The Report Committee for Carl William Schwab
certifies that this is the approved version of the following report:

Object Avoidance and Wall Following Using the Kinect

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Christine Julien

William Bard

Object Avoidance and Wall Following Using the Kinect

by

Carl William Schwab, B.S.E.E.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2011

Abstract

Object Avoidance and Wall Following Using the Kinect

Carl William Schwab, M.S.E.

The University of Texas at Austin, 2011

Supervisor: Christine Julien

The range camera in Microsoft's Kinect, intended for the Xbox 360 gaming console, offers a powerful alternative to the many standard sensors used in robotics for gathering spatial information about a robot's surroundings. The recently-released Kinect is the first commercially available product to provide depth data of its resolution and accuracy with a price tag within reach of many robotics projects.

The work described in this paper explores the feasibility of using this sensor by developing a robot that relies solely on the Kinect for sensory data. This robot successfully performs standard navigational procedures, demonstrating the possibility of integrating spatial information from the Kinect into a real-time robotics application. This paper documents the techniques used to integrate the Kinect into the system, highlighting the key benefits and limitations of the sensor.

Table of Contents

List of Figures

Chapter 1: Introduction and Overview

A wide range of sensors exist today for object detection in robotics. The two primary types of sensors used for near-range object detection are cameras [1, 2, 3] and single-direction range finders such as ultrasonic [4, 5] and infrared sensors [6, 7]. These two approaches both have their limitations. Algorithms for pulling depth data from camera imagery are complex and processor intensive [8]. Meanwhile single-direction sensor approaches are limited by the simplicity of these sensors: several are often required for robust coverage [9] and their accuracy varies based on the surfaces encountered [10]. Range cameras, or cameras capable of returning distance measurements rather than color, have until now been prohibitively expensive and low-resolution [11].

Microsoft's Kinect for Xbox 360 is a recently-released sensor meant for the Xbox 360 gaming console. This sensor's intended purpose is to interpret players' movements to enhance the gaming experience, allowing players to use their bodies as the controller. This is done through a number of sensors within the Kinect itself as well as an internal processor. These sensors include a standard RGB camera, a sophisticated array of microphones, and most interestingly for robotics, a relatively high-resolution range camera [12]. With Microsoft's recent release of the beta version of a software development kit (SDK) for this device, and the device's relatively affordable price tag, the door is now wide open for the integration of this sensor into robotics applications.

The work in this report explores using the Kinect's range camera data for object detection and wall-following. These two functions were chosen because they exercise common, essential navigational features of many robotics applications and as such serve as a good test for analyzing the value of the Kinect in this field. A robot, shown in Figure

1, was built to use the Kinect sensor as the sole source of environmental data for this navigation. Overall, the system was successful in demonstrating the intended robotics functions, although a few limitations to the Kinect's capabilities were identified.

Figure 1: The robot developed for testing the Kinect

The remainder of this paper is broken down into several chapters. Related Work provides a brief survey of relevant work and technologies. The System Overview chapter provides a high-level overview of the system's components, after which the Kinect Integration chapter describes the techniques used for integrating the Kinect into the system. This is followed by chapters providing a system evaluation, thoughts on potential future work, and an overall conclusion.

Chapter 2: Related Work

Robotics projects often use simple proximity or range sensors for object detection. These range from ultrasonic [4, 5] and infrared [6, 7] sensors for distance measurements (in one direction) to push-button and capacitance sensors for detecting nearby or touching objects. These sensors typically have the benefit of being economical and straight-forward to integrate into software. However, they do come with drawbacks. Serious consideration needs to be put into the exact placement of these sensors in order to detect all possible obstacles, and even then there are usually coverage gaps [9, 10]. In comparison, data-rich sensors such as cameras or the Kinect (while still having coverage gaps) offer more flexibility in that they capture more data than required allowing software to extrapolate the desired details later. Because of the rigid connection between single-direction sensor placement and software, updates to improve object avoidance often require hardware changes, which are typically more difficult and costly than software updates.

