FRAMEWORK FOR THE AUTOMATED DISASSEMBLY OF ELECTRONIC WASTE USING THE SOAR COGNITIVE ARCHITECTURE

BY

NICHOLAS M. DIFILIPPO

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

MECHANICAL ENGINEERING AND APPLIED MECHANICS

UNIVERSITY OF RHODE ISLAND

2016

DOCTOR OF PHILOSOPHY DISSERTATION

OF

NICHOLAS M. DIFILIPPO

APPROVED:

Dissertation Committee:

Major Professor Musa Jouaneh

David Chelidze

Walter Besio

Nasser H. Zawia
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND
2016

ABSTRACT

An experimental study was conducted to investigate the feasibility of integrating the cognitive architecture Soar with a robotic system to aid in the disassembly of electronic waste (e-waste). This study was broken up into different parts, first, to examine how well the different sensors and tools that composed the system worked before finally integrating the cognitive architecture Soar to assess how it improved the systems performance. Due to the increasing amounts of e-waste and the adverse effects it has on human health, a robotic system that can aid in the disassembly of e-waste is essential.

The first part in this study was to investigate the performance of three different models of the Microsoft Kinect sensor using the OpenNI driver from Primesense. The study explored the accuracy, repeatability, and resolution of the different Kinect models' abilities to determine the distance to a planar target. An ANOVA analysis was performed to establish if the model of the Kinect, the operating temperature, or their interaction were significant factors in the Kinect's ability to calculate the distance to the target. Different sized gauge blocks were also used to test how well a Kinect could reconstruct precise objects. Machinist blocks were used to examine how accurately the Kinect could reconstruct objects set up on an angle and determine the location of the center of a hole. All the Kinect models could determine the location of a target with a low standard deviation (less than 2 mm). At close distances, the resolutions of all the Kinect models were 1 mm. Through the ANOVA analysis, the best performing Kinect at close distances was the Kinect model 1414, and at farther distances was the Kinect model 1473. The internal temperature of the Kinect sensor influenced the distance

reported by the sensor. Using different correction factors, the Kinect could determine the volume of a gauge block and the angles machinist blocks were setup at, with under a 10 percent error.

After the Kinect's performance was characterized, the study continued and investigated the performance of an automated robotic system, which used a combination of vision and force sensing to remove screws from the back of laptops. This robotic system used two webcams, one that is fixed over the robot and the other mounted on the robot, as well as a sensor-equipped (SE) screwdriver. Experimental studies were conducted to test the performance of the SE screwdriver and vision system. The parameters that were varied included the internal brightness settings on the webcams, the method in which the workspace was illuminated, and color of the laptop case. A localized light source and a higher brightness setting as the laptop's case became darker produced the best results. In this study, the SE screwdriver successfully removed 96.5% of the screws from laptops.

Since a method to locate and remove screws from a laptop automatically, and guidelines to find holes based on the laptop color and camera brightness were established, the study continued and added the cognitive architecture language Soar to the system. Soar's long term memory module, semantic memory, was used to remember pieces of information regarding laptop models and screw holes. The system was trained with multiple laptop models, and the method in which Soar was used to facilitate the removal of screw was varied to determine the best performance of the system. In all cases, Soar could determine the correct laptop model and in what orientation it was placed in the system. Remembering the locations of holes decreased all the trial times for all the different laptop models by at least 60%. The system performed the best when the amount of training trials that were used to explore circle locations was limited as this decreased the total trial time by over 10% for most laptop models and orientations.

ACKNOWLEDGMENTS

I would like to take this opportunity to first thank my major advisor, Dr. Musa Jouaneh, for his continued support throughout my time as a graduate student. I am thankful for the advice and guidance that he has given me and for allowing me to explore different topics in the field of robotics to select a dissertation topic. I would also like to thank the members of my committee, Dr. Walter Besio, Dr. David Cheldize, Dr. Mohammad Faghri, and Dr. Richard Vaccaro for taking time out of their schedule and agreeing to be part of my doctoral experience.

I would like to recognize all the colleagues who have helped me in some way with my project; Dan Sherman, Angela Reisch, Gregory Turner, Manuel Muller, Mike Franzblau, Drew Davis, Zaid Odufuye, and Dr. John D. Jones. I am especially appreciative of Mike Northup who has helped me prepare and edit papers. I also would like to thank Jim Byrnes and Rob D'Ambrosca for helping with all the electrical questions I had, and Dave Ferreria and Joe Gomez in the machine shop for fabricating or modifying whatever I needed.

I would also like to take this opportunity to recognize my family, especially, my Mom, Dad, Grandma Di, and brother Mark. Without their help, love, and support, it would not have been possible for me to complete this project. A special thank you to my dog Boomer who was always willing to go on a walk when I just needed to relax, and was happy to see me whenever I came home. Finally, I would also like to thank my girlfriend Gina Tripodi for her continued love and support throughout these last four years especially whenever I became discouraged and needed someone to talk to.

DEDICATION

To my Grandma and Grandpa Palumbo, and my Grandpa Di

PREFACE

This dissertation is prepared using the manuscript format in accordance with the University of Rhode Island Graduate School Guidelines. This dissertation is composed of three manuscripts that have been combined to satisfy the requirements of the department of Mechanical, Industrial, and Systems Engineering.

Chapter 1 provides a review of the literature that details the motivation of this work. Topics touched upon include e-waste and the problems that it causes as well as potential robotics solutions using low cost tooling and sensors, and different robotic learning techniques. This chapter serves to provide relevant research in the field as well as an introduction to the different topics presented in this dissertation.

Chapter 2 details an experimental study that investigated the performance of three different Kinect sensor models. The accuracy, repeatability, and resolution of these different models were explored and an ANOVA analysis was performed to determine if the Kinect model, Kinect operating temperature, or their interaction were significant factors in how the Kinect measured the distance to an object. The Kinect's ability to reconstruct gauge blocks, lying flat on the ground, and machinist blocks, set up at various angles, were also investigated. This chapter was published in IEEE Sensors Journal.

Chapter 3 outlines a novel method that combines vision and force sensing to automatically identify and remove screws from the backside of different laptop models. This chapter introduces a sensor-equipped screwdriver and the computer logic used to identify and remove the screws. A study was undertaken to determine what combination of lighting, and camera parameters would result in the robot finding and removing the highest quantity of screws on the different color cases. This chapter has been submitted to IEEE Transactions on Automation Science and Engineering.

Chapter 4 focuses on using the Soar cognitive architecture to improve the robot's performance when removing screws from a laptop. Using Soar's long term memory modules, more specifically semantic memory, the robot can remember locations that had been checked for screws leading to a decrease in the overall test time. This chapter has been prepared to be submitted to IEEE Transactions on Automation Science and Engineering.

Chapter 5 provides a summary of the major findings on the material presented in this dissertation. Suggestions for future work using the Soar cognitive framework will also be provided.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Electronic Waste

Electronic waste (e-waste) is a problem that is quickly growing all over the world. While the market demand of electronic products is constantly increasing, the life cycles of these products are shortening, causing a build-up of e-waste that must be addressed. Some of the heavy metals that e-waste is composed of, (Lead (Pb), Mercury (Hg), Zinc (Zn), Copper (Cu), Cadmium (Cd), among others [1]), are detrimental to the health and well-being of both humans and the environment. It is not uncommon to see developing countries dispose of e-waste by burning, burying, or dumping it in the sea [2]. In these situations, heavy metals can be spread through the air as dust particles, or leach into the soil and water supplies contaminating them [3].

The village of Guiyu, located in the Guangdong region of China, is one example of a village heavily invested in the recycling of e-waste. About 80% of families have at least one family member that is engaged in some form of e-waste recycling operations [4]. In Guiyu, the process in which e-waste is recycled is very crude and not many preventative measures are taken to protect the workers from the hazardous materials that they are disassembling. Workers will melt solder off circuit boards over makeshift grills only using fans as a precautionary measure [5]. Many studies have been conducted to measure the wellness of the people there and the levels of heavy metals found in the region [4-14].

The Taizhou region of China, another area heavily invested in the recycling of e-waste, also shows elevated levels of heavy metals in the soil which can pose a significant threat to human health [15-17]. Fu *et al*. [15] found that the soil was heavily contaminated with Cd, Cu, and Hg and rice had high levels of Pb and Cd. Liang *et al*. [16] measured the amount of Polybrominated diphenyl ethers (PBDEs) in hens and found that they contained an elevated level of Decabromodiphenyl ether (BDE 209), concluding that there was a possibility for the risk of the transmission of these toxins to humans through the food chain. Ha *et al*. [18] found a high level of heavy metals in the soil around recycling sites in Bangalore, India that were at levels comparable or slightly lower than those in Guiyu, China.

The United Nations University- Institute for the Advanced Study of Sustainability (UNU-IAS) estimates the amount of e-waste generated globally in 2014 was 41.8 million metric tons, and this number is expected to grow to 50 million metric tons by 2018. In 2014, the latest year for which data is available, 7.1 million tons of e-waste was generated in the U.S. alone (a 630% increase compared to 1999), but only 15% was recycled [19].

In 2009, Robinson [3] argued that the number of computers a country has is indicative of the e-waste that country will produce. He stated in the next 10 years, Eastern Europe, Latin America, and China will all become major producers of e-waste. The above data proves there is a genuine need to recycle and recover reusable components from these discarded products.

Countries have been struggling with how to regulate and adapt to this influx of new of e-waste. Some countries in Asia and Europe have begun to enact “take-back” laws which require manufacturers of the product to be responsible for the end of life (EOL) disposal [2]. The United States does not currently have an all-encompassing federal law. However, some states, such as California, have adapted laws where the cost is passed as a small fee ($6-10) on to the consumer when the product is purchased. Maine mandates cities and towns are responsible for the collection and transportation of the e-waste to a consolidation facility where it is sorted according to manufacturers, who are then billed. Maryland requires manufacturers to pay $5,000 a year to sell their products in the state or pay $500 and set up their own collections [20].

The Basel Convention, the framework for regulating the international movement of e-waste, was endorsed by 116 nations in 1989 [21-22]. As of September 2016, 184 countries have ratified the convention and only the United States and Haiti have signed but not ratified the convention [23]. However, some countries including the United States, Canada, and Japan still export their e-waste to developing countries. A loophole in the law allows for companies to export old electronics as used goods to developing nations as a means to try to bridge a "digital divide." This loophole is exploited as much of the e-waste exported does not work as it was never meant to be used by the developing country [2]. In lieu of this, China is attempting to ban all e-waste imports from other countries. However, this is met with resistance by people profiting off the importing of e-waste into China and thus, the practice continues [24].

E-waste is recycled by destructive, semi-destructive, and non-destructive disassembly methods. Destructive disassembly is when the product is destroyed in order to disassemble it and is suitable if the objective is to recycle materials such as metals or plastics from the part. Semi-destructive disassembly will destroy parts of the object such as screws and snap-fits, while non-destructive disassembly is essentially the reverse engineering of the assembly cycle and is attractive if the goal of the disassembly is to reuse parts in different applications, or if the part that is being removed is hazardous. Manual (non-destructive) disassembly is costly and time-consuming since most products are designed for assembly and not disassembly. As a result, many electronic devices are simply fed into large shredders and processed to extract different materials [25]. While some plastics and metals can be easily separated during this process, separation of different types of metals and plastics is a difficult task. Shredding also prevents the re-use of components in new or refurbished products. Currently, destructive and semi-destructive disassembly are the most commonly used disassembly methods.

1.2 Robotics

According to the Roadmap for U.S. Robotics Report [26], robotics is a key transformative technology that can revolutionize manufacturing. However, several issues need to be addressed in order to increase the use of robots in manufacturing operations such as the automated disassembly of e-waste. These issues include the high cost of the robotic work cell, the creation of cost-effective force sensing tooling, and the need for structured software that will support the robot’s interaction with humans.

The high cost of the primary robotic hardware is only a fraction of the total cost of creating a robotic work cell. Since most industrial robots cannot operate in unstructured environments, a large amount of money is spent on designing and fabricating additional equipment, such as jigs and fixtures, so that parts and features are located at precise, pre-specified locations. If a robot could learn what tasks it needs to perform from using sensors and human guidance in a non-engineered environment, then these costs can be substantially reduced.

The Microsoft Kinect sensor is a low-cost sensor which integrates many different electrical and mechanical components. Some of these components include a traditional red-green-blue (RGB) camera, a depth sensor consisting of an infrared (IR) camera and projector, a microphone, and a geared direct current (DC) motor. This geared DC motor allows the base of the sensor to tilt up and down (± 27º). The Kinect works by projecting an IR speckle pattern on its surroundings and compares the pattern to a reference pattern. If an object in the scene is farther or closer than the reference plane, the speckles that are on the object will shift and by using a correlation procedure, the Kinect sensor can determine the distance to that object [27-28].

The Microsoft Kinect sensor was originally designed to be used to control games on the Microsoft Xbox 360 gaming console. Players can control characters or menus in a game by using their hand with the Kinect sensor rather than a controller. Multiple libraries are available to developers that open up the potential applications of the Kinect sensor to fields such as computer vision [29], 3D mapping [27] [30-31], robotics [32-36], medicine [37-44], human tracking [45-46], as well as others.

Examples of some of these applications include Shirwalker *et al.* [35] and Afthoni *et al*. [36] who used the Kinect sensor to obtain positions of an operator's joints to control a robotic arm. Alnowami *et al*. [38] and Tahavori *et al*. [39-40] used the

Kinect to monitor patients' respiratory breathing patterns while they are receiving external beam radiotherapy. It was found that the Kinect has mm-level errors at distances between 85 and 115 cm away from a patient and can determine various types of breathing patterns. Ning and Guo [43] showed that the Kinect sensor can be used to assess the spinal loading of the lumbosacral (L5/S1) joint which is directly associated with low back disorder. Yang *et al*. [44] assessed how the Kinect sensor performed in measuring the postural stability of a person. They reported that although the Kinect sensor needs to be calibrated by a set of linear equations, it was possible to measure a person's standing balance with the Kinect sensor comparable to standard testing equipment.

Besides vision, a robot interacting with the environment needs force information to guide that interaction. Sensors that are currently available, such as load cells, are not very cost-effective. It would be advantageous to develop tooling that utilizes reliable, low-cost force sensors such as force sensing resistors (FSRs).

Several researchers have investigated the design of disassembly tooling and grippers. Rebafka *et al.* [47] proposed a flexible unscrewing tool that creates its own acting surfaces to improve loosening. Park and Kim [48] discussed the development of a 6-axis force/moment sensor for an intelligent robot's gripper. Feldmann *et al.* [49] detailed the design of a drill driver, which can be used for three different methods of disassembling joining members. The drill driver could use an existing working point, drill to create a working point to remove a fastener, or drill to destroy the joint. Zuo *et al.* [50] presented a novel disassembly tool, wherein a screwnail is used as an end effector and a self-connection that results from the screwnail indentation provides a reliable closure to transmit forces and torques required for various dismantling operations. Seliger *et al.* [51] presented an unscrewing tool designed to optimize flexibility over accuracy since different variations of models of products can exist. The tool consists of a modified drill bit with movable needles (grippers) that are able to hold objects down while the disassembly steps are performed. Peeters *et al.* [52] designed a prying tool that could pull apart housing components to remove LCD screens. Schumacher and Jouaneh [53-54] used force sensing resistors in order to create a disassembly tool that could remove snap fits and spring-loaded batteries from calculators.

1.3 Learning

Unlike current industrial robots, new industrial robots will depend on humans to learn how to operate and improve their skills. This development requires the creation of structured software to support such interaction and learning so robots can execute their tasks more effectively. A lot of research has been undertaken involving the interaction of leveraging learning between humans and robots [55-61].

Learning techniques are widely used for teaching a robot how to correctly perform an action. Kormushev *et al.* [55] provided multiple examples of robots learning how to perform tasks in real-world environments. One robot that can flip pancakes is programmed by kinesthetic teaching (directly moving the robot’s limbs) with a reward factor for successful flips. Another robot used a vision system for feedback to teach itself how to shoot a bow and arrow so that the arrow will hit the center of a target.

Reinforcement learning algorithms described by Peters *et al.* [56] and Guenter *et al.* [57] allow the robot to create its own solution if the system is not able to perform the appropriate task. This type of learning method would work well if the robot was trying to learn from autonomous trial and error. Adams *et al.* [58] described a method of learning called Mixed Initiative Control that allows for a human operator to collaborate with the robot while the robot performs a task. Grollman and Jenkins [59] introduced a method called Dogged Learning which allows the robots behavior to be fine-tuned as the human requirements change. Thomaz *et al.* [60] used a modified reinforcement learning technique that gives a reward function to the robot based on how the robot performs a task while, additionally, allowing human interaction. This allows for the human to change the reward that the robot is receiving. There are different ways that robots can know that the task that they are trying to perform is correct or successful.

For a robot to make decisions based on an understanding of the parts presented to it, some form of intelligence should be built into the robot's control system. Kurup and Lebiere [61] showed that an algorithmic approach, where the user programs all possible scenarios that the robot can handle, is not efficient as the program will not be able to handle a scenario unless it was preprogrammed. Instead, an approach that uses cognition is preferred as it corresponds more with the way humans process and make decisions. Cognitive architectures make use of some form of high-level abstract reasoning to determine the next action to take and can incorporate different learning methods.

Two of the more popular cognitive architectures are Soar [62], and ACT-R [63]. ACT-R (current version is 7.0.11) and Soar (current version is 9.4) each have pros and cons associated with them. Jones *et al.* [64] identified that these versions of ACT-R and Soar have constraints that are opposites of each other in their respective low-level reasoning tasks. ACT-R only allows for one production rule to work at a time even if more rules are available. Using ACT-R, multiple passes must be made to make decisions, which increases computational time. Soar allows for multiple rule instantiations to fire at once. However, the computation time in Soar increases because only one type of operator rule is allowed to be selected at a time and the operators have to be proposed through different types of rules.

Hanford [65] suggested that Soar is a better choice to use for robots since it does not limit the access to working memory (short term), and can use all knowledge to figure out how to proceed with a situation. Johnson [66] also compared the two architectures and stated the only similarity between the two architectures is that they both organize control around a single-goal hierarchy. In conflict resolution, Soar tries to select an operator while ACT-R tries to select a rule. Soar tries to select an action based on all available knowledge, and if an impasse is detected, create a sub-goal. On the other hand, ACT-R will halt if it detects an impasse.

The more a robot performs a task the better it should become at performing the task, regardless of whether a learning method or a cognitive architecture is used. Depending on the type of robot and task performed, use of a cognitive architecture or a learning method may be more appropriate. For disassembly operations, a cognitive architecture is a better choice since the basic premise of how to disassemble an object will remain the same but the condition or layout of parts may change between models. This means the agent will have to reason out a disassembly plan, if a previous action worked, or what the next step will be.

Laird and Rosenbloom [67] used a robot called Robo-Soar [68], which was a PUMA arm that used a vision system to obtain orientation and position of objects in a workspace and to align these objects until a light comes on and then press a button. Hanford *et al.* [69-70] also used Soar to control mobile robots. With these robots, Soar used different sensor information to decide how the robot should navigate to a GPS location while avoiding obstacles. A robot named Rosie learned to play new games and perform other tasks using mixed initiative interactions through natural language [71-72] and visual demonstrations [73]. Rosie used semantic memory when learning to play a game to store all action knowledge, failure conditions, and goal states in order to determine a legal play

Vongbunyong *et al.* [74-76] described a system that disassembled LCD TV's using semi-destructive disassembly methods and a cognitive robotic agent. A vision system is used to obtain information about the product being disassembled and a cognitive agent using the language IndiGolog was used to determine the next disassembly step. The system presented here uses learning and revision primarily in order to eliminate redundant moves that are performed during a disassembly operation.

These issues described above are very important when considering the use of robots to perform disassembly of EOL electronic products. Unlike assembly operations on a production line, where product features and locations are known in advance, a robotic disassembly system for EOL products needs to work with a large variety of products of unknown sizes and, possibly, with damaged or missing features.

1.4 References

[1] A. Chen, K. N. Dietrich, X. Huo, and S. Ho, “Developmental neurotoxicants in e-waste: an emerging health concern.,” *Environ. Health Perspect.*, vol. 119, no. 4, pp. 431–438, Apr. 2010.

[2] I. C. Nnorom and O. Osibanjo, “Overview of electronic waste (e-waste) management practices and legislations, and their poor applications in the developing countries,” *Resour. Conserv. Recycl.*, vol. 52, no. 6, pp. 843–858, Apr. 2008.

[3] B. H. Robinson, “E-waste: an assessment of global production and environmental impacts.,” *Sci. Total Environ.*, vol. 408, no. 2, pp. 183–191, Dec. 2009.

[4] Y. Li, X. Xu, J. Liu, K. Wu, C. Gu, G. Shao, S. Chen, G. Chen, and X Huo, “The hazard of chromium exposure to neonates in Guiyu of China.,” *Sci. Total Environ.*, vol. 403, no. 1, pp. 99–104, Oct. 2008.

[5] A. O. W. Leung, N. S. Duzgoren-Aydin, K. C. Cheung, and M. H. Wong, “Heavy Metals Concentrations of Surface Dust from e-Waste Recycling and Its Human Health Implications in Southeast China,” *Environ. Sci. Technol.*, vol. 42, no. 7, pp. 2674–2680, Mar. 2008.

[6] W. J. Deng, P. K. K. Louie, W. K. Liu, X. H. Bi, J. M. Fu, and M. H. Wong, “Atmospheric levels and cytotoxicity of PAHs and heavy metals in TSP and PM2.5 at an electronic waste recycling site in southeast China,” *Atmos. Environ.*, vol. 40, no. 36, pp. 6945–6955, Nov. 2006.

[7] X. Huo, L. Peng, X. Xu, L. Zheng, B. Qiu, Z. Qi, B. Zhang, D. Han, and Z. Piao, “Elevated blood lead levels of children in Guiyu, an electronic waste recycling town in China.,” *Environ. Health Perspect.*, vol. 115, no. 7, pp. 1113–1117, Jul. 2007.

[8] W. Qu, X. Bi, G. Sheng, S. Lu, J. Fu, J. Yuan, and L. Li, “Exposure to polybrominated diphenyl ethers among workers at an electronic waste dismantling region in Guangdong, China.,” *Environ. Int.*, vol. 33, no. 8, pp. 1029–1034, Nov. 2007.

[9] X. Z. Yu, Y. Gao, S. C. Wu, H. B. Zhang, K. C. Cheung, and M. H. Wong, “Distribution of polycyclic aromatic hydrocarbons in soils at Guiyu area of

China, affected by recycling of electronic waste using primitive technologies," *Chemosphere*, vol. 65, no. 9, pp. 1500–1509, Nov. 2006.

[10] A. O. W. Leung and A. S. Wong, "Spatial Distribution of Polybrominated Diphenyl Ethers and Polychlorinated Dibenzo- p -dioxins and Dibenzofurans in Soil and Combusted Residue at Guiyu , an Electronic Waste Recycling Site in Southeast China," *Environ. Sci. Technol.*, vol. 41, no. 8, pp. 2730–2737, Mar. 2007.

[11] D. Wang, Z. Cai, G. Jiang, A. Leung, M. H. Wong, and W. K. Wong, "Determination of polybrominated diphenyl ethers in soil and sediment from an electronic waste recycling facility.," *Chemosphere*, vol. 60, no. 6, pp. 810–816, Aug. 2005.

[12] C. S. C. Wong, N. S. Duzgoren-Aydin, A. Aydin, and M. H. Wong, "Evidence of excessive releases of metals from primitive e-waste processing in Guiyu, China.," *Environ. Pollut.*, vol. 148, no. 1, pp. 62–72, Jul. 2007.

[13] X. Xu, Y. Zhang, T. A. Yekeen, Y. Li, B. Zhuang, and X. Huo, "Increase male genital diseases morbidity linked to informal electronic waste recycling in Guiyu, China.," *Environ. Sci. Pollut. Res.*, vol. 21, no. 5, pp. 3540–3545, Mar. 2014.

[14] W. Ni, Y. Chen, Y. Huang, X. Wang, G. Zhang, J. Luo, and K. Wu, "Hair mercury concentrations and associated factors in an electronic waste recycling area, Guiyu, China.," *Environ. Res.*, vol. 128, pp. 84–91, Jan. 2014.

[15] J. Fu, Q. Zhou, J. Liu, T. Wang, Q. Zhang, and G. Jiang, "High levels of heavy metals in rice (Oryza sativa L.) from a typical E-waste recycling area in southeast China and its potential risk to human health.," *Chemosphere*, vol. 71, no. 7, pp. 1269–1275, Apr. 2008.

[16] S. X. Liang, Q. Zhao, Z. F. Qin, X.R. Zhao, Z. Z. Yang, and X. B. Xu, "Levels and Distribution of Polybrominated Diphenyl Ethers in Various Tissues of Foragin Hens from an Electronic Waste Recycling Area in South China," *Environ. Toxicol. Chem.*, vol. 27, no. 6, pp. 1279–1283, Jan. 2008.

[17] J. K. Y. Chan and G. H. U. A. Xing, "Body Loadings and Health Risk Assessment of Polychlorinated Dibenzo- p -dioxins and Dibenzofurans at an Intensive Electronic Waste Recycling Site in China," *Environ. Sci. Technol.*, vol. 41, no. 22, pp. 7668–7674, Oct. 2007.

[18] N. N. Ha, T. Agusa, K. Ramu, N. P. C. Tu, S. Murata, K. A. Bulbule, P. Parthasaraty, S. Takahashi, A. Subramanian, and S. Tanabe, "Contamination by trace elements at e-waste recycling sites in Bangalore, India.," *Chemosphere*, vol. 76, no. 1, pp. 9–15, Jun. 2009.

[19] C. P. Baldé, F. Wang, R. Kuehr, and J. Huisman, "The Global E-Waste Monitor - 2014," United Nations University, IAS - SCYCLE, Bonn, Germany, 2015. [Online]. Available: https://i.unu.edu/media/unu.edu/news/52624/UNU-1stGlobal-E-Waste-Monitor-2014-small.pdf. Accessed on: October 21, 2016.

[20] J. R. Gregory and R. E. Kirchain, "A Comparison of North American Electronics Recycling Systems," in *Proc. 2007 IEEE Int. Symp. Electronics and Environment*, pp. 227–232.

[21] "Basel Convention on the control of Transboundary Movements of Hazardous Wastes and their Disposal," 2014. [Online]. Available: http://www.basel.int/Portals/4/Basel Convention/docs/text/BaselConventionText-e.pdf. Accessed on : Oct. 20, 2016.

[22] D. P. Hackett, "An assessment of the Basel convention on the control of Transboundary Movements of Hazardous Wastes and their Disposal," *Am. Univ. Int. Law Rev.*, vol. 5, no. 2, pp. 291–323, 1990.

[23] "Parties to the Basel Convention," 2013. [Online]. Available: http://www.basel.int/Countries/StatusofRatifications/PartiesSignatories/tabid/1290/Default.aspx. Accessed on: Oct. 20, 2016.

[24] H. G. Ni and E. Y. Zeng, "Law Enforcement and Global Collaboration are the Keys to Containing E-Waste Tsunami in China," *Environ. Sci. Technol.*, vol. 43, no. 11, pp. 3991–3994, Jun. 2009.

[25] J. Cui and E. Forssberg, "Mechanical recycling of waste electric and electronic equipment: a review," *J. Hazard. Mater.*, vol. 99, no. 3, pp. 243–263, May 2003.

[26] "A Roadmap for U . S . Robotics," 2013. [Online]. Available: https://robotics-vo.us/sites/default/files/2013 Robotics Roadmap-rs.pdf. Accessed on: Oct. 20, 2016.

[27] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of Kinect depth data for indoor mapping applications.," *Sensors*, vol. 12, no. 2, pp. 1437–1454, Jan. 2012.

[28] D. Um, D. Ryu, and M. Kal, "Multiple Intensity Differentiation for 3-D Surface Reconstruction With Mono-Vision Infrared," *IEEE Sens. J.*, vol. 11, no. 12, pp. 3352–3358, Dec. 2011.

[29] M. R. Andersen, T. Jensen, P.Lisouski, A.K. Mortensen, M. K. Hansen, T.Gregersen, and P. Ahdrendt., “Kinect Depth Sensor Evaluation for Computer Vision Applications”, Dept. Eng. Electr. Comput. Eng., Aarhus Univ., Aarhus, Denmark. Tech. Rep. ECE-TR-6, Feb. 2012.

[30] P. Henry, M. Krainin, E. Herbst, X. En, and D. Fox, “RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments,” *Int. J. Rob. Res.*, vol. 31, no. 5, pp. 647–663, Feb. 2012.

[31] N. Rafibakhsh, J. Gong, M. K. Siddiqui, C. Gordon, and H. F. Lee, “Analysis of XBOX Kinect Sensor Data for Use on Construction Sites : Depth Accuracy and Sensor Interference Assessment,” in *Construction Research Congr.*, 2012, pp. 848–857.

[32] M. Tölgyessy and P. Hubinský, “The Kinect Sensor in Robotics Education,” in *Proc. 2nd Int. Conf. Robotics Education*, 2011, pp. 143–146.

[33] J. Stowers, M. Hayes, and A. Bainbridge-Smith, “Altitude control of a quadrotor helicopter using depth map from Microsoft Kinect sensor,” in *2011 IEEE Int. Conf. Mechatronics*, pp. 358–362.

[34] R. A. El-iaithy, J. Huang, and M. Yeh, “Study on the Use of Microsoft Kinect for Robotics Applications,” in *Position Location and Navigation Symp.*, 2012, pp. 1280–1288.

[35] S. Shirwalkar, A. Singh, K. Sharma, and N. Singh, “Telemanipulation of an industrial robotic arm using gesture recognition with Kinect,” in *2013 Int. Conf. Control Automation, Robotics and Embedded Syst*, pp. 1–6.

[36] R. Afthoni, A. Rizal, and E. Susanto, “Proportional Derivative Control Based Robot Arm System Using Microsoft Kinect,” in *2013 Int. Conf. Robotics, Biomimetics, Intelligent Computational Syst.*, pp. 25–27.

[37] L. Gallo, A. P. Placitelli, and M. Ciampi, “Controller-free exploration of medical image data : experiencing the Kinect,” in *Int. Symp. Computer-Based Medical Syst.*, 2011, pp. 1–6.

[38] M. Alnowami, B. Alnwaimi, F. Tahavori, M. Copland, and K. Wells, “A quantitative assessment of using the Kinect for Xbox360 for respiratory surface motion tracking,” in *Proc. SPIE Medical Imaging 2012: Image-Guided Procedures, Robotic Interventions, and Modeling*, vol. 8316. pp T1-T8.

[39] F. Tahavori, M. Alnowami, and K. Wells, “Marker-less respiratory motion modeling using the Microsoft Kinect for Windows,” in *Proc. SPIE Medical Imaging 2014: Image-Guided Procedures, Robotic Interventions, and Modeling*, vol. 9036, pp K1-K10.

[40] F. Tahavori, M. Alnowami, J. Jones, P. Elangovan, E. Donovan, and K. Wells, "Assessment of Microsoft Kinect Technology (Kinect for Xbox and Kinect for Windows) for Patient Monitoring during External Beam Radiotherapy," in *Nuclear Science Symp. and Medical Imaging Conf.*, 2013, pp. 1–5.

[41] R. Clark Y. H. Pua, K. Fortin, C. Ritchie, K. E. Webster, L. Denehy, and A. L. Bryant., "Validity of the Microsoft Kinect for assessment of postural control," *Gait Posture*, vol. 36, no. 3, pp. 372–377, Jul. 2012.

[42] B. Lange, C. Y. Chang, E. Suma, B. Newman, A. S. Rizzo, and M. Bolas, "Development and evaluation of low cost game-based balance rehabilitation tool using the Microsoft Kinect sensor.," in *Proc. 2011 Ann. Int. Conf. IEEE Engineering Medicine and Biology Soc.*, pp. 1831–1834.

[43] X. Ning and G. Guo, "Assessing Spinal Loading Using the Kinect Depth Sensor: A Feasibility Study," *IEEE Sens. J.*, vol. 13, no. 4, pp. 1139–1140, Apr. 2013.

[44] Y. Yang, F. Pu, Y. Li, S. Li, Y. Fan, and D. Li, "Reliability and Validity of Kinect RGB-D Sensor for Assessing Standing Balance," *IEEE Sens. J.*, vol. 14, no. 5, pp. 1633–1638, May 2014.

[45] D. S. Alexiadis, P. Kelly, P. Daras, N. E. O'Connor, T. Boubekeur, and M. Ben Moussa, "Evaluating a dancer's performance using kinect-based skeleton tracking," in *Proc. 19th ACM Int. Conf. Multimedia*, 2011, pp. 659–662.

[46] B. Southwell and G. Fang, "Human Object Recognition Using Colour and Depth Information from an RGB-D Kinect Sensor," *Int. J. Adv. Robot. Syst.*, vol. 10, Mar. 2013.

[47] U. Rebafka, G. Seliger, A. Stenzel, and B. Zuo, "Process model based development of disassembly tools," *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.*, vol. 215, no. 5, pp. 711–722, May 2001.

[48] J. J. Park and G. S. Kim, "Development of the 6-axis force/moment sensor for an intelligent robot's gripper," *Sensors Actuators A Phys.*, vol. 118, no. 1, pp. 127–134, Jan. 2005.

[49] K. Feldmann, S. Trautner, and O. Meedt, "Innovative Disassembly Strategies Based on Flexible Partial Destructive Tools," *Annu. Rev. Control*, vol. 23, pp. 159–164, 1999.

[50] B. R. Zuo, A. Stenzel, and G. Seliger, "A novel disassembly tool with screwnail endeffectors," *J. Intell. Manuf.*, vol. 13, no. 3, pp. 157–163, Jun. 2002.

[51] G. Seliger, T. Keil, U. Rebafka, and A. Stenzel, "Flexible disassembly tools," in *Proc. 2001 IEEE Int. Symp. Electronics and Environment*, pp. 30–35.

[52] J. R. Peeters, P. Vanegas, C. Mouton, W. Dewulf, and J. R. Duflou, "Tool Design for Electronic Product Dismantling," *Procedia CIRP*, vol. 48, pp. 466–471, Jul. 2016.

[53] P. Schumacher and M. Jouaneh, "A force sensing tool for disassembly operations," *Robot. Comput. Integr. Manuf.*, vol. 30, no. 2, pp. 206–217, Apr. 2014.

[54] P. Schumacher and M. Jouaneh, "A system for automated disassembly of snap-fit covers," *Int. J. Adv. Manuf. Technol.*, vol. 69, pp. 2055–2069, Jul. 2013.

[55] P. Kormushev, S. Calinon, and D. Caldwell, "Reinforcement Learning in Robotics: Applications and Real-World Challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, Jul. 2013.

[56] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement Learning for Humanoid Robotics," in *Proc. 3rd IEEE-RAS Int. Conf. Humanoid Robots*, 2003, pp. 1–20.

[57] F. Guenter, M. Hersch, S. Calinon, and A. Billard, "Reinforcement learning for imitating constrained reaching," *Adv. Robot.*, vol. 21, no. 13, pp. 1521–1544, 2007.

[58] J. A. Adams, P. Rani, and N. Sarkar, "Mixed-Initiative Interaction and Robotic Systems," in *Proc. AAAI Workshop Supervisory Control Learning and Adaptive Syst.*, 2004, pp. 6–13.

[59] D. H. Grollman and O. C. Jenkins, "Dogged Learning for Robots," in *Proc. 2007 IEEE Int. Conf. Robotics and Automation*, pp. 2483–2488.

[60] A. L. Thomaz, G. Hoffman, and C. Breazeal, "Real-Time Interactive Reinforcement Learning for Robots," in *Proc. of AAAI Workshop on Human Comprehensible Machine Learning*, 2005, pp. 1-5.

[61] U. Kurup and C. Lebiere, "What can cognitive architectures do for robotics?," *Biol. Inspired Cogn. Archit.*, vol. 2, pp. 88–99, Oct. 2012.

[62] J. E. Laird, A. Newell, and P. S. Rosenbloom, "Soar: An architecture for general intelligence," *Artif. Intell.*, vol. 33, no. 1, pp. 1–64, Sep. 1987.

[63] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin, “An integrated theory of the mind.,” *Psychol. Rev.*, vol. 111, no. 4, pp. 1036–60, Oct. 2004.

[64] R. M. Jones, C. Lebiere, and J. A. Crossman, “Comparing Modeling Idioms in ACT-R and Soar,” in *Proc. 8th Int. Conf. Cognitive Modeling*, 2004, pp. 49–54.

[65] S. D. Hanford, “A Cognitive Robotic System based on the Soar Cognitive Architeture for Mobile Robot Navigation, Search, and Mapping Missions.,” Ph.D. dissertation, Pennslyvania State University, 2011.

[66] T. R. Johnson, “Control in Act-R and Soar Act-R,” in *Proc.19th Annu. Conf. Cognitive Science Society*, 1997, pp. 343–348.

[67] J. E. Laird and P. S. Rosenbloom, “Integrating Execution , Planning , and Learning in Soar for External Environments,” in *8th Nat. Conf. Artificial Intelligence*, 1990, pp. 1022–1029.

[68] E. S. Yager, E. Laird, M. Tuck, and M. Hucka, “Learning in Tele-autonoumous Systems Using Soar,” in *Proc. NASA Conf. Space Telerobotics*, 1989, pp. 416–424.

[69] S. D. Hanford, O. Janrathitikarn, and L. N. Long, “Control of Mobile Robots Using the Soar Cognitive Architecture,” *J. Aerosp. Comput. Information, Commun.*, vol. 6, no. 2, pp. 69–91, Feb. 2009.

[70] S. D. Hanford, O. Janrathitikarn, L. N. Long, and I. Introduction, “Control of a Six-Legged Mobile Robot Using the Soar Cognitive Architecture,” in *2008 AIAA Aerospace Sciences Meeting*, 2008, pp. 1–14.

[71] J. Kirk and J. Laird, “Interactive Task Learning for Simple Games,” *Adv. Cogn. Syst.*, vol. 3, pp. 11–28, Jul. 2014.

[72] P. Lindes and J. E. Laird, “Toward Integrating Cognitive Linguistics and Cognitive Language Processing,” in *Proc. 14th Int. Conf. Cognitive Modeling*, 2016, pp. 86–92.

[73] J. Kirk, A. Mininger, and J. Laird, “Learning task goals interactively with visual demonstrations,” *Biol. Inspired Cogn. Archit.*, Aug. 2016, In press- corrected proof.

