=9>.

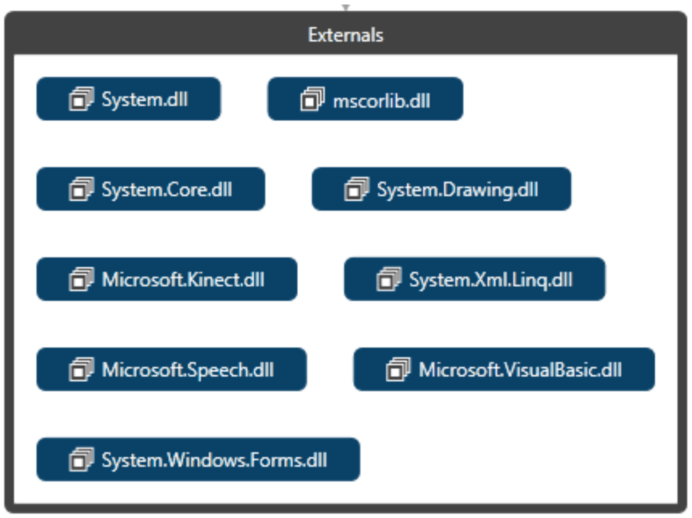
Swisslog. *Swisslog's Newest Mobile Robot Designed for Hospital-Wide Materials Transport*. Denver, 12 August 2010. <http://www.prweb.com/releases/2010/08/prweb4369204.htm>.

XInput. "XInput Versions." *Windows Dev Center*. Microsoft, n.d. <http://msdn.microsoft.com/en-us/library/windows/desktop/hh405051(v=vs.85).aspx>.

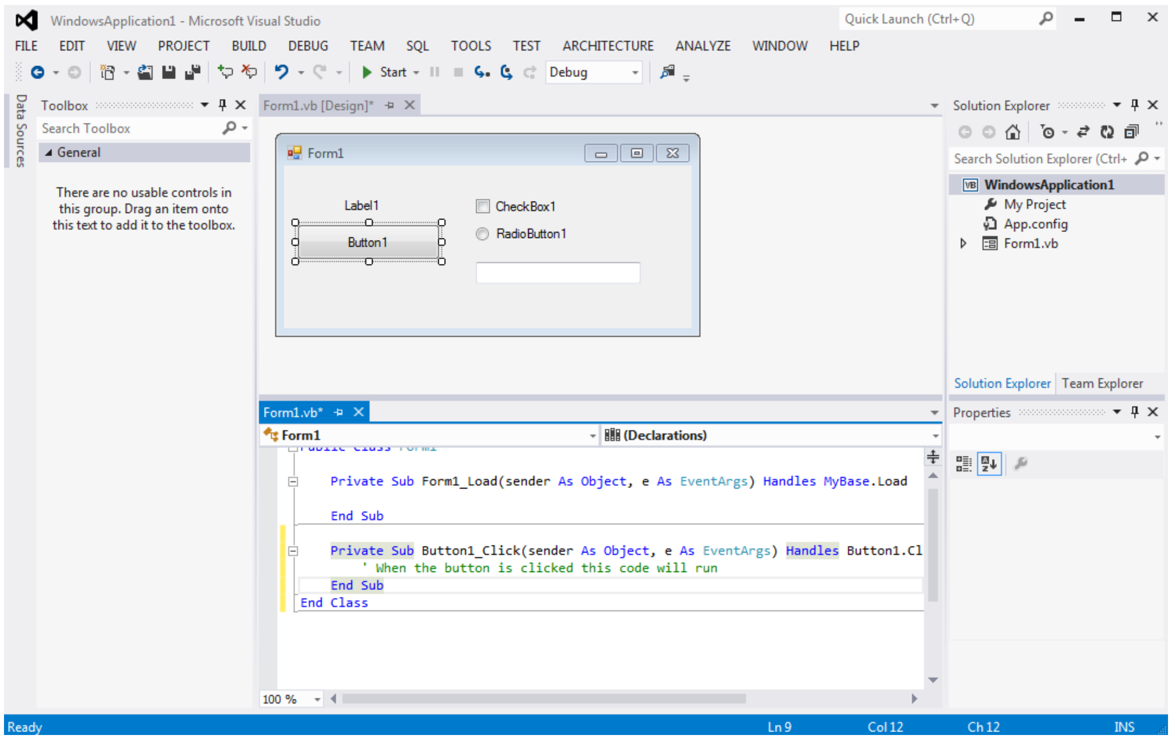
Appendix A

VB Examples

The Image below represents the externally referenced assemblies used in collaboration with our Visual Basics code, the figure is derived from Visual Studio's Dependency Graph function.