Extracting navigational information from video sources in real-time has become increasingly common in the last two decades due to advances in processing power, although many of the techniques have been around longer [12, 13]. Such computer vision techniques typically rely on first being able to extract visual features from images [12]. These features are often corners, points, or blocks of pixels which may be identified and tracked from one video frame to the next. Once these features are identified, other techniques may then be used to extract the desired information. Stereo vision, or the use of two cameras to provide depth perception, is the most common computer vision technique used for gathering depth information in robotics, as demonstrated in [1, 2, 3]. This technique works in the same way that humans perceive depth by leveraging a simple

fact of geometry: displacement of an object in the views of two side-by-side lenses is indirectly proportional to the object's distance from the lenses [13]. Computer vision does have its drawbacks: it is reliant on sufficient visual features in the environment, sufficient processing power, and complex algorithms. However, freely available and highly-optimized computer vision libraries, such as OpenCV [14], have made such algorithms readily available to those interested, significantly decreasing the effort required to develop computer vision applications.

Range imaging, or the use of range cameras, is an emerging technology used to essentially collect an array of depth readings similar to how a standard camera returns color readings. Prior to the release of the Kinect, the commercially available range cameras had much lower resolutions and came with price tags upwards of $9,000 [11], a price high enough to make them impractical for most common applications. Despite the prohibitive price tag, range imaging has been a subject of some interesting research including hand-posture recognition [16], geometric object classification [17], and map building [18].

With range imaging now much more accessible because of the Kinect, we are likely to see a surge of research in this field in the next few years. To date, only limited research has been published on robotics projects using the Kinect. These have included a robot using the Kinect for localization and navigation in a known setting [19] and a helicopter using the Kinect for flight control [20]. The sensor has, however, led to a number of interesting projects amongst hobbyists. The list at [21] includes attempted applications of the Kinect (with varying levels of success) such as telepresence, autonomous car navigation, and gesture navigation.

Chapter 3: System Overview

This chapter provides a high-level overview of the components of the system introduced in Chapter 1. It is divided into hardware and software sections.

HARDWARE

The system is comprised of several key hardware components: the iRobot Create, a laptop computer, the mechanical structure, and the Kinect sensor.

iRobot Create

The iRobot Create is a mobile robotics platform intended to provide a starting point for robotics projects; it is used in this system for exactly that reason. The iRobot Create provides the robot's mobility and serves as the base of the robot. All of the other components are mounted on this base. Although the Create is well-equipped with over 30 sensors, none of these were used for this system as this would have conflicted with the objective of navigating solely with data from the Kinect sensor.

The Create is connected to the laptop via a USB cable and a third-party RooStick adapter from RoboDynamics. The RooStick adapter provides the required level-shifting in order for a standard USB port to communicate with the serial port of the Create. This serial interface is used by the laptop in order to send control commands to the Create.

Laptop Computer

A light-weight laptop computer running Windows 7 is used as the main controller of the system. This laptop receives and interprets data from the Kinect and then issues commands to the Create to move the robot. Software was written to run on this laptop and make all data-interpretation and navigational decisions required for the robot. This

software is elaborated on in the following Software section, and the data interpretation techniques are elaborated on in the Kinect Integration chapter.

Mechanical Structure

The system is connected together using a mechanical structure of light-weight metal braces, Styrofoam, packing tape, and a shoestring. The main intent of this structure is to position the Kinect in a downwards-facing fashion directly above the base of the robot; the reasons for this positioning are elaborated on in Kinect Integration chapter.

Kinect Sensor

The Kinect sensor is mounted on top of the mechanical structure in such a fashion that it looks directly down on the remainder of the robot. This position provides it with the ability to survey the base of the robot and surrounding obstacles.

The Kinect is connected to the laptop via a USB connection. This connection is used to communicate with and retrieve data from the sensor. The Kinect is powered directly from pins on the cargo bay connector of the iRobot Create through a modified power cable.

As the Kinect sensor is the main focus of this system, further details on its integration are provided in Chapter 4.