[74] S. Vongbunyong, S. Kara, and M. Pagnucco, “Application of cognitive robotics in disassembly of products,” *CIRP Ann. - Manuf. Technol.*, vol. 62, no. 1, pp. 31–34, Jan. 2013.

[75] S. Vongbunyong, S. Kara, and M. Pagnucco, “Basic behaviour control of the vision-based cognitive robotic disassembly automation,” *Assem. Autom.*, vol. 33, no. 1, pp. 38–56, 2013.

[76] S. Vongbunyong, S. Kara, and M. Pagnucco, “General plans for removing main components in cognitive robotic disassembly automation,” *2015 6th Int. Conf. Automation, Robotics and Applications,* pp. 501–506.

CHAPTER 2

CHARACTERIZATION OF DIFFERENT MICROSOFT KINECT SENSOR MODELS

by

Nicholas M. DiFilippo and Musa K. Jouaneh

Published in *IEEE Sensors Journal*, vol. 15, no. 8 pp. 4554-4564, Aug. 2015.

Corresponding Author: Musa Jouaneh

Department of Mechanical, Industrial, and Systems Engineering

University of Rhode Island

103C Gilbreth Hall, 2 East Alumni Ave.

Kingston, RI, 02881, U.S.A.

Phone : +1-401-874-2349

Email Address : jouaneh@uri.edu

Abstract

This experimental study investigates the performance of three different models of the Microsoft Kinect sensor using the OpenNI driver from Primesense. The accuracy, repeatability, and resolution of the different Kinect models' abilities to determine the distance to a planar target was explored. An ANOVA analysis was performed to determine if the model of the Kinect, the operating temperature, or their interaction were significant factors in the Kinect's ability to determine the distance to the target. Different sized gauge blocks were also used to test how well a Kinect could reconstruct precise objects. Machinist blocks were used to examine how well the Kinect could reconstruct objects setup on an angle and determine the location of the center of a hole. All the Kinect models were able to determine the location of a target with a low standard deviation (< 2 mm). At close distances, the resolutions of all the Kinect models were 1 mm. Through the ANOVA analysis, the best performing Kinect at close distances was the Kinect model 1414, and at farther distances was the Kinect model 1473. The internal temperature of the Kinect sensor had an effect on the distance reported by the sensor. Using different correction factors, the Kinect was able to determine the volume of a gauge block and the angles machinist blocks were setup at, with under a 10 percent error.

Keywords: Kinect sensor, Kinect accuracy, 3-D image reconstruction, Kinect for Xbox, Kinect for Windows, OpenNI, depth

2.1 Introduction

The Microsoft Kinect is a low-cost sensor that integrates many components. It is composed of a traditional RGB camera, a depth sensor consisting of an infra-red (IR) camera and projector, a microphone, and a built-in motor. The built-in motor allows the base of the Kinect sensor to tilt [1]. The Kinect sensor was originally designed to be used to control games on the Microsoft Xbox 360 gaming console. Using their hand with the Kinect sensor rather than a controller, players are able to control characters or menus in a game. Instead of being limited to only gaming applications, multiple libraries are available that open up potential applications of the Kinect sensor to fields involving computer vision [2], 3D mapping [3-5], robotics [6-10], medicine [11-18], human tracking [19, 20], as well as others.

Examples of these applications include the work of Shirwalker *et al*. [9] and Afthoni *et al*. [10] who have both used the Kinect sensor to obtain gestures that an operator makes in order to control a robotic arm. Alnowami *et al*. [12] and Tahavori *et al*. [13, 14] used the Kinect to monitor patients' respiratory breathing patterns while they are receiving external beam radiotherapy. It was found that the Kinect performs quite well and can determine various types of breathing patterns compared to the equipment that is currently used. Ning and Guo [17] showed that the Kinect sensor can be used to assess the spinal loading of a person. Yang *et al*. [18] assess how the Kinect sensor performs in measuring the postural stability of a person. It was reported that although the Kinect sensor needs to be calibrated by a set of linear equations, it was able to measure a person’s standing balance comparable to how standard testing equipment is able to. Obdrzálek *et al*. [21] used the Kinect sensor to estimate human poses and compared this with other known techniques. They found that with controlled postures, such as standing and exercising arms, the Kinect’s performance is comparable with motion capture techniques. However, the Kinect’s estimation of general postures can be off by as much as 10 cm and the Kinect skeletal tracking often fails due to self-occlusions of body parts.

The three main libraries that allow programmers access to the Kinect’s camera and depth information are OpenKinect, the Microsoft Kinect Software Development Kit (SDK), and OpenNI [1]. The first library available for developers, OpenKinect , was released in November 2010 via the hacker community [22]. PrimeSense, creator of the hardware in the Kinect sensor, released the OpenNI open source SDK in December 2010. OpenNI is a framework that uses the middleware library, NiTE, to enhance the Kinect sensors gesture recognition and tracking abilities. In June 2011, Microsoft released the Microsoft Kinect SDK [23] which allows development of applications using C++, C#, and Visual Basic.

The two types of Kinect sensors available are the Kinect for Xbox 360 and the Kinect for Windows (K4W). The main difference between these sensors is the range at which the sensors can return depth values for using the Microsoft Kinect SDK. While the Kinect for Xbox 360 can only determine the distance of an object between 800 and 4000 mm, the K4W possesses a near mode option allowing it to change this range to see objects between 400 and 3000 mm. Additionally, the Kinect for Xbox 360 has two common models, model 1414 and model 1473. The OpenNI drivers and the Microsoft Kinect SDK work with all of the sensors however; the OpenKinect libraries currently only support the Kinect model 1414. At the time of this writing, Microsoft has released a new Kinect for Windows v2. This sensor has a full 1080p video, a higher depth fidelity, and can track up to 6 people. However, it is only available on Windows 8 or 8.1 and can only interface with the Microsoft SDK V2 [24]. This new Kinect sensor works by using time of flight (ToF) technology [25] where a light source transmits a modulated light signal that travels to an object and is then reflected back to the sensor. The Kinect uses phase detection that measures how long it takes the light to travel to the object and back and can then determine the depth of an object from this time.

The Kinect Sensor works by using a speckle pattern 3-D mapping [4, 26]. This is when the sensor projects an IR speckle pattern on the scene and compares this pattern to a reference pattern. If an object in the scene is farther or closer than the reference plane, the speckles that are on the object will shift and using a correlation procedure, the Kinect sensor can determine the distance of that object.

Various researchers have attempted to characterize the Kinect sensor and determine the optimal range to use it. Khoshelham and Elberink [4] explained the mathematical model of how the Kinect sensor determines the distance of an object and how to align the depth and RGB images. They concluded, using the OpenKinect framework and the RANSAC plane fitting method with the standard deviation of the residuals on the point cloud data, that the optimal distance to use the Kinect sensor is within 1000 - 3000 mm. At greater distances, the low resolution and noise of the sensor will reduce the accuracy of the data.

Alnowami *et al.* [12] observed a nonlinear relationship between the true depth and the Kinect pixel depth intensity, and they determined that the optimal performance lies between 800 mm and 1500 mm using the Microsoft SDK. Molnar *et al.* [27] reported a standard deviation of less than 1 mm to up to 10 mm at close ranges and between 7 and 50 mm at 3500 mm using OpenNI drivers. They found the optimal range of the Kinect sensor to be between 700 and 2000 mm. Andersen *et al.* [2] showed that the depth estimates of a pixel compared to the actual distance follow a linear relationship after being linearized by the OpenNI framework. The resolution of the Kinect was shown to be nonlinear and continues to degrade as the distance from an object increases.

Macknojia *et al.* [28] compared the performance of a K4W and a Kinect for the Xbox 360. They used the Microsoft SDK for the K4W sensor and the OpenNI framework for the Xbox Kinect sensor. They found that both devices have close to the same quantization error (the Xbox Kinect slightly overestimates it) and the optimal operating range was up to 2000 mm with an error of 10 mm. They also discovered that both Kinect sensors could not reconstruct transparent surfaces (since glass and clear materials are IR transparent), but could reconstruct curves and shiny painted surfaces on a car. Stommel *et al.* [29] describe a method in which missing depth values are estimated to eliminate gaps in images.

Stoyanov *et al.* [30] compared the Kinect against two ToF cameras (SwissRanger SR-4000 and Fotonic B70) to a laser range sensor (SICK LMS-200) and found that in short distances (< 3m), the Kinect sensor performed the closest to the laser range sensor. Smisek *et al.* [31] also compared the Kinect to two different sensors, a SwissRanger SR-4000 ToF sensor and a 3.5 M pixel SLR stereo camera. They concluded that the Kinect was much better than the ToF camera and comparable to the stereo camera.

Mallick *et al.* [32] review the work that has been done on the characterization of the noise in the Kinect sensors depth image. They conclude that the three main types of noise that occur are spatial, temporal, and interference noise. Spatial noise happens in a single frame, temporal noise happens over multiple frames, and interference noise happens when two or more Kinects are looking at the same scene.

The aim of this research is to compare the performance of different Kinect models using the OpenNI framework. OpenNI was chosen because the alignment of the depth and color images is performed automatically; it works with all the Kinect models, and is cross platform. In some of the tests reported in this paper, we also used the Microsoft Kinect SDK 1.8. The Kinect sensor will also be used to reconstruct 3-D objects of known dimensions.

The remainder of this paper is organized as follows. The next section discusses the methods used to characterize the different sensors. This is followed by Section 3 which shows and discusses the results obtained. Section 4 discusses the 3-D reconstruction of gauge and machine blocks.

2.2 Methods

2.2.1 Accuracy, Repeatability, and Resolution

The experimental setup is shown in Fig 1. A 254 x 254 mm target machined out of plastic and covered with paper to make it non IR-transparent was attached to a stage mounted on a 2 m DryLin linear slide (Igus). A NEMA 23 stepper motor **(**1.8 deg/step, Minebea Co.**)** was attached to the linear slide. The stage was controlled by a data acquisition card (PCIM- DAS1602/16) which moved the target back in 100 mm increments. The Kinect sensor was mounted on one end of the linear stage using a Kinect wall mount. This allowed for the model of the Kinect sensor to be interchanged while ensuring positional accuracy. The stepper motor and linear stage were calibrated by a displacement gauge (Mitutoyo) and a linear distance of 100 mm yielded an accuracy of +/-0.01 mm. When a test was started the stage would move towards the Kinect sensor to a homing position located 100 mm in front of the Kinect. The homing position was reached when the limit switch was triggered. Then the linear stage would move back in 100 mm intervals and the Kinect sensor recorded the depth of the target at every interval. The accuracy of each sensor was measured by constructing an area of 50x50 pixels in the center of the target. Twenty five images at 30 FPS were taken and the depth value of every pixel in the 50x50 pixel area was averaged to obtain a final value. For each interval, a total of 62500 pixels were used (25x50x50). The distance returned by the Kinect is the distance from the object to the plane of the Kinect's camera. An area of 50 x 50 pixels was chosen so the region of interest that the Kinect was looking at would be at the center of the target at the closest and farthest positions of travel. The repeatability of each sensor was measured by the standard deviation of these final values. A test was completed once the limit switch opposite the Kinect sensor was triggered. Distances greater than 1800 mm were not considered because of the physical limitations of the slider used. Also, distances less than 600 mm were not explored because 500mm is very close to the limit of the Kinect in determining distances. The resolution step is the smallest increment that the Kinect can see at a distance. To determine the resolution of a Kinect sensor, pixels in the upper left, center, and lower right portion of the target were chosen and their values were recorded over 5000 images. The initial pixel value was subtracted from the subsequent pixel values and histograms of the differences were used to determine the resolution step. After a Kinect was placed on the holder, the distance from each end of the Kinect was measured to the target to ensure the Kinect was parallel to the target.

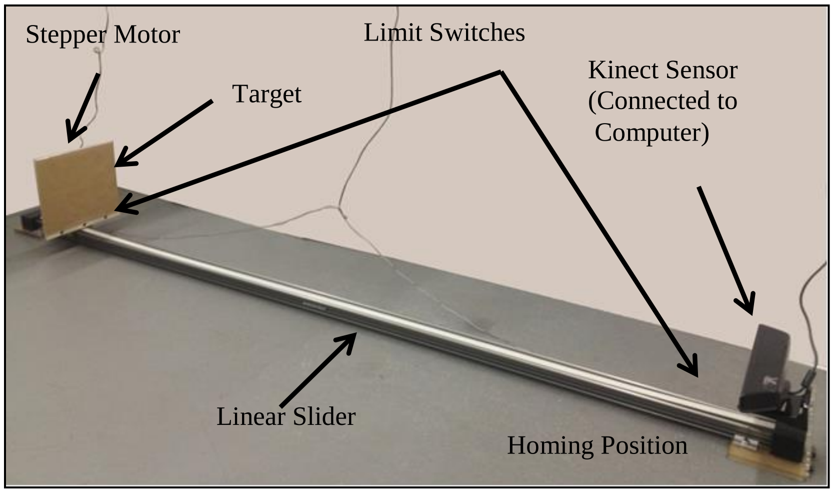


Fig. 1. Experimental setup of the linear slide used for the accuracy, repeatability, resolution, and ANOVA trials.

2.2.2 Steady State Temperature

To determine the amount of time it took a Kinect sensor to reach an internal steady state temperature, a temperature sensor (LM35) was connected to an Arduino Uno and interfaced to Matlab 2012a. The LM35 temperature sensor is a precision integrated-circuit that outputs an analog voltage proportional to the temperature in centigrade. This sensor does not require any external calibration and is accurate to ±1/4°C. The temperature sensor was attached directly above the infrared camera projector and LED on top of the plastic casing. This location was chosen because it was the location on the Kinect that heats up when the sensor is in use. The Kinect sensor was turned on and the OpenNI example program SimpleViewer was run for the duration of the test. The results of the trials were averaged, and the time constant (τ) was determined to be 1500 seconds. For the system to reach temperature stability, the Kinect sensor has to run for 4τ or ~6000 seconds (100 minutes). Fig. 2 shows the temperature response of all three Kinect models. This operating temperature will be referred to as the high temperature in the results, while the low temperature will be referred to as the room temperature the Kinect is at before being powered on which for this experiment is ~21 - 23°C. The Kinect sensor was set up to look at a stationary target 1800 mm away and images were recorded at 30 FPS over a period of 1200 minutes. A 50 x 50 pixel area was averaged over 25 images to obtain a depth distance. This test was repeated using the OpenNI driver and Microsoft Kinect SDK 1.8. The results of Fig. 2 will be discussed in Section 2.3.

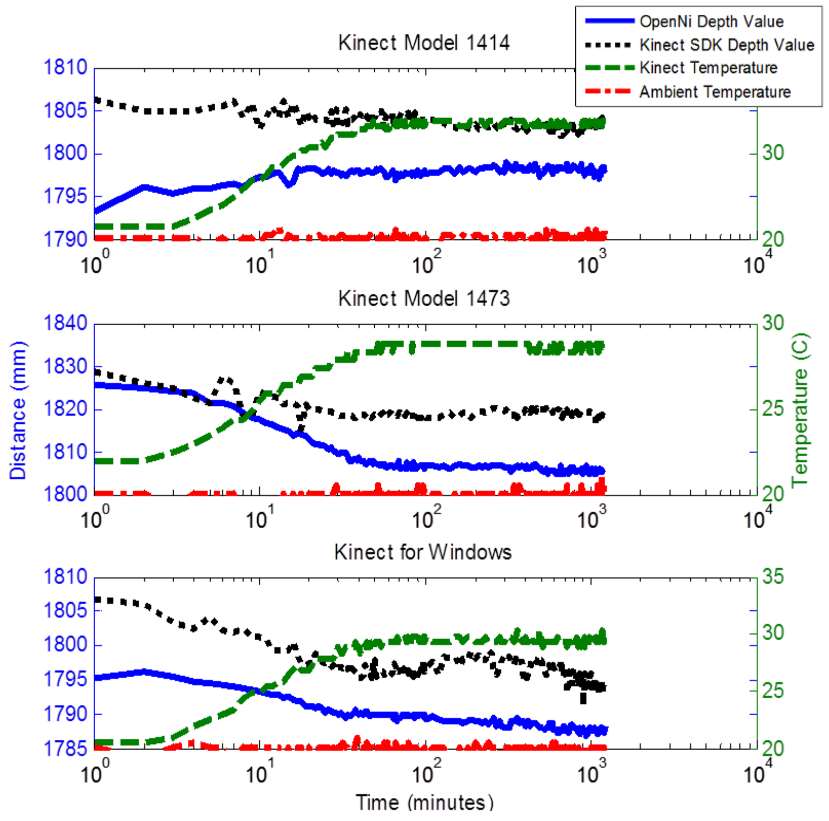


Fig. 2. Temperature response of the three Kinect models.

2.2.3 Statistical Design

The design used for the experiment was a full factorial design whose effects model is shown in (1). Here, μ is the overall mean effect, τ_i is the effect of the *i*th level of the Kinect model and β_j is the effect of the *j*th level of the operating temperature. $(\tau\beta)_{ij}$ is the interaction effect between the Kinect model and the operating temperature, and (ε_{ijk}) is the random error term. y_{ijk} is the distance returned by the Kinect sensor when the Kinect model is at the *i*th level, the operating temperature is at the *j*th level and it is the *k*th replicate. Using Minitab's DOE tool, a random run order for the experiment was generated and a total of 18 tests were performed.

$$y_{ijk} = \mu + \tau_i + \beta_j + (\tau\beta)_{ij} + \varepsilon_{ijk} \tag{1}$$

An ANOVA test (*p*= 0.05) was performed in Minitab on the response (depth data) of the Kinect sensor at 600 to 1800 mm in 100 mm increments. This was used to examine if the Kinect model, the operating temperature, and/or and the interaction between the Kinect model and operating temperature were significant factors that influence the depth data. The ANOVA model was verified using Bartlett's test for equal variance (*p*=0.05) and checking the normality of residuals (*p*=0.05). A comparison of the means was performed using Tukey's test with a 95% confidence interval and the final recommendations were made by looking at the interactions plot.

2.2.4 3-D Image Reconstruction

Reconstructing objects and locating holes is an important step in automated vision guided assembly and disassembly operations. In a disassembly operation, the vision system needs to be able to identify holes in order to try to remove a screw. It also requires precise reconstruction of objects and holes that might not always be flat or completely intact.

The Kinect model used for image reconstruction was the 1414 model which was mounted on a camera tripod on the top of a leveled surface 600 mm away from the object as shown in Fig. 3. This Kinect was chosen because it performed the best at close distances. Closer distances are better for applications involving precise reconstruction because the resolution step of the Kinect is lower (1mm). These results are talked about more in Section III. Thus the purpose of the work in this section is to investigate how well the Kinect could reconstruct and identify features on these objects. The intent is not to perform comparison between different Kinect models.

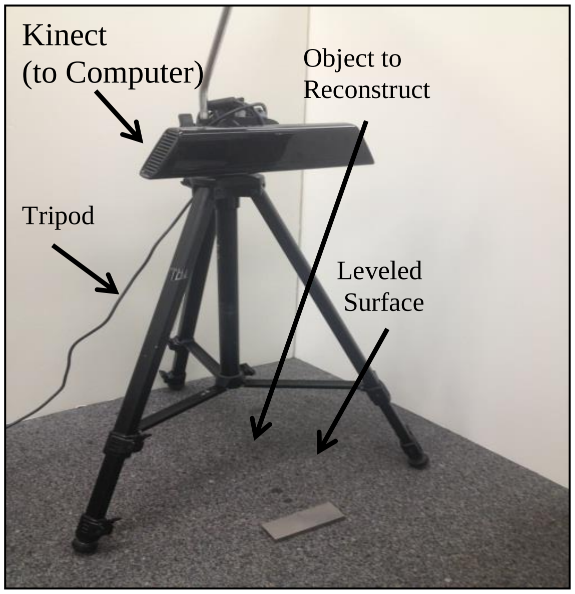


Fig. 3. Experimental Setup for 3D image reconstruction of gauge and machinist blocks.

The Kinect sensor was then leveled using its depth image, making sure all sides of the image returned the same depth value. Using the 150 x 90 pixel RGB image in Fig. 4 the number of pixels a 101.6 mm gauge block took up could be determined by subtracting the pixels of both sides of the gauge block on the same row (X_2 - X_1 where Y_1 = Y_2.) A millimeter to pixel (mm/px) resolution of 1.116 was calculated for this setup. Using the built in Matlab function "surf" and various sized gauge blocks, the Kinects sensor ability to accurately reconstruct 3D objects was tested. The experimental volumes of the gauge blocks were determined by separating the background from the gauge block using a height threshold.

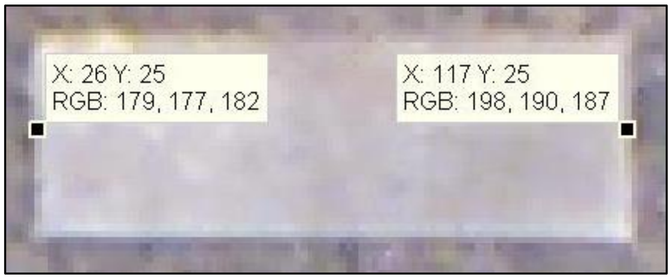


Fig. 4. Gauge block used for determining pixel to mm ratio. The X and Y values are used to determine the length of the gauge block in pixels.

Gauge blocks were chosen to be the objects reconstructed by the Kinect sensor because they are manufactured to known dimensions and tolerances. This makes them ideal to test the performance of the Kinect sensor. Machinist blocks with a dimension of 25.4 x 50.8x 76.2 mm and containing six 12.7 mm through holes were used to test the Kinect sensors ability to reconstruct an angle an object was at and an object containing holes. The machinist blocks were angled using a height support and the angle was verified with a protractor. The angle of the machinist block was experimentally determined by using the depth image to obtain the y_1, y_2, $z_{1,}$ and z_2 points shown in Fig.5. The angle of the machinist block could then be calculated using trigonometric relationships.

The Hough circle transform method (Tao Peng, Matlab Central File Exchange) was performed on the RGB image to obtain the location of the center of the holes for each of the six holes on the machinist block. The labels of these holes are given in Fig. 6a. Fig. 6b shows the calculation of the offset (r) between the between the center of the hole (A) and the center of the circle found with the circle transform algorithm (*A'*). This offset is given by (2). The offset was calculated with and without adjusting for the Kinect sensors position being directly above to the machinist block and a rotation in the X-Y plane. The theoretical locations of the center of the holes are determined by using the mm/px resolution, the dimensions of the machinist block, and the row of pixels in the RGB image that the top of the block is at. These center hole locations work well for a flat object but as soon as the object is at an angle, the adjacent values A_1, A_2, and A_3 in Fig. 6c become a more accurate estimation of the location of the centers. Using similar triangle relations, the equations for the A_i values are given by (3) while the H_i values are found from the dimensions of the machinist block. The machinist block used in this setup has a hypotenuse of 7.62 mm.

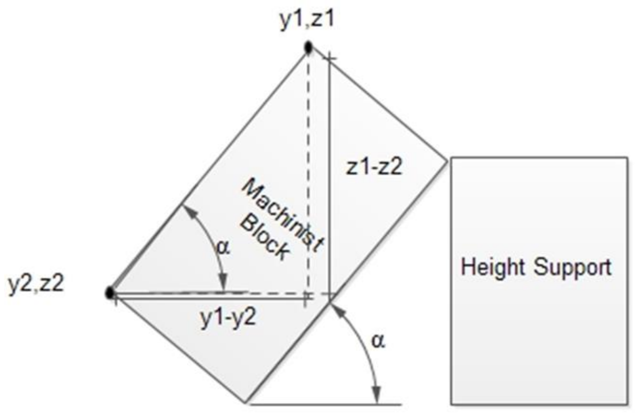


Fig. 5. Schematic showing how the angle of the machinist block is determined.

The other adjustment, shown in Fig. 6d is performed to offset any rotation that the machinist block may have undergone in the x-y plane of the image. Since the locations of the center of the holes are determined by their distance from the side of the block (in px), O_1, O_2, and O_3 allow for a more accurate starting pixel point. These points are also determined using similar triangles relations shown in (4). The values for H_i are found by using the dimensions of the machinist block. Fig. 6d only shows this adjustment on the side of the machinist block however, the same procedure will work for the rotation of the top part of the machinist block.

Using the depth image obtained from the Kinect sensor, at the location of the center holes, the number of pixels in a 12x12 square that have a depth value of zero are added up. The size of the square was determined by calculating the pixel equivalent of the area of a 12.7 mm hole. This can be used to tell if there are patches of 0 depth pixels. These patches of pixels with 0 depth values can be used to validate a hole at these locations.

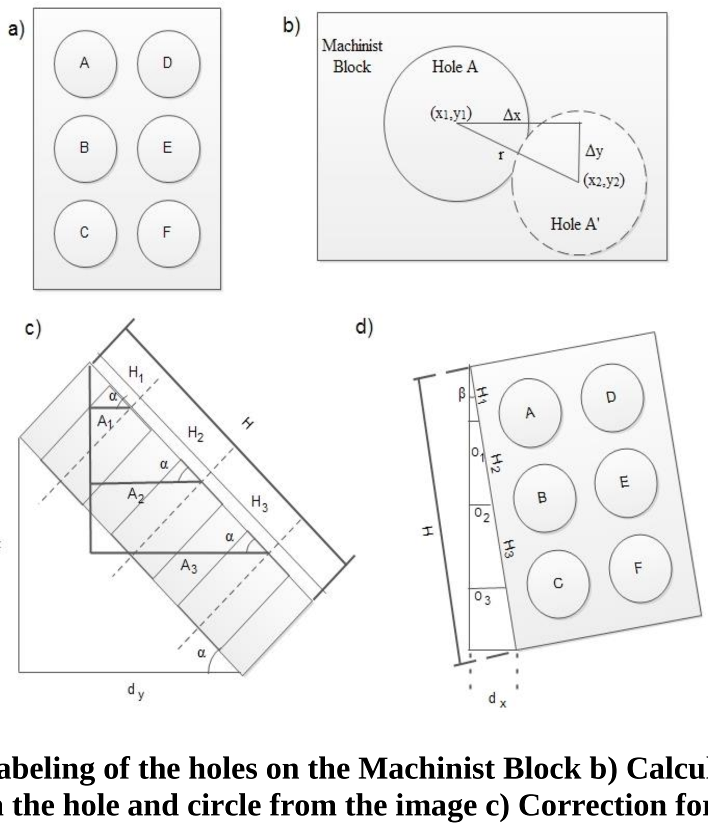


Fig. 6. a) Labeling of the holes on the Machinist Block b) Calculation of the offset between the hole and circle from the image c) Correction for the way the Kinect is positioned over the machinist block d) Correction for rotation of machinist block.

$$\Delta x = x_2 - x_1 \qquad (2)$$
$$\Delta y = y_2 - y_1$$
$$r = \sqrt{(\Delta x^2 + \ \Delta y^2)}$$

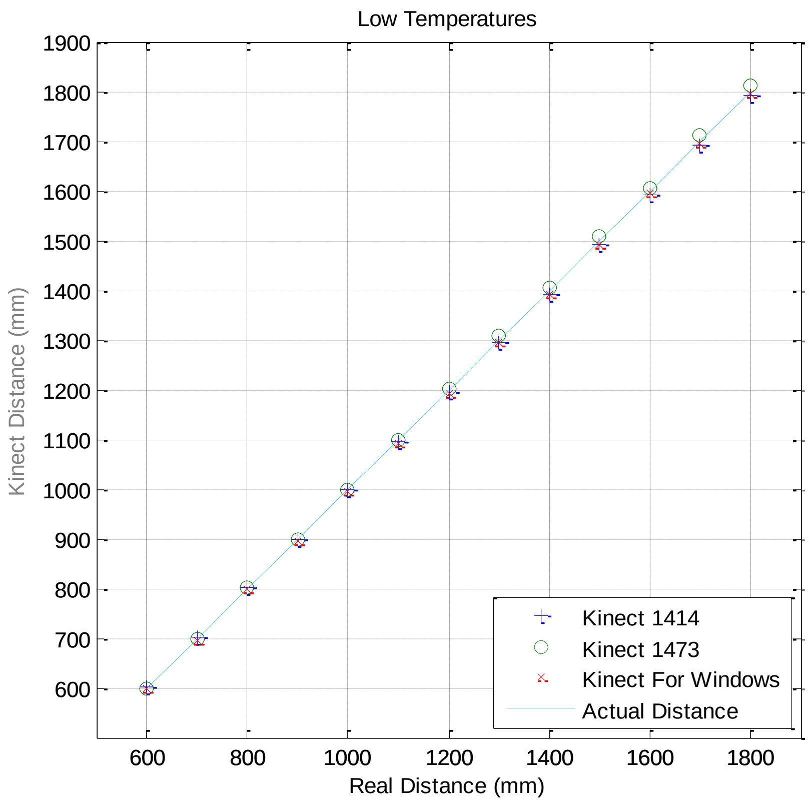
$$A_i = H_i \cos(\alpha) \quad where\ \alpha = \cos^{-1}\left(\frac{d_y}{H}\right) \qquad (3)$$

$$O_i = H_i\ \sin(\beta) \ \ where\ \beta = \sin^{-1}\left(\frac{d_x}{H}\right) \qquad (4)$$

2.3 Results

2.3.1 Accuracy and Repeatability

Each model of the Kinect sensor was used for three trials at a high and a low temperature as part of the ANOVA test (Section 2.3.3). The averaged results of all three Kinect models are shown in Fig. 7. This figure plots the actual distance the target is from the Kinect sensor vs. the distance calculated by the Kinect sensor at different temperatures. The standard deviations of all the points from these trials with all the Kinect models are very small with the greatest standard deviation being just over 2mm (Kinect 1414 at 1700mm). Due to the small size of the standard deviation and the scale of the graph these error bars are not shown. At low temperatures (~21 ºC - temperature when Kinect was just turned on) and close distances less than 800 mm, all the calculated results are close to the actual distances (within 2.1mm), but at greater distances than 1300 mm, the Kinect model 1473 starts to calculate distances farther away than the actual distance. At a high temperature (temperature after turning on the Kinect for more than 100 minutes), the Kinect 1473 improves the distances it reads as the target gets farther away while the other two Kinect models get worse and underestimate the depth.



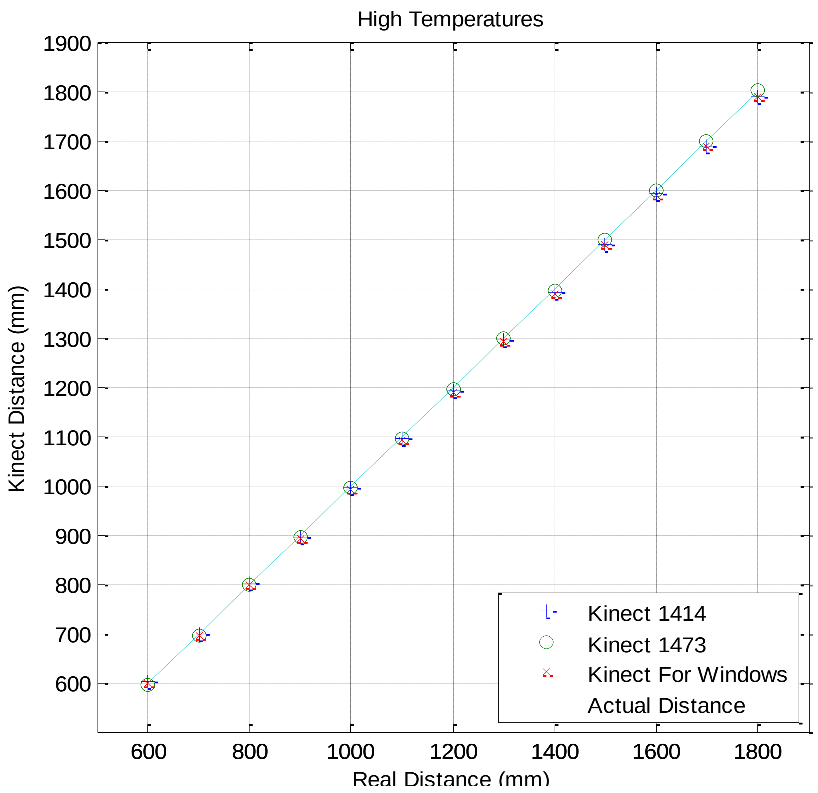


Fig. 7. Accuracy graph of the Kinect sensor at high and low temperatures (Actual distance of object vs. distance returned by Kinect sensor)

2.3.2 Depth Resolution

The depth resolution histogram of each Kinect sensor was obtained at different distances. A histogram of the resolution at 600mm with the Kinect model 1414 is shown in Fig. 8. This figure shows a resolution of 1 pixel. These histograms are summarized in Fig. 9. As the distance from a target increases, the resolution of the Kinect sensor for all models becomes coarser. The most accurate resolutions occur at the closer distances (distances < 700 mm) when the resolution of all the models are 1 mm. At 1800 mm, the resolutions of the models are still less than 10 mm. All of the different Kinect models exhibit the same upward sloping graph and the same resolution. The results presented agree with the results presented by Andersen *et al.* [2] and Macknojia *et al.* [28] at these distances. Andersen *et al.* [28] reported approximately a 1 mm resolution at a distance of 600mm to around a 10 mm resolution at a distance 1800mm. Macknojia *et al.* [28] also reported a 1 mm resolution at a distance of 600 mm to 9 mm at a distance of 1800 mm. Both of these authors also demonstrated that at distances of around 3000mm, the resolution can be as poor as 30 mm. This was not explored with this setup due to the size of the linear slider.

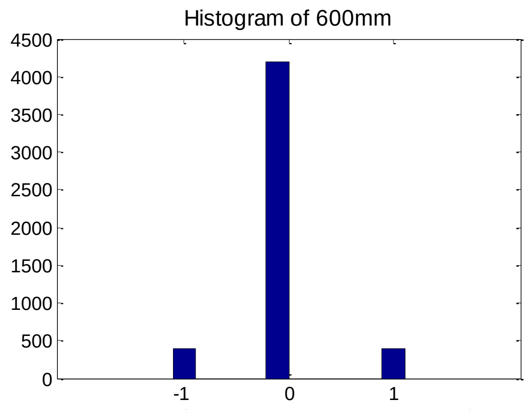


Fig. 8. Histogram of the resolution data from 600mm.

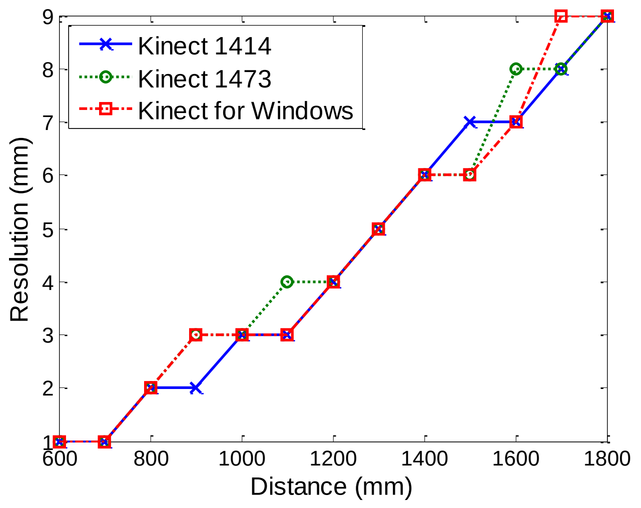


Fig. 9. Resolution of different Kinect sensors.

2.3.3 ANOVA Testing

For all of the distances, the *Kinect Model* and the *Temperature* were significant factors with *p*-values of 0.000. The *p*-value of the interaction between these two factors was a significant (*p* value <0.05) at all distances except for 600mm where the *p*-value was 0.089. However, since this value is close to 0.05 and can be interpreted as moderately significant, it was determined to treat the interaction of this distance as significant.

The results for the comparison of the means test for the Kinect models are shown in Table 1. Since there was interaction between the *Kinect Model* and *Operating Temperature* was taken to be a significant factor at all the distances, the comparison of the means had to be performed for both the low and high temperatures. In Table 1, the grouping category shows how the response of the different Kinect models compared with each other. If the response of the Kinect models were not statistically different, then the letter under grouping would be the same. If the responses of the Kinect models were statistically different, then the letters would be different. For example, at 600 mm under the *Temperature Low* column, the response of the Kinect model 1473 and K4W are not statistically different from each other, thus both have a grouping of “A”. The Kinect model 1414 is statistically different from both of the other models so it has a grouping of “B”.

Table 1 shows that in 11 of the 26 trials all of the Kinect sensors have a response that is significantly different from each other. It also shows that in 7 of the 26 trials the response of the Kinect 1414 and K4W are not significantly different while the response between them and the Kinect 1473 is. The Kinect 1414 and K4W are more likely to produce results that are not statistically different at distances when the target was farther away (<1200 mm) at a low temperature and (<1700 mm) at high temperatures.

Table 2 provides a summary of the Interaction plots as well as showing the most consistent performing Kinect model. In the second row labeled *Kinect Model (ANOVA)*, the table shows the model that performed the “best” when the target was at different distances from the Kinect. The row labeled *Temp (ANOVA)* shows at what temperature the Kinect performed the “best” at. The row *Best Model (ANOVA)* is used to show that the Kinect 1414 performs the best at close distances (<900mm). It also shows the Kinect 1473 performs best at farther distances and performs the most consistently over the most distances. Both the Kinect 1414 and 1473 have distances where they performed best at low temperature and distances where they performed best at high temperatures. There are two models listed under 900 mm because according to Table 2, there is no significant difference between these two models at a low temperature.

Table 1. Tukey Results and Comparison of the Mean of Different Kinect Models

Temperature Low							
Distance (mm)	600	700	800	900	1000	1100	1200
Model							
1473	A	A	A	A	A	A	A
1414	B	B	A	A	B	B	B
K4W	A	C	B	B	C	C	B
Distance (mm)	1300	1400	1500	1600	1700	1800	
Kinect							
1473	A	A	A	A	A	A	
1414	B	B	B	B	B	B	
K4W	B	B	B	C	B	C	
Temperature High							
Distance (mm)	600	700	800	900	1000	1100	1200
Model							
1473	A	A	A	A	A	A	A
1414	B	B	B	B	A	A	B
K4W	A	A	A	C	B	B	C
Distance (mm)	1300	1400	1500	1600	1700	1800	
Model							
1473	A	A	A	A	A	A	
1414	B	B	B	B	B	B	
K4W	C	C	C	C	B	B	

Fig. 2, shows that as the temperature of the Kinect increases, the depth response continues to change. It also shows that the response stabilizes once the Kinect reaches a high steady state temperature. For all three Kinect models, the Microsoft Kinect SDK returned a depth distance that was farther away than the depth distance returned by OpenNI. The Kinect model 1414 had its depth response increase when using the OpenNI drivers and decrease while using the Microsoft Kinect SDK.