This Screenshot demonstrates the creation of a VB form utilizing objects such as Buttons, Labels, Checkboxes, and RadioButtons.



```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    End Sub

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Cl
        ' When the button is clicked this code will run
    End Sub
End Class
```

Appendix B

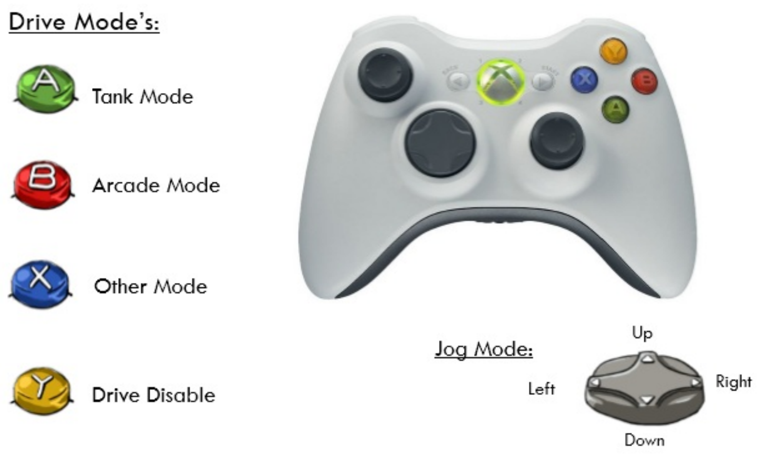
Gamepad

This is the default mapping and is designed around the standard Xbox 360 Common Controller gamepad, and is exposed as a Gamepad usage type.

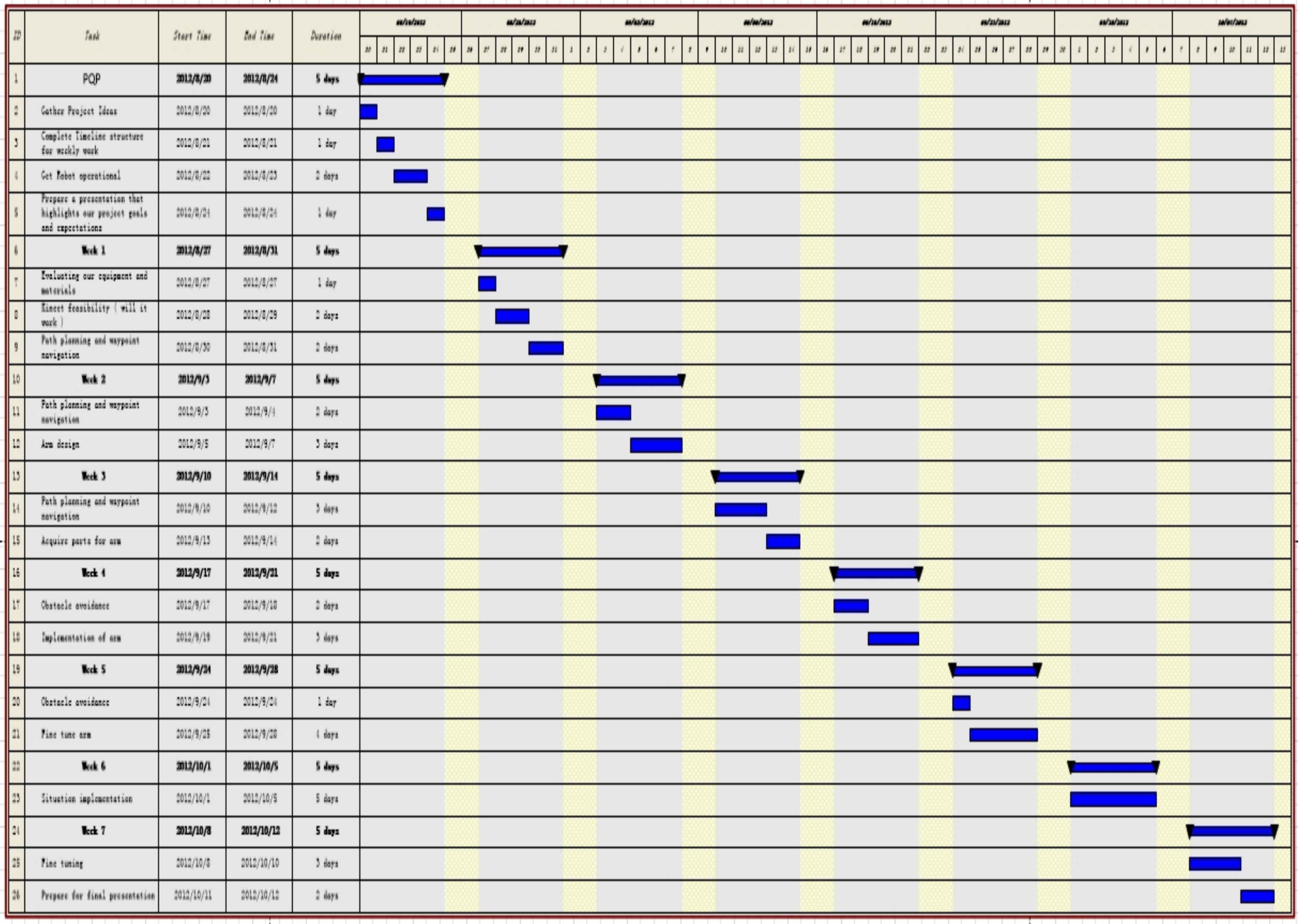
Control	DirectInput HID Mapping
Left Stick	X, Y
Right Stick	Rx, Ry
Left Trigger + Right Trigger	Rx*
D-Pad Up, Down, Left, Right	Hat Switch
A	Button1
B	Button 2
X	Button 3
Y	Button 4
LB (left bumper)	Button 5
RB (right bumper)	Button 6
BACK	Button 7
START	Button 8
LSB (left stick button)	Button 9
RSB (right stick button)	Button 10