SOFTWARE

The system software can be broken down into several components that approximately correspond to the hardware components. Software wrappers were written to read data from the Kinect sensor and to control the iRobot Create motors. A high-level control application was then written to control the overall system by calling into these hardware wrappers. The remainder of this section elaborates on these components and the technologies used.

Kinect Wrapper

Microsoft released the beta version of the first Kinect SDK in June of 2011, providing the first official means of programmatically interfacing with the Kinect. Prior to this release a number of outside parties had reverse engineered the communication link to the Kinect, however these approaches were unsupported and, according to Microsoft, technically illegal to use.

Using the Kinect SDK, a C#.NET wrapper was written to encapsulate the required functionality of the Kinect including the data-interpretation algorithms described in greater detail in the Kinect Integration chapter. The intent of this wrapper was to simplify the integration of the Kinect into the high-level control application.

iRobot Create Wrapper

The iRobot Create is controlled through a USB port on the laptop. This connection is treated like a standard COM port, allowing it to be controlled by sending and receiving serial commands. A C# wrapper was written to handle sending all of required serial commands to the iRobot Create. This consisted of the commands for initializing and closing the connection with the device and controlling the Create's movement. None of the Create's sensor readings were encapsulated as part of this because they were not required by this system.

The Create is capable of rotating in position, which was a contributing factor for its being chosen for this project. Additionally, the driving commands for the Create are optimized for turning. One of the expected input parameters for drive commands is the turn radius, making it straight-forward to command the device to proceed on paths with a gentle turn. This configurable turning radius ability was leveraged in the navigation algorithm to have it arc in order to relocate walls that it had strayed from.

Main Control Application

The main control application handles all high-level algorithms and control. It leverages the functionality provided by the two instrument wrappers and essentially makes decisions based on the Kinect input as to which direction to command the Create to move.

The main application was written as a C#.NET Forms application in Microsoft Visual Studio Express. This environment was chosen mainly because the Kinect SDK was designed for Visual Studio.

The main application was developed to display a GUI so as to provide debugging feedback during operation. A number of screenshots from the GUI have been used in this paper to illustrate the Kinect data and the logic used in the algorithms.

Chapter 4: Kinect Integration

This chapter details how the Kinect was integrated into the system in order to successfully navigate the robot around and along obstacles. It covers retrieving, processing, and utilizing the raw data returned from the sensor.

RAW DATA

The Kinect is capable of returning several types of raw data including depth information, pictures, "skeletal" data, and audio. The two novel data types are the depth and skeletal data, of which the depth data has been the focus of this effort. The Kinect sensor is itself comprised of several sensors and a processor. It exposes a combination of the raw data and processed data through the available API. The skeletal data is a prime example of the processed data made available; based on a combination of picture and depth data, the Kinect is able to isolate people in its view and pinpoint their current positions, returning nodes roughly corresponding to body positions.

For this particular system, the depth data is of primary interest. The Kinect returns raw depth data which is then converted to a two-dimensional array of data points containing depth information. This can be thought of as being analogous to a standard picture; however instead of each pixel representing a color in the red-green-blue color planes, each pixel represents a depth.