For the Kinect models 1473 and K4W, using either driver the depth response decreased and then reached a steady state as the Kinect's temperature increased and reached steady state. Using the OpenNI drivers, the depth response of the Kinect 1473 changes about 20mm while the depth response of the Kinect 1414 differs about 5mm as the Kinect goes from a low to a high temperature. The K4Ws depth response changes about 6 mm from a low temperature to a high temperature but after about 350 minutes the depth response begins to drop again. Using the Microsoft Kinect SDK the magnitude of the depth response change is about 10mm Kinect 1473 and is about 3mm for the Kinect 1414. The K4W depth response changes about 10 mm and begins to drop again after about 250 minutes.

Table 2. Summary of Interactions Plots

Distance (mm)	600	700	800	900	1000	1100	1200
Kinect Model (ANOVA)	1414	1414	1414	1473	1473	1473	1473
Temp (ANOVA)	High	Low	High	Low	Low	Low	Low
Best Model (ANOVA)	1414	1414	1414	1414/ 1473	1473	1473	1473
Best Model High Temp	1414	1414	1414	1414	1473/ 1414	1473/ 1414	1473
Distance (mm)	1300	1400	1500	1600	1700	1800	
Kinect Model (ANOVA)	1473	1473	1473	1473	1473	1473	
Temp (ANOVA)	High	Low	High	High	High	High	
Best Model (ANOVA)	1473	1473	1473	1473	1473	1473	
Best Model High Temp	1473	1473	1473	1473	1473	1473	

Since the response of the Kinect is more stable at a high steady state temperature, the row *Best Model at High Temp* in Table 2 shows which Kinect Model performs the best at a high temperature. Again, the Kinect model 1414 performs the best at close distances while the Kinect 1473 performs the best at farther distances. All of the results for the distances shown in Table 1 and Table 2 satisfied the normality and equal variance assumption with a *p*-value above 0.05.

2.4 3-D Reconstruction

2.4.1 Gauge Blocks

Fig. 10a shows a single frame of the gauge block as captured by the Kinect sensor which was mounted 600mm away. The figure shows that there is a distortion in the image. The left and right spikes shown in the image are pixels whose depth values are zero. Applying a correction factor improved the image (See Fig. 10b). Looking at the pixel depth values on the y-axis in Fig 10a, a *Height Difference* of around 8 mm is determined. On the left side of the image, the depth value is around 391 mm and on the right side around 399 mm (It should be noted this value has been subtracted from 1000 in order to invert the image for viewing. The distance from the Kinect to the table was 600 mm). A linear function was developed to be used as a correction factor to flatten out the image that allows for a better estimate of the volume of the gauge block. This method works the best when it is possible to distinguish the background from the object(s) of interest. The correction formula is given by (5). In this equation, *i* represents the row of the image, and *j* represents the column of the image. The correction factor (*CF*) from (5) given by (6). The negative sign (-) in (6) shifts the depth values up to the correct height. In (6), N is the last column. In Fig. 10, 150 columns of the image were adjusted therefore, N would be 150.

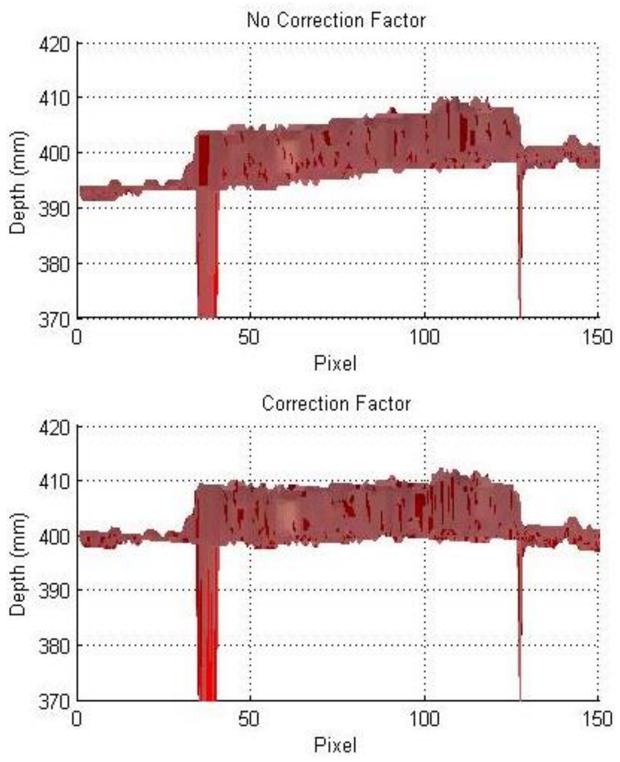


Fig. 10. Image of gauge block without correction factor. b. Image of gauge block with correction factor applied. This image was taken with the Kinect set up 600mm away from the gauge block.

$$Image(i,j) = Image(i,j) + CF \tag{5}$$

$$CF = \left(\frac{Height\ Difference}{N} * j - Height\ Difference\right) \tag{6}$$

Fig 11. shows the effect different correction factors and cutoffs have on the percent error in estimating the volume of different sized gauge blocks. Two parameters are shown in the legend in the figure. The first value is the *Height Difference* (in pixels) from (5) and the second value is the *Cutoff Height* (in mm) used to determine where the background ends and the object begins for the setup used in this experiment. Trying to determine the volume of an object without a correction factor led to an underestimation of the objects volume. This is due to the fact part of the object would be beneath the cutoff height and the program would perceive it as background. By introducing a correction factor, the ability to accurately reconstruct an objects volume greatly increases. The most consistent combination of *Height Difference* and *Cutoff Height* is 6 and 598 mm for this geometry. By using these values, all the percentage errors are lower than 10 percent. The percent error also decreased as the length of the gauge blocks increased. When the 101.6 mm gauge block was analyzed, the percent error of the volume was under 10 % regardless of the *Cutoff Height* and *Height Difference* values. Fig. 11 shows there is a genuine need to calibrate the Kinect's output in order to get meaningful data while trying to reconstruct a 3-D object. These correction factors show that the output from the Kinect sensor needs to be calibrated for fine 3D reconstruction and cannot just be used as is.

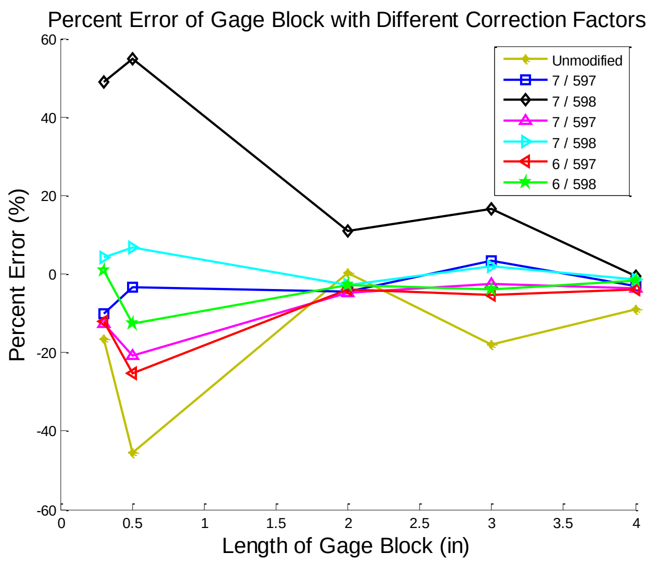


Fig. 11. Percent error of volume vs. length of gauge block in gauge block reconstruction.

2.4.2 Machinist Blocks

2.4.2.1 Angle

The angle the machine blocks were setup at versus the angle calculated by the

Kinect sensor are shown in Fig. 12. The angles are plotted with and without using a correction factor. All of the angles calculated just by using the depth image, are less than the actual angle that the machinist block was setup at. An average correction factor for the angle in this setup was determined to be 1.3708 (R^2 value of .9957). This correction factor was determined by dividing the actual angle by the depth angle. Multiplying the correction factor by the previous angle greatly reduces the percent error between the actual angle and the angle calculated. In Fig. 12, all of the adjusted angles are more accurate than the unadjusted angles. These results agree with the results presented by Tahavori *et al.* [14] in which they showed the Kinect underestimated the angles.

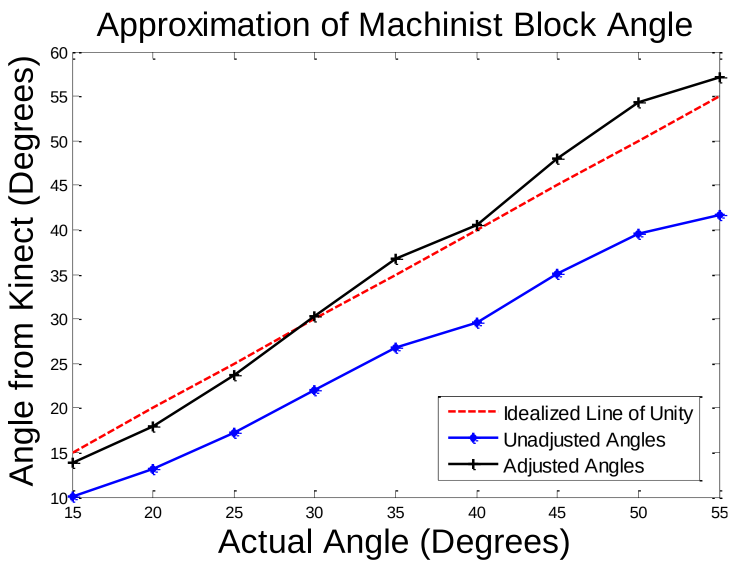


Fig. 12. The unadjusted and adjusted machinist block angles determined by the Kinect sensor

2.4.2.2 Holes

The offset of the center of the circles found with the circle detection algorithm from the centers of the hole in the machinist blocks are shown in Fig 13. These results show that correcting for the position of the Kinect sensor using (3) does not improve the accuracy of this offset value. However, correcting for a rotation of the machinist block using (4) does improve the offset. The best combination (lowest offset) is found correcting for the rotation of the machinist block but not correcting for the position of the Kinect sensor. With these parameters, the offset between the centers of the circles for all of the holes are less than 10 mm. As the angle of the machinist block increases, the offset between the two circles also increases. This method worked for the machinist blocks when the angle of tilt was less than 35 degrees. When the machinist block was positioned at an angle greater than 35 degrees, the circle detection algorithm was unable to determine the circles for the holes as they become elliptical.

The number of pixels returning a depth value of 0 at the center of the circles are shown in Fig 14. This figure shows that clusters of pixels with a depth of zero are around all of the holes of the machinist blocks (labeled A-F). At lower angles, there is a higher pixel count around the holes than at higher angles which signifies that the Kinect is able to determine there is a hole in that location. However, this method only worked when the angle of the machinist block was less than 40 degrees. When the machinist blocks were placed at angles greater than 40 degrees, the depth images from the Kinect sensor were not reliable, as parts of the depth image of the machinist block were missing and thus had pixels with depths values of 0.

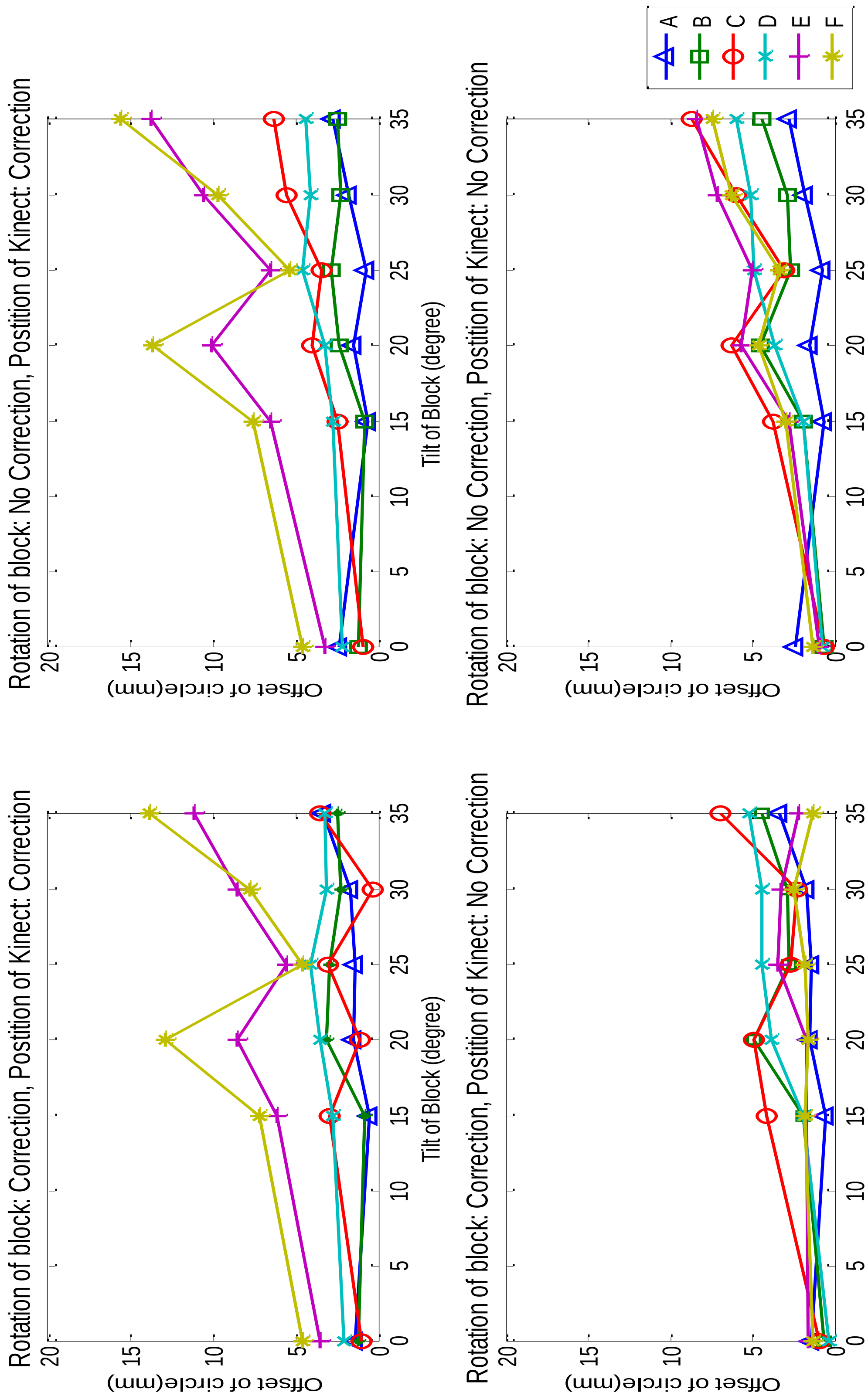


Fig. 13. Percent error of volume vs. length of gauge block in gauge block reconstruction.

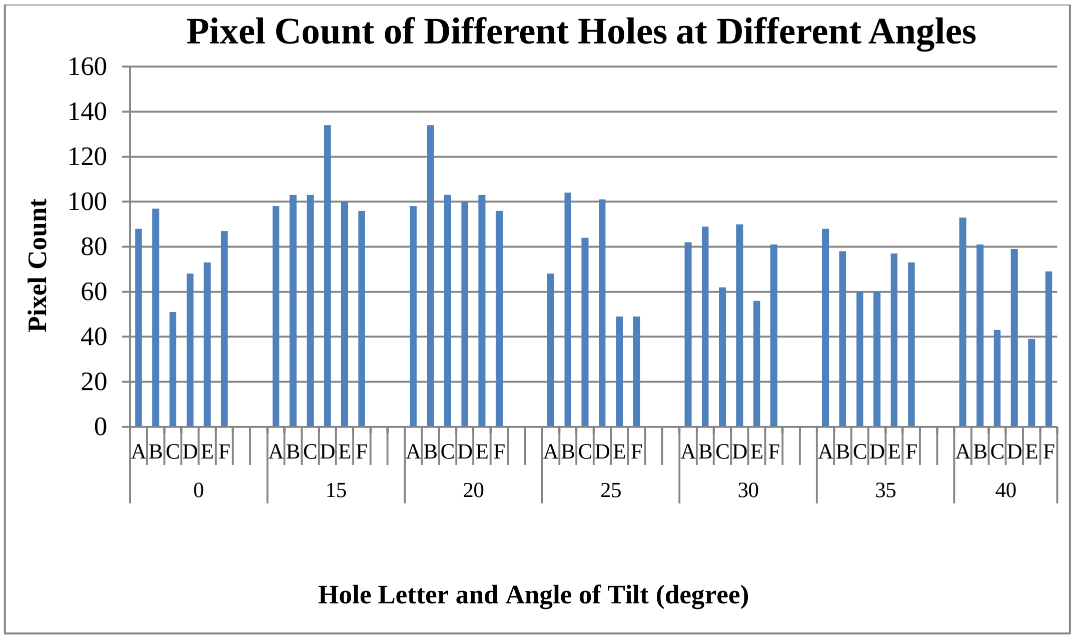


Fig. 14. Zero depth pixel clusters around the location of holes in the machinist blocks. The letters A-F in the legend, correspond to hole location.

2.5 Conclusions

Overall, the Microsoft Kinect sensor is an inexpensive sensor that is capable of producing acceptable results when determining the distance of an object from the sensor. All of the Kinect models give very accurate results at close distances (600 mm to 800 mm). At a low operating temperature, the Kinect 1473 overestimates the depth to an object as the distance increases. At a high operating temperature, the Kinect 1473 performs better as the target moves farther away (towards 1800mm) while the other two Kinect models perform worse and underestimate the distance. The standard deviation of all the Kinect sensors is low (< 2.1mm) at all the distances tested which shows the Kinect sensor has good repeatability.

The resolution of all the Kinect sensors are best at close ranges (<700mm) as the resolution at these distances is 1 mm. At a distance of 1800 mm from the sensor, the resolution is still less than 10 mm. However, this may make it unattractive for applications requiring precise measurements.

From the ANOVA study, the Kinect model and the operating temperature the Kinect sensor is at, and their interactions are both significant factors in the depth response. The interaction plots show that the Kinect model 1414 is the most consistent model at close distances (<1100mm) while the Kinect 1473 produces the best results at farther distances (>1000mm). The overlap in distances of 1000 and 1100mm between these two Kinects are because the depth response between the two models is not statistically significant.

It is also shown that the Kinect depth response changes with the temperature of the Kinect sensor and the software driver used. This response becomes more stable when the Kinect reaches a high steady state temperature. For all the Kinect models, the Microsoft SDK returned a depth distance at high steady state temperature that was further away than the depth returned by OpenNI. Therefore it may be beneficial to have the Kinect warm up before using it as this depth response change can be rather large.

The Kinect sensor is also able to accurately reconstruct a 3D object and determine the objects volume. However, it was important to use a correction factor and correctly pick a cutoff value to separate the background from the object. It is not a robust solution as a change of just one pixel in the cutoff value can drastically alter the results and is geometry dependent. It does however show that there is a need for calibration of the Kinect’s output for fine measurements.

Using the Kinect depth image, the Kinect tends to underestimate the actual angle of an object. It is also possible to account for the angle that an object was setup at after applying another correction factor. The Kinect sensor was able to estimate the offset from the center of a circle found with a circle detection algorithm to the center of a hole in the machinist block. The best results were obtained by adjusting for a rotation of the machinist block while not adjusting for the position of the Kinect sensor above the machinist block.

Additionally, clusters of pixels containing a depth value of zero can also be used to indicate a hole at certain angles provided the angles are not too steep (< 40 degrees). This would be important in automated computer vision applications. Both the RGB and depth images should be used in conjunction with each other in order to confirm the results and determine if a hole is actually present.

2.6 References

[1] J. Han, L. Shao, D. Xu, and J.Shotton, "Enhanced Computer Vision with Microsoft Kinect Sensor : A Review," *IEEE Trans. Cybern*, vol. 43, no. 5, pp. 1318–1334, Oct 2013.

[2] M. R. Andersen, T. Jensen, P.Lisouski, A.K. Mortensen, M. K. Hansen, T.Gregersen, and P. Ahdrendt., "Kinect Depth Sensor Evaluation for Computer Vision Applications", Dept. Eng. Electr. Comput. Eng., Aarhus Univ., Aarhus, Denmark. Tech. Rep. ECE-TR-6, Feb. 2012.

[3] P. Henry, M. Krainin, E. Herbst, X .En,, and D. Fox "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments," *Int. J. Rob. Res.*, vol. 31, no. 5, pp. 647–663, Feb. 2012.

[4] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of Kinect depth data for indoor mapping applications.," *Sensors*, vol. 12, no. 2, pp. 1437–1454, Jan. 2012.

[5] N. Rafibakhsh, J. Gong, M. K. Siddiqui, C. Gordon, and H. F. Lee "Analysis of XBOX Kinect Sensor Data for Use on Construction Sites : Depth Accuracy and Sensor Interference Assessment," in *Construction Research Congr.,* 2012, pp. 848–857.

[6] M. Tölgyessy and P. Hubinský, “The Kinect Sensor in Robotics Education,” in *Proc. 2nd Int. Conf. Robotics Education*, 2011, pp. 143–146.

[7] J. Stowers, M. Hayes, and A. Bainbridge-Smith “Altitude control of a quadrotor helicopter using depth map from Microsoft Kinect sensor,” in *2011 IEEE Int. Conf. Mechatronics*, pp. 358–362.

[8] R. A. El-iaithy, J. Huang, and M. Yeh, “Study on the Use of Microsoft Kinect for Robotics Applications,” in *Position Location and Navigation Symp.*, 2012, pp. 1280–1288.

[9] S. Shirwalkar, A. Singh, K. Sharma, and N. Singh, “Telemanipulation of an industrial robotic arm using gesture recognition with Kinect,” *2013 Int. Conf. Control Automation, Robotics and Embedded Syst.*, pp. 1–6.

[10] R. Afthoni, A. Rizal, and E. Susanto, “Proportional Derivative Control Based Robot Arm System Using Microsoft Kinect,” in *2013 Int. Conf. Robotics, Biomimetics, Intelligent Computational Syst.*, pp. 25–27.

[11] L. Gallo, A. P. Placitelli, and M. Ciampi, “Controller-free exploration of medical image data : experiencing the Kinect,” in *Int. Symp. Computer- Based Medical Syst.*, 2011, pp. 1–6.

[12] M. Alnowami, B. Alnwaimi, F. Tahavori, M. Copland, and K. Wells, “A quantitative assessment of using the Kinect for Xbox360 for respiratory surface motion tracking,” in *Proc. SPIE Medical Imaging 2012: Image-Guided Procedures, Robotic Interventions, and Modeling*, vol. 8316. pp T1-T8.

[13] F. Tahavori, M. Alnowami, and K. Wells, “Marker-less respiratory motion modeling using the Microsoft Kinect for Windows,” in *Proc. SPIE, Medical Imaging 2014: Image-Guided Procedures, Robotic Interventions, and Modeling*, vol. 9036, pp K1-K10.

[14] F. Tahavori, M. Alnowami, J. Jones, P. Elangovan, E. Donovan, and K. Wells, “Assessment of Microsoft Kinect Technology (Kinect for Xbox and Kinect for Windows) for Patient Monitoring during External Beam Radiotherapy,” in *Nuclear Science Symp. and Medical Imaging Conf.*, 2013, pp. 1-5.

[15] R. A. Clark, Y. H. Pua, K. Fortin, C. Ritchie, K. E. Webster, L. Denehy, A. L. Bryant, “Validity of the Microsoft Kinect for assessment of postural control.,” *Gait Posture*, vol. 36, no. 3, pp. 372–377, Jul. 2012.

[16] B. Lange, C.-Y. Chang, E. Suma, B. Newman, A. S. Rizzo, and M. Bolas, “Development and evaluation of low cost game-based balance rehabilitation tool using the Microsoft Kinect sensor,” in *Proc 2011 Annu. Int. Conf. IEEE Engineering Medicine and Biology Soc.*, pp. 1831–1834.

[17] X. Ning and G. Guo, "Assessing Spinal Loading Using the Kinect Depth Sensor: A Feasibility Study," *IEEE Sens. J.*, vol. 13, no. 4, pp. 1139–1140, Apr. 2013.

[18] Y. Yang, F. Pu, Y. Li, S. Li, Y. Fan, and D. Li, "Reliability and Validity of Kinect RGB-D Sensor for Assessing Standing Balance," *IEEE Sens. J.*, vol. 14, no. 5, pp. 1633–1638, May 2014.

[19] D. S. Alexiadis, P. Kelly, P. Daras, N. E. O'Connor, T. Boubekeur, and M. Ben Moussa, "Evaluating a dancer's performance using kinect-based skeleton tracking," in *Proc.19th ACM Int. Conf Multimedia*, 2011, pp. 659–662.

[20] B. Southwell and G. Fang, "Human Object Recognition Using Colour and Depth Information from an RGB-D Kinect Sensor," *Int. J. Adv. Robot. Syst.*, vol. 10, Mar. 2013.

[21] S. Obdrzálek, G. Kurillo, F. Ofli, R. Bajcsy, E. Seto, H. Jimison, and M. Pavel, "Accuracy and robustness of Kinect pose estimation in the context of coaching of elderly population.," *in Proc. 2012 Annu. Int. Conf. IEEE Engineering Medicine Biology Society*, pp. 1188–1193.

[22] "History - OpenKinect." [Online]. Available: http://openkinect.org/wiki/History, Accessed on: October 20, 2016.

[23] "Microsoft Releases Kinect for Windows SDK Beta for Academics and Enthusiasts." [Online]. Available: http://www.microsoft.com/en-us/news/press/2011/jun11/06-16mskinectsdkpr.aspx, Accessed on: October 20, 2016.

[24] A. Kadambi, A. Bhandari, and R. Raskar, "3D Depth Cameras in Vision: Benefits and Limitations of the Hardware With an Emphasis on the First- and Second-Generation Kinect Models," in *Computer Vision and Machine Learning with RGB-D Sensors*, L. Shao, J. Han, P. Kohli, and Z. Zhang, Eds. Cham: Springer International Publishing, 2014, pp. 3–26.

[25] J. Sell and P. O. Connor, "The Xbox One System on a Chip and Kinect Sensor," *IEEE Micro*, vol. 34, no. 2, pp. 44–53, 2014.

[26] D. Um, D. Ryu, and M. Kal, "Multiple Intensity Differentiation for 3-D Surface Reconstruction With Mono-Vision Infrared," *IEEE Sens. J.*, vol. 11, no. 12, pp. 3352–3358, Dec. 2011.

[27] B. Molnar, C. K. Toth, and A. Detrekoi, "Accuracy Test of Microsoft Kinect for Human Morphologic Measurements," *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. XXXIX-B3, pp. 543–547, Aug. 2012.

[28] R. Macknojia, A. Chávez-Aragón, P. Payeur, and R. Laganière, "Experimental Characterization of Two Generations of Kinect ' s Depth Sensors," in *IEEE Int. Symp. Robotic and Sensors Environments*, 2012, pp. 150–155.

[29] M. Stommel, M. Beetz, and W. Xu, "Inpainting of Missing Values in the Kinect Sensor's Depth Maps Based on Background Estimates," *IEEE Sens. J.*, vol. 14, no. 4, pp. 1107–1116, Apr. 2014.

[30] T. Stoyanov, A. Louloudi, H. Andreasson, and A. J. Lilienthal, "Comparative Evaluation of Range Sensor Accuracy in Indoor Environments," in *Proc. 2011 IEEE Int. Conf Comput. Vision Workshops*, pp. 1154–1160.

[31] J. Smisek, M. Jancosek, and T. Pajdla, "3D with Kinect," in *Consumer Depth Cameras for Computer Vision—Research Topics and Applications: Advances in Computer Vision and Pattern Recognition.* New York, NY, USA: Springer-Verlag, 2013, ch. 1, pp. 3-25.

[32] T. Mallick, P. P. Das, and A. K. Majumdar, "Characterizations of Noise in Kinect Depth Images: A Review," *IEEE Sens. J.*, vol. 14, no. 6, pp. 1731–1740, Jun. 2014.

CHAPTER 3

A SYSTEM COMBINING FORCE AND VISION SENSING FOR AUTOMATED SCREW REMOVAL ON LAPTOPS

by

Nicholas M. DiFilippo and Musa K. Jouaneh

Submitted to *IEEE Transactions on Automation Science and Engineering*

Corresponding Author: Musa Jouaneh

Department of Mechanical, Industrial, and Systems Engineering

University of Rhode Island

103C Gilbreth Hall, 2 East Alumni Ave.

Kingston, RI, 02881, U.S.A.

Phone : +1-401-874-2349

Email Address : jouaneh@uri.edu

Abstract

This paper investigates the performance of an automated robotic system, which uses a combination of vision and force sensing to remove screws from the back of laptops. This robotic system uses two webcams, one that is fixed over the robot and the other mounted on the robot, as well as a sensor-equipped (SE) screwdriver. Experimental studies were conducted to test the performance of the SE screwdriver and vision system. The parameters that were varied included the internal brightness settings on the webcams, the method in which the workspace was illuminated, and color of the laptop case. A localized light source and higher brightness setting as the laptop's case became darker produced the best results. In this study, the SE screwdriver was able to successfully remove 96.5% of the screws.

Keywords: Automated Disassembly, Computer Vision, Disassembly Tooling, Electronic Waste, Robotic Disassembly

3.1 Introduction

Electronic waste (e-waste) is a growing concern all over the world. While consumer demand for electronic products continues to grow, the lifecycles of products are shortening, causing a stockpile of discarded e-waste that must be dealt with. E-waste is composed of many heavy metals [1] that are harmful to both human health and the environment. In developing countries, common methods of e-waste disposal are burning, burying, or dumping it in the sea [2]. In 2014, 7.7 million metric tons of e-waste was generated in the U.S., but only 15% of it was recycled. On a global scale, 41.8 million metric tons of e-waste was generated in 2014 and this number is expected to keep growing to 49.8 million metric tons by 2018 [3].

E-waste is recycled by destructive, semi-destructive, and non-destructive methods. Destructive methods are suitable if the aim of the disassembly process is to recycle materials from the part, however, the part will be destroyed. Semi-destructive disassembly will destroy parts of the object such as screws and snap fits. Non-destructive disassembly aims to reverse engineer the assembly cycle. Non-destructive disassembly is appealing if the goal is to reuse parts in a different product or if the disassembled part is hazardous. Manual (non-destructive) disassembly is costly and time-consuming since most products are designed for assembly not disassembly.

Robotics is a technology that can revolutionize manufacturing. However, several issues need to be addressed in order to increase the use of robots in manufacturing operations such as the automated disassembly of e-waste. Issues include the high cost of robotic work cells, creation of cost-effective force sensing tooling, and the ability to work in an unstructured environment [4].

The high cost of the primary robotic hardware is only a fraction of the total cost of creating a robotic work cell. Since most industrial robots cannot operate in unstructured environments, large amounts of money are spent designing and fabricating additional equipment, so that parts and features are located at precise pre-specified locations. If a robot was able to learn what tasks it needs to perform from sensors in a non-engineered environment, these expenses can be substantially reduced. A robot needs force information to guide interactions with an environment. It would be advantageous to develop tooling that utilizes reliable, low-cost force sensors such as force-sensing resistors (FSR) [5-9] since current sensors such as load cells are expensive.

Several researchers have investigated the design of disassembly tooling and grippers. Rebafka *et al*. [10] proposed a flexible unscrewing tool that could create its own acting surfaces to improve loosening. Park and Kim [11] discussed the development of a 6-axis force-moment sensor for an intelligent robot’s gripper. Feldmann *et al*. [12] detailed the design of a drill driver, which could use or create a working point to transmit torque to remove a fastener, or drill to destroy the fastener. Zuo *et al*. [13] presented a screwnail that could create an indentation in a product and attach to transmit forces and torques required for dismantling operations. Seliger *et al*. [14] presented an unscrewing tool consisting of a modified drill bit with movable needles to hold objects down while disassembly steps were performed. This tool was designed for flexibility over accuracy since different variations of products can exist. Peeters *et al.* [15] designed a prying tool that could pull apart housing components to remove LCD screens.

This paper builds upon previous work performed by Schumacher and Jouaneh [5][16] where a disassembly tool was created that used FSRs to remove snap fits and spring-loaded batteries from calculators. The tool used two FSRs to detect horizontal forces and one FSR to detect vertical forces on the tool tip. The force applied to the snap-fit was monitored with the FSRs in order to fully depress but not damage it. OpenCV’s template-matching algorithms were used to identify the type of device and its orientation in the workspace. After a match was determined, the x-y position of the snap-fit was retrieved from a lookup table and robot would begin the disassembly process. Using feedback from the FSRs, the system would first remove the battery cover and then the batteries from the device. The use of a Kinect sensor helped make the system more robust due to minor variations that could occur loading the device. The current work extends the previous work to products that it has no prior information on. This means that if a product placed in a workspace was missing a battery cover or had some other physical defect, the system would still be able to perform disassembly operations.

Disassembly of products is a difficult task because of the uncertainty regarding their physical structure and conditions. Tian *et al.* described a probability analysis method of disassembly cost with a new evaluation parameter, cost disassemblability degree, [17] and chance constrained programming models [18] in order to determine an efficient disassembly sequence. Tang and Zhou [19] presented an extended disassembly petri net to help develop the best disassembly sequence for a product. Kim *et al.* [20] developed a partly automated disassembly system capable of generating and adapting disassembly plans based on the product, as well as determining whether the disassembly step should be performed manually or by an automated process. Guo *et al.* [21] introduced a model using petri nets to determine the disassembly sequence for large scale products which was optimized for disassembly operators and tools.

The motivation for this work was to create a robotic system that could recycle e-waste and minimize the need for a human operator. This work presents a novel approach to finding and removing screws that combines vision and force sensing. This approach can later be augmented with a cognitive architecture and using information learned from these methods, that will facilitate the disassembly process.

The rest of this paper is organized as follows. The next section discusses the overall setup of the automated system. Section 3.3 goes into further detail about the two main components, the cameras and sensor-equipped (SE) screwdriver. The testing methodology and an overview and explanation of the logic of the system are given in Section 3.4 while the results of the testing are detailed in Section 3.5. Concluding remarks and a short discussion are given in the final section.

3.2 Overall System Setup

The robotic platform used was an ENCore 2s System (MRSI North Billerica, MA) modified to be controlled by a Galil DMC-2143 controller. This controller was interfaced with Matlab and a graphical user interface (GUI) was designed to control the system. A wooden frame to hold panels of black fabric was constructed around the robot and is shown in Fig. 1. The fabric allows for the ambient fluorescent light to be eliminated and a uniform lighting to illuminate the workspace. The black fabric is attached to the frame by Velcro which allows the panels to be removed for maintenance. The workspace platform is constructed out of an aluminum sheet covered with black felt to absorb the IR beam emitted from the Microsoft Kinect sensor. The felt also aids parts in sliding when they are clamped to the edge of the workspace. There is a cutout that allows a checkerboard to be placed level to the surface to set the origin for the camera system. Two linear actuators were added to the workspace to provide automated clamping capability. The SE screwdriver and webcam were mounted on the robot and are shown in Fig. 2. A second webcam was mounted above the robot allowing it to see the entire workspace. In the following section, each of the components will be explained in greater detail.

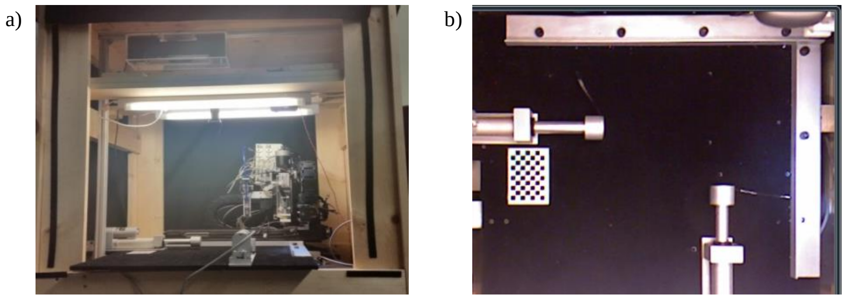


Fig. 1 a) View of disassembly module with the front panel removed. b)Top down view of the workspace.

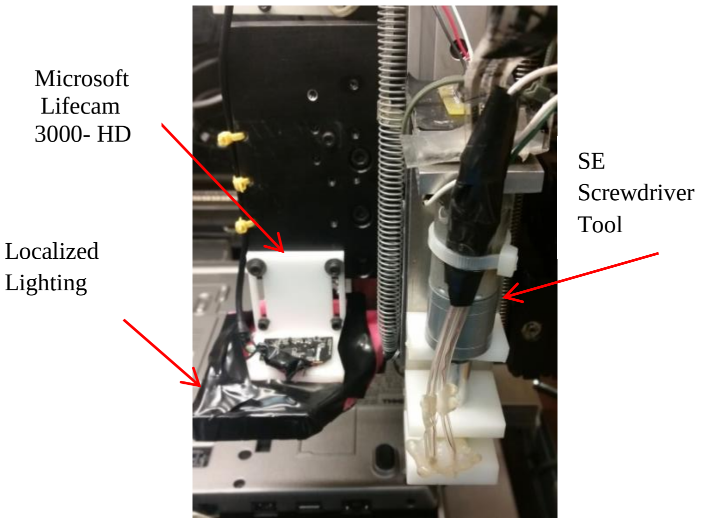


Fig. 2. Disassembly module consisting of the SE screwdriver tool, webcam, and localized lighting

3.3 System Components

3.3.1 Camera System

The robot uses two different cameras in order to identify laptop features. The first camera is an overhead camera that can view the entire workspace and is used to find circles on the laptop that may contain screws. The second camera is mounted on the robot and used to find and center screw holes. Both cameras are the Microsoft Lifecam 3000-HD which was chosen because it can capture images in high definition (1280x720 pixels) as opposed to the Kinect sensor which has a lower resolution (640x480 pixels). A webcam calibration process was used to remove lens distortion, which is greatest near the edges of an image, from the overhead camera. The calibration process consisted of using a 7x9 checkerboard pattern (8.5mm square) and the Camera Calibrator App in Matlab. The checkerboard is moved around the workspace and images are taken with it at different angles in order to calculate the camera's intrinsic matrix parameters listed in Table 1.