Note (*): This is combined so that Rz exhibits the centering behavior expected by most titles for rotation; this does mean it is not possible to see all possible trigger combination values through DirectInput.

Controller drive modes



Appendix C



ID	Task	Start Time	End Time	Duration
1	**PQP**	**2012/8/20**	**2012/8/24**	**5 days**
2	Gather Project Ideas	2012/8/20	2012/8/20	1 day
3	Complete Timeline structure for weekly work	2012/8/21	2012/8/21	1 day
4	Get Robot operational	2012/8/22	2012/8/23	2 days
5	Prepare a presentation that highlights our project goals and expectations	2012/8/24	2012/8/24	1 day
6	**Week 1**	**2012/8/27**	**2012/8/31**	**5 days**
7	Evaluating our equipment and materials	2012/8/27	2012/8/27	1 day
8	Kinect feasibility (will it work)	2012/8/28	2012/8/29	2 days
9	Path planning and waypoint navigation	2012/8/30	2012/8/31	2 days
10	**Week 2**	**2012/9/3**	**2012/9/7**	**5 days**
11	Path planning and waypoint navigation	2012/9/3	2012/9/4	2 days
12	Arm design	2012/9/5	2012/9/7	3 days
13	**Week 3**	**2012/9/10**	**2012/9/14**	**5 days**
14	Path planning and waypoint navigation	2012/9/10	2012/9/12	3 days
15	Acquire parts for arm	2012/9/13	2012/9/14	2 days
16	**Week 4**	**2012/9/17**	**2012/9/21**	**5 days**
17	Obstacle avoidance	2012/9/17	2012/9/18	2 days
18	Implementation of arm	2012/9/19	2012/9/21	3 days
19	**Week 5**	**2012/9/24**	**2012/9/28**	**5 days**
20	Obstacle avoidance	2012/9/24	2012/9/24	1 day
21	Fine tune arm	2012/9/25	2012/9/28	4 days
22	**Week 6**	**2012/10/1**	**2012/10/5**	**5 days**
23	Situation implementation	2012/10/1	2012/10/5	5 days
24	**Week 7**	**2012/10/8**	**2012/10/12**	**5 days**
25	Fine tuning	2012/10/8	2012/10/10	3 days
26	Prepare for final presentation	2012/10/11	2012/10/12	2 days