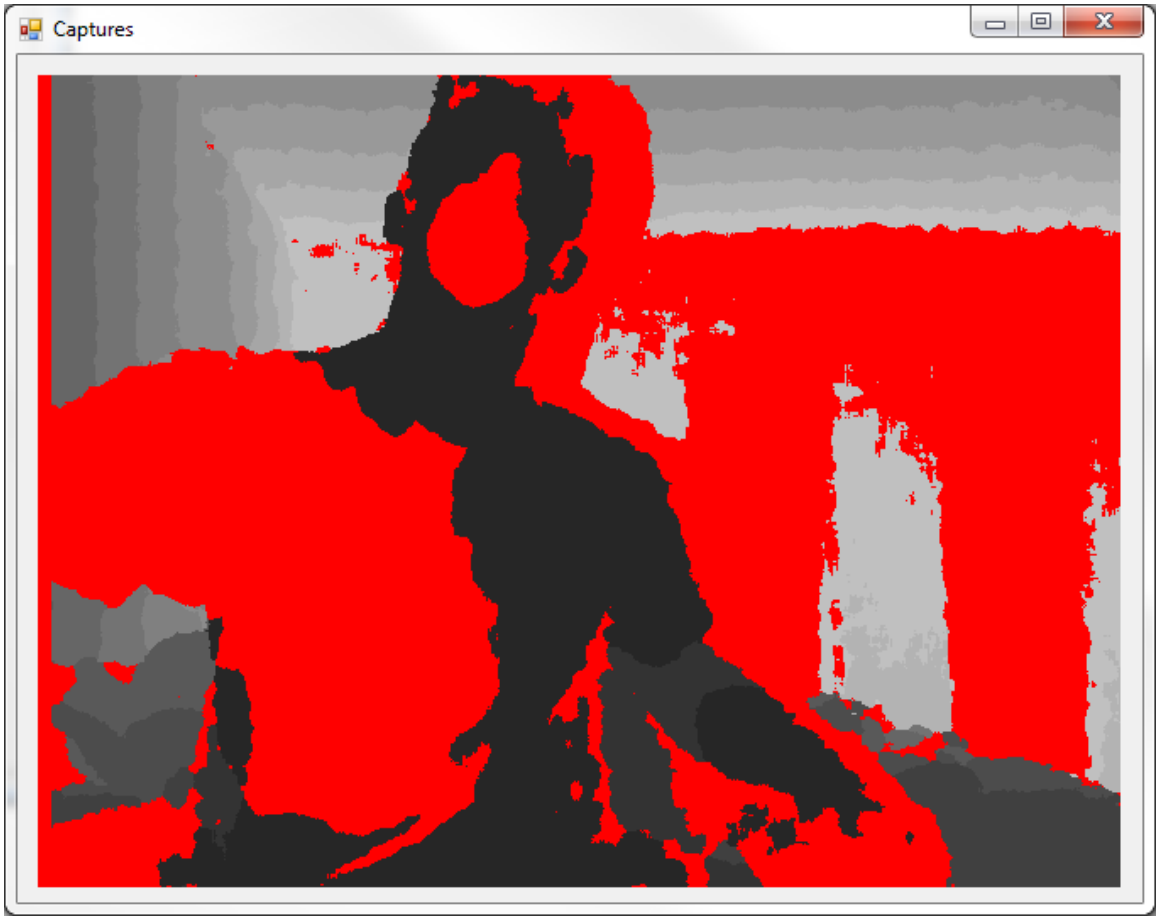

Figure 2: Kinect depth data represented as an image

As part of the initial investigation of the device, I converted the depth ranges to arbitrary colors and then displayed this as a video in a GUI to get a better understanding of how the device worked. A screenshot of this GUI is shown in Figure 2. In this screenshot, closer objects are dark in color and objects that are further away are lighter in color. Also of importance, there are noticeable red regions in this image; these are pixels for which the Kinect was unable to ascertain depth data.

After preliminary investigation, it became clear that a few factors can lead to the Kinect being unable to identify the depth for a particular pixel. Surfaces that are too close or too far away return no depth data. The minimum detectable range is approximately 0.9

meters, and the maximum detectable range is approximately six meters. The other more interesting observation is that shadows are also present in the depth data. Pixels at the edge of objects often return no depth data as a result of this shadow effect.

The raw array of depth data returned by the Kinect returns the distance of points relative to the perpendicular plane running through the lens of the Kinect. Thus, if the Kinect is facing a wall and held perpendicular to it, all readings will be approximately the same value.

SENSOR PLACEMENT

The main consideration for sensor placement was the distance limitation of the Kinect. Before initial experimentation, the preliminary concept was to mount the sensor in a forward-facing fashion so as to detect all objects in front of the device. However, this design was scrapped due to the minimum distance threshold of the Kinect sensor. Since it is not capable of distinguishing between points that are closer than 0.9 meters, farther away than 6 meters, or in shadow regions, it would have been difficult to implement a successful navigation algorithm with this design short of keeping a 0.9 meter buffer between all objects.