Table 1. Intrinsic Properties for the Microsoft Lifecam 3000HD

Symbol	Properties	Values
c_x	Optical center x	654.4 px
c_y	Optical center y	367.7 px
f_x	Focal length x	1129.5 px
f_y	Focal length y	1129.5 px
s	Skew coefficient	2.1673

After calibration, the location of an object (in mm) in the workspace is given by solving (1) for X and Y. In this equation, f_x and f_y are the focal length of the camera in pixels, s be the skew coefficient, and c_x and c_y are the optical center in pixels of the length and width. R and T are the rotation and translation values of how the world is transformed relative to the camera (extrinsic matrix) while x and y are the pixel's locations. The extrinsic matrix is calculated by the location of the checkerboard and the world coordinates origin.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & T_1 \\ R_{21} & R_{22} & T_2 \\ R_{31} & R_{32} & T_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{1}$$

Finally, using the transformation matrix K_c^w, the coordinate system is transformed from the edge of the checkerboard to the corner of the workspace as shown in Fig. 3. Using (2) - (4), it is possible to take the location of an object and transform its coordinates from the camera coordinates to world coordinates. Solving (4) yields a 4x1 matrix containing the X and Y locations of the object with respect to the world origin.

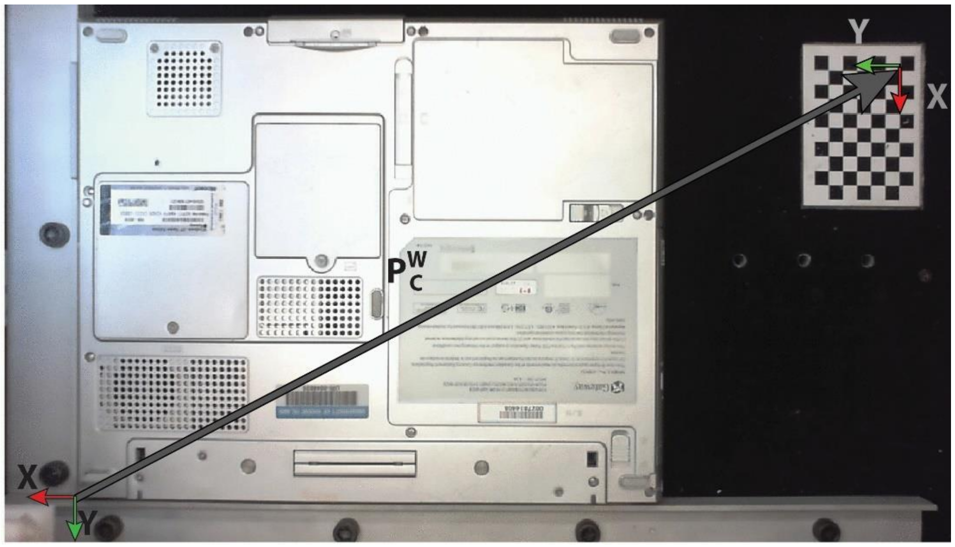


Fig. 3. Coordinate transformation from the camera coordinates to the world coordinate workspace.

$$K_c^w = \left[\begin{array}{ccc|c} & R_C^W & & P_C^W \\ & 3x3 & & 3x1 \\ 0 & 0 & 0 & 1 \end{array}\right] \tag{2}$$

$$R_C^W = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \quad P_C^W = \begin{bmatrix} -477 \\ -251 \\ 0 \end{bmatrix}, \quad and\ p^c = \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} \tag{3}$$

$$p^w = K_c^w * p^c = \begin{bmatrix} X_{workspace} \\ Y_{workspace} \\ 0 \\ 1 \end{bmatrix} \tag{4}$$

The overhead camera’s resolution (~0.5 mm/px) is not high enough to place the SE screwdriver on the head of a screw. When determining the circle’s center, an error of a few pixels could result in an error in the placement of the SE screwdriver of a few millimeters. A camera mounted on the robot’s head needs to be used to center the SE screwdriver with respect to a hole. Since the distance from the center of the robot camera to the tip of the SE screwdriver is known, once the screw is centered with the robot camera, the robot can be moved by this offset and place the SE screwdriver directly on the screw head. This camera does not need to go through the same calibration process because the center of a lens does not have distortion. The robot camera was removed from the plastic housing and a plano-convex optical lens (10mm diameter, 50mm focal length Edmunds Optics) was used to make the camera focus on the laptop at short range. A lighting fixture consisting of four white LEDs with a large viewing angle (> 100 degrees) was attached to the outside of the webcam.

3.3.2 Sensor Equipped Screwdriver

The motivation behind this tool was to create a screwdriver capable of probing and removing Philips head screws from a laptop in a non-destructive manner. The screwdriver needed to be able to determine if a screw was located at a position identified by the vision system and when the screw had been completely removed from the hole. The SE screwdriver, shown in Fig. 4, consists of an inner and an outer shell, connected by a low friction linear slide that allows the shells to move relative to one another. A small extrusion on the top of the inner shell triggers a FSR when the tip of the SE screwdriver makes contact with a surface. A pneumatic cylinder connected to the outer shell allows the tool to keep a constant pressure on a screw when it is trying to remove it and to retract above the product when the robot needs to move.

An accelerometer on top of the outer shell is used to determine when a screw has been completely loosened from a screw hole. Due to the geometry of a screw thread, when a screw thread ends and continues to rotate, it will fall downward every complete rotation, creating an acceleration along the axis of the screw which can be picked up by the accelerometer. The signal from the accelerometer is amplified and sent through a set-reset latch circuit on a custom Arduino shield, which makes sure two pulses from the accelerometer are detected before it signals the screw has been loosened. The driving force of the SE screwdriver is a dc motor controlled by an Arduino Motorshield R3 which allows for the motor’s current to be easily monitored. If the SE screwdriver is engaged with a screw, the current will increase as the motor stalls out. The SE screwdriver also has an electromagnet that surrounds the tip and when energized, magnetizes it allowing the SE screwdriver to remove the screw from the hole.

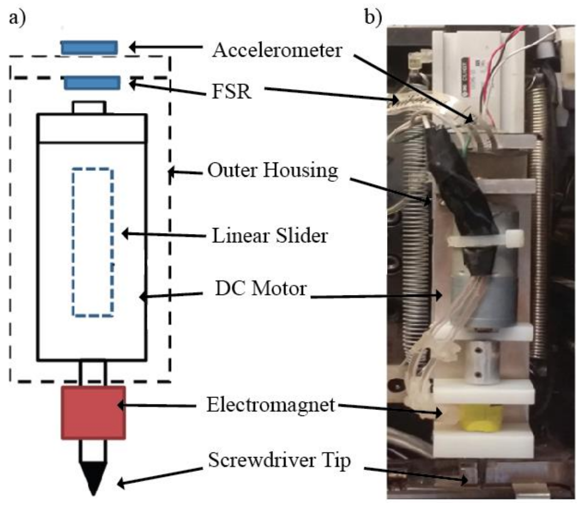


Fig. 4. a) Sketch of screwdriver. b) Prototype of screwdriver (Linear slider is hidden behind shell).

3.4 Testing Methodology

3.4.1 Sensor Equipped Screwdriver Tests

The SE screwdriver follows the state transition diagram shown in Fig. 5 starting in the state 'XY Move' where it moves to a desired position. After the SE screwdriver has reached that position, the state changes to 'Mate', the pneumatic cylinder extends, and the screwdriver lowers towards the part. When the tip of the screwdriver makes contact with the part002C the pressure supplied by the pneumatic cylinder will keep the outer shell in place which allows the inner shell to move upward and trigger the FSR. The state will change to 'Tighten' and the SE screwdriver will check to see if a screw is present by either having the accelerometer trigger (screwdriver motion akin to stripping a screw) or the motor current increase (motor stalled out). If neither of these conditions are met, and a timeout value is exceeded, then no screw is present and the state will move to 'Retract'. If a screw is found, the state will change to 'Loosen' where the screwdriver will loosen the screw until the accelerometer triggers (screw removed) or a timeout value is exceeded. This process will repeat for all of the screws that are found.

An aluminum block (50 x 70 x 25.4 mm) containing six rows for different screw types (#10-32 [¾", ½", ¼"], #10-24 [¾", ½", ¼"], #8-32 [½", ¼"], #6-32 [½", ¼"], #4-40 [¼"], #2-56 [¼"]) with four holes each was used to test how well the SE screwdriver could remove screws. The holes in the block not containing screws were used to test how well the SE screwdriver could distinguish between holes with and without screws. Five trials were performed (120 holes) in which the screw removal order was defined, and three trials (72 holes) were performed in which the order was randomized. The locations of the center of the holes were preprogrammed into the system for these tests.

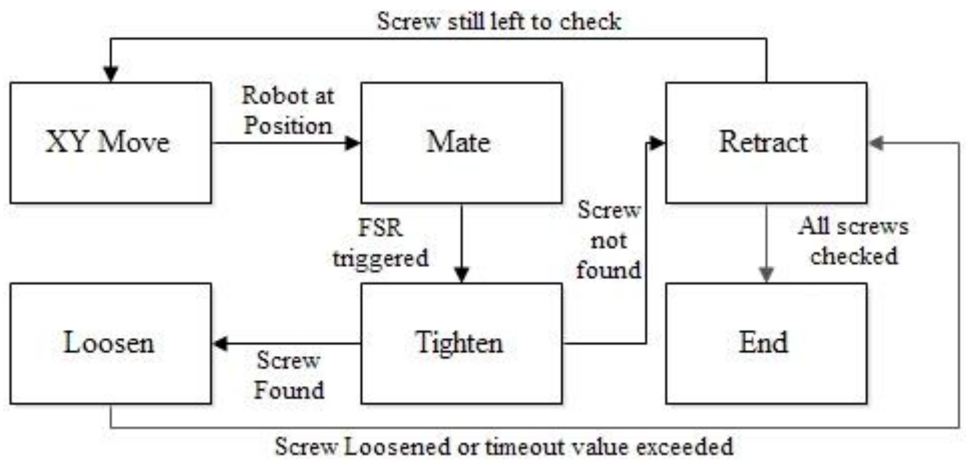


Fig. 5. SE screwdriver state transition diagram

3.4.2 Computer Vision Algorithm

The computer vision algorithm used to find the holes is shown in Fig. 6. An image is taken and a 5x5 Gaussian blur (standard deviation [std] = 5) and Prewitt edge detection (sensitivity threshold value [stv] = 0.75) are applied. The edges are dilated, any enclosed regions filled in, and all clusters touching the borders are removed. A structured diamond element is used to erode the image and find any connected components. The original image and results of these steps are shown in Fig. 7. If no connected components are found, then the stv is decreased by 0.03 and the process repeats. If the stv reaches 0, it is reset to 0.75 and std value is increased by 2. If the std reaches 15 and a hole is still not found, the std is reset to 2, the stv is reset to 0.75, and the algorithm will repeat the previous steps again but use a disk structured element to close any contours in the image. When a connected component is found, its area and length-width ratios are checked to ensure they are within certain bounds. If so, the centroid is used to center the hole with the robot webcam. The values of these parameters were determined by experimentation and were chosen to be sensitive enough to detect small changes in the image as they were varied, but also to limit the time of a test.

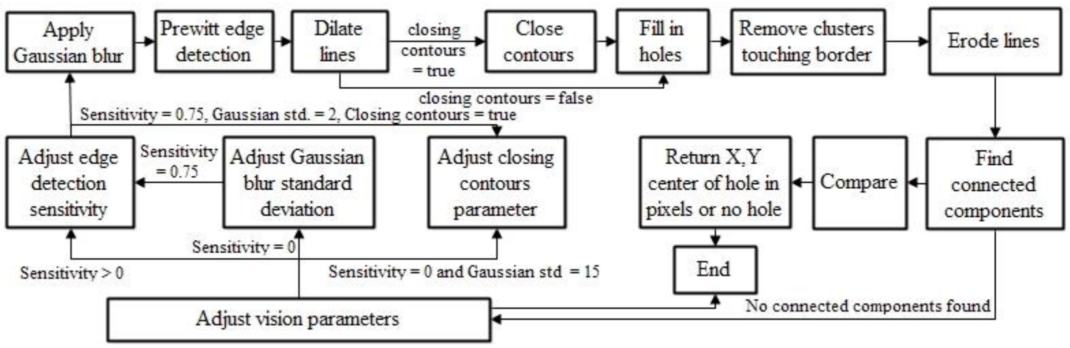

Apply Gaussian blur
Prewitt edge detection
Dilate lines
closing contours = true
Close contours
Fill in holes
Remove clusters touching border
Erode lines
closing contours = false
Sensitivity = 0.75, Gaussian std. = 2, Closing contours = true
Adjust edge detection sensitivity
Sensitivity = 0.75
Adjust Gaussian blur standard deviation
Adjust closing contours parameter
Return X,Y center of hole in pixels or no hole
Compare
Find connected components
Sensitivity = 0
Sensitivity > 0
Sensitivity = 0 and Gaussian std = 15
End
No connected components found
Adjust vision parameters

Fig. 6. Computer vision logic.

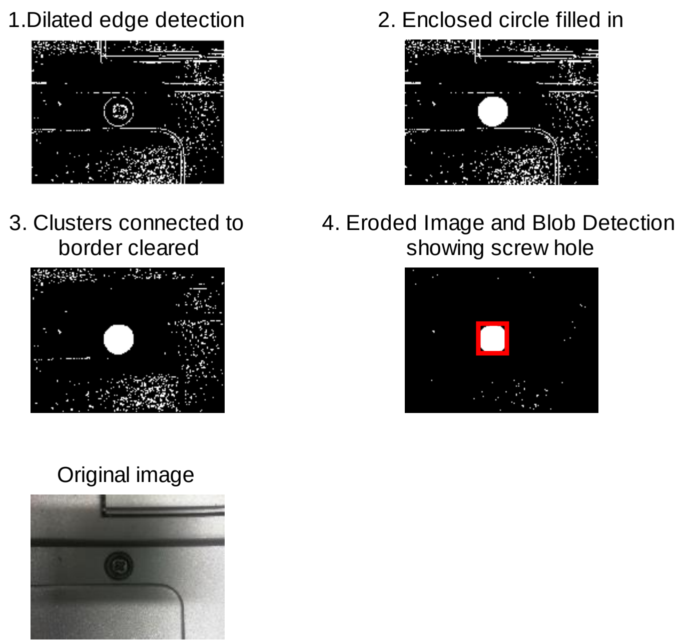

1.Dilated edge detection
2. Enclosed circle filled in
3. Clusters connected to border cleared
4. Eroded Image and Blob Detection showing screw hole
Original image

Fig. 7. Computer vision results at different stages in the screw hole detection process.

3.4.2 Laptop Screw Removal Tests

A set of 36 trials were performed that varied the type of lighting, the cameras brightness levels, and the color of laptop cases. The overhead camera brightness level was varied at 30, 90, and 150, and the robot camera brightness level was varied at 30 and 90. The two types of lighting tested were overhead lights mounted in the frame and the localized lighting fixture around the robot camera. Three different laptops models were used and consisted of light, medium and dark cases all of which used Phillips head screws.

A flow chart detailing the disassembly process is shown in Fig. 8. An initialization process homes the robot and the laptop is clamped. Then a snapshot of the workspace is taken, the image is undistorted, and circles are located on the laptop using a Hough circle transform (Tao Peng, Matlab Central). The program moves the robot camera over every circle and attempts to detect a hole. The distance from the center of the camera to the center of the centroid is calculated, and the robot is moved to offset this error until the error is less than 0.5 mm. After the hole has been centered, the SE screwdriver is moved over the hole to check for a screw. If a screw was found, then other circles within a certain distance (10 mm) are discarded. This entire process is repeated for all remaining holes. A visual of the robot performing these steps is shown in Fig. 9.

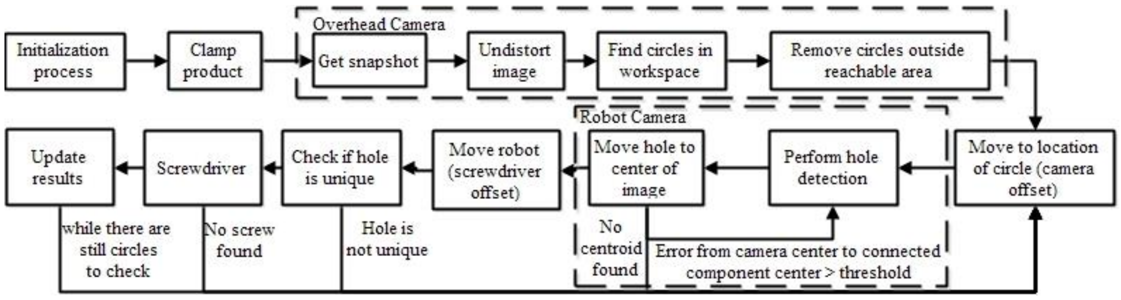

Overhead Camera
Initialization process
Clamp product
Get snapshot
Undistort image
Find circles in workspace
Remove circles outside reachable area
Robot Camera
Update results
Screwdriver
Check if hole is unique
Move robot (screwdriver offset)
Move hole to center of image
Perform hole detection
Move to location of circle (camera offset)
while there are still circles to check
No screw found
Hole is not unique
No centroid found
Error from camera center to connected component center > threshold

Fig. 8. Flow chart of automated disassembly process

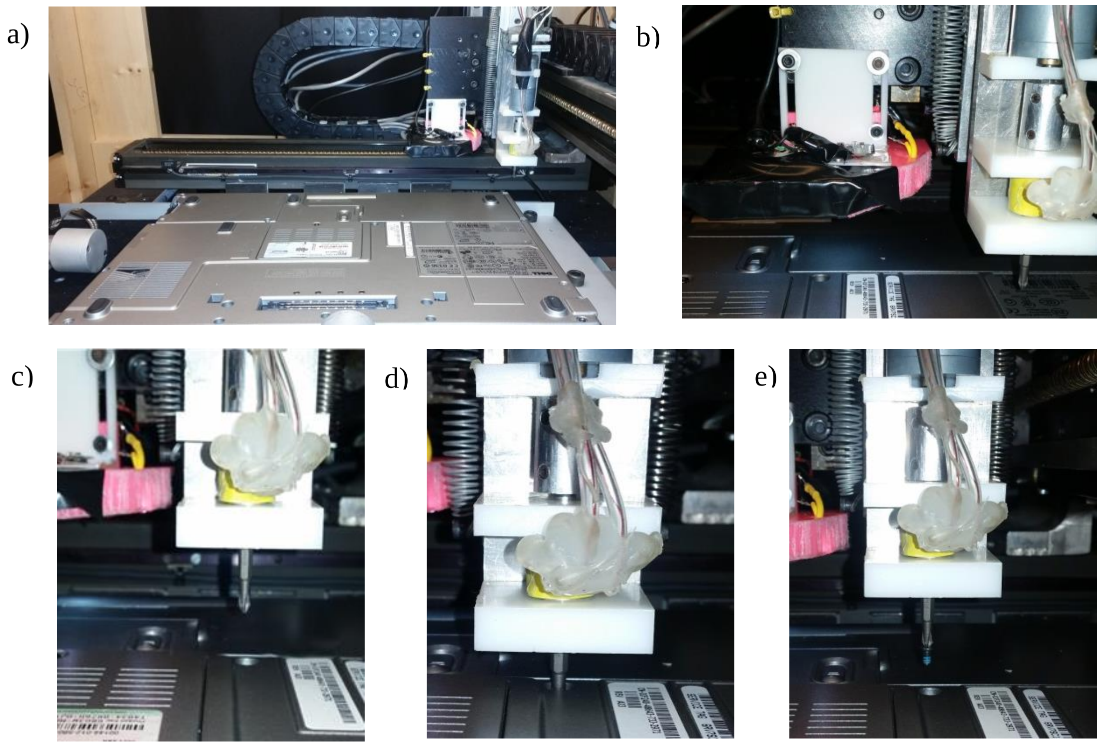


Fig. 9. a) The robot homes itself and finds circles in the workspace. b) The robot moves the camera with localized lighting over the circle to center a hole. c) The robot moves the SE screwdriver over the hole. d) The SE screwdriver is lowered into the hole and removes the screw. e) The SE screwdriver removes the screw from the hole.

3.5 Results

3.5.1 SE screwdriver Testing Results

A summary of how well and with which method the SE screwdriver was able to determine if a screw was present is shown in Table 2. When the screw removal order was preset, the SE screwdriver was able to correctly determine if a screw was present in all 120 cases. In 60 cases there was no screw in where a screw was present, the accelerometer was used to determine the screw presence 56 times (93.3%), while current monitoring was used four times (6.7%). When the removal order was random, it was correct in all 72 cases and 19 (52.8%) times it used the accelerometer, and 17 times (47.2%) it used current monitoring to determine the screw presence.

The time the SE screwdriver took to unscrew the different lengths and types of screws is shown in Fig. 10. There is a positive linear relationship between the length of the screw and the time that it takes to unscrew it as well as a positive relationship in the number of threads per inch that a screw has and the amount of time it takes to unscrew it.

Table 2. Summary of method used to find Screws.

			Method used to determine screw presence	
Test	Percentage	No Screw	Accelerometer	Motor Current
Set Removal Order	100	60	56	4
Random Removal Order	100	26	19	17

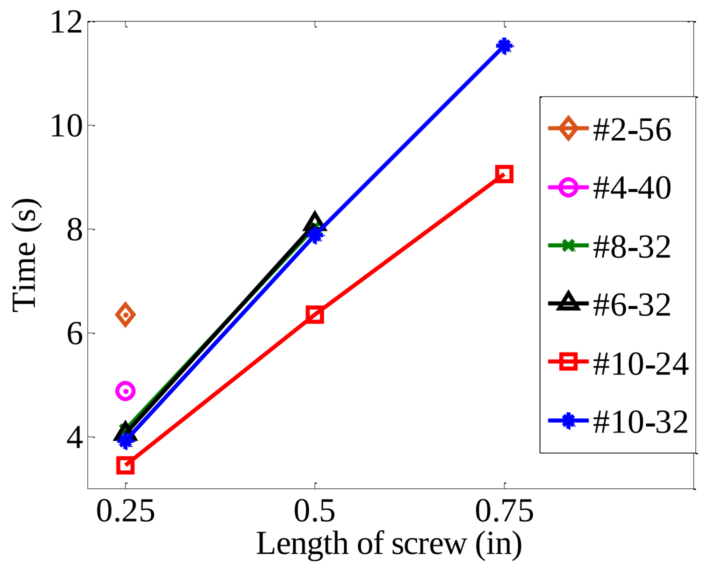


Fig. 10. The amount of time taken to remove screws of different lengths and threads.

3.5.2 Automated Laptop Screw Removal

The results of the automated screw removal test are shown in Table 3, where the first four columns list the parameters varied and the next four present the results. The number of circles, located by the overhead camera, are listed in the column "*Total Number of Circles Found"*. These circles include screw holes on the laptop, in addition to shapes or geometries perceived as circles. Every circle represents a location that the SE screwdriver has to move to and check during a test. The number of screw holes that were correctly identified by the overhead camera are listed in the column "*Total Number of Screw Holes Found"*. The column "*Total Number of Screw Holes in Workspace"* lists the number of screw holes that are present in the reachable workspace on the laptop. The column "*Percentage of Screw Holes Found"* is the "*Total Number of Screw Holes Found"* divided by *"Total Number of Screw Holes in Workspace".* Finally, the row labeled "*Total"* shows cumulative statistics for all of the trials for each model of laptop.

The results show the localized lighting helps when the laptop case is darker as the average number of screws found goes from 16.7% to 30.8%. For the dark laptop, the best combination is when the overhead camera brightness is 150, robot camera brightness is 90, and localized lighting is used. The localized lighting eliminates shadows cast by the robot as it illuminates the laptop from under the robot. The results for varying the lighting and brightness parameters of the robot camera are presented in Fig. 11 and show a combination of using the localized lighting and higher robot camera brightness increases the system's ability to find a hole. Table 3 shows the results were best for each laptop model for all the trials when localized lighting was used and the robot camera brightness was 90.

Table 3. Summary of Identifying Screw Holes in Laptops with Different Parameters

Laptop Model	Localized Lighting	Robot Camera Brightness	Overhead Camera Brightness	Total Number of Circles Found	Total Number of Screw Holes Found	Total Number of Screw Holes in Workspace	Percentage of Screw Holes Found
Light	No	30	30	50	12	15	80.0
			90	70	11	15	73.3
			150	82	11	15	73.3
		90	30	53	11	15	73.3
			90	65	7	15	46.7
			150	69	7	15	46.7
	Yes	30	30	53	10	15	66.7
			90	68	13	15	86.7
			150	71	11	15	73.3
		90	30	57	11	15	73.3
			90	69	7	15	46.7
			150	63	8	15	53.3
Total					119	180	66.1
Medium	No	30	30	60	8	13	61.5
			90	59	8	13	61.5
			150	77	7	13	53.8
		90	30	52	10	13	76.9
			90	50	10	13	76.9
			150	84	9	13	69.2
	Yes	30	30	60	7	13	53.8
			90	52	9	13	69.2
			150	72	9	13	69.2
		90	30	60	9	13	69.2
			90	59	7	13	53.8
			150	79	11	13	84.6
Total					104	156	66.7
Dark	No	30	30	58	4	13	30.8
			90	54	2	13	15.4
			150	62	2	13	15.4
		90	30	45	2	13	15.4
			90	48	2	13	15.4
			150	67	1	13	7.7
	Yes	30	30	53	4	13	30.8
			90	43	3	13	23.1
			150	60	2	13	15.4
		90	30	60	3	13	23.1
			90	59	5	13	38.5
			150	79	7	13	53.8
Total					37	156	23.7

The number of circles located, is dependent on the overhead camera brightness. Fig. 12 shows the overhead brightness has a positive linear relationship on the number of circles found, which, up to a certain point, increases the chance that more screw holes will be found. The lighter the case, the faster this point will occur as the entire image will eventually turn white. A tradeoff for allowing more circles to be found is that the time of the test increases. As the laptop case color gets darker, the system's ability to locate holes increases as the overhead camera brightness increases.

Fig. 13 shows an image summary of the results for the laptops with different color cases. The reachable region of the workspace is shown by the enclosed area of the rectangle. A '○' represents a screw was found and removed at that location. An 'X' indicates the computer vision system found a circle, but a screw was not found when probed by the SE screwdriver. A '+' indicates the circle found was discarded because it was within 10 mm of a screw hole and a '△' shows that the computer vision system was not able to find a circle. The '□' also means the computer vision system did not find a screw at that particular circle location, however, at this location, the robot made at least one movement.

The time statistics for the tests are summarized in Table 4. These results show that the clamping time is not significant as it is an order of magnitude smaller than the robot movement time and screw removal time (srt), and two orders of magnitude smaller than the computer vision time (cvt). cvt is the greatest due to the number of circles that need to be explored and the various parameters associated with the computer vision algorithm. It should be noted that the srt time for the dark laptop is significantly lower because there were fewer holes detected.

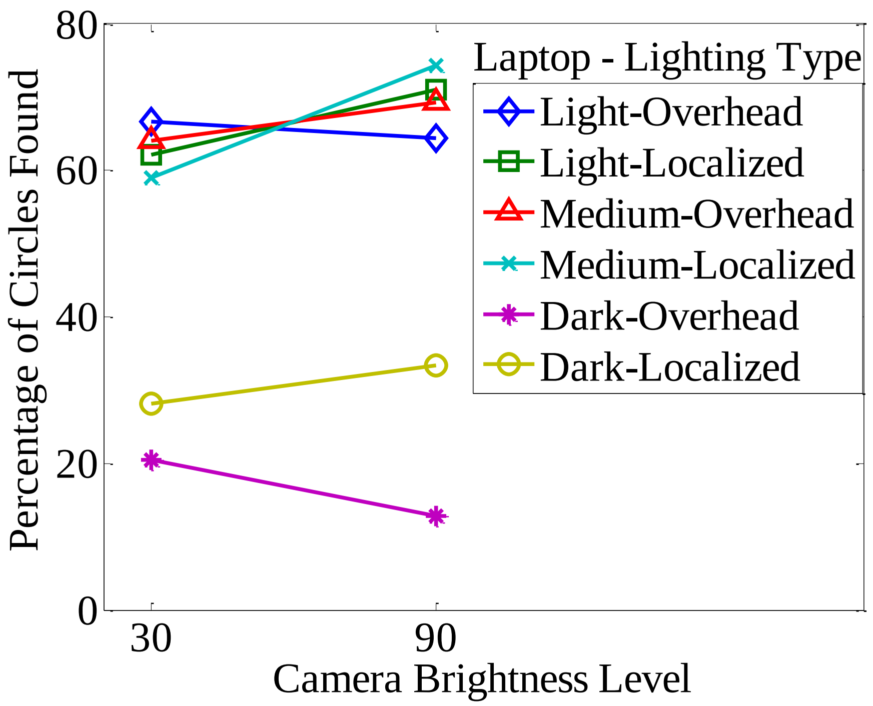


Fig. 11. The percentage of screw holes successfully found vs. the robot camera's brightness setting and the color of laptop case.

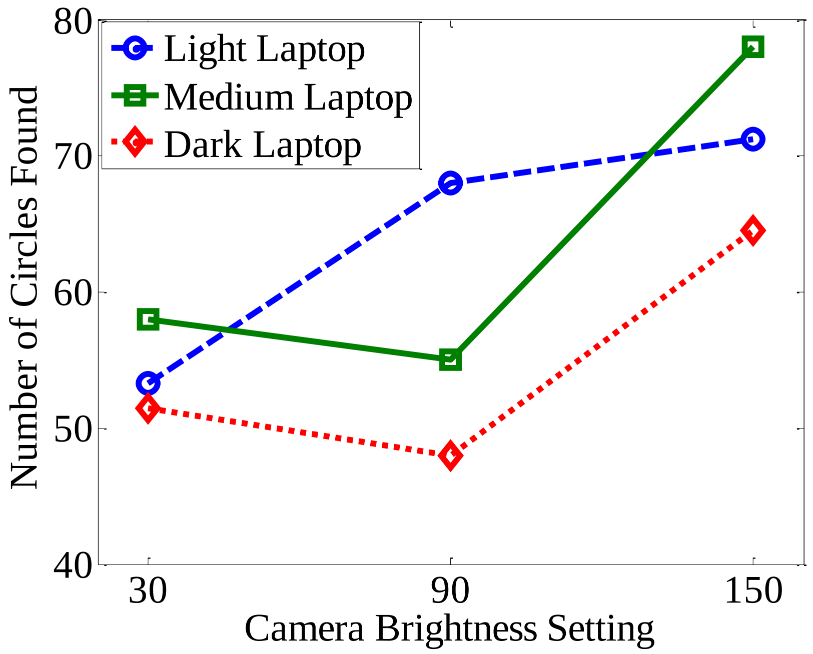
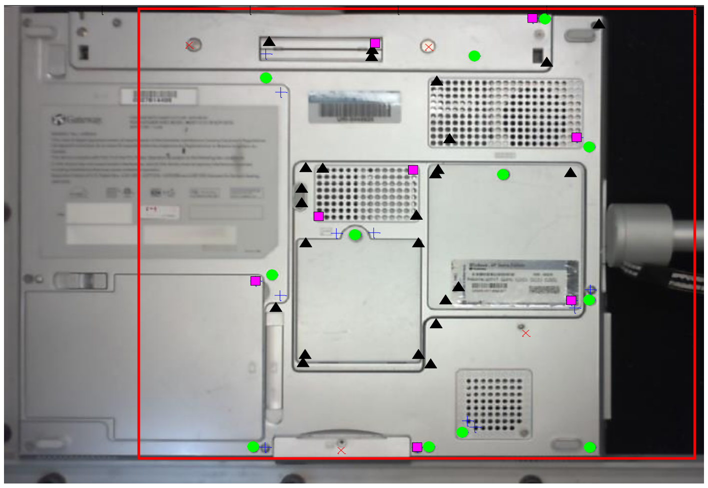
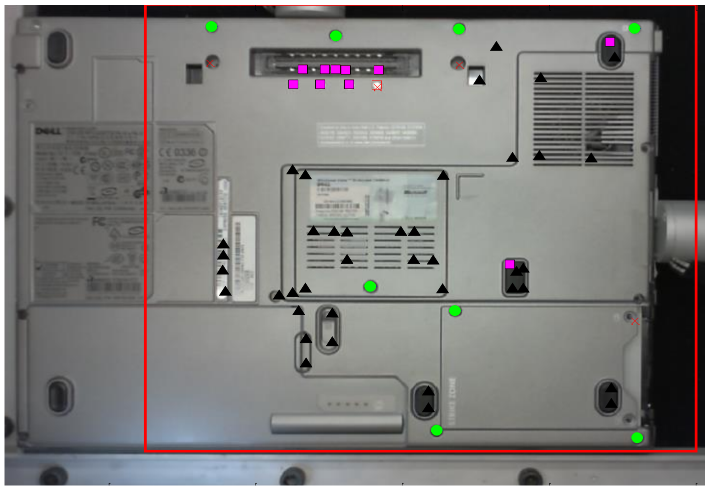


Fig. 12. Number of circles that are found versus the overhead cameras brightness level.

a)



b)



c)

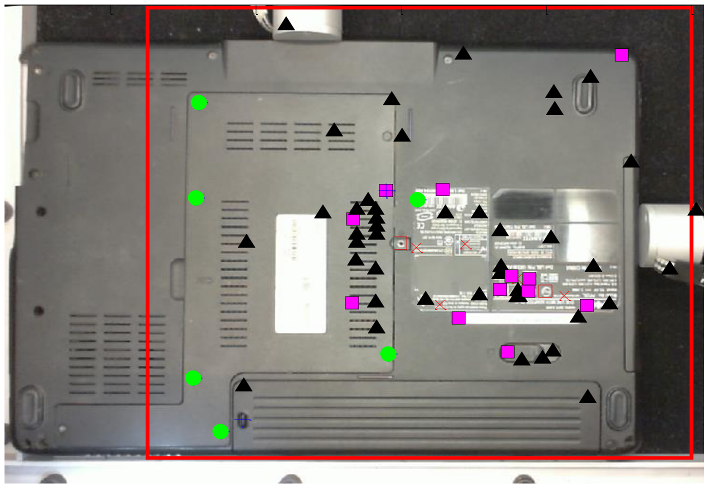


Fig. 13. a) Results for the laptop with a light case. b) Results for the laptop with the medium case. c) Results for laptop with a dark case. Circles mean that the screwdriver was able to determine a screw while X's mean that the screwdriver was not able to determine a screw. Plus signs indicate the screw hole was discarded due to proximity to a screw that was found and triangles and squares mean the computer vision algorithm was not able to determine a hole.

The performance of the SE screwdriver does not depend on how well the computer vision system worked or even the model of the laptop used. In order to not penalize the SE screwdriver for “missing” holes that the computer vision module was not able to find, the performance of the SE screwdriver is defined as the ‘screws found’ divided by total ‘screw holes that the computer vision found’. The SE screwdriver performed well and correctly removed 251/260 (96.5%) of screws in screw holes. In 27 of the 36 trials, the SE screwdriver determined if a screw was in the hole correctly 100% of the time. Even when the SE screwdriver missed a hole on a laptop, it never missed more than one.

Table 4. Timing Results for Automated Screw Removal Test

Laptop Model	Clamping Time (s)		Robot Movement Time (s)		Computer Vision Time (s)		Screwdriver Time (s)	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
Light	11.2	0.8	337.8	60.0	2671.7	967.9	166.2	36.4
Medium	8.8	0.4	357.4	55.2	2654.8	509.3	139.8	31.0
Dark	10.1	0.4	294.7	43.1	3013.3	332.2	55.3	26.0

3.6 Conclusion

This paper presented a new method that combines vision and force sensing in order to automatically find and remove screws from a laptop without preprogramming their locations. This method would be the first step in disassembling laptops or other products with a plastic casing covering internal electronics. The results presented in this paper show that this is a viable strategy for removing screws from laptops and could be incorporated into a system that uses a cognitive architecture to facilitate a disassembly process to recycle and reuse materials from e-waste.

A significant challenge is trying to automatically locate screw holes on the laptop. The results show this is composed of two tasks. The first involves finding holes with a computer vision module and the second involves probing the hole with a SE screwdriver. The lighting and the brightness of the cameras are the two major parameters that need to be adjusted and are dependent on how dark the laptop case is.

The darker the laptop case is, the higher the brightness the cameras should use. The higher the brightness in the overhead camera, the more circles will be found and the greater chance that all of the screws will be found. A greater brightness with the robot camera creates a higher contrast with a hole making it easier to find. The best results occurred when using the localized lighting as it was able to light the workspace from underneath the robot, eliminating shadows that would occur if the robot was lit from above. A screwdriver tool was also introduced that was able to tell if a screw was present and when that screw had been completely loosened by using an accelerometer.

A current drawback of this system is the length of time it takes to complete a test. Next, the cognitive architecture Soar will be added to the robot and will allow it to remember the locations of the holes to reduce the time. Soar [22] will be able to use the methods presented in this paper to find screws and will be used in order to select the hole to remove a screw from. The time will be reduced because if a circle has already been explored and a screw was not found, that location will not need to be explored again. Optimization of different image parameters such as the brightness levels for the different laptops or parameters governing the Hough circle transform, the number of screws that are found on each laptop could be improved.

3.7 References

[1] A. Chen, K. N. Dietrich, X. Huo, and S. Ho, “Developmental neurotoxicants in e-waste: an emerging health concern.,” *Environ. Health Perspect.*, vol. 119, no. 4, pp. 431–438, Apr. 2010.

[2] I. C. Nnorom and O. Osibanjo, “Overview of electronic waste (e-waste) management practices and legislations, and their poor applications in the developing countries,” *Resour. Conserv. Recycl.*, vol. 52, no. 6, pp. 843–858, Apr. 2008.

[3] J. Baldé, C.P., Wang, F., Kuehr, R., Huisman, "The Global E-Waste Monitor - 2014". United Nations Univ., IAS - SCYCLE, Bonn, Germany. 2015. [Online]. Available: https://i.unu.edu/media/unu.edu/news/52624/UNU-1stGlobal-E-Waste-Monitor-2014-small.pdf. Accessed on: October 21, 2016.

[4] “A Roadmap for U . S . Robotics From Internet to Robotics,” 2013. [Online]. Available: https://robotics-vo.us/sites/default/files/2013 Robotics Roadmap-rs.pdf. Accessed on : Oct. 20, 2016.