A design involving mirrors was briefly considered in order to allow a backwards-facing Kinect to detect objects ahead of it. While experimentally this proved feasible, it would have led to a complicated mechanical structure.

The final design implementation had the Kinect mounted directly above the Create (approximately 1 meter off the ground). This design had several significant advantages: it allowed the Kinect to continuously observe over 290 degrees around the perimeter of the Create, simplifying the navigational algorithms and it eliminated any need to remember a history of nearby objects. The significant drawback to this

implementation is the mechanical structure sticking over a meter out of the top of the robot, limiting its mobility.

DATA PROCESSING

Due to the sheer volume of data returned by the Kinect, it was necessary to develop an algorithm for making sense of the data. The sensor returns an array with 640x480 pixels of depth data. This data is processed through a multiple-step approach outlined in this section.

The first step is to determine which data points correspond to obstacles and which ones do not. This is done by setting up a low and high threshold for depth data; any readings outside of this range (including pixels with undetermined depth readings) are considered to be obstacles. These thresholds were determined through experimentation after the Kinect was mounted on the robot. The minimum threshold was originally set at such a value that the base of the robot was just within it, yet the floor was not. The intention of this was to allow robot to know the location of the base. The maximum threshold was set by placing the robot on the top of a step and finding the value for which it would detect the top of my hand when it was several centimeters from the plane of the top of the step. These thresholds were left intentionally loose at first to account for vibrations in the robot as well as well as for wiggle room in the sensor's exact placement on the tower. Using these thresholds, all depth data is converted to a binary result representing whether or not an object is visible at that particular location. Eventually these thresholds were tightened to within about 2 cm of the floor on either side.

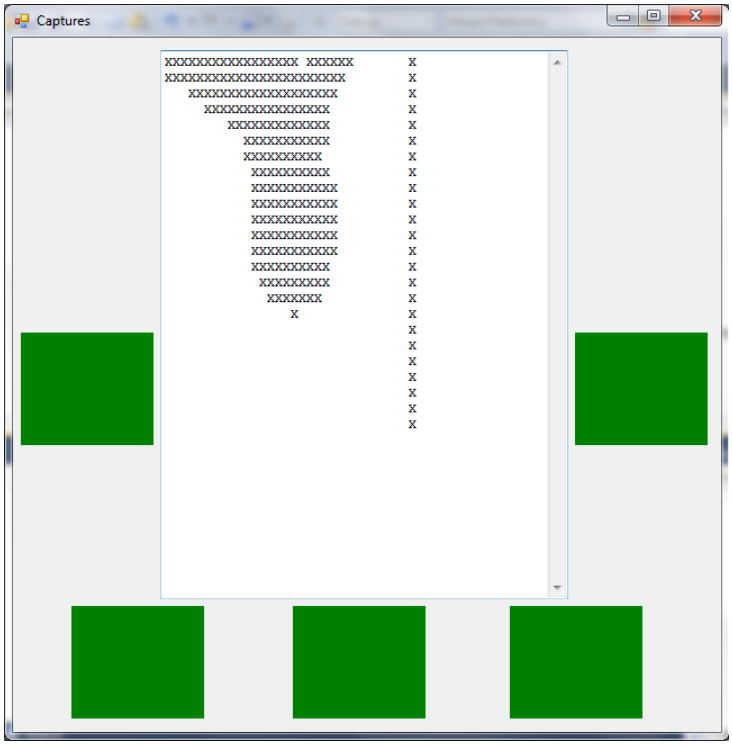

Figure 3: Screenshot of GUI with no obstacles detected

After converting the depth array to a binary array, the next step is to condense this data into smaller regions of data. Each 20x20 region of binary results is converted to a single element in a new array in an attempt to make the data more manageable. If ten or more percent of the pixels in the 20x20 region warn of a possible object, then the overall result of that region is flagged for an object. The ten percent tolerance is intended to filter out noise from shadows and movement which tends to cause sporadic bad depth measurements.

From this smaller array of binary data, I defined “soft sensor regions” corresponding to regions outside of the approximate edges of the robot. If any of the elements in these soft sensor regions indicate the presence of an object, then a “soft sensor” in turn indicates that an object is present. These soft sensors are then exposed

through the Kinect wrapper, allowing the navigation algorithms to simply work with the high-level soft-sensor data.

Figures 3 and 4 are both screenshots from the GUI of the main control application. X's in these screenshots represent 20x20 regions where the Kinect has detected something outside of the defined thresholds. The object in the middle of each of these is the base of the robot, the region of X's connecting the base to the top of the image is the mechanical structure, and the white space is the floor. The green and red boxes represent the values returned by the soft sensors. Figure 4 illustrates a situation in which one of the soft sensors was triggered by a can in the proximity of the robot. The vertical lines on the right side of each image are remnants of the data returned by the Kinect and are ignored by the soft sensors.

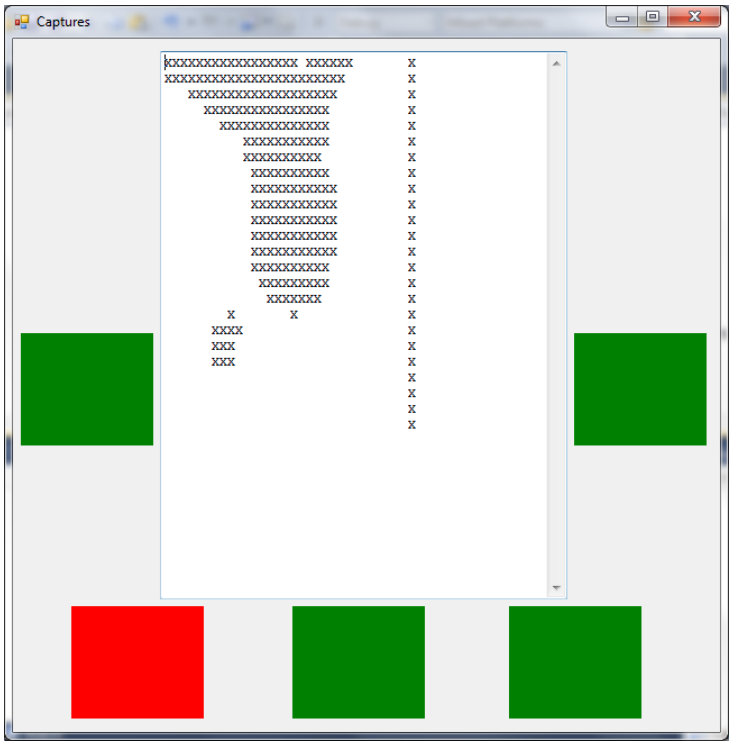

Figure 4: Screenshot of GUI with one obstacle present

NAVIGATION ALGORITHMS

The main control application uses the soft sensor data returned from the Kinect wrapper to make movement decisions and then calls into the Create wrapper to move the robot. After initial experimentation and tweaking, the algorithms were refined down to a straight-forward set of steps. Initially, the robot drives forward in a straight line until an object is detected. After that, it simply adheres to the following rules:

- If no obstacles are detected, then drive forward while veering right.
- If only the right soft sensor detects an obstacle, then drive straight forward.
- If other sensors detect obstacles, rotate in position to the left.

These rules were intended to ensure that the robot always kept objects (such as walls) on its right side, while never running into objects ahead of it. The initial implementation of the robot followed more complicated logic, but this proved to be unnecessary and more difficult to debug.

Chapter 5: Evaluation

The overall goal of this project was to demonstrate the Kinect's ability to provide sufficient sensory information to perform basic navigation. Object detection and wall following were chosen to illustrate this, and as such the system was evaluated on its ability to detect objects and follow their perimeters.

The system was tested in an indoor environment, on a flat floor, with obstacles of varying size as well as stairs. Overall, the robot was able to successfully avoid all of the standard obstacles used that were greater than approximately 3 cm in height. The robot was also able to successfully identify and avoid the drop-off from the step.