[5] P. Schumacher and M. Jouaneh, “A force sensing tool for disassembly operations,” *Robot. Comp. Integr. Manuf.*, vol. 30, no. 2, pp. 206–217, Apr. 2014.

[6] A. Gopalai, A. Senanayake, and D. Gouwanda, “Determining level of postural control in young adults using force-sensing resistors,” *IEEE Trans. Inf. Technol. Biomed.*, vol. 15, no. 4, pp. 608–614, Jul. 2011.

[7] C. Castellini and V. Ravindra, “A wearable low-cost device based upon Force-Sensing Resistors to detect single-finger forces,” in *Proc. 5th IEEE RAS/EMBS Int. Conf. Biomedical Robotics and Biomechatronics*, 2014, pp. 199–203.

[8] C. Zerpa, D. Jackson, P. Sanzo, and D. Kivi, “Examining the Reliability of a Force Sensing Resistor as a Possible Tool to Assess Carpal Tunnel Syndrome,” *Sci. Technol.*, vol. 5, no. 1, pp. 1–4, 2015.

[9] R. A. Romeo, F. Cordella, L. Zollo, D. Formica, P. Saccomandi, E. Schena, G. Carpino, A. Davalli, R. Sacchetti, and E. Guglielmelli, “Development and preliminary testing of an instrumented object for force analysis during grasping,” in *Proc. Annu. Int. Conf. IEEE Engineering Medicine and Biology Society*, 2015, pp. 6720–6723.

[10] U. Rebafka, G. Seliger, A. Stenzel, and B. Zuo, “Process model based development of disassembly tools,” *Proc. Inst. Mech. Eng., Part B J. Eng. Manuf.*, vol. 215, no. 5, pp. 711–722, May 2001.

[11] J. J. Park and G.-S. Kim, “Development of the 6-axis force/moment sensor for an intelligent robot’s gripper,” *Sens. Actuators, A*, vol. 118, no. 1, pp. 127–134, Jan. 2005.

[12] K. Feldmann, S. Trautner, and O. Meedt, “Innovative Disassembly Strategies Based on Flexible Partial Destructive Tools,” *Annu. Rev. Control*, vol. 23, pp. 159–164, 1999.

[13] B.R. Zuo, A. Stenzel, and G. Seliger, “A novel disassembly tool with screwnail endeffectors,” *J. Intell. Manuf.*, vol. 13, no. 3, pp. 157–163, Jun. 2002.

[14] G. Seliger, T. Keil, U. Rebafka, and A. Stenzel, “Flexible disassembly tools,” in *Proc. 2001 IEEE Int. Symp. Electronics and Environment*, pp. 30–35.

[15] J. R. Peeters, P. Vanegas, C. Mouton, W. Dewulf, and J. R. Duflou, “Tool Design for Electronic Product Dismantling,” *Procedia CIRP*, vol. 48, pp. 466–471, Jul. 2016.

[16] P. Schumacher and M. Jouaneh, “A System for Automated Disassembly of Snap-fit Covers,” *Int. J. Adv. Manuf. Technol.*, vol. 69, pp. 2055–2069, Jul. 2013.

[17] G. Tian, M. Zhou, J. Chu, and Y. Liu, “Probability evaluation models of product disassembly cost subject to random removal time and different removal labor cost,” *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 2, pp. 288–295, Apr. 2012.

[18] G. Tian, M. Zhou, and J. Chu, “A Chance Constrained Programming Approach to Determine the Optimal Disassembly Sequence,” *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 4, pp. 1004–1013, Oct. 2013.

[19] Y. Tang and M. Zhou, “A Systematic Approach to Design and Operation of Disassembly Lines,” *IEEE Trans. Autom. Sci. Eng.*, vol. 3, no. 3, pp. 324–329, Jul. 2006.

[20] H. Kim, R. Harms, and G. Seliger, “Automatic Control Sequence Generation for a Hybrid Disassembly System,” *IEEE Trans. Autom. Sci. Eng.*, vol. 4, no. 2, pp. 194–205, Apr. 2007.

[21] X. Guo, S. Liu, M. Zhou, and G. Tian, “Disassembly Sequence Optimization for Large-Scale Products With Multiresource Constraints Using Scatter Search and Petri Nets,” *IEEE Trans. Cybern.*, to be published, doi : 10.1109/TCYB.2015.2478486.

[22] J. E. Laird, A. Newell, and P. S. Rosenbloom, “Soar: An architecture for general intelligence,” *Artif. Intell.*, vol. 33, no. 1, pp. 1–64, Sept. 1987.

CHAPTER 4

USING THE SOAR COGNITIVE ARCHITECTURE TO REMOVE SCREWS FROM DIFFERENT LAPTOP MODELS

by

Nicholas M. DiFilippo and Musa K. Jouaneh

In preparation to be submitted to *IEEE Transactions on Automation Science and Engineering*

Corresponding Author: Musa Jouaneh

Department of Mechanical, Industrial, and Systems Engineering

University of Rhode Island

103C Gilbreth Hall, 2 East Alumni Ave.

Kingston, RI, 02881, U.S.A.

Phone : +1-401-874-2349

Email Address : jouaneh@uri.edu

Abstract

This paper investigates an approach that uses the cognitive architecture Soar to improve the performance of an automated robotic system which uses a combination of vision and force sensing to remove screws from laptop cases. Soar's long term memory module, semantic memory, was used to remember pieces of information regarding laptop models and screw holes. The system was trained with multiple laptop models and the method in which Soar was used to facilitate the removal of screws was varied to determine the best performance of the system. In all cases, Soar could determine the correct laptop model and in what orientation it was placed in the system. Soar was also used to remember what circle locations that were explored contained screws and what circles did not. Remembering the locations of the holes decreased a trial time by over 60%. The system performed the best when the number of training trials used to explore circle locations was limited, as this decreased the total trial time by over 10% for most of the laptop models and orientations.

Keywords: Automated Disassembly, Cognitive Architecture, Disassembly Tooling, Electronic waste, Robotic Disassembly, Soar Cognitive Architecture.

4.1 Introduction

Electronic waste (e-waste) is a growing concern in the world today as the number of electronics used in everyday life continues to grow while the lifespan of electronics continues to shrink [1]. In developing countries, the most prevalent ways of discarding e-waste are through burning, burying, or dumping it in the sea [2]. In areas of developing countries such as Guiyu China, where entire villages are heavily invested in the recycling of e-waste, the methods in which e-waste is recycled are crude and not many preventative measures are taken to protect the workers from the hazardous materials that they are disassembling. This is a concern because many of the heavy metals found in e-waste are very toxic to both humans and the environment. Due to this, many studies have been conducted to measure the wellness of the people and the levels of heavy metals found in the region [3–5].

Currently the methods that are used to disassemble e-waste are destructive, semi-destructive, and non-destructive disassembly. Destructive methods are used when the aim of the disassembly process is to recover materials such as metals and plastics from the waste. Typically, with destructive methods, the e-waste is fed into large shredders and then processed to retrieve different types of metals [6]. Semi-destructive methods of recovering e-waste involve cutting screws or wires in an attempt to disassemble the product, while non-destructive disassembly methods attempt to reverse engineer the assembly process. Non-destructive disassembly is an attractive option when the goal of the recycling operation is to be able to reuse the product for different applications. Although manual non-destructive methods can yield the highest amount of recycled product, manual disassembly is time consuming and if the proper working conditions are not met, dangerous for the workers performing the operation [7].

Schumacher and Jouaneh [8] created a disassembly tool that utilized low-cost force-sensing resistors (FSRs) and could remove snap-fits and batteries from calculators. This snap-fit removal tool was placed on a robotic system that used a

Microsoft Kinect and a Visual Basic program to control an EnCore 2s robot [9]. This system used a database that contained the preprogrammed locations of the batteries for different orientations of the calculator. The Kinect was used to make the system more robust against minor variations that may have occurred when the device was loaded in the workspace. DiFilippo and Jouaneh [10] extended the system to work with a variety of laptops that the system had never seen before. A webcam with an overview of the entire workspace was used to determine locations of features on the laptop, and a sensor-equipped (SE) screwdriver tool was developed that would go and explore those locations to determine if a screw was present or not. The drawback of this system, which this work aims to address, was that the system did not remember the locations of the screws. Therefore, for every trial, every circle that was previously located needed to be checked again which lead to relatively long trial times ranging from 40-60 minutes.

For a robot to make decisions based on an understanding of the parts and situations presented to it, some form of intelligence should be built into the robot's control system. Bannat *et al.* [11] detailed the evolution of production, planning, and cognition systems and their growth in the future as they pertain to flexible, adaptive production systems for manufacturing. Stenmark *et al.* [12] presented a knowledge based method that leverages cloud computing to extend the capabilities of a robotic system in a manufacturing setting. Kurup and Lebiere [13] showed that an algorithmic approach, where the user programs all possible scenarios that the robot can handle, is not efficient as the program will not be able to handle a scenario unless it was preprogrammed. Instead, an approach that uses cognition is preferred, as it corresponds more with the way that humans process and make decisions. Cognitive architectures make use of some form of high-level abstract reasoning to determine the next action to take and can incorporate different learning methods.

Popular cognitive agents include Soar [14], ACT-R [15], and EPIC [16-17]. ACT-R (current version 7.0.11) and Soar (current version 9.4) each have pros and cons associated with them. Jones *et al.* [18] identified that ACT-R and Soar have constraints that are opposites of each other in their respective low-level reasoning tasks. ACT-R only allows for one production rule to work at a time even if more rules are available. Using ACT-R, multiple passes must be made in order to make decisions, which increases computational time. Soar allows for multiple rule instantiations to fire at once. However, the computation time in Soar increases because only one type of operator rule is allowed to be selected at a time and the operators must be proposed through different types of rules. Hanford [19] suggested that Soar is a better choice to use for robots since it does not limit the access to working memory, and can use all knowledge to figure out how to proceed with a situation. Johnson [20] also compared the two architectures and stated the only similarity between the two is that they both organize control around a single goal hierarchy. In conflict resolution, Soar tries to select an action based on all available knowledge, and if an impasse occurs, create a sub-goal to solve the problem while ACT-R will halt if it detects an impasse.

Laird and Rosenbloom [21] used a robot called Robo-Soar [22], which was a PUMA arm, that used a vision system to obtain the orientation and position of objects in a workspace. The goal of Robo-Soar was to align these objects until a light came on and then press a button. Hanford *et al.* [23-24] used Soar to control mobile robots.

With these robots, Soar used different sensor information to decide how the robot should navigate to a GPS location while avoiding obstacles. One drawback in these robotic applications is that Soar's long-term memory is not used, so when the agent is initiated it cannot draw from its previous experiences. A Soar agent named Rosie learned to play new games and perform other tasks using mixed initiative interactions through natural language [25-26], and visual demonstrations [27]. Rosie used semantic memory when learning to play a game to store all action knowledge, failure conditions, and goal states to determine a legal play. Another system called LUCIA [28] was developed that works in conjunction with Rosie and uses grammatical rules to turn sentences into operations that Rosie can use in order to pick up or move objects. Mininger and Laird [29] developed a mobile robot that uses Soar to perform tasks and search for an object even if it cannot see the object it is looking for.

Vongbunyong *et al.* [30–32] described a system to perform semi-destructive disassembly of LCD TV's where a cognitive robotic agent is used. This cognitive agent used the language IndiGolog to determine the next disassembly step. A vision system is used to obtain information about the product being disassembled. The system presented used learning and revision to eliminate redundant moves performed during the disassembly operation.

These issues described above are very important when considering the use of robots to perform disassembly of end-of-life (EOL) electronic products. Unlike assembly operations on a production line, where product features and locations are known in advance, a robotic disassembly system for EOL products needs to work with a large variety of products of unknown sizes and, possibly, with damaged or missing features.

The rest of this paper is organized as follows; the section 4.2 will give a brief introduction to Soar and how Soar works. It will also discuss the procedure used to train Soar's semantic memory with the different laptop models and holes. Section 4.3 will discuss the results obtained during the trials, and Section 4.4 will contain concluding remarks.

4.2 Methods

4.2.1 Soar Cognitive Architecture

Soar is a general cognitive architecture developed by John Laird, Allen Newell, and Paul Rosenbloom at Carnegie Mellon University in 1983 [33]. Soar works by using productions (rules) to test, propose, and select operators (rules) that are in working memory to try to obtain a specified goal state. Working memory is a representation of how Soar views the workspace in its current configuration and is part of Soar's short term memory module. Soar stores information as working memory elements (WME) consisting of an identifier, attribute (preceded by a '^'), and value. The value can either be a terminal node comprising of a constant (string or numerical value) or a non-terminal node which links to another identifier. Fig. 1 shows a graphical representation of WMEs of Soar's memory structure that is created the first time a Soar agent is started. All the WMEs are connected back to the state S1 which is always the name of the top-level state. The state S1 always has three attributes, ^superstate, ^type, and ^io. The attribute ^superstate has a constant value of nil, ^type has a constant value of state, and ^io has a value of another identifier I1, which in turn has two attributes which also link to more identifiers.

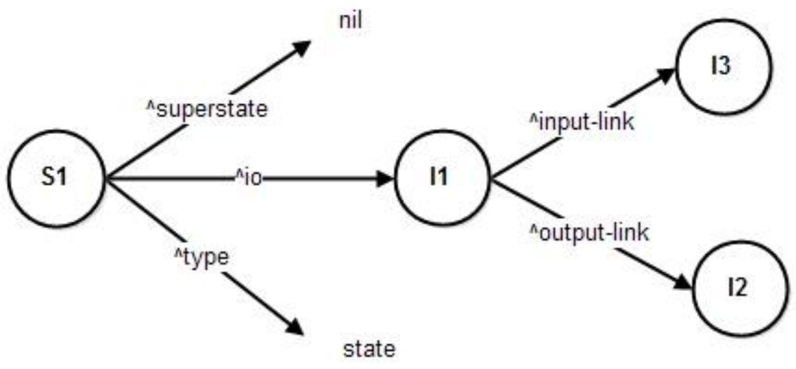


Fig. 1. Soar Initial State.

The decision cycle that Soar goes through is shown in Fig. 2 and consists of an input phase, a state elaboration/propose operators phase, a select operators phase, an apply operator phase, and an output phase. During the input phase, Soar checks to see if any new information has been placed on its input-link through sensors or external environments. The next phase is when Soar elaborates the working memory elements and proposes different operators. Here, the operators are fired and retracted in parallel and the firing of one operator could lead to the firing or retraction of other operators. Once there are no more rules left to fire, the system has reached a state of quiescence and proceeds to the select operators phase. In this phase, Soar uses the rule preferences that it knows to select one operator among a possibility of multiple operators. In the apply operator phase, Soar will execute the selected operator as well as make changes to working memory and in the output phase, put any appropriate information on the output link. The information placed on the output-link can be taken and used by an external environment.

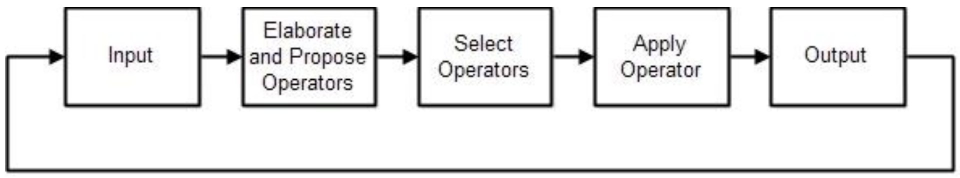


Fig. 2. Soar Decision Cycle.

While working memory represents short term memory, Soar also has the ability to use long-term memory modules to improve its performance in performing a task. The three types of long-term memory modules Soar can currently use are procedural memory, episodic memory, and semantic memory. Procedural memory uses a reward function and reinforcement learning to change how an operator is selected. Semantic memory is where previous knowledge about the workspace is stored, and episodic memory is used to remember past experiences as snapshots of the workspace This allows Soar to move through these snapshots in an attempt to find the best match [34-35].

In order to communicate with various programming languages, Soar uses the Soar Markup Language (SML) which allows for information to be taken off and put on the input and output links very easily. This project communicates with Soar using Python and makes use of the long-term semantic memory module.

4.2.2 Workspace Setup

The experimental setup consists of an EnCore 2s System (MRSI North Billerica, MA) that has been modified to be controlled with a Galil DMC-2413 four-axis digital motion controller. The robot was enclosed in a wooden frame to control the lighting needed for the computer vision modules.

A Matlab R2014a client and a Python server are connected and communicate via TCP/IP protocols. The Python script interfaces with Soar and its semantic memory module utilizing the Soar Markup Language (SML), while Matlab is used to communicate with the positioning, vision, and tooling systems. The linear actuators (Midwest Motion) used for clamping are connected to an Arduino R3 and Arduino Motorshield R3. The Sensor-Equipped (SE) screwdriver is connected to a second Arduino R3, Motorshield R3, and a custom shield designed to interface with the SE screwdriver accelerometer. A Microsoft Kinect was placed above the workspace to make depth measurements and a Microsoft HD-3000 webcam was placed over the workspace to determine screw holes on the laptop. Another Microsoft HD-3000 webcam was placed on the head of the robot and is used to make the fine movement adjustments when the robot is centering a screw. A complete overview of the connections of the different systems can be seen in Fig. 3.

4.2.3 Laptop Selection and Orientation using Soar

A clamping system composed of two linear actuators was used to push the laptop against the walls at the edge of the workspace. By utilizing the potentiometer feedback from the linear actuators, and calibrating its output of the distance from the wall to the edge of the clamp, it was possible to obtain the length and width of the laptop after it has been clamped in the workspace. The average thickness of the laptop was determined using a Microsoft Kinect sensor located above the workspace. From previous work [36], the Kinect was placed approximately 720 mm above the workspace plate. This distance was as close as the Kinect could be mounted to see the entire workspace and not interfere with the overhead camera used to obtain screw coordinates.

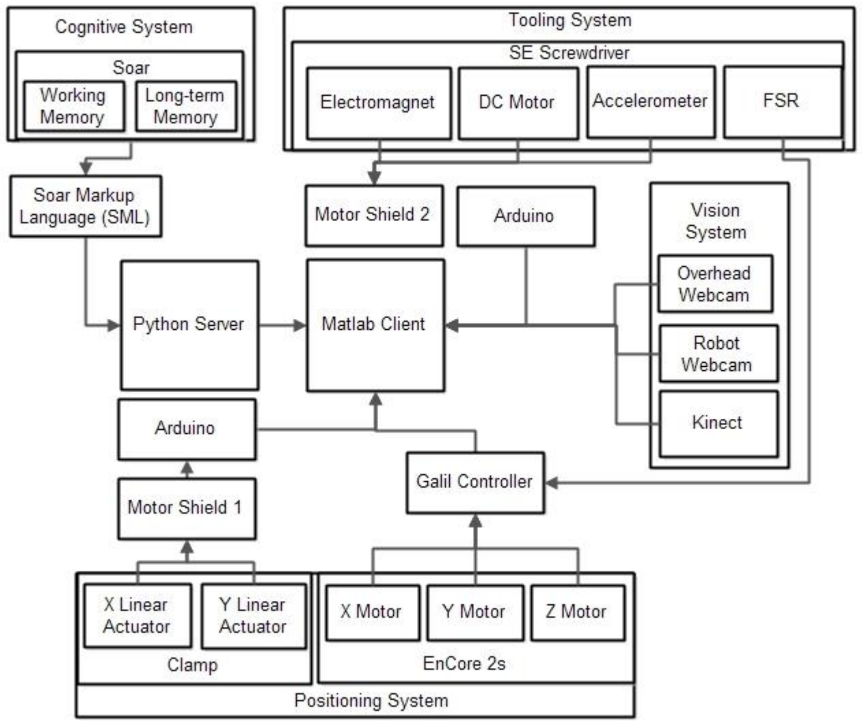


Fig. 3. Overview of the systems connection and communication

Using the Microsoft Kinect, a background image of the workspace was taken with the linear actuators fully retracted. In Fig 4., the region that is enclosed in the box is the region of interest (ROI) that the pixel distances were calculated. This ROI slightly extends past the railing on the left and bottom sides to accommodate laptops taller than the railing. Five readings were taken at each pixel to minimize noise, saved to a CSV file, and averaged using Matlab. Once a laptop was placed on the surface, another measurement was taken with the Kinect. Due to the fact the Kinect gives a measurement with respect to how far an object is from it, the distance will be greater when there is no laptop in the workspace. Since the only change in the ROI is the laptop, these two measurements can be subtracted from each other at a pixel-by-pixel basis.

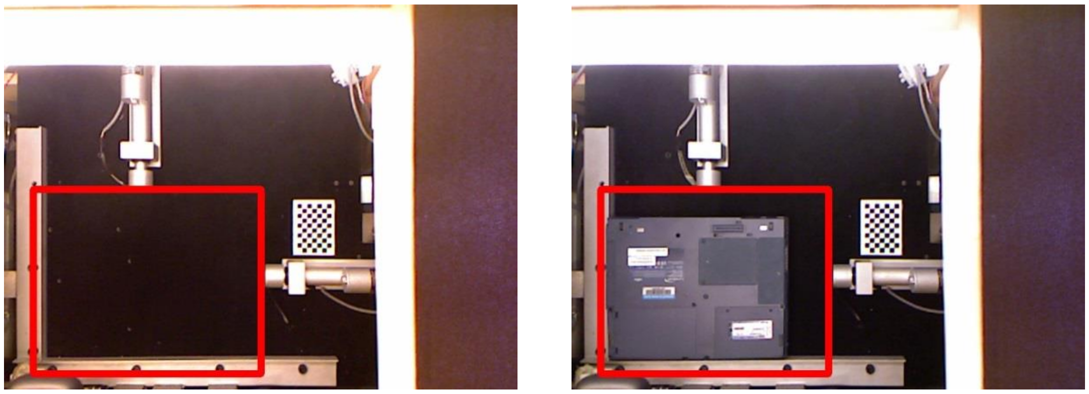


Fig. 4. Background image of workspace without laptop. b) Second image of workspace taken with the laptop.

Algorithm 1 shows how the average thickness of a laptop was obtained. The average thickness of a laptop was used because the cover of laptops can slant, resulting in a non-uniform thickness across the width of the laptop. In this algorithm, the thickness difference is the absolute value of the difference between the background thickness and the laptop thickness. These pixel values are only counted towards the total thickness if they are between 10 and 100 mm. If they are less than 10 mm, they can be disregarded as noise and if they are more than 100 mm, that means one of the pixels had a depth that the Kinect was unable to determine and was 0.

Since the geometries of many laptops are not just rectangular prisms and contain curved features and contours, it is possible that when a laptop is inserted into the workspace at one orientation, different parts of the laptop may become the points of contact between the clamp and the wall than when it is inserted at a different orientation. To account for this, the length, width, and thickness are measured for a laptop when it is in both a “0-degree” and “180-degree” orientation. The current workspace setup only allows the laptop to be placed in these two orientations rather than at four orientations every 90 degrees.

Algorithm 1. Determine Average Thickness of Laptop from Microsoft Kinect

```
Thickness Difference = ABS (background thickness-laptop thickness)
FOR number of pixels in ROI
    IF Thickness Difference > 10 AND Thickness Difference < 100 THEN
        Increment counter by 1
        Add Thickness Difference to the Total Thickness
    ENDIF
ENDFOR
Average Thickness = Total Thickness / Counter
```

Soar's semantic memory can be used to store the different laptop models and the parameters that define these laptops. Soar was trained with the dimensions of eight different laptop models and then used to determine which of the laptop models was placed in the workspace and in which orientation. This was repeated 10 times for each laptop with a random run order; five times, the model was placed in a "0-degree" orientation, and the other five times the model was rotated and placed in a "180-degree" orientation. For each trial, the length, width, and thickness of the laptop was recorded as well as the successful query number, and laptop identifier that Soar determined. Fig. 5 shows the data structure used to store each of the laptops attributes and how it can be extended to hold a m-number of attributes. In Soar, the '@' sign symbolizes a long-term identifier (LTI) indicating that the specified identifier resides in the long-term memory module.

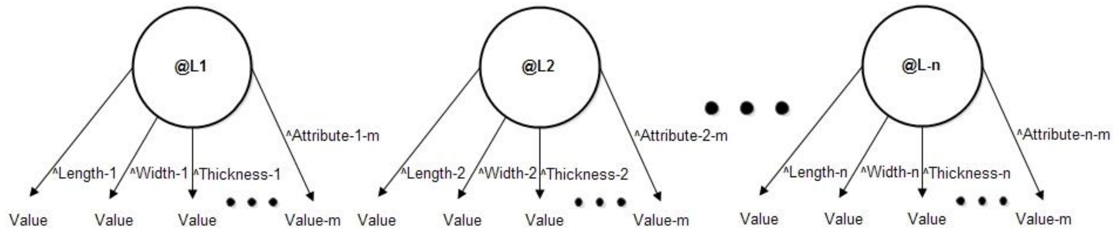


Fig. 5 Soar Semantic Memory

The logic employed by Soar to determine if a laptop is already in the database is shown in Fig. 6. The first step is to obtain the length, width, and thickness of the laptop from the linear actuators and the Kinect. Next, a query is performed on the database and these dimensional values are compared to the laptops "0-degree" attributes to determine if the laptop model is already in the database. To perform a query in Soar with semantic memory, the command "<s> ^smem.command.query.<attribute> <value>" needs to be used (In Soar, the dot (.) notation is a short-hand way of connecting multiple attributes which only link identifiers). If <attribute> is "name", then the command translates to "on the main state conduct a semantic memory query that matches any LTI with any name". This command needs to be augmented with a math-query to restrict the attributes that are being compared to within an upper and lower bound. The command for a math-query is "math-query.<attribute>.<condition> <value>" where <attribute> is the attribute that you are comparing and <condition> can either be greater-or-equal, less-or-equal among a few others which are listed in the Soar manual [37]. Examples of both these commands within the Soar syntax can be found in Fig. 7.

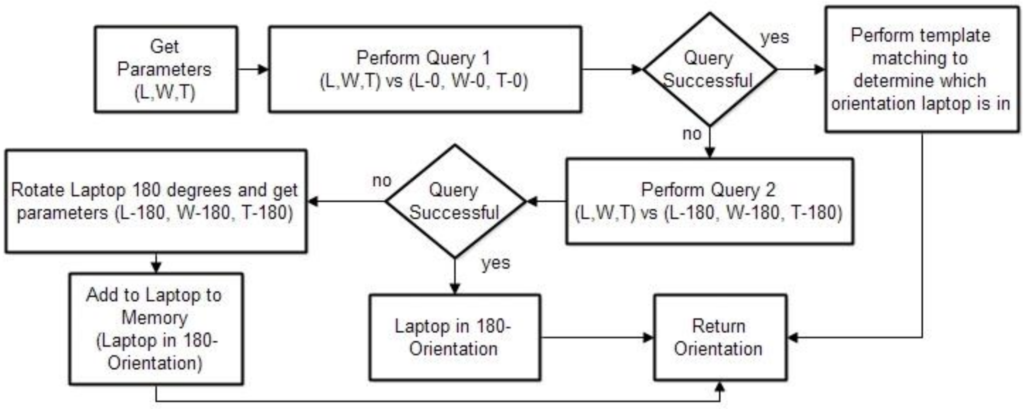

Get Parameters (L,W,T)
Perform Query 1 (L,W,T) vs (L-0, W-0, T-0)
Query Successful
yes
no
Perform template matching to determine which orientation laptop is in
Perform Query 2 (L,W,T) vs (L-180, W-180, T-180)
Query Successful
no
yes
Rotate Laptop 180 degrees and get parameters (L-180, W-180, T-180)
Add to Laptop to Memory (Laptop in 180-Orientation)
Laptop in 180-Orientation
Return Orientation

Fig. 6. Soar logic for retrieving laptop from memory

```
sp {apply-check-database
    (state <s> ^name smemory    ^smem <sm>
                  ^operator <o>    ^memory-process.laptop <laptop>
     (<laptop> ^query <q>)
     (<sm> ^command <cmd>)
     (<o>  ^name  check-database*laptop*q1        ^length  <length-
bounds>
            ^width <width-bounds>   ^thickness <thick-bounds>)
     (<length-bounds>  ^upper  <length-upper>      ^lower  <length-
lower>)
     (<thick-bounds> ^upper <thick-upper>   ^lower <thick-lower>)
     (<width-bounds>  ^upper  <width-upper>         ^lower  <width-
lower>)
-->
     (<laptop> ^query <q> -)
     (<laptop> ^query (+ 1 <q>))
     (<cmd> ^math-query <mq>    ^query.name <any-name>)
     (<mq> ^length <mql>   ^width <mqw>   ^thickness <mqt>)
     (<mqt>   ^less-or-equal   <thick-upper>          ^greater-or-equal
<thick-lower>)
     (<mqw>   ^less-or-equal   <width-upper>        ^greater-or-equal
<width-lower>)
     (<mql>   ^less-or-equal   <length-upper>         ^greater-or-equal
<length-lower>)}
```

Fig. 7. Example Soar code that checks the data base for laptop

If the first query is successful, an image template matching algorithm must be performed since it is not possible to tell what orientation the laptop is in. If the query is unsuccessful, then the database is queried again, and compared to the laptop's attributes in the '180-degree' orientation. If the second query is successful, then the laptop is in the '180-degree' orientation. If the query fails, then the laptop needs to be added to the database, so the clamps will unclamp and the laptop will need to be rotated to obtain the dimensional parameters of the other orientation.

When a laptop is added to memory, a template image must be created for the laptop in both orientations. The steps taken to find and create the template image are shown in Fig. 8. Fig. 8a) shows the result once the image with the laptop in the workspace is subtracted from the image without the laptop in the workspace. In these images, the only change is the laptop so when the images are subtracted from each other, only the laptop is left and the rest of the image is black. Next, Fig. 8b) shows the same image after a Prewitt edge detection filter is performed, and Fig. 8c) shows the image after the lines have been dilated. Finally, in Fig. 8d), the contours are closed over the laptops area, and the size and location of the laptop in the image is determined. Since the location of the laptop is now known, the laptop's image can be extracted from the main image and saved as a template for both orientations of the laptop. These template file names are added to semantic memory as attributes of that laptop.

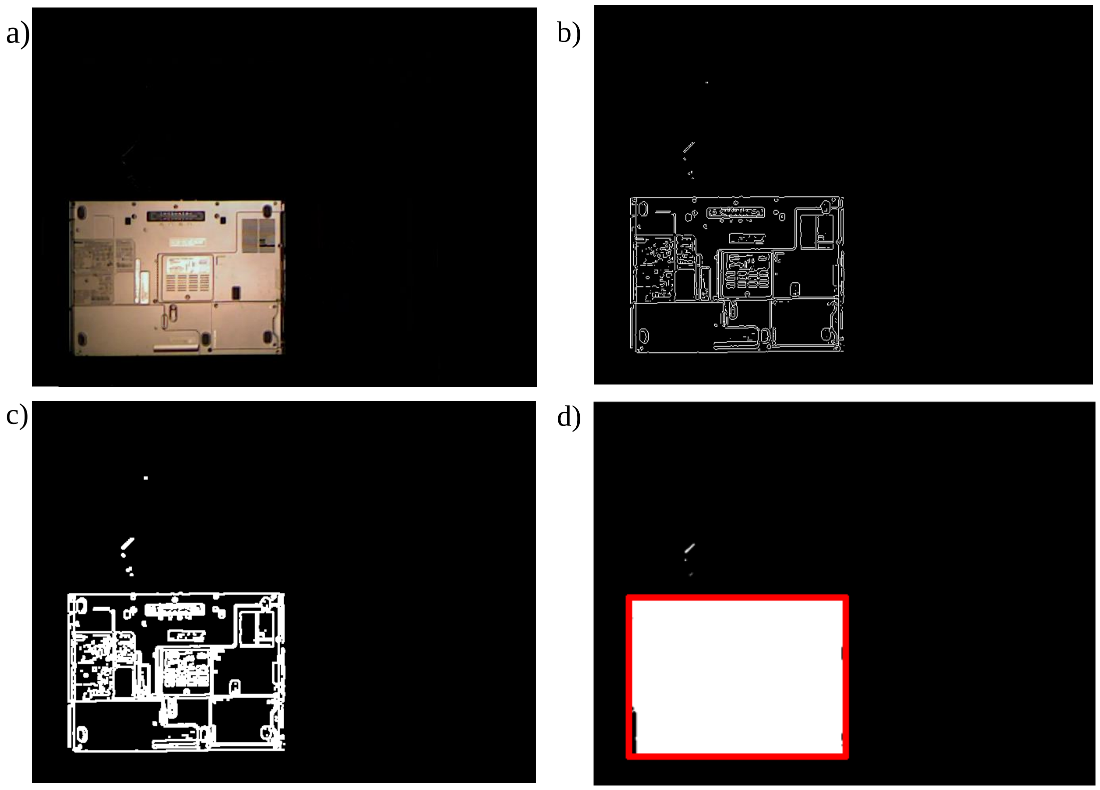


Fig. 8. Creating a template from the laptop placed in the workspace. a) Background subtraction of the image with a laptop and the image without a laptop. b) After performing edge detection using the Prewitt method. c) Dilation of the lines found with edge detection. d) filling in contours and determining the region of the image containing the laptop

The template matching algorithm used is the sum of absolute difference (SAD) shown in (1) [38]. In this equation, let I1 be the template image, I2 be the target image, and x and y be the size of the template image. In SAD, the template is moved over the target image, the pixel values are subtracted pixel-by-pixel, and the sum of these values give the score for the current region. The best match therefore becomes the region with the lowest score.

$$SAD(x,y) = \sum_{i=0}^{m_{rows}} \sum_{j=0}^{n_{cols}} |I_1(i,j) - I_2(x+i, y+j)| \tag{1}$$

4.2.4 Laptop Hole Removal Using Soar

Soar’s semantic memory was also used to evaluate how well the systems performance could improve while removing screws from the backside of a laptop. Previously [10], every time the system ran a test on a laptop, every circle location obtained from the overhead camera had to be explored to determine if a screw was present or not. Using semantic memory will cut down on the trial times as circles that have had their location already explored will not have to be explored again. Soar will be able to ignore circles that do not have a screw in them and only focus on the circles that have not yet been explored or have had screws found.

To test how Soar improves the robot’s performance in removing screws from laptops, the disassembly program was run on a laptop until the length of a test converged. The testing was performed with two methods. The first method was when circles were identified with the overhead camera during every trial. The second method was when circles were only identified during the first training run, and the locations of the screws were retrieved from memory after the first run.

The eight laptop models were placed in the workspace a total of 10 times each. For five of the trials the laptop was placed in the “0-degree” configuration and for the other five trials, the laptop was placed in the “180-degree” configuration. For all of these trials, circles were identified with the overhead camera, and the length of a test as well as the number of screws that were correctly determined and removed were recorded. An additional three trials were performed where Soar did not identify circles from the overhead camera and just removed screws by retrieving their locations from semantic memory.

Due to how semantic memory is currently constructed within the Soar framework, it is not possible to use dot notation to explore levels that are not directly under the LTI as it leads to increasingly complex queries. Instead, this work proposes that each hole can be treated as its own LTI and can be linked to a specific laptop by having its own attribute that consists of the laptops ^name value. By linking the LTIs of the laptop model and the holes in the laptop via the attribute ^name when queried, all subsequent queries will only select LTIs that pertain to the laptop in the workspace. This is shown in Fig. 9 where the laptops have LTIs of @L1 and @L2 and the holes have LTIs of @H1-@H5. The LTI @L1 has an attribute ^name "laptop1". When this laptop is chosen as the laptop that is in the workspace, only the holes that also have an attribute ^name laptop1 will be returned. In Fig. 9, LTIs that are linked between the holes and laptops are visually shown with the same shade in their identifier.

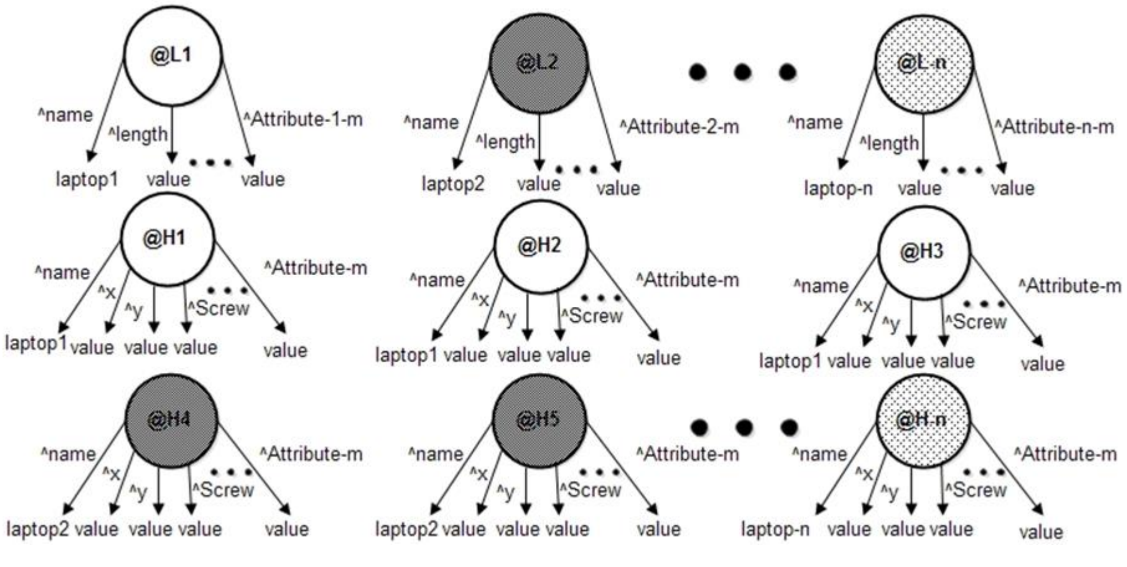


Fig. 9. Semantic memory incorporating circles

It also becomes easier to bring the LTI elements into working memory and compare their contents in short term memory rather than continually querying and comparing with semantic memory. The general logic flow used with Soar is shown in Fig. 10. When the program is started, the database will be checked for a laptop. While the details of checking for a laptop are shown in Fig. 6, the result is the laptop is already in the database, or the laptop is not in the database and needs to be added. Next, the locations of all the circles that were found by the overhead camera in Matlab, as well as the current position of the end effector, are put on the input-link and transferred to Soar's working memory. Then all the holes in Soar's long term memory are queried and brought to working memory one-by-one until the query fails. When the query fails, it means that there are no more holes in semantic memory left to bring to working memory. If laptop is not already in the database, the failure will occur after the first attempt. Next, a circle that was from the input link is picked and its x,y coordinates are compared to the upper and lower bounds of all of the holes in memory. In case of multiple matches, preference is given to circles that are closer to the current position of the robot, and circles that contain a screw. Once a hole has been chosen there are three possibilities:

1) **The circle location is not already in the database and must be explored-** When a circle is explored it uses the same procedure as described in [88] with minor variations in order to speed up segments of a test. First, before the initial movement adjustment, the image used to center a screw is slightly smaller (350 x 400 px compared to 600 x 800 px). If the robot makes a movement adjustment, the image becomes even smaller (300 x 300 px) as the screw should be close to the center of the webcam. Since the image is now smaller, the algorithm no longer attempts to close contours, but does attempt to locate a hole by toggling the localized lighting

module; First by checking the hole with the light on and then with the light off. The different lighting methods combined with both the laptop color and the depth of the screw holes can result in higher contrast levels. If a screw is removed, the program remembers the computer vision constraints used to find the hole and the coordinates at which the hole is actually at. The program also transposes the coordinates of the hole so it can be found on the laptop when the laptop is placed in the other orientation. After a screw has been centered, the SE screwdriver is used to check the area for a screw. Attributes are added to the LTI of the hole to save the information determined by the computer vision, robot, and screwdriver and is placed in Soar's semantic memory.