While stationary, the robot's depth thresholds were calibrated to detect objects as short as half a centimeter off the ground anywhere within its line of sight. However, while in motion, these thresholds had to be expanded due to less accurate readings from the Kinect due to vibrations in the tower and shifting of the Kinect in its mounting. Maintaining the minimum threshold at about 2 cm proved to consistently detect obstacles without introducing false positive detections. A more robust structure, slower speeds, and a smoother navigational algorithm could all theoretically allow for tightened depth thresholds.

The robot, as would be expected, did run into an overhanging table top which was higher than the Kinect sensor yet shorter than the top of the tower.

Once obstacles were detected, the robot was able to successfully follow their perimeter until a new obstacle was detected. With the soft sensor settings as they were, the robot maintained a buffer of approximately 20 cm between itself and objects. However, this buffer could easily be modified by adjusting the soft sensor locations in software.

The downwards-facing sensor does have limitations, such as the blind spot above the sensor as illustrated by the overhanging table top incident. A clear advantage of a forward-facing sensor, as had originally been looked into before the minimum depth range limitation was discovered, is that it would be able to see all objects as they approach.

Chapter 6: Future Work

This system is really a starting point for further exploration into the Kinect's applicability to robotics projects. The sensor itself provides a rich assortment of data, some of which, with some creativity, certainly could be applied to more advanced and interesting projects. For instance, the skeletal data could be used to have a robot follow, mimic, or take orders from people.

The high-resolution depth data returned by the Kinect could theoretically be passed into computer vision algorithms intended to handle video. Computer vision algorithms and processing power have become much more powerful over recent years and this combination of technologies could be applied towards such applications as 3-D environment mapping and autonomous off-road vehicle navigation.

Extensions of this particular project are also possible. One of the struggles with the current project is the instability of the Kinect mounting on the tower. Rather than building a sturdier tower, this issue could be resolved in software by extracting the plane of the floor from the depth data, and then using this plane as a reference point when detecting obstacles. Such an approach would allow for much tighter thresholds by essentially making the shifts in the Kinect location irrelevant.

Chapter 7: Conclusion

Overall, the Kinect proved to be applicable to this robotics project and was able to serve as a single-sensor alternative to the traditional sensors used for navigation. The Kinect proved to have a number of advantages over traditionally used sensors for obstacle detection, as well as some notable disadvantages. Nonetheless, the Kinect is certainly a tool with great potential in the field of robotics.

One of the big advantages that the Kinect holds over standard distance sensors is the ability to relegate many decisions to software. With standard distance and proximity sensors, much of the software logic is tied to the hardware and coverage is limited to what the sensors can sense. With an approach using the Kinect, much more granular information is available about the environment, allowing many decisions on threshold and region coverage to be handled in software.

The Kinect has several significant advantages over using computer vision-based approaches for object detection. In environments with limited features or lighting (e.g., a room with white walls, a white floor, and white obstacles or simply outside at night), obstacles are very difficult if not impossible to detect through computer vision. The Kinect is agnostic of its environment's visual features. Additionally, computer vision relies on more complex and processor-intensive algorithms than needed to work with the Kinect data, requiring significant computer resources and software development time.

The Kinect does have notable disadvantages over other mechanisms for obstacle avoidance. The most significant of these is its inability to detect objects outside of the 0.9 - 6 meter range which can be very restrictive, as was demonstrated by the need to mount the Kinect on top of a tower in this system. The Kinect is also more costly than many basic distance sensors and could be overkill for many projects for this reason.

References

[1] Sabe, K.; Fukuchi, M.; Gutmann, J.; Ohashi, T.; Kawamoto, K.; Yoshigahara, T., "Obstacle avoidance and path planning for humanoid robots using stereo vision," *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, Volume 1, pp. 592-597, 26 April-1 May 2004.