2) **The circle is already in the database and contains no screw or is not a screw hole**- When the circle from the input-link matches the circle in working memory and there is no screw, then the robot is told to wait and Soar proceeds to its next decision cycle.
3) **The circle is already in the database and contains a screw**- When the circle from the input-link matches the circle in working memory and there is a screw at that location, then the robot is moved to the x,y location and attempts to center and remove the screw. To speed up the computer vision while it is locating and centering a screw, the ROI is reduced (300 x 300 px) and the computer vision parameters it uses are retrieved from the hole's semantic memory attributes.

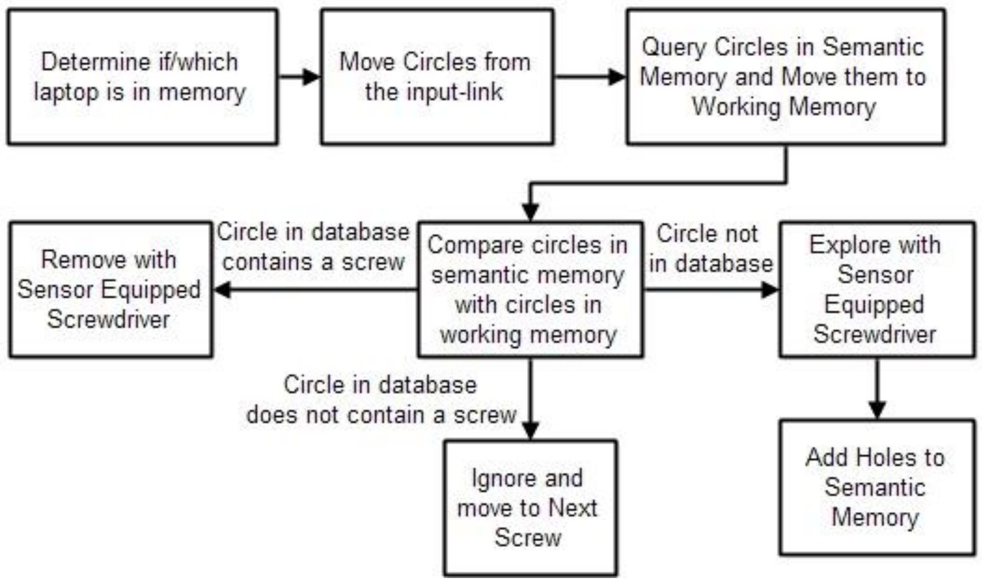

Determine if/which laptop is in memory
Move Circles from the input-link
Query Circles in Semantic Memory and Move them to Working Memory
Remove with Sensor Equipped Screwdriver
Circle in database contains a screw
Compare circles in semantic memory with circles in working memory
Circle not in database
Explore with Sensor Equipped Screwdriver
Circle in database does not contain a screw
Ignore and move to Next Screw
Add Holes to Semantic Memory

Fig. 10. Logic of using Soar to determine if a hole has a screw.

When Soar decides to explore the circle location with the SE screwdriver, it is assigning a reward based on if the screw is found or not. This model assumes that a score of a 1.0 means a screw is present and a 0.0 means a screw is not present and each hole starts with a score of 0.5. If the screw is not found, then that location is marked with a ‘no’ when the LTI is modified (-0.5). However, if a screw is found, then that location is marked with a ‘yes’ (+0.5).

4.3 Results and Discussion

4.3.1 Laptop Thickness

The average thicknesses of the laptops, as calculated by the Kinect sensor, are shown in Fig. 11. The triangle that is pointed upward shows the maximum thickness that was measured on laptop and the triangle that is pointed downward shows the minimum thickness measured on the laptop. The thickness that was computed by the Kinect is shown as a circle with error bars indicating one standard deviation. The results show that the Kinect can compute an average thickness that falls between the minimum and maximum thickness. The measurements taken by the Kinect are also extremely repeatable as the standard deviation of the thickness is low with the highest standard deviation being 0.324 mm for Laptop 3. Laptop 4 was equipped with an extended life battery explaining why the maximum thickness of that laptop is so high. The average thickness is much closer to the minimum thickness of the laptop because the extended battery only covered a small area of the back of the laptop.

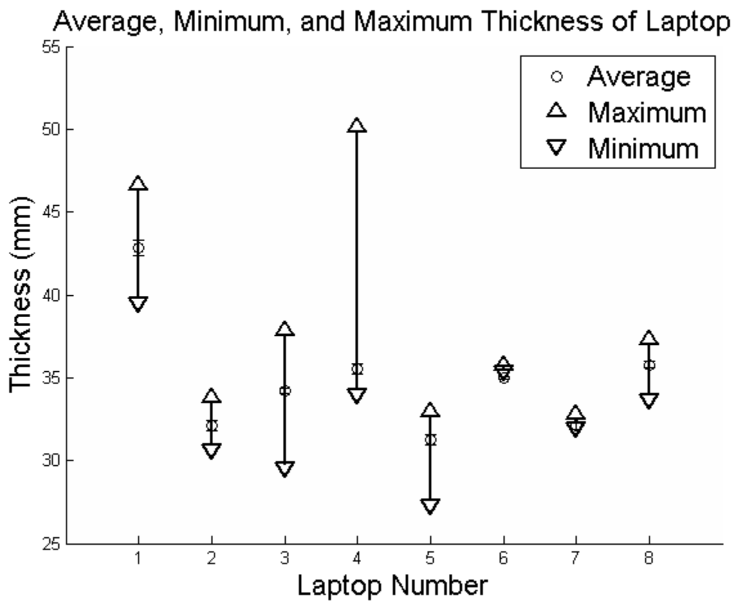


Fig. 11. Average Thickness of a Laptop

The results when Soar was used to determine the orientation and model of a laptop are shown in Table 1. The ‘Orientation’ column in Table 1 identifies whether the laptop was placed in a ‘0-degree’ or a ‘180-degree’ orientation. The column ‘Query’ displays whether the first or second query was the successful query when the laptop was compared to values in semantic memory. If query 1 was the successful query, the template matching procedure had to be implemented since the program would be unable to determine what orientation the laptop was in. The highest standard deviation (std) of length, width, and thickness is 0.321 mm thus, four stds (99.9% of population) is 1.3 mm. This can be seen with laptop models 3, 7, and 8, as all these parameters are less than 1.3 mm of the corresponding parameter in the other orientation. For all the other laptop models that were determined by query 1 when the laptop was placed in the “0-degree” orientation, and query 2 when the laptop was placed in the “180-degree” configuration, at least one of their parameters was higher than 1.3 mm of the corresponding parameter in the other orientation. However, if query 2 was the successful query, then by the process of elimination, the laptop is in the “180-degree” configuration. In this experiment, Soar could determine the correct laptop and orientation 80 out of 80 times. The “Length”, “Width”, and “Thickness” columns are the experimental results obtained from the two linear actuators, and the Kinect sensor. The ± values in these columns represent one std of the measured quantity. All measurements have low standard deviations demonstrating that the measurement methods are repeatable.

4.3.2 Holes

The number of trials it takes for the time of a disassembly routine to converge is shown in Fig. 12. The first disassembly trial for a laptop always takes the longest amount of time because all the circles located by the overhead camera need to be explored. For all the different runs, by the second trial, the trial time has decreased by ~60% or more. The system performs better when the circles are only explored during the initial training trial rather than every trial. With this method, the system reaches its best performance time during the second trial and the test times stay consistent throughout all the trials.

Table 1. Results for the Laptop Selection Test

Laptop	Orientation (degree)	Query	Success Rate (%)	Length (mm)	Width (mm)	Thickness (mm)
1	0	1	100	307.48 ± 0.012	266.11 ± 0.036	43.06 ± 0.157
1	180	2	100	307.32 ± 0.019	261.13 ± 0.228	43.46 ± 0.045
2	0	1	100	332.41 ± 0.010	274.11 ± 0.006	31.60 ± 0.047
2	180	2	100	332.43 ± 0.038	270.23 ± 0.321	31.87 ± 0.094
3	0	1	100	354.80 ± 0.010	254.12 ± 0.012	34.04 ± 0.137
3	180	1	100	355.16 ± 0.010	253.86 ± 0.063	33.42 ± 0.027
4	0	1	100	343.39 ± 0.022	279.80 ± 0.207	35.24 ± 0.010
4	180	2	100	343.26 ± 0.004	283.38 ± 0.022	35.32 ± 0.009
5	0	1	100	290.72 ± 0.086	242.51 ± 0.016	30.06 ± 0.154
5	180	2	100	289.93 ± 0.128	245.16 ± 0.021	30.32 ± 0.041
6	0	1	100	359.49 ± 0.074	264.43 ± 0.031	35.45 ± 0.032
6	180	2	100	359.07 ± 0.020	261.75 ± 0.013	35.52 ± 0.139
7	0	1	100	292.65 ± 0.124	234.74 ± 0.021	32.15 ± 0.196
7	180	1	100	292.86 ± 0.006	234.91 ± 0.007	32.11 ± 0.222
8	0	1	100	356.11 ± 0.024	262.36 ± 0.022	34.69 ± 0.221
8	180	1	100	356.41 ± 0.062	262.04 ± 0.011	35.07 ± 0.171

If the circles are located and explored during every trial, then a comparable trial time doesn't start to occur until at least the third trial. Fig.13 shows the visual results of the circles that the system explored for the 2nd run shown in Fig 12. with the '○' data marker. In Fig. 13, a '△' means the computer vision system was unable to determine a screw, a 'X' means the SE screwdriver was unable to find and remove a screw, and a green circle means that the screw was successfully removed. As the number of trials progress, the number of circles that are explored continue to decrease until none or only a couple of circles are explored during the trial.

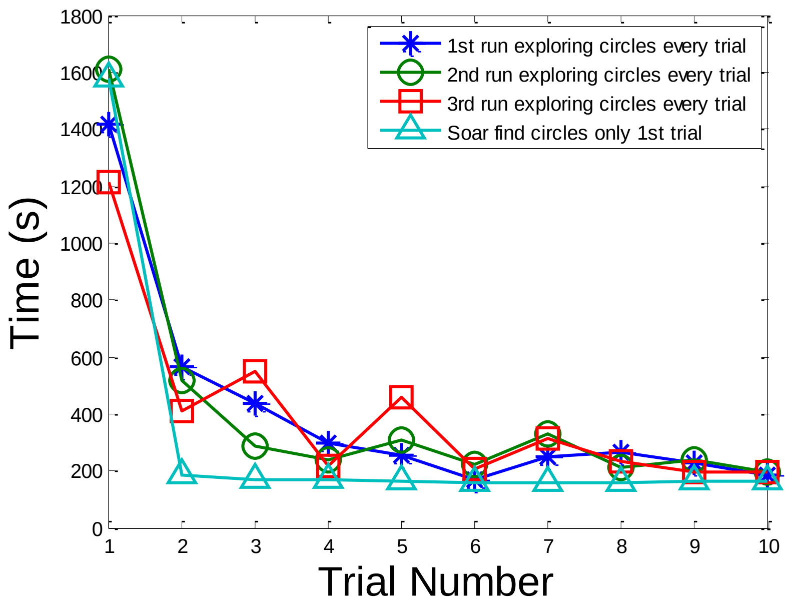


Fig. 12. Number of trials it takes for the testing time to remove screws from a laptop using Soar to converge.

The results showing the percentage of correct holes found by the overhead camera and the percentage of screws that were successfully removed from the laptop by the SE screwdriver are shown in Table 2. These results are an improvement over the previous findings [10] as the parameters governing the Hough circle transform such as the radius size of a circle were relaxed. The results are broken up based on the orientation the laptop was placed in the workspace. The column '*New Circles Explored by the Robot*', lists the number of new circles that the robot went and explored with the robot camera and SE screwdriver. As the trial number increased for each laptop in every orientation, the number of holes that needed to be explored decreased, until a trial would yield a few holes that needed to be explored. The screws that had been previously found and saved in semantic memory were still removed without needing to be explored. The column '*Possible Screws that could be removed*' lists how many screws could be found after adjusting for if a screw had been missed and that circle had incorrectly been placed in memory as not containing a screw. The column '*Holes Found*' lists the percentage of the screw holes that were successfully found by the computer vision system based on the total number of screw holes that could possibly be found for that trial. Finally, the column '*Screws removed*' indicates how many of the screws were successfully removed based on how many holes containing screws were found by the computer vision system.

Entries that are italicized indicate that a screw that was on the laptop was missed the first time a screw hole was discovered. This would occur by either the vision system not being able to determine the screw hole or the SE screwdriver hitting an obstruction on the laptop or the workspace and not being able to remove a screw. The number in the parenthesis denotes the number of screws that were missed and had their positions incorrectly placed in semantic memory as not containing a screw. The column the parentheses are in indicates whether the error occurred with the computer vision system ('*Holes Found*') or the SE screwdriver ('*Screws Removed*'). This is also reflected in the '*Possible screws that could be removed'* column. For example, with laptop 2, there are 14 circles that contain screws and during the first trial, two of these holes were not found by the computer vision system and placed in memory as not containing a screw. During the next trial, since those two circles were in memory as not containing a screw, there were only 12 possible screws that the system would be able to remove. Adding up the errors in both orientations, six errors occurred via the vision system where the robot camera was unable to find the screw and six errors occurred via the SE screwdriver. Of the six errors with the SE screwdriver, four occurred because the screwdriver was not able to fully mate with the screw because of a collision with a feature on the laptop or with the clamp.

Due to the setup of the tools on the robot, the entire laptop cannot be reached by the robot when it is placed in the workspace. The reachable region of the workspace is shown as the rectangle that is on the laptops in Fig. 13. This means that the entire laptop is not explored when the laptop is introduced in its initial 180-degree orientation. This results in a higher number of circles that need to be explored during the first trial for both orientations of the laptop since when the laptop is turned around there is a portion of the laptop that is being analyzed for the first time.

The results of the tests broken up by different testing time categories are shown in Fig. 14 and Fig 15. The main testing time categories are: the parameter time, computer vision time, robot movement time, and the screwdriver removal time. The parameter time for these tests was defined as the amount of time it takes the system to clamp the laptop in the workspace, take measurements of the length, width, and thickness, and find circles with the overhead camera. The parameter time always took the most time when a laptop was placed in the workspace and was not in memory. When a laptop needed to be added to memory, the laptop would then have to be unclamped, rotated, and re-clamped. These trials are always the first trial when the laptop was placed in the '180-degree' orientation. When a laptop was placed in the '0-degree' orientation and a test was run the parameter time is always less, even during the first trial, because the laptop had already been added to memory.

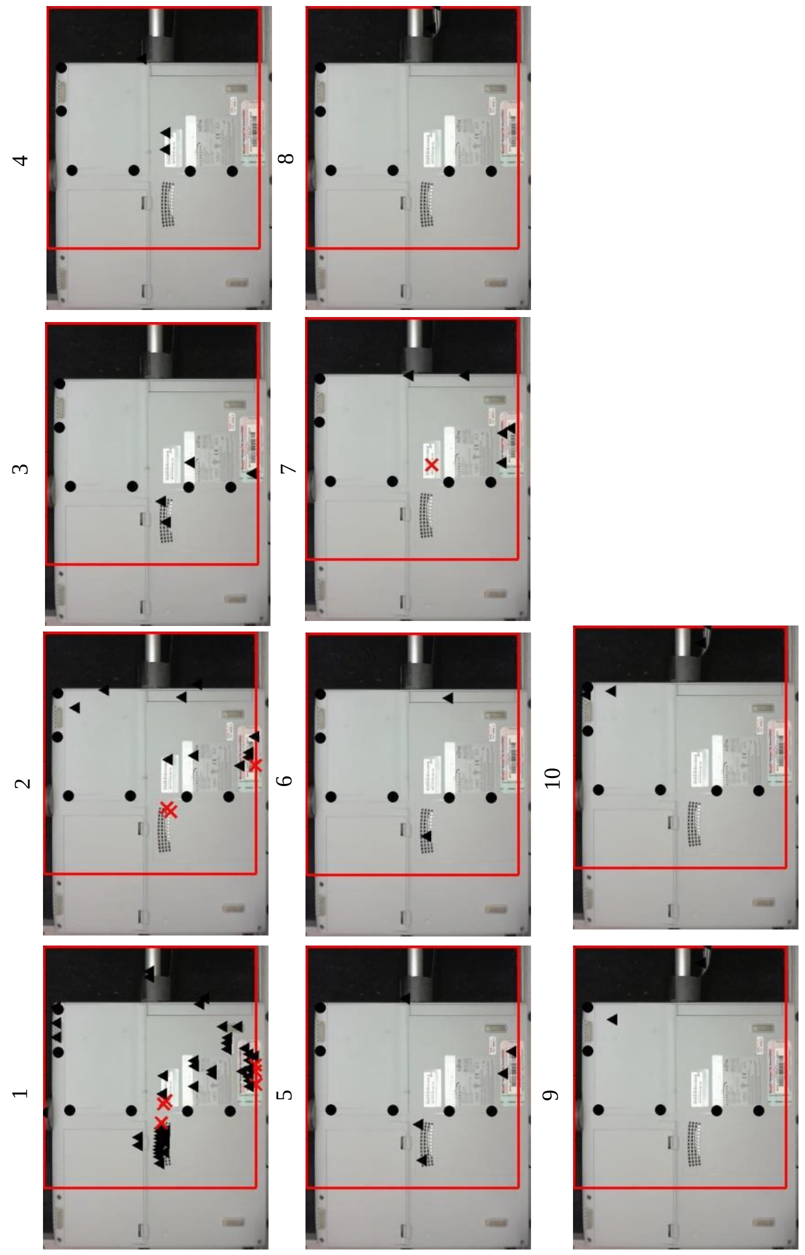


Fig. 13. Visual results for the convergence test for laptop model 1. The numbers over the laptop indicate the corresponding trial number in Fig 12. (2nd run). The '●' indicates that a screw was found and successfully removed, a 'X' indicates the screwdriver did not find a screw, and the '▲' indicates that the computer vision did not find a screw.

Table 2. Results on how well the system could find circles and remove screws from the laptop.

Laptop	**180-degree orientation**				
	Trial	New Circles Explored by Robot	Possible Screws that could be removed	Holes Found By Vision (%)	Screws Removed (%)
1	1	46	6	100	100
	2	1	6	100	100
	3	1	6	100	100
	4	0	6	100	100
	5	0	6	100	100
2	1	61	14	85.7 (2)	100
	2	12	12	*91.7*	100
	3	4	12	*100*	100
	4	2	12	*100*	91.7
	5	2	12	*91.7*	90.9
3	1	89	13	92.3	100
	2	10	13	84.6	100
	3	2	13	84.6	100
	4	9	13	92.3	91.7
	5	4	13	92.3	100
4	1	60	15	93.3	100
	2	1	15	93.3	92.9
	3	1	15	93.3	100
	4	0	15	93.3	100
	5	1	15	93.3	100
5	1	41	9	100	100
	2	20	9	100	88.9
	3	5	9	100	100
	4	4	9	88.9	88.9
	5	4	9	88.9	100
6	1	66	11	100	100
	2	5	11	100	90.9
	3	7	11	100	100
	4	3	11	100	100
	5	4	11	100	100
7	1	34	11	63.6 (3)	100
	2	7	8	*87.5*	100
	3	2	8	*87.5*	100
	4	1	8	*87.5*	100
	5	2	8	*87.5*	100
8	1	60	16	100	100
	2	4	16	100	100
	3	0	16	93.8	100
	4	0	16	93.8	100
	5	0	16	93.8	100

Laptop	**0-degree orientation**				
	Trial	New Circles Explored by Robot	Possible Screws that could be removed	Holes Found By Vision (%)	Screws Removed (%)
1	1	35	7	100	100
	2	11	7	100	100
	3	3	7	100	100
	4	2	7	100	100
	5	1	7	100	100
2	1	47	12	91.7 (1)	100
	2	5	11	*90.9*	100
	3	2	11	*90.9*	90.9
	4	5	11	*90.9*	100
	5	2	11	*100*	100
3	1	86	16	75	85.7 (2)
	2	15	14	85.7	*100*
	3	16	14	78.6	*100*
	4	5	14	71.4	*100*
	5	2	14	92.9	*100*
4	1	29	19	94.7	100
	2	2	19	94.7	100
	3	0	19	94.7	100
	4	2	19	94.7	100
	5	3	19	94.7	100
5	1	52	8	100	87.5 (1)
	2	5	7	100	*100*
	3	4	7	100	*100*
	4	3	7	100	*100*
	5	2	7	100	*100*
6	1	38	11	100	90.9 (1)
	2	14	10	100	*100*
	3	5	10	90	*100*
	4	2	10	100	*100*
	5	1	10	90	*100*
7	1	22	7	85.7	83.3 (1)
	2	6	6	*83.3*	*100*
	3	0	6	*100*	*100*
	4	0	6	*100*	*100*
	5	2	6	*83.3*	*100*
8	1	34	14	85.7	92.3 (1)
	2	3	13	*92.3*	*100*
	3	1	13	*92.3*	*100*
	4	0	13	*92.3*	*100*
	5	1	13	*92.3*	*100*

The robot movement time was the time it took the robot to move from circle to circle and the computer vision time was the amount of time that that system spent trying to identify and centering screw holes. Both the robot movement time and the computer vision time were the greatest in the first trial because the system had to explore every hole. The correlation between these two time variables has an r-score > 0.97 for every laptop model indicating there is a strong positive correlation. The screw removal time was the length of time the disassembly sequence was actively trying to remove the screw with the SE screwdriver. The other time was the amount of time that the program either spent in Soar trying to determine the next hole to remove, or time spent waiting for a response between Matlab and the hardware during a communication timeout. When the computer vision system identified a large cluster of holes, or when there were many holes in the database, the computational time Soar required to make a hole selection increased.

Fig. 14 shows the results for when circles were found and explored every trial, and Fig. 15 shows the results when the circles were only explored during the initial trial. Fig. 15 shows improvement over the trials in Fig. 14 as the time of a trial is faster and more consistent. When circles were only found during the initial trial, the computer vision time is only the amount of time that the system spends trying to center a screw while the computer vision time for Fig. 14 is spent both exploring and centering screws. The results in Table 3 show the performance between the two methods used to remove the screws from laptops. In this table the method where circles are located during every trial is referred to as '*Method 1*' and the method where circles are only found during the second trial is referred to as '*Method 2'.* For every laptop model, finding circles with '*Method 2'* decreased the overall time of a trial, and for 15 out of 16 laptops and orientations decreased the trial run time by over 10%. The ± values in these columns represent one std of the measured quantity and shows that '*Method 2*' is much more repeatable than '*Method 1*' due to the fact the time of '*Method 1*' can greatly increase by exploring new circles. The column '*Difference*' shows that that for some laptops, it is possible for '*Method 1*' to save over a minute of testing time during a trial. The average time that was reported for each orientation and method was calculated by averaging the time of the trials with five or less new circles explored from Table 2.

Using the results of trials where the circles were only explored for the first trial (Method 2), data revealing the best run conditions could be calculated and is displayed in Table 4. All this data is broken up by the orientation that the laptop was placed in the workspace. The column '*Average screw removal time*' is the average disassembly trial time for three runs after the training run. The column '*Number of Screws Removed*' lists the most number of screws that were removed from the laptop model during a trial. The column labeled '*Screw Removal Time Limit*' is the fastest time that the laptop could be disassembled while being adjusted for the screws that the system missed. To calculate this limit, the average parameter time was used for each model of laptop. This accounts for the fact that laptops could have different dimensions and so the clamping time could vary. The robot movement time was calculated by measuring the distance between holes that were closest to each other on the back of the laptop, for all the holes in the reachable workspace. Additional distances that the laptop moved such as switching from the robot camera to the SE screwdriver as well as moving up and down were also added in to the total distance. This total distance was divided by the speed the robot was traveling at to obtain the optimal robot movement time.

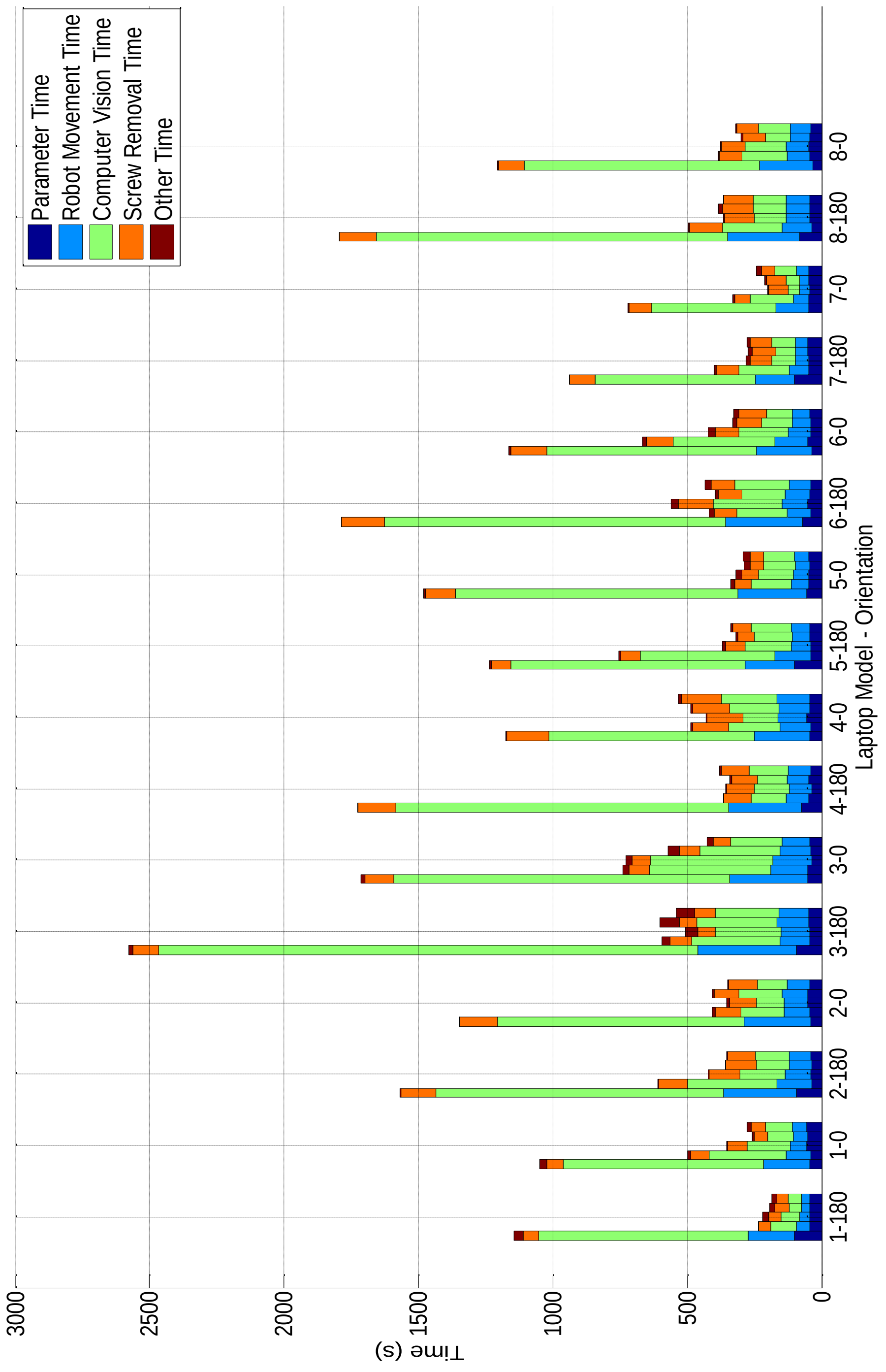


Fig. 14. Results showing the length of a trial broken up by category when circles were found and explored during every trial.

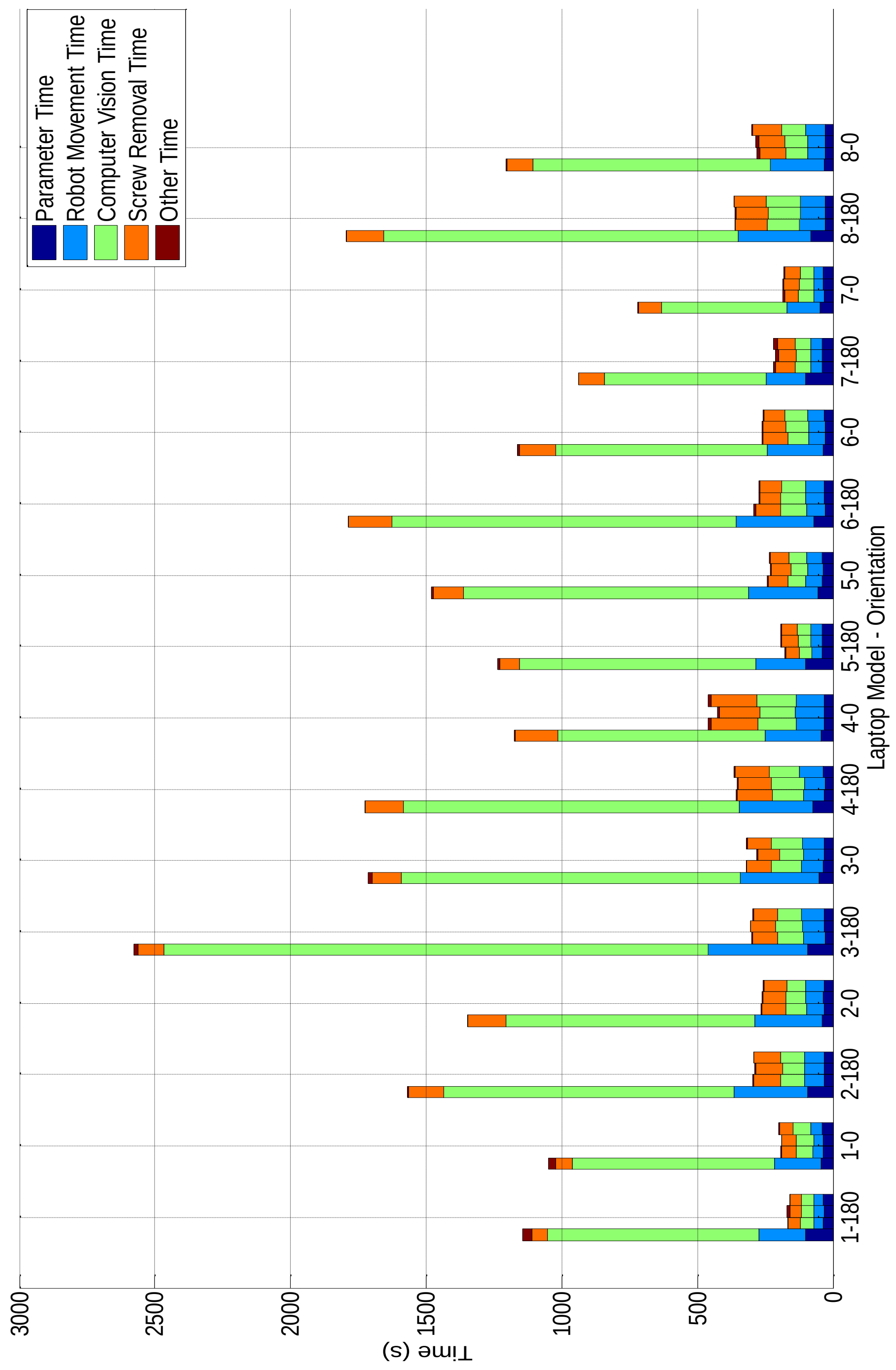


Fig. 15. Results showing the length of a trial broken up by category when circles were only found and explored during the first trial.

Table 3. Results showing improvement from when circles were found and explored during every trial (method 1) and when circles were only found during 1st trial (Method 2).

180-degree orientation				
Laptop Model	**Method 1 Average (s)**	**Method 2 Average (s)**	**Difference (s)**	**Percent Decrease**
1	208.17 ± 23.20	164.33 ± 5.79	43.84	21.06
2	377.30 ± 37.80	289.28 ± 5.10	88.02	23.33
3	554.60 ± 76.99	296.96 ± 4.54	257.64	46.46
4	361.06 ± 19.72	346.29 ± 8.63	14.76	4.09
5	328.69 ± 15.91	230.86 ± 4.50	97.83	29.76
6	414.85 ± 26.80	275.60 ±14.61	139.25	33.57
7	276.26 ± 3.88	216.63 ± 5.34	59.63	21.58
8	402.31 ± 62.03	360.97 ± 2.52	41.34	10.27
0-degree orientation				
Laptop Model	**Method 1 Average (s)**	**Method 2 Average (s)**	**Difference (s)**	**Percent Decrease**
1	296.60 ± 51.58	199.39 ± 1.10	97.21	32.77
2	350.48 ± 1.72	257.19 ± 3.50	93.29	26.62
3	478.13 ± 72.99	302.34 ± 23.50	175.79	36.77
4	483.56 ± 41.03	426.04 ± 12.27	57.52	11.90
5	299.49 ± 16.21	187.77 ± 8.23	111.72	37.30
6	329.22 ± 3.39	257.60 ± 4.00	71.62	21.75
7	219.45 ± 20.64	184.45 ± 3.31	34.99	15.95
8	345.51 ± 41.42	280.4 ± 14.19	65.12	18.88

The fastest computer vision time that was produced by a trial was 6.5 seconds per hole. This time was used for all laptops and multiplied by how many holes were in the reachable workspace to get an optimal computer vision time. Finally, for every laptop, the fastest screwdriver removal time was used which was between 7.0 and 8.2 seconds. The screw removal times for individual laptops were used because the screws used on the different laptops could vary in type and size.

The backs of different laptop models have different number of screws on them so it is tough to compare the total disassembly times of laptop models. However, if the disassembly time is normalized to a ‘per screw’ basis, it is possible to compare how well the system could remove the screws from the different laptop models. The column '*Screwdriver Removal Time per Screw*' shows the average time the SE screwdriver was trying to remove an individual screw per test, and for all the tests, this time was between 6.95 and 9.56 seconds per screw. The column '*Average Screw Removal Time per Screw*', accounts for the robot movement and computer vision time as well as the screw removal time. These time categories are added together and divided by the number of screws in the reachable workspace. This means the average time to travel to, center, and remove a screw takes between 22.78 and 30.95 seconds per screw for the different laptops. For all columns, the ± represent one std of the measured quantity and show that Method 1 is very repeatable on a 'per screw' basis. Ideally, the '*Average Screw Removal Time*' divided by the '*Number of Screws*' should match the time in the column '*Average Screw Removal Time per Screw*'. If these two times do not match, then it means the computer vision system or the screwdriver was unsuccessful in removing a screw in at least one of the trials with that laptop model.

4.4 Conclusions

This paper presented the results of incorporating the cognitive architecture Soar, to an automated system capable of determining the locations of screw holes on a laptop. The locations of these screw holes were not preprogrammed and were found by combining force and vision sensing. The semantic memory of Soar was used to remember the location of circles that contained screws and the locations that did not contain screws.

Table 4. Results for removing screws when the circles were only explored for the first trial.

180-degree orientation					
Laptop Model	Average Screw Removal Time (s)	Number of Screws Removed	Screw Removal Time Limit (s)	Screwdriver Removal Time per Screw (s)	Average Screw Removal Time per Screw (s)
1	164.33 ± 5.79	6	144.6	7.15 ± 0.31	27.39 ± 0.97
2	289.28 ± 5.10	12	253.1	8.20 ± 0.04	24.11 ± 0.43
3	296.96 ± 4.54	12	255.9	7.38 ± 0.71	24.21 ± 2.14
4	346.29 ± 8.63	14	303.8	9.05 ± 0.20	24.74 ± 0.62
5	230.86 ± 4.50	9	209.2	7.94 ± 0.12	25.65 ± 0.50
6	275.60 ±14.61	11	222.5	7.95 ± 0.82	26.54 ± 0.56
7	216.63 ± 5.34	7	179.2	9.56 ± 0.52	30.95 ± 0.76
8	360.97 ± 2.52	16	322.3	7.34 ± 0.04	22.78 ± 0.16
0-degree orientation					
Laptop Model	Average Screw Removal Time (s)	Number of Screws Removed	Screw Removal Time Limit (s)	Screwdriver Removal Time per Screw (s)	Average Screw Removal Time per Screw (s)
1	199.39 ± 1.10	7	170.7	7.61 ± 0.08	28.48 ± 0.16
2	257.19 ± 3.50	10	238.6	7.62 ± 0.24	25.72 ± 0.35
3	302.34 ± 23.50	13	259.3	6.95 ± 0.25	24.54 ± 0.44
4	426.04 ± 12.27	18	371.2	8.98 ± 0.67	23.67 ± 0.68
5	187.77 ± 8.23	7	173.4	8.49 ± 0.20	26.82 ±1.17
6	257.60 ± 4.00	10	233.2	8.30 ± 0.69	25.76 ± 0.40
7	184.45 ± 3.31	6	155.4	9.26 ± 0.51	30.84 ± 0.55
8	280.4 ± 14.19	12	252.8	8.84 ± 0.82	24.80 ± 0.16

Overall, the results show that it is possible to integrate a cognitive architecture such as Soar to aid in a disassembly sequence. Soar is a good cognitive system to choose for work like this as it has a good framework in place to easily accept input from sensors and different languages and give output commands. Its long-term memory module, semantic memory, is well suited for remembering information about specific LTI such as laptop models and holes. One limitation to using Soar in this fashion is if when the database starts to grow very large (around 150 holes), Soar begins to take a long time (order of seconds instead of milliseconds) to decide about what hole is next due to all the proposal rules and elaboration rules between the different hole identifiers. This delay only occurs when Soar looks for new circles during every trial and does not occur when Soar is only trained during the first trial as only the holes that have screws are proposed.

Using Soar, the run times of the different laptop models could be cut down. The system could further increase its speed by eliminating the need for computer vision after a screw has been found. To do this, the exact location of the screw must be saved to semantic memory. The location must be found and coordinates saved for both orientations of the laptop as the transposition of coordinates from one orientation to the other are not perfect. Although this will increase the training time of the laptop, it will cut out over a minute for some of the laptop models and further reduce the disassembly time to 2-3 minutes when the system is removing the screws.

The system presented in this paper performs the best and has the fastest run times when the trials that it can explore circles are limited to one training run. While exploring circles in every trial gives the system the greatest chance to find a screw it might have missed, it results in longer run times. The number of trials that the system performs when it is exploring screws can be adjusted so there is a tradeoff between the fastest trial times and its accuracy in finding screws. The reward factor the system currently gives is +0.5 if a screw is found or -0.5 if a screw is not found. These reward scores could also be modified to change the systems behavior when it is training. By changing the scores, it would mean that the system would have to find a screw or not find a screw multiple times before that location was saved as containing a screw or not thus improving the confidence that the system has found all the screws it can remove.

The role that Soar plays in the disassembly process could also be extended to control the different systems connected to the robot and not just selecting the next hole to remove a screw from. For example, it may be possible to use a different module in Soar's long term memory, such as episodic memory, to remember the disassembly steps and order for the different models of laptops. Soar could also be used to control different disassembly tools that are created by using sub goals to organize and manage the different states and operators rules that Soar would be able to choose.

4.5 References

[1] A. Chen, K. N. Dietrich, X. Huo, and S. Ho, "Developmental neurotoxicants in e-waste: an emerging health concern.," *Environ. Health Perspect.*, vol. 119, no. 4, pp. 431–438, Apr. 2010.

[2] I. C. Nnorom and O. Osibanjo, "Overview of electronic waste (e-waste) management practices and legislations, and their poor applications in the developing countries," *Resour. Conserv. Recycl.*, vol. 52, no. 6, pp. 843–858, Apr. 2008.

[3] A. O. W. Leung, N. S. Duzgoren-Aydin, K. C. Cheung, and M. H. Wong, "Heavy Metals Concentrations of Surface Dust from e-Waste Recycling and Its Human Health Implications in Southeast China," *Environ. Sci. Technol.*, vol. 42, no. 7, pp. 2674–2680, Mar. 2008.

[4] A. O. W. Leung and A. S. Wong, "Spatial Distribution of Polybrominated Diphenyl Ethers and Polychlorinated Dibenzo- p -dioxins and Dibenzofurans in Soil and Combusted Residue at Guiyu , an Electronic Waste Recycling Site in Southeast China," *Environ. Sci. Technol.*, vol. 41, no. 8, pp. 2730–2737, Mar. 2007.

[5] X. Huo, L. Peng, X. Xu, L. Zheng, B. Qiu, Z. Qi, B. Zhang, D. Han, and Z. Piao, "Elevated blood lead levels of children in Guiyu, an electronic waste recycling town in China.," *Environ. Health Perspect.*, vol. 115, no. 7, pp. 1113–1117, Jul. 2007.

[6] J. Cui and E. Forssberg, "Mechanical recycling of waste electric and electronic equipment: a review," *J. Hazard. Mater.*, vol. 99, no. 3, pp. 243–263, May 2003.

[7] C. Meskers, C. Hagelueken, S. Salhofer, and M. Spitzbart, "Impact of pre-processing routes on precious metal recovery from PCs," in *Proc. EMC*, 2009, pp 527-540.

[8] P. Schumacher and M. Jouaneh, "A force sensing tool for disassembly operations," *Robot. Comput. Integr. Manuf.*, vol. 30, no. 2, pp. 206–217, Apr. 2014.

[9] P. Schumacher and M. Jouaneh, "A system for automated disassembly of snap-fit covers," *Int. J. Adv. Manuf. Technol.*, vol. 69, pp. 2055–2069, Jul. 2013.

[10] N. M. DiFilippo and M. K. Jouaneh, "A System Combining Force and Vision Sensing for Automated Screw Removal on Laptops," *IEEE Trans. Autom. Sci. Eng.*, In review.

[11] A. Bannat, T. Bautze, M. Beetz, J. Blume, K. Diepold, C. Ertelt, F. Geiger, T. Gmeiner, T. Gyger, A. Knoll. C. Lau, M. Ostgathe, G. Reinhart, W. Roesel, T. Ruehr, A. Schuboe, K. Shea, I.S.g. Wersborg, S. Stork, W. Tekouo, F. Wallhoff, M. Wiesbeck, and M. Zaeh "Artificial cognition in production systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 1, pp. 148–174, 2011.

[12] M. Stenmark, J. Malec, K. Nilsson, and A. Robertsson, "On Distributed Knowledge Bases for Robotized Small-Batch Assembly," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 519–528, Apr. 2015.

[13] U. Kurup and C. Lebiere, "What can cognitive architectures do for robotics?," *Biol. Inspired Cogn. Archit.*, vol. 2, pp. 88–99, Oct. 2012.

[14] J. E. Laird, A. Newell, and P. S. Rosenbloom, "Soar: An architecture for general intelligence," *Artif. Intell.*, vol. 33, no. 1, pp. 1–64, Sep. 1987.

[15] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin, "An integrated theory of the mind.," *Psychol. Rev.*, vol. 111, no. 4, pp. 1036–60, Oct. 2004.

[16] R. Chong, "Inheriting Constraint in Hybrid Cognitive Architectures: Applying the EASE Architecture to Performance and Learning in a Simplified Air-Traffic Control Task," *Air Force Res. Lab.*, no. AFRL-HE-WP-TR-2005–0120, 2004.

[17] D. Kieras and D. Meyer, "An Overview of the EPIC Architecture for Cognition and Performance with Application to Human-Computer Interaction," *Univ. Michigan*, no. TR-95-ONR-EPIC-5, 1995.

[18] R. M. Jones, C. Lebiere, and J. A. Crossman, "Comparing Modeling Idioms in ACT-R and Soar," in *Proc. 8th Int. Conf. Cognitive Modeling*, 2004, pp. 49–54.

[19] S. D. Hanford, "A Cognitive Robotic System based on the Soar Cognitive Architeture for Mobile Robot Navigation, Search, and Mapping Missions," Ph.D. dissertation, Pennslyvania State Univ., 2011.

[20] T. R. Johnson, "Control in Act-R and Soar Act-R," in *Proceedings of the 19th Annu. Conf. Cognitive Science Society*, 1997, pp. 343–348.

[21] J. E. Laird and P. S. Rosenbloom, "Integrating Execution , Planning , and Learning in Soar for External Environments," in *8th Nat. Conf. Artificial Intelligence*, 1990, pp. 1022–1029.

[22] E. S. Yager, E. Laird, M. Tuck, and M. Hucka, "Learning in Tele-autonoumous Systems Using Soar," in *Proc. NASA Conf. Space Telerobotics*, 1989, pp. 416–424.

[23] S. D. Hanford, O. Janrathitikarn, and L. N. Long, "Control of Mobile Robots Using the Soar Cognitive Architecture," *J. Aerosp. Comput. Information, Commun.*, vol. 6, no. 2, pp. 69–91, Feb. 2009.

[24] S. D. Hanford, O. Janrathitikarn, L. N. Long, and I. Introduction, "Control of a Six-Legged Mobile Robot Using the Soar Cognitive Architecture," in *2008 AIAA Aerospace Sciences Meeting*, 2008, pp. 1–14.

[25] J. Kirk and J. Laird, "Interactive Task Learning for Simple Games," *Adv. Cogn. Syst.*, vol. 3, pp. 11–28, Jul. 2014.

[26] S. Mohan, J. Kirk, A. Mininger, and J. Laird, "Agent Requirements for Effective and Efficient Task-Oriented Dialog Interactive Task Learning Agent : Rosie," in *Proc. AAAI 2015 Fall Symp.* pp. 94–99.

[27] J. Kirk, A. Mininger, and J. Laird, "Learning task goals interactively with visual demonstrations," *Biol. Inspired Cogn. Archit.*, Aug, 2016, In press-corrected proof.

[28] P. Lindes and J. E. Laird, "Toward Integrating Cognitive Linguistics and Cognitive Language Processing," in *Proc. 14th Int. Conf. Cognitive Modeling*, 2016, pp. 86–92.

[29] A. Mininger and J. Laird, "Interactively Learning Strategies for Handling References to Unseen or Unknown Objects," *Adv. Cogn. Syst.*, vol. 4, pp 1-16, Jun. 2016.

[30] S. Vongbunyong, S. Kara, and M. Pagnucco, "Application of cognitive robotics in disassembly of products," *CIRP Ann. - Manuf. Technol.*, vol. 62, no. 1, pp. 31–34, Jan. 2013.

[31] S. Vongbunyong, S. Kara, and M. Pagnucco, "Basic behaviour control of the

vision-based cognitive robotic disassembly automation," *Assem. Autom.*, vol. 33, no. 1, pp. 38–56, 2013.

[32] S. Vongbunyong, S. Kara, and M. Pagnucco, "General plans for removing main components in cognitive robotic disassembly automation," *2015 6th Int. Conf. Automation, Robotics and Applications*, pp. 501–506.

[33] A. Newell and J. Laird, "A universal weak method," Carnegie Mellon Univ., PA, Dec. 1983, Rep. no. DRC-15-22-S3. Accessed on Oct. 21, 2016.

[34] S. Mohan, "From Verbs to Tasks: An Integrated Account of Learning Tasks from Situated Interactive Instruction," Ph.D dissertation. Univ. Michigan, 2015.

[35] J. E. Laird and S. Mohan, "A case study of knowledge integration across multiple memories in Soar," *Biol. Inspired Cogn. Archit.*, vol. 8, pp. 91–97, 2014.

[36] N. M. DiFilippo and M. K. Jouaneh, "Characterization of Different Microsoft Kinect Sensor Models," *IEEE Sens. J.*, vol. 15, no. 8, pp. 4554–4564, Aug. 2015.

[37] J. E. Laird and C. B. Congdon, "The Soar User ' s Manual," 2015. [Online]. Available: http://web.eecs.umich.edu/~soar/downloads/Documentation/SoarManual.pdf. Last accessed: Oct. 20, 2016.

[38] S. Mehta, A. Misra, a Singhal, P. Kumar, and a Mittal, "A high-performance parallel implementation of sum of absolute differences algorithm for motion estimation using CUDA," in *HiPC Conf.*, 2010, vol. 2, No. 4, pp 6-11.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

The main goal of this dissertation was to investigate how well the Soar cognitive architecture could be integrated with a robotic agent to help it automatically disassemble e-waste. This was broken up into smaller studies that first characterized how accurately different Microsoft Kinect models could determine the distance to an object. An ANOVA test was performed to establish if the operating temperature, the model of Kinect, or their interactions were significant factors in determining the distance to an object. The Kinect sensor was also used to reconstruct gauge blocks and objects on machinists blocks set up on an angle.

All the different Kinect models give very accurate results when they are placed between 600 and 800 mm away from the target. At all the distances tested, the standard deviation of all the Kinect models is lower than 2.1 mm, which shows the sensor has good repeatability. At distances up to 700 mm, the resolution of all the models is also very good at 1 mm. At 1800 mm, the farthest distance that the sensor was tested; the resolution of the Kinect was 9 mm which shows that the Kinect may not be suitable at that distance for applications requiring precise measurements.

The operating temperature, the model of the Kinect used, and the interactions of the two are all significant factors when the Kinect is determining the distance to a target. Until the temperature reaches a high steady state operating temperature, the output from the Kinect continues to change. Therefore, it is advised to wait until the

Kinect has reached a high steady state temperature, which takes approximately 100 minutes, before using it for applications that require precise measurements. The Kinect sensor is also able to accurately reconstruct 3D objects and determine the angle that an object is at, however calibration is needed for both these parameters before the Kinect's output is accurate. From overhead, the Kinect was also able to determine the holes in the machinists block provided that the angle is not too steep - less than 40 degrees.

A significant challenge in the automated disassembly process is automatically locating the holes that contain screws. After characterizing the Kinect, a method was created that could automatically find and remove screws from the backside of laptops. This approach used two cameras; one that was mounted over the workspace (overhead camera) and another that was mounted on and moved with the robot (robot camera). The overhead camera would identify possible screw locations on the laptop and the robot camera would center the hole for the screwdriver. A SE screwdriver was designed and could go and explore locations on the laptops that were identified by the overhead camera. The screwdriver could probe an area to determine if a screw was present or not. The SE screwdriver used two low cost sensors, a FSR and an accelerometer to determine when the SE screwdriver had made contact with a surface and when a screw had been completely loosened respectively.

Trials were conducted that varied camera parameters and the color of the laptop to determine which combinations helped the system find and remove the most screws. The results show that the lighting used to illuminate the surface and the brightness of the cameras are two major parameters that need to be adjusted and are dependent on the darkness of the laptop case. The darker the case, the higher the brightness level of the cameras should be. The brightness of the overhead camera will affect how many circles will be found over the surface of the laptop and up to a certain point, the higher the brightness, the more circles will be found on the laptop, thus increasing the chance that all the circles will be found on a laptop. As the laptop case gets darker, the robot camera's brightness level must increase as it creates a higher contrast between the hole and laptop case making it easier to find and center the hole.

One drawback to the method described above is that without a way to remember the location of the holes, the time of the test will take between 40 to 60 minutes. To improve on the systems performance, Soar was incorporated into the robotic system and used to determine what model of laptop was placed in the workspace and what circles locations on the laptop contained screws and what locations did not. Using Soar the trial time decreased by approximately 60% for all the laptop models tested. Soar was run on the different laptop models using two different methods. The first method allowed Soar to continually train itself with circle locations every trial while the second method limited the number of these training trials to just the first trial. The second method was faster than the first method and decreased the time of a test by at least 10% for 17 out of the 18 laptops.

Overall, the system presented in this dissertation shows that it can automatically locate and remove screws from the backside of laptops. Removing screws is one of the first steps in the complex operation of recycling e-waste. The Soar cognitive architecture improved the speed that the system could remove screws and showed that a cognitive architecture can be beneficial in creating disassembly plans. Sub-goals and the ability to keep Soar organized are going to become even more important as more sub-systems are integrated with the robotic system and the disassembly operations performed become more advanced. Soar's ability to automatically move into a sub-goal makes it a strong platform to solve the problem when an impasse occurs. The introduction of new sub-systems will introduce a large amount of input from various sensors which Soar can handle well since it uses all working memory to decide on what operator to choose next. Additionally, it is very easy to interface with Soar from different programming languages using the SML as all the hard work to put information to and get information from the input-link and the output-link has been taken care of.

A drawback of using Soar is that as the number of circles the overhead camera locates increases, the number of holes that are moved from semantic memory into working memory also increases. Therefore, once they are in short term memory, Soar will need more time to elaborate the state and perform state preferences between all the potential operators that represent the different holes. The time it takes to sort these proposals out is typically on the order of seconds instead of milliseconds and speeds up as more and more of the circles have been chosen and there are fewer holes to perform these preferences between. This slowdown is more prominent when the number of holes gets to around 150 for an individual laptop, or when there are clusters of circles found within a small region of a laptop. This problem is minimized when Soar only selects circles that contain screws to remove. Since there are typically less than 20 screws on the backside of the laptop, the next hole is selected quickly because not that many operators need to be proposed.

Another limitation of Soar is that the syntax used to create Soar productions is quite complicated, which leads to a very steep learning curve when trying to write a Soar agent. There are a couple of tools that are included with any distribution of Soar, Visual Soar and the Soar debugger, that aim to help in the debugging and writing of Soar agents. Visual Soar is an editing program that provides internal support for debugging Soar programs such as checking Soar specific syntax. The Soar debugger allows you to step through code and pause whenever you want during a decision cycle to see what working memory looks like at that moment. However, sometimes it is more convenient to register for print events using the SML and print them like debug statements in other languages. A more detailed explanation of Soar productions and operators as well as the SML are given in Appendix – A.

5.2 Future Work

In the work presented in this dissertation, the Soar cognitive architecture was used to determine which laptop model was placed in the workspace. It was also used to decide which hole to select next during the disassembly process since it could remember if there was in a screw in that location or not. If there was a screw in that hole, the screwdriver would go to remove it and if there was not a screw in the hole then the robot would move on to the next hole. If the hole was not in memory because it had not been explored yet, then the screwdriver would go and explore that hole to find out if a screw was present or not.

Instead of just using Soar to remember the locations of circles found by the overhead camera and if they contain screws or not, the role of Soar can be expanded to control the other systems connected to the robot. Currently, the SE screwdriver is controlled by a state transition diagram, however, this logic can be migrated to Soar. This is also true for other systems, such as the clamping system or any other tools that may be developed in the future. In order to keep track of these different systems, Soar would have to implement a framework where it uses sub-goals to manage and organize the states needed. An example of how the different states can be managed is shown in Fig. 1.

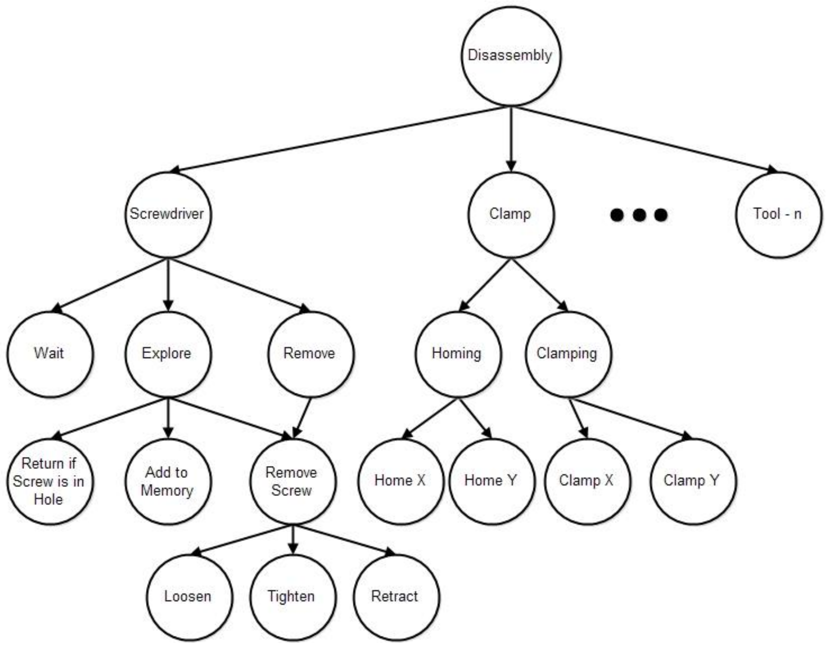


Fig. 1. Sub-goal diagram for Soar

When using Soar, every circle on the diagram is a different sub-goal and state that Soar can move into when trying to solve a problem. By moving into these new states, different sub-goals and states become available to Soar that were not accessible under the original state. For example, if the process was started in the state 'D*isassembly*', there are two different states it could choose: '*Screwdriver'* or '*Clamp'*. Both of these states have sub-goals that are not accessible from the other state. If the screwdriver state is chosen, then it can lead to sub-goals such as wait, remove, or explore. If the clamp state is chosen, it can lead to goals such as home or clamp. These goals can lead to further sub-goals and this can continue until the output is a low-level command that can control a motor or other piece of hardware.

The role of Soar can also be expanded by using different long term memory modules, since only semantic memory is currently used. The other long term memory modules that Soar supports are episodic memory, which uses past experiences of the workspace to help the system learn, and procedural memory which uses a reward function through the use of reinforcement learning to change how an operator is selected. While semantic memory could still be used to determine which laptop has been placed in the workspace, episodic memory could store the specific disassembly routines for each of the laptop models. For example, instead of unscrewing all of the screws on the laptop at once, it may be more efficient to only unscrew certain screws in a specific region of a laptop and then use another tool to perform a different action in the disassembly sequence.

A drawback of using the current memory logic is if the screwdriver or vision system is unsuccessful in finding a screw when a screw is actually in that location, that location will be saved in semantic memory as not containing a screw and subsequently ignored in the future. One way to fix this problem would be to adjust the way the system is trained and require some interaction with a human operator. After the system has attempted to find the holes with the overhead camera during training, the human operator could mark any holes that it had missed.

Another way to address this problem is to modify the reward factor given when determining if a location contains a hole or does not contain a hole. The reward factor the system currently gives is +0.5 if a screw is found or -0.5 if a screw is not found. Since the system starts with of score 0.5 and this score can range from 0.0 (no screw present) to 1.0 (screw present), after one trial the system reaches of the range limits. By changing the reward, it would mean that the system would have to find a screw or not find a screw multiple times before that location was saved as containing a screw or not thus improving the confidence that the system has found all the screws it can remove. These scores can also have different values as so finding a screw might have more weight than not finding a screw. The tradeoff for changing the rewards to a less aggressive factor is that the number of training trials would have to go up and ultimately, the amount of time of a test would increase.

APPENDICES

APPENDIX A – Description of Soar code and syntax

Soar is a general cognitive architecture developed by John Laird, Allen Newell, and Paul Rosenbloom at Carnegie Mellon University in 1983. Soar works by using productions to test, propose, and select operators (rules) that are in working memory to try to obtain a specified goal state. Working memory is a representation of how Soar views the workspace in its current configuration and is part of Soar's short term memory module. Soar stores information as *working memory elements* (WME) consisting of an identifier, an attribute (preceded by a '^'), and a value. The value can either be a terminal node consisting of a constant (string or numerical value) or a non-terminal node which links to another identifier. Figure 1 shows a graphical representation of WMEs of Soar's working memory structure that is created the first time a Soar agent is started. All of the WMEs are connected back to the state S1 which is always the name of the top-level state. The state S1 always has three attributes, ^*superstate*, ^*type*, and ^*io*. The attribute ^*superstate* has a constant value of nil, ^*type* has a constant value of state, and ^*io* has a value of another identifier I1, that in turn has two attributes which also link to more identifiers.

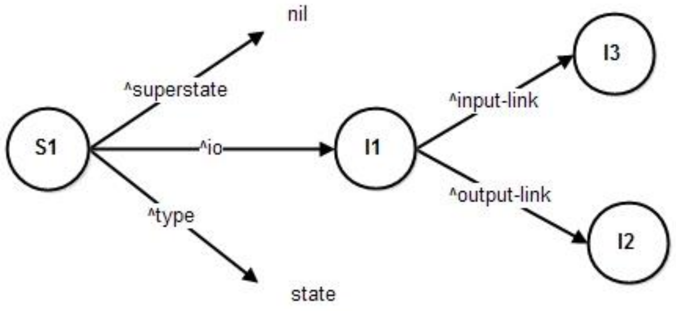


Figure 1. Soar Initial State

When using Soar, there is a shortcut notation known as “dot notation” which adds a dot (.) in between multiple attributes that only link identifiers. This dot notation can be used for an arbitrary number of levels in Soars working memory and is useful for shortening and making the rules written for Soar easier to read. A general rule of thumb is to use dot notation unless you need to get information off of a certain state or a state has two attributes that you need to access. Looking at Figure 1, the state I3 can be represented in two different ways. The first is with the conventional notation that uses variables to link states. In Soar, anything that is enclosed by < > is considered a variable. A variable can be used to store a value that needs to be tested later and can be placed either as an attribute or a value of a WME.

```
(state <s> ^io <io>)
(<io> ^input-link <il>)
```

The second way to represent the state I3 uses the more compact dot notation which shortens the rule:

```
(state <s> ^io.input-link <il>)
```

The decision cycle that Soar goes through is shown in Figure 2. This cycle is repeated until either a goal state is reached or an impasse occurs that prevents Soar from making progress in solving the problem. The first phase is the input phase and is where Soar checks to see if there has been any new information that has been placed on its input link via sensors or external environments. In order to put information on the input-link, different programming languages must communicate with Soar using the Soar Markup Language (SML). After Soar has checked its input-link, it will move to the ‘elaborate and propose’ operator phase. Here, Soar proposal operators are fired in parallel and the firing of one operator could lead to the firing or retraction of other operators. Once there are no more operators left to fire, the system has reached a state of quiescence and proceeds to the next phase, which is the ‘select operators’ phase. In this phase, Soar uses the rules preferences that it has been given in order to select a single operator. In the ‘apply operator’ phase, Soar will execute the selected operator and make changes to working memory. When Soar gets to the output phase, Soar will put any appropriate information on the output-link. Once the information is on the output-link, the different programming language can again access it by using the SML.

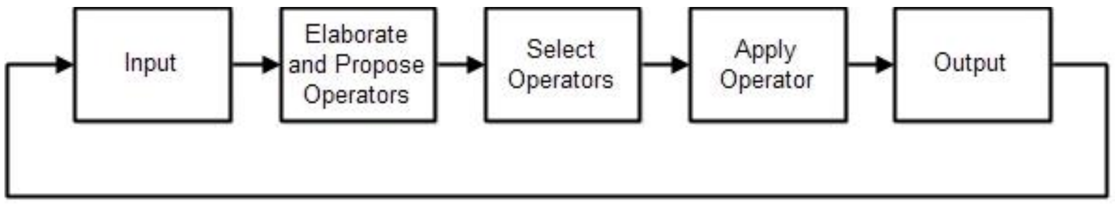


Figure 2. Soar decision cycle

In Soar, the rules that are written are called productions, (which is why every rule starts with ‘sp’ or ‘Soar production’) and two of the main types of productions are the proposal and apply operators. During the proposal sequence of a Soar decision cycle, all of the different proposal operators that match the current conditions of the workspace are fired and eventually only one is chosen. A rule in Soar can essentially be broken up like an IF-THEN statement. Soar separates these rules with the ‘-->’ symbol. Everything that is above this arrow symbol can be construed as the IF part of the statement and everything below this symbol can be taken as the THEN part of the statement. Typically, proposal and apply operators are written together as shown in Figure 3. Here, this particular rule is the first rule that is fired after a Soar agent is started and the working memory shown in Figure 1 is created. All Soar rules start with ‘state <s>’ referring to the identifier S1, although <s> is technically just a variable and can be any combination of letters, it is good convention to keep it a ‘<s>’

The proposal rule (propose*initialize*smemory) tests that the Soar agent exists and Soar is running (<s> ^superstate nil) and that there is no attribute in Soar memory directly connected to S1 called ‘name’ (this is a test for negation). If both of these conditions are true, Soar then puts a preference operator with the ^name initialize-smemory on the main state. The plus sign next to the operator means that this operator is a proposal operator. Since this is the only rule that matches in working memory, this operator (apply*initialize*smemory) will be selected and applied. First, the rule checks to make sure that the operator with the attribute ^name initialize-smemory is selected. If this is true then it sets up WMEs in Soar. This process is shown in Figure 4. During the next decision cycle, since there is now a “^name” on state S1, this operator does not match anymore and is removed from memory.

```
#Proposal Rule
sp {propose*initialize-smemory
   (state <s> ^superstate nil
              -^name)
-->
   (<s> ^operator <o> +)
   (<o> ^name initialize-smemory)}

#Operator Apply Rule
sp {apply*initialize-smemory
   (state <s> ^operator <op>)
   (<op> ^name initialize-smemory)
-->
   (<s> ^name smemory
        ^laptop-info <li>
        ^smem-holes <smem-holes>
        ^memory-process <mp>)
   (<mp> ^laptop <laptop>
        ^memory <memory>)
   (<laptop> ^query 0)}
```

Figure 3. Soar example showing a proposal and apply operator.

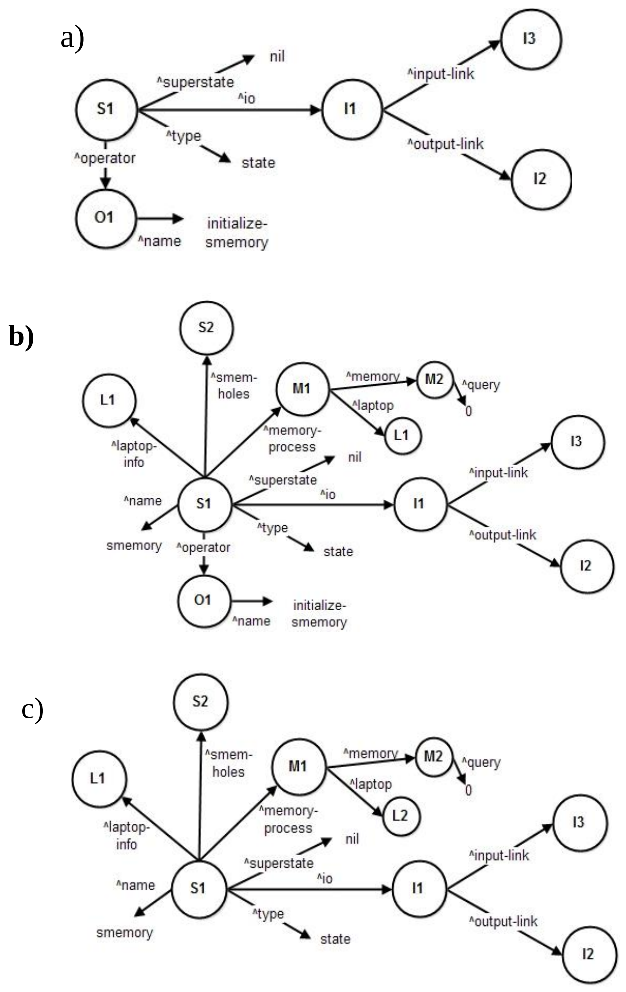


Figure 4. Soar Memory a) an operator is selected. B) an operator is applied. c) an operator is removed and the WMEs created persist.

Persistence of Memory Elements

The two different types of WMEs that can be added to memory are rules that are operator supported (o-support) and rules that are instantiation supported (i-support). The important distinction between these two types of rules is that the changes made to working memory by o-supported rules persist even after the operator that makes the changes has been removed from working memory. The example in Figure 4 shows an example of these o-supported rules. Even after initialize-smemory has been applied and removed from working memory, the WMEs it added stayed in working memory. If a WME is i-supported, it means that when the rule that created them is no longer in working memory, the WME is removed. The rules that have i-support are operator proposal rules, operator preference rules, elaborations, or state elaboration rules. Since WMEs that are o-supported will remain in working memory even after the rule that created them is removed, they must be explicitly told to be removed from working memory. An example of removing a WME is shown in

Figure 5. In this example, a semantic memory query is removed from working memory with the minus (-) sign.

```
#Example of removing an element from working memory
sp {apply-clean-up-query
   (state <s> ^name smemory
              ^operator <o>
              ^smem.command <cmd>)
   (<cmd> ^<q> <query>)
   (<o> ^name clean-up-query
        ^<q> <query>)
-->
   (<cmd> ^<q> <query> -)}
```

Figure 5. Soar example showing how to remove an WME from working memory.

Soar Elaborations

Elaborations are a type of Soar rule that create new WMEs. Elaborations rules don’t need to be chosen with a proposal rule as they fire during every decision cycle when their IF part of the rules matches working memory. Elaborations create WMEs that have i-support, so if an elaboration is removed from working memory, all of their changes are also removed. In the example in Figure 6, the rule checks to make sure that the name of the state is smemory and that there are dimensions for the length and width on the input link. In this rule, the { } are used to group the alternative attributes that are acceptable in between the double angle brackets (<< >>) and save them to the variable <type>. Now <type> will match both the ‘length’ and ‘width’, and create attributes for both under laptop-info. Both ‘length’ and ‘width’ will contain the upper and lower bounds of the value plus and minus 2. A visual of what the state looks like before and after this rule is shown in Figure 7 (the output link and parts of the state that are not tested in this rule have been omitted). If the length and the width of the laptop were to change during a test, then new bounds would be automatically computed.

```
#Example with elaboration
sp {compute-bounds-on-laptop-input
   (state <s> ^name smemory
              ^io.input-link.laptop.dimensions <d>
              ^laptop-info <li>)
   (<d> ^{<< length width >> <type>} <value>)
-->
   (<li> ^<type> <lit>)
   (<lit> ^upper (+ <value> 2)
          ^lower (- <value> 2)) }
```

Figure 6. Soar example showing elaboration rule.

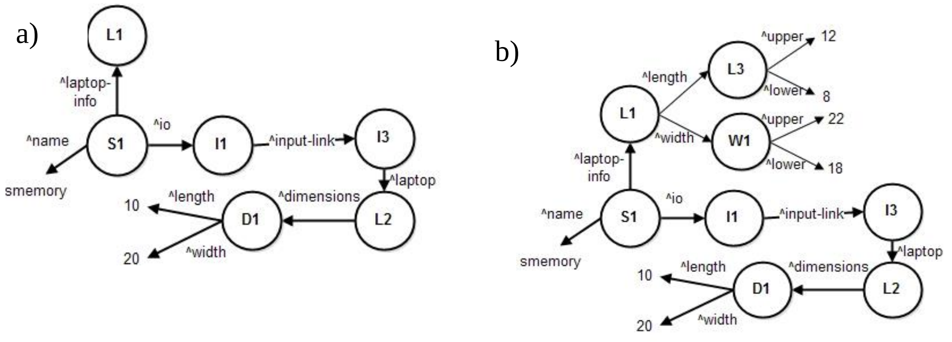


Figure 7. Soar example showing a Soar elaboration rule with variables matching multiple attributes. A) before the rule is applied. B) After the rule is applied.

Soar Preferences

During a decision cycle, Soar allows for preference production rules, generated by the user, to be employed. These rules help Soar decide which operator to select when there are multiple rules that have been proposed. These preferences can either be given to the rule when it is proposed, or the preferences can be given to differentiate two operators from each other. Figure 8 gives two examples of Soar rules with preferences. The first example shows the preference operator when operator <o> (a hole that contains a screw) is preferred over operator <o1> (a hole that does not contain a screw). The second example shows an operator being proposed that has acceptable (+) and best (>) preference. Using preferences, it is possible for operators to be rejected (-), be the worst choice (<) where the operator is only selected if nothing better is proposed, or be indifferent (=). Indifferent is an important preference (and what was used in this dissertation to distinguish between two choices with identical preference) because if two operators are proposed and have the same preference level, and all of the operator preferences are exhausted; Soar is allowed to choose one of the operators at random. If indifferent was not selected, Soar would have a tie impasse that occurs, and would have to go into subgoals.

```
#Example with operator preference comparison
sp {prefer-match-with-a-screw
   (state <s> ^name smemory
            ^operator <o> +
^operator <o1> +)
   (<o> ^screw yes)
   (<o1> ^screw no)
-->
   (<s> ^operator <o> > <o1>)}

#Example with operator preference in the proposal
sp {propose-no-match-after-first-query
   (state <s> ^name smemory
            ^smem <sm>
           -^laptop-in-memory
            ^memory-process <mp>)
   (<mp> ^laptop.query 1
        -^memory.status complete)
     (<sm> ^result.failure <failure>)
-->
   (<s> ^operator <o> + >)
   (<o> ^name failed*query-1)}
```

Figure 8. Soar example showing operator preferences.

Math Operators

Unlike many programming languages that you may be familiar with, Soar is not conducive to performing complex mathematical operations. Soar uses prefix notation, which means in order to add two numbers together, they must be prefixed by the mathematical operator. For example, to add 4 and 5 together, the syntax would look like (+ 4 5). This can be extended to variables so if <v> = 4 and <z> = 5, the syntax would look like (+ <v> <z>). A downside to this is that mathematical operations can get rather complex rather quickly. A more complicated example is shown where an average is updated with a new number. This is shown below with the Soar syntax first, and in regular math syntax for clarification.

```
( * ( + (* <old-Average> <Old-Count>) <new-number> ) ( / (+ 1 <old-count>) ) )
```

$$((Old\ Average * Old\ Count) + New\ Number) * \left(\frac{1}{Old\ Count + 1}\right)$$

Subgoals

In Soar, an impasse occurs when Soar cannot make a decision based on the current contents of working memory. There are four different types of impasses that can occur. The first type of impasse is a *tie impasse* which is when multiple operators tie in preference level when an operator is being selected and no preferences are left to distinguish between them. The second type is a *conflict impasse* which is when two operators have conflicting preferences (operator A > operator B and operator B > operator A). The third type is a *constraint-failure impasse* which is when a value or operator has both a require and prohibit preference. The fourth type of impasse is a *no-change impasse* where a new operator is not selected in the decision cycle. A no change impasse can either be a *state no-change impasse* where there are no acceptable operators or an *operator no-change impasse* where no productions match during the application phase of a decision cycle.

In order to solve these impasses, Soar can create new states known as substates or subgoals in order to try to resolve them. When Soar is using the different substates it will makes use of the ^superstate attribute which is the state it just came from. The initial state that Soar started in is known as the top-level state. In the substate, operators can be selected and applied with the goal of resolving an impasse and returning to the top-level state. It is also possible that this substate will result in a new impasse and another substate will be created.

The Soar agent used for this dissertation was designed to only run in the top-level state and did not use subgoals. However in the future work section of the conclusion, a process was proposed on how to extend the use of Soar in disassembly operations in which the use of subgoals would need to be implemented.

Semantic Memory

Semantic memory is part of the long term memory module in Soar and is where previous knowledge about the workspace is stored. This allows Soar to move through these snapshots in an attempt to find the best match. In Soar, the '@' sign symbolizes a long-term identifier (LTI) indicating that the specified identifier resides in the long term memory module.

In order to use Soar's semantic memory, the file named _firstload.soar needs to be modified with the following commands:

```
smem --init                                    #initializes the database clears semantic memory
smem --set database file                       #Tells Soar to use a file for semantic memory
smem --set path C:\Python27\Scripts\smem.db    #Path of the SQLite database
smem --set lazy-commit off                     #Tells when Soar can write to SQLite database
smem --set learning on                         #Turns on semantic learning
```

Some notes about the above code:

- Semantic memory can be maintained in the computer's memory or on a disk. The contents of semantic memory will only persist between trial runs when semantic memory was maintained on a disk and the database was set to file.
- Lazy-commit was set to off when using Python and Idle, as changes made when Soar was run were not recorded in semantic memory when this was set to on. When C++ was used, Lazy-commit did not have to be set to off.
- The path command doesn't always work properly, however, it consistently places the database in the main directory where the Soar agent is. If this parameter is not set, then the database (which is saved as a SQLite database) will go to some other location on the computer that can be hard to locate.

To retrieve information from long term memory, the semantic memory database has to be queried (This dissertation only used the cue-based-retrieval method as it allowed the base query to be modified with a math-query). In order to perform a query, you need to use the command <s> smem.command.query.<*attribute*> <value>. The smem.command.query memory links are built-in to Soar. If you are interested in performing mathematical comparison on the query, the query command must be augmented with a math-query. An example of a math-query is shown in Figure 9 where there is a query that will match any name but the math query restricts the results to being greater than <length-lower> and less than <length-higher>. A math-query supports conditions like greater-or-equal and less-or-equal natively and a list of all of conditions can be found in the Soar manual.

```
#Example query and math-query
sp {apply-check-database
    (state <s> ^name smemory
               ^smem <sm>
               ^operator <o>
     (<sm> ^command <cmd>)
     (<o> ^name check-database
          ^length <length-bounds>)
     (<length-bounds> ^upper <length-upper>
                      ^lower <length-lower>)
-->
    (<cmd>  ^query.name <any-name>
            ^math-query <mq>    )
     (<mq> ^length <mql>
     (<mql> ^less-or-equal <length-upper>
            ^greater-or-equal <length-lower>)}
```

Figure 9. Soar example showing a query and math query with the semantic memory database

In order to check the results, the memory structure (state <s> ^smem.result <r>) is created by Soar. The state <r> is modified based on the results of the query. When the database is queried, there are three main outcomes. The first is that the query is successful and the LTI that you want is retrieved from memory. In this case, the result state is shown below where the LTI that is found is under the retrieved attribute:

```
(state <s> ^smem.result <r>)
(<r> ^success <val>
     ^retrieved <LTI>)
```

The second is that the query is successful but there is no LTI that matches in working memory:

```
(state <s> ^smem.result <r>)
(<r> ^failure <val>)
```

The third is that the command of the query is not a valid command:

```
(state <s> ^smem.result <r>)
(<r> ^bad-cmd <cmd>)
```

It should be noted that semantic memory will only return **ONE** long-term identifier that matches the parameters of the query (based on the largest bias value) even if more satisfy the requirements. If this is the case, more parameters need to be added to the query to pare down the results that match.

To store a values as a LTI and save them in semantic memory, the command that needs to be used is ^smem.command.store <LTI>. The <LTI> will then need to be expanded so any attribute that needs to be saved is connected to it. An example is shown in Figure 10 where a laptop is added to semantic memory with name, length, width, height, template, and count attributes.

```
#Example store identifier
sp {apply-add-laptop-to-memory
   (state <s> ^name smemory
              ^operator.name add-new-laptop-to-memory
              ^io.input-link.laptop <ill>
              ^smem.command <cmd>)
   (<ill> ^dimensions <d>
          ^filenames <fn>)
   (<fn> ^template <t>)
   (<d> ^height <h>
        ^length <l>
        ^width <w>)
-->
   (<cmd> ^store <laptop>)
   (<laptop> ^name (make-constant-symbol <laptop>)
             ^length <l>
             ^width <w>
             ^height <h>
             ^template <t>
             ^count 1)
   (<s> ^stored-laptop <laptop>)}
```

Figure 10. Soar example to store an identifier in semantic memory

Writing and Debugging

Soar comes with a couple of tools that make writing and debugging Soar agents easier. One of these tools is called Visual Soar which is a text editor that natively supports writing Soar agents. It allows you to check Soar syntax as you are writing code and is a good way to keep your project organized. Another program that comes with all releases of Soar is the Soar Java debugger. This debugger allows you to step through a Soar agent and see how the contents of working memory change as the agent makes decisions. This is a good way to debug code and make sure that working memory is changing how you think it should be changing. Finally, it is possible to insert write statements that can help with debugging variables or the logic flow. A write statement in Soar has the syntax (write (crlf)|Hello World|) where the (crlf) stands for carriage return line feed.

Soar Markup Language (SML)

Soar is able to communicate with various programming languages by using the Soar Markup Language (SML). This dissertation was written using SML to communicate in Python and this section of the Appendix will cover different SML commands that were used in this dissertation using Python code. Before the SML can be used, it must first be compiled for the language that you want to use. A SML guide is located at http://soar.eecs.umich.edu/articles/articles/soar-markup-language-sml (last accessed October 21, 2016). This guide shows how to compile the SML in various languages and has a quick start guide that deals with showing how to use the SML using C++. The guide located online contains more methods and functions that may be helpful for different applications but were not used in this project. In Python the first thing you must do is append your system path to point to wherever Soar is located on your computer. After the path has been appended, you can import the Python_sml_ClientInterface

```
sys.path.append('C:/Soar/SoarSuite_9.4.0-Windows_32bit/bin')
import Python_sml_ClientInterface as sml
```

Next, you need to create the kernel and the agent for Soar to run as well as load the production that you have written.

```
k = sml.Kernel.CreateKernelInNewThread()
a = k.CreateAgent('disassembly')
lP = a.LoadProductions("disassembly.soar")
```

Registering for Print Events

It is a good idea setup your code to be able to register for print events from Soar. This allows you to display statements that are generated by Soar which is invaluable especially during the debugging process.

```
 trace = ""
callbackp = a.RegisterForPrintEvent(sml.smlEVENT_PRINT, callback_print_message, trace)
```

Underneath your main function you will need two more functions to finish setting up to register for the print events.

```
def register_print_callback(kernel, agent, function, user_data=None):
        agent.RegisterForPrintEvent(sml.smlEVENT_PRINT, function, user_data)

def callback_print_message(mid, user_data, agent, message):
    print(message.strip())
```

Now whenever you want to print out the trace message, all you have to do is use print(trace). It is also possible to tell Soar to execute a ‘command line’ command. A couple examples have been shown below. These commands will print what is connected to state 7, print what is connected to state S1 for 100 levels of a tree structure, and print semantic memory respectively.

```
print a.ExecuteCommandLine('print s7')
print a.ExecuteCommandLine('p -d 100 s1')
print a.ExecuteCommandLine('smem --print')
```

Connecting to the Input Link

First, you must create a connection using the SML to the input link by using GetInputLink. After this connection has been made, you can create working memory structures to suit your application. When you are creating these structures, if you are creating identifiers that only link to other identifiers, you need to use the CreateIdWME method which takes two arguments. One is the identifier that you are linking to and the other is the name of the attribute that links them. Currently, integers, floats, and string WMEs are supported when creating terminal nodes and will use CreateIntWME, CreateFloatWME, or CreateStringWME respectively. Each of these methods takes three arguments. One is the identifier it links to, one is the attribute that links the identifier to the value, and the last argument is the value or constant.

```
pInputLink = a.GetInputLink()
pIDScrewdriver = a.CreateIdWME(pInputLink, "screwdriver")
pWME0 = a.CreateStringWME(pIDScrewdriver,"screw-present", "na")
pWMEHoleSensitivity = a.CreateFloatWME(pIDScrewdriver,"sensitivity", 0.75)
pWMEHoleBrightness = a.CreateIntWME(pIDScrewdriver,"brightness", 0)
```

It is also possible to update a WME that is on the input link instead of just creating it. This method takes two arguments. The first is the identifier that you are trying to change and the second is the value that you want to change it to. It is important to make sure that the value that you are trying to put on the input-link is of the same variable type that it has been declared as (ie. You ***CANNOT*** put an integer on a WME that has been designated as a string) or the program will crash.

```
a.Update(pWMEHoleBrightness, 125)
```

Decision Cycle

There are a couple of different ways to control Soar's decision cycle. The work in this dissertation used a flag called *endSoar* that was used as a condition on a while loop. Once a goal state was reached in Soar, this flag was changed to True and Soar would end. Every time the loop was repeated, the decision cycle ran one time. A counter called *SoarCounter* was also used as protection to make sure that if an impasse occurred Soar would not run forever. This counter number was adjusted as necessary and chosen to be high to ensure it would not interfere and end the program prematurely when Soar was running and making decision.

```
while endSoar != True:
        a.RunSelf(1)
        SoarCounter = SoarCounter + 1
        if SoarCounter == 1000:
             endSoar = True
```

Output Link

During every loop of the decision cycle, the output link needs to be checked to see if any new commands were generated and put on it. If new commands were put on the output link, the attribute name and value need to be retrieved in order to be processed. The output link needs to be in the form (<O3> ^<attribute-name> <A1>) (<A1> ^<command> value) so it can be checked easily.

```
numberCommands = a.GetNumberCommands()
        for i in range(numberCommands):
```

```
pCommand = a.GetCommand(i)
attribute = pCommand.GetCommandName()
value = pCommand.GetParameterValue("cmd")
```

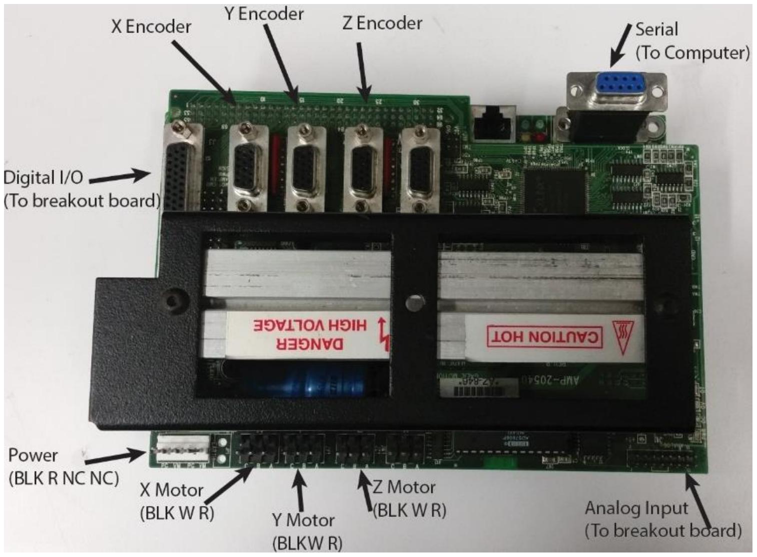
After the output link has been checked, a flag ^*status complete* was added to command identifier on the output link to mark the command as completed. In this project, this was used to verify with Soar that the information had been retrieved from the output link that information could be removed from the output link.

```
a.CreateStringWME(pCommand, "status", "complete")
```

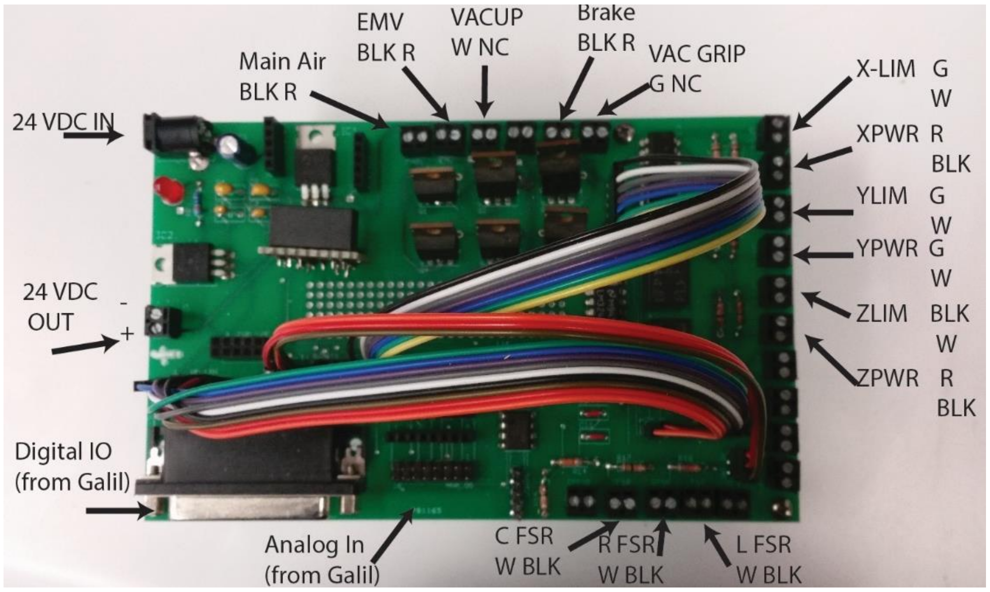
APPENDIX B – Wiring diagrams and location of code

All of the Code written for this dissertation is located on the computer connected to the EnCore 2s robot in the mechatronics/robotics lab as well as on file with Dr. Musa Jouaneh. Below is the wiring diagrams and schematics for the various hardware that is connected to the EnCore 2s robot.

Galil DMC 2143 and AMP-20540

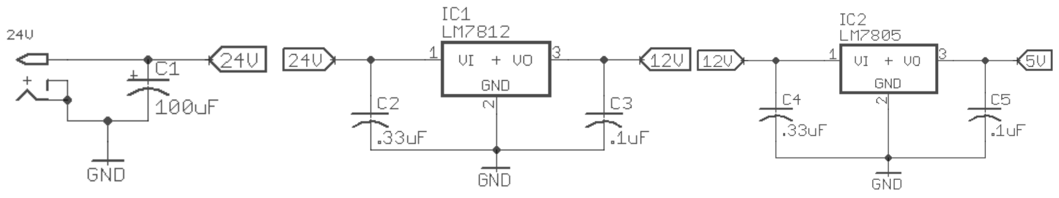


Galil Breakout board

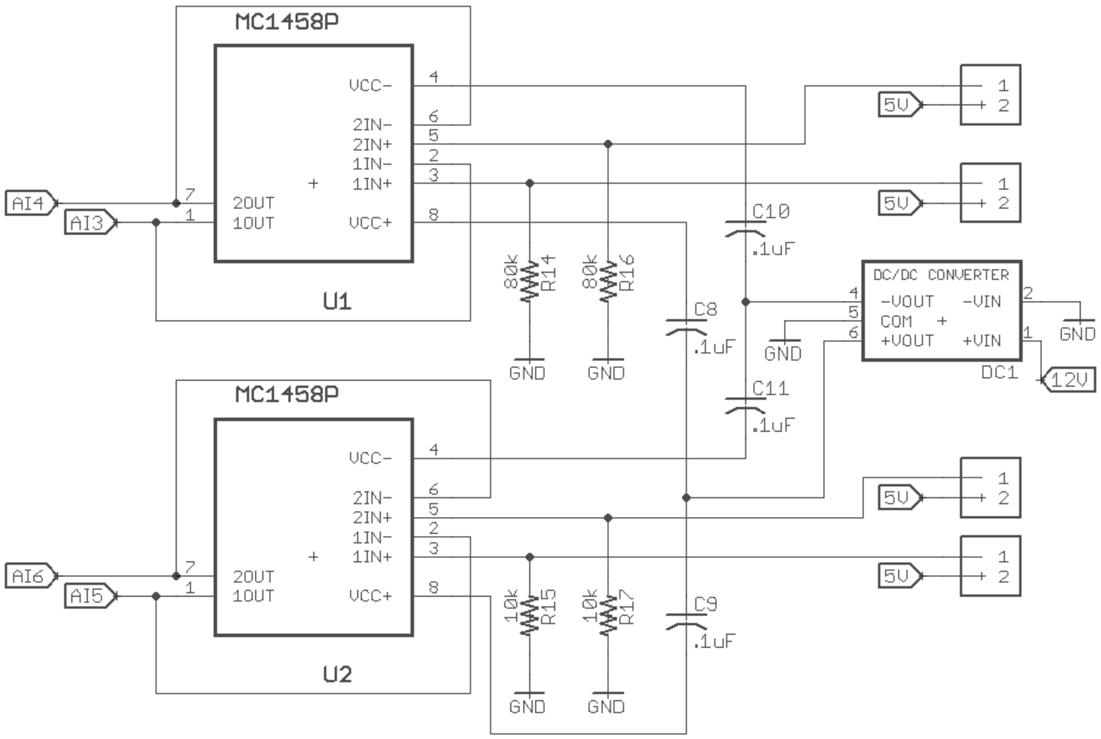


Galil Breakout Board Schematic

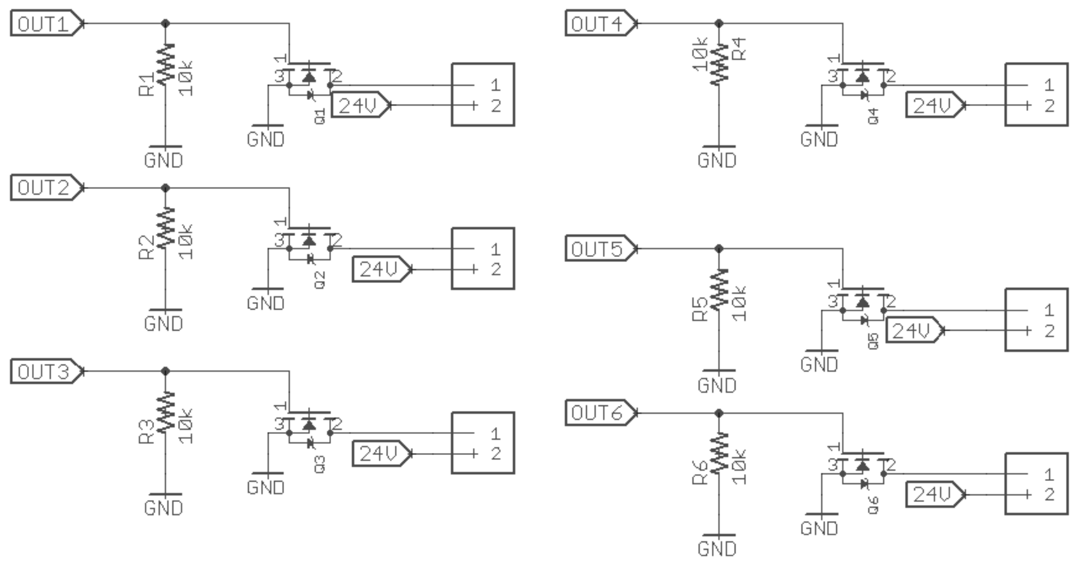
Power Circuits



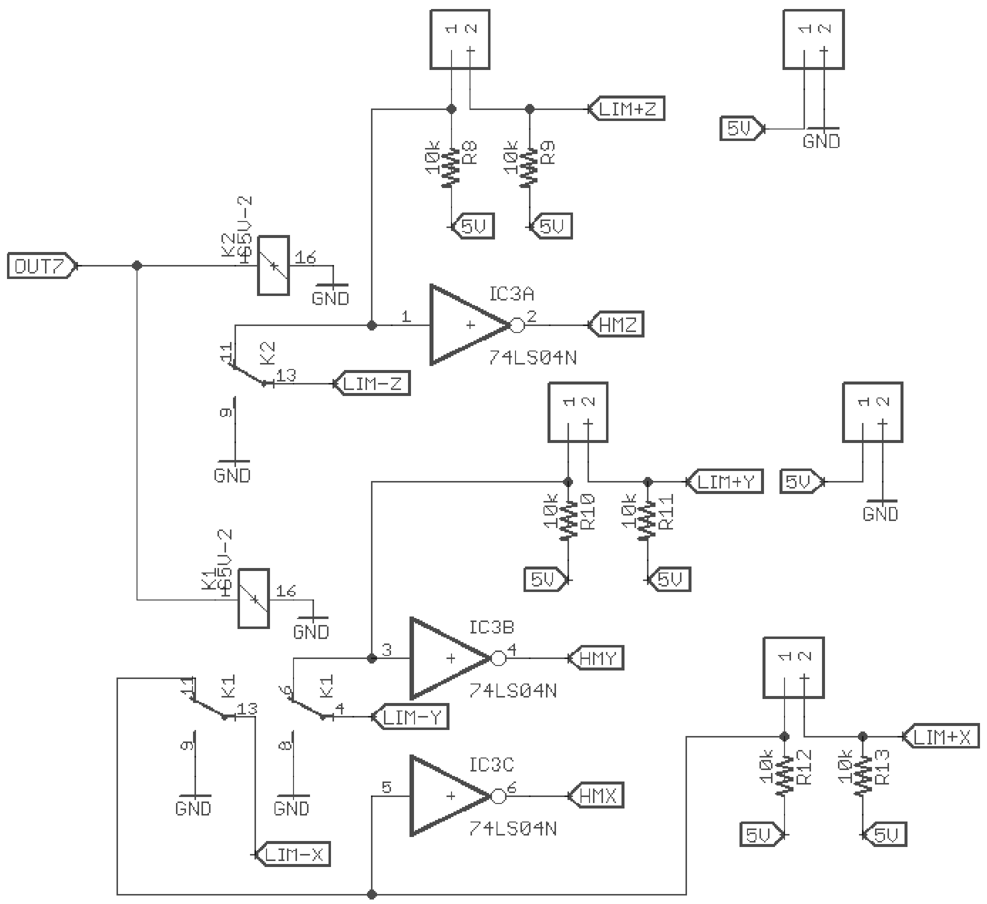
Amplifier Circuit



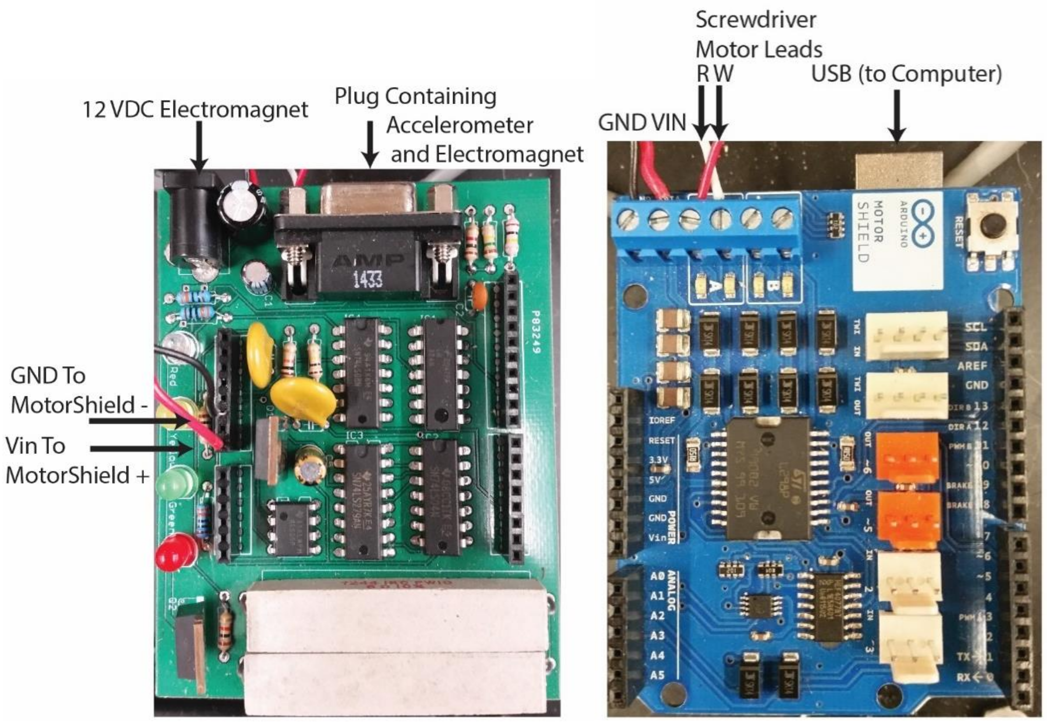
I/O Circuit



Homing Circuit

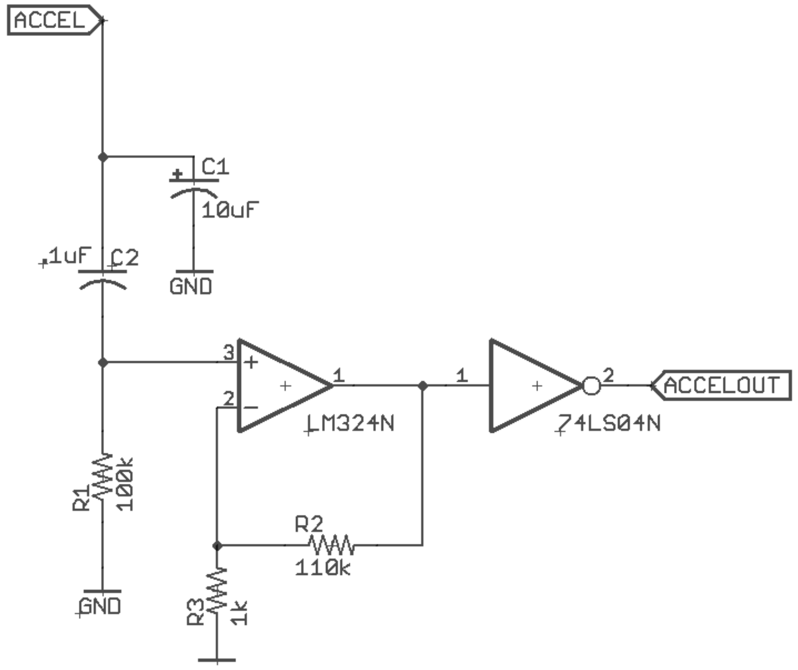


Arduino Controlling Screwdriver

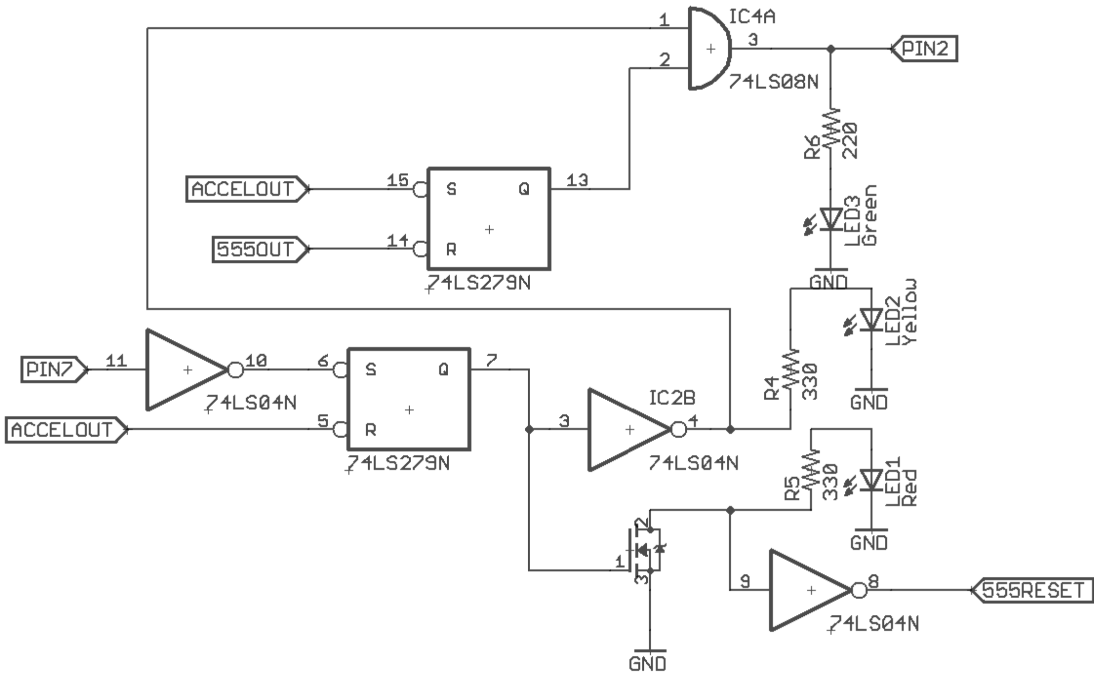


Arduino Screwdriver Schematic

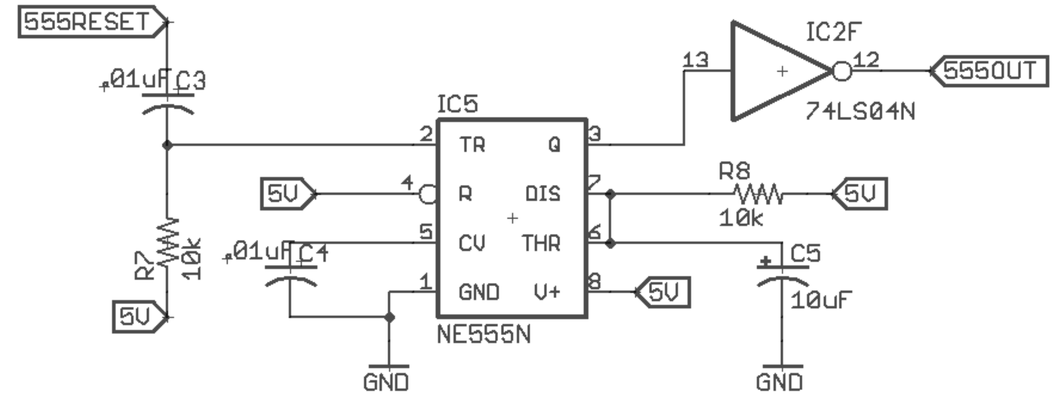
Amplifier



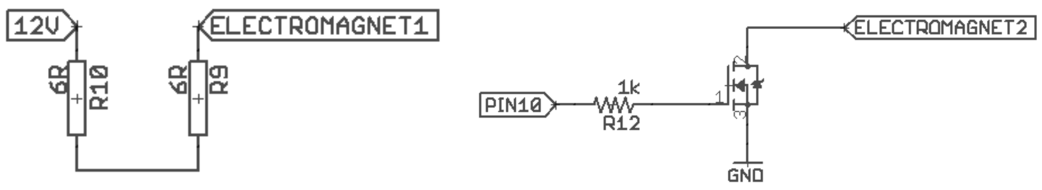
Set Reset



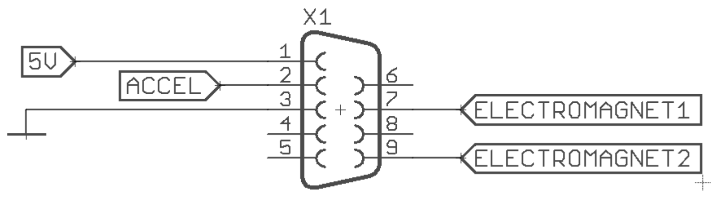
555 Timer



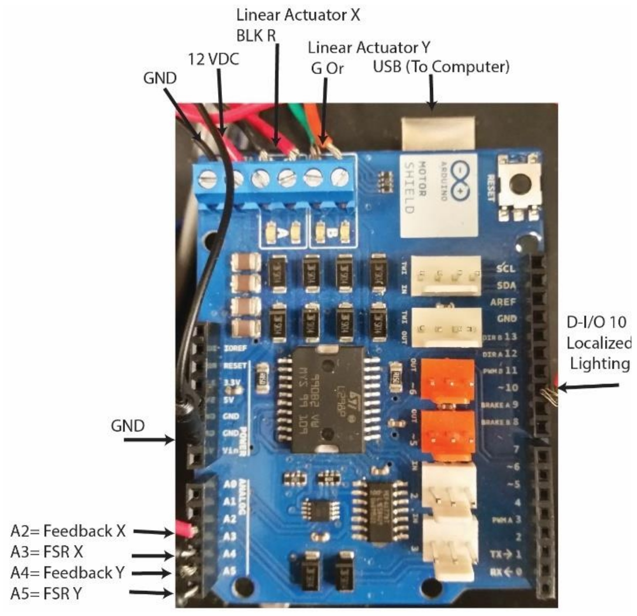
Electromagnet



Plug

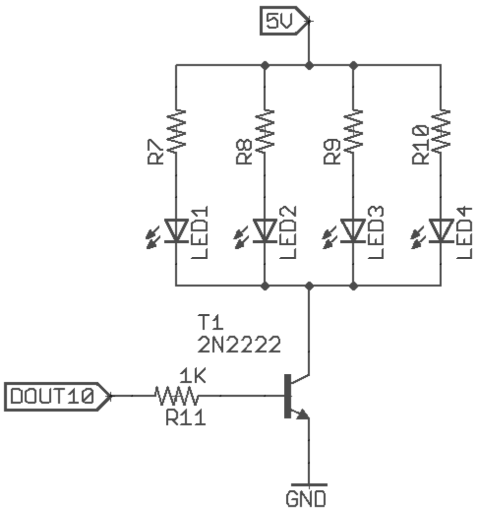


Arduino Clamp

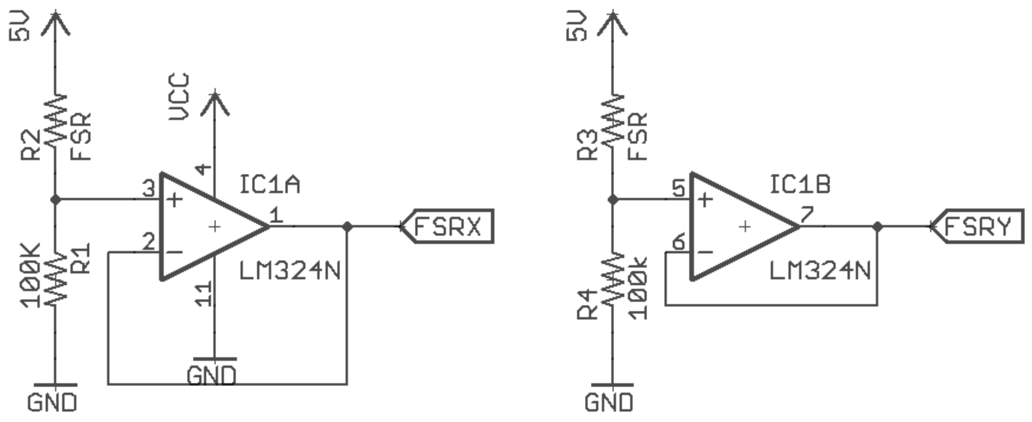


Arduino Clamp Schematic

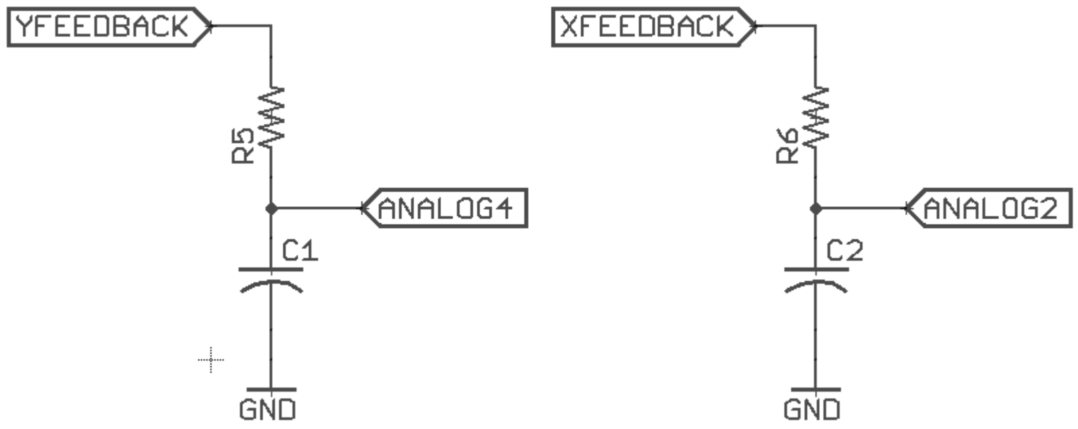
Lights



FSR Amplification



Feedback Filtering



APPENDIX C – Procedure for lens calibration to remove lens distortion

This appendix will detail lens distortion and the procedure for removing it using Matlab’s built-in calibration app. This section summarizes the Matlab help pages on distortion found at: https://www.mathworks.com/help/vision/ug/single-camera-calibrator-app.html. There is another camera calibration app for Matlab created by Jean-Yves Bouget, however, this section will not discuss how to use that software. Lens distortion is important to remove from a camera when trying to make precise measurements as the distortion could lead to significant errors. The highest amount of distortion will occur near the edges of an image and there will be little to no distortion in the middle of an image.

Matlab has the Camera Calibration app which is included with the Computer Vision System Toolbox that helps find the intrinsic parameters of the camera you are using to remove lens distortion. The two types of distortion that Matlab can correct for are radial distortion and tangential distortion. Radial distortion can either be negative (pincushion) or positive (barrel). Tangential distortion occurs when the camera sensor is not completely parallel with the camera lens. These three types of distortion are shown in Figure 1.

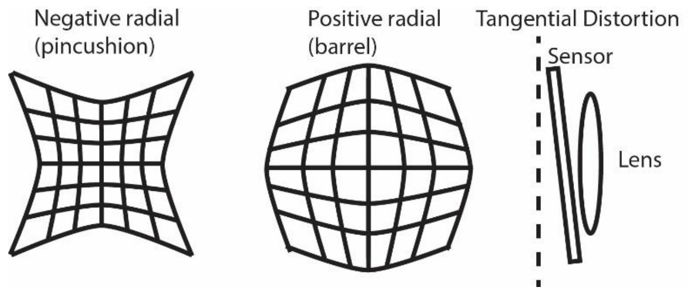
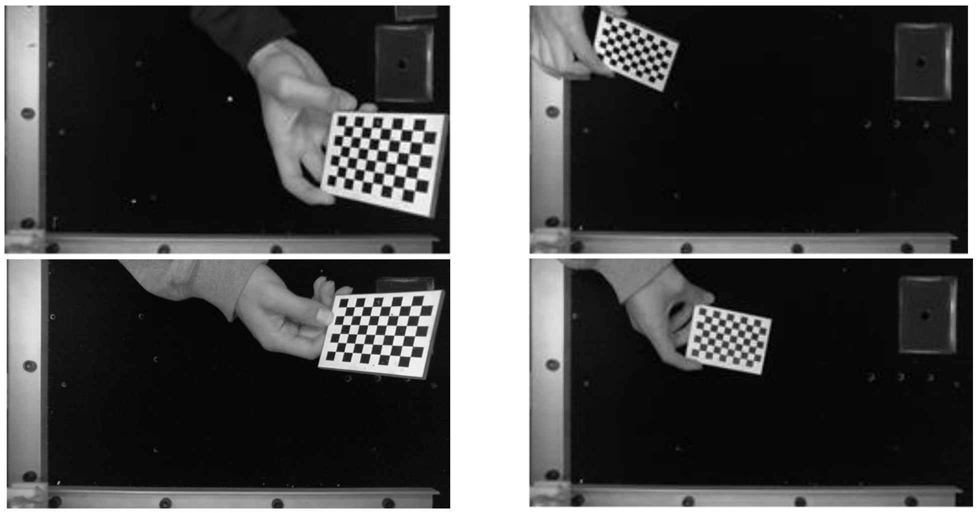