[2] Konolige, K; Agrawal, Motilal; Bolles, R.; Cowan, C; Fischler, M.; Gerkey, B., "Outdoor mapping and navigation using stereo vision," *Springer Tracts in Advanced Robotics*, Volume 39/2008, pp. 179-190, 2008.

[3] Huh, K.; Park, J.; Hwang, J.; Hong, D., "A stereo vision-based obstacle detection system in vehicles," *Optics and Lasers in Engineering*, Volume 46, Issue 2, pp. 168-178, 2008.

[4] Marioli, D.; Sardini, E.; Taroni, A., "Ultrasonic distance measurement for linear and angular position control," *IEEE Transactions on Instrumentation and Measurement*, Volume 37, Issue 4, pp. 578-581, 1988.

[5] Borenstein, J.; Koren, Y., "Obstacle avoidance with ultrasonic sensors," *IEEE Journal of Robotics and Automation*, Volume 4, Number 2, April 1988.

[6] Benet, G.; Blanes, F.; Simo, J.E.; Perez, P., "Using infrared sensors for distance measurement in mobile robots," *Robotics and Autonomous Systems 1006*, pp. 1-12, 2002.

[7] Huang, L., "Wall-following control of an infrared sensors guided wheeled mobile robot," *International Journal of Intelligent Systems Technologies and Applications*, Volume 7, Number 1, pp. 106-117, 2009.

[8] Lopez, M.; Sergiyenko, O.; Tyrsa, V., "Machine vision: approaches and limitations," *Computer Vision*, pp. 538, November 2008.

[9] Shillon, S.; Chakrabarty, K., "Sensor placement for effective coverage and surveillance in distributed sensor networks," *Wireless Communications and Networking*, Volume 3, pp. 1609-1614, 20-22 March 2003.

[10] Society of Robots, "Infrared vs. ultrasonic – what you should know", 27 January 2008, http://www.societyofrobots.com/member_tutorials/node/71.

[11] Hizook, "Low-cost depth cameras (aka ranging cameras or RGB-D cameras) to emerge in 2010?", 29 March 2010, http://www.hizook.com/blog/2010/03/28/low-cost-depth-cameras-aka-ranging-cameras-or-rgb-d-cameras-emerge-2010.

[12] Kinect for Windows, "Kinect for Windows Features," viewed 21 November, 2011, http://kinectforwindows.org/features.

[13] Brown, C.; Terzopoulos, D., "Real-time computer vision," 1994.

[14] Jarvis, R., "A perspective on range finding techniques for computer vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 5, Number 2, March 1983.

[15] OpenCV, "OpenCVWiki", http://opencv.willowgarage.com/wiki.

[16] Malassiotis, S.; Strintzis, M.G., "Real-time hand posture recognition using range data," *Image and Vision Computing*, Volume 26, Issue 7, pp. 1027-1037, 2 July 2008.

[17] Shin, J.; Gächter, S.; Harati, A.; Pradalier, C.; Siegwart, R., "Object classification based on geometric grammar with range camera," *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, pp. 2443-2448, 12-17 May 2009.

[18] Prusak, A.; Melnychuk, O.; Roth, H.; Schiller, I.; Koch, R., "Pose estimation and map building with a PMD-camera for robot navigation," *International Journal of Intelligent Systems Technologies and Applications*, Volume 5, Number 3-4, pp. 355-364, 2008.

[19] Cunha, J.; Pedrosa, E; Cruz, C.; Neves, A.; Lau, N., "Using a depth camera for indoor robot localization and navigation," *Robotics Science and Systems 2011 Workshop on Advanced Reasoning with Depth Cameras*, July 2011.

[20] Stowers, J.; Hayes, M.; Bainbridge-Smith, A., "Quadrotor helicopter flight control using Hough transform and depth map from a Microsoft Kinect sensor," *Conference on Machine Vision Applications*, Nara, Japan, 13-15 June 2011.

[21] IEEE Spectrum, "Top 10 robotic Kinect hacks," 7 March 2011, http://spectrum.ieee.org/automaton/robotics/diy/top-10-robotic-kinect-hacks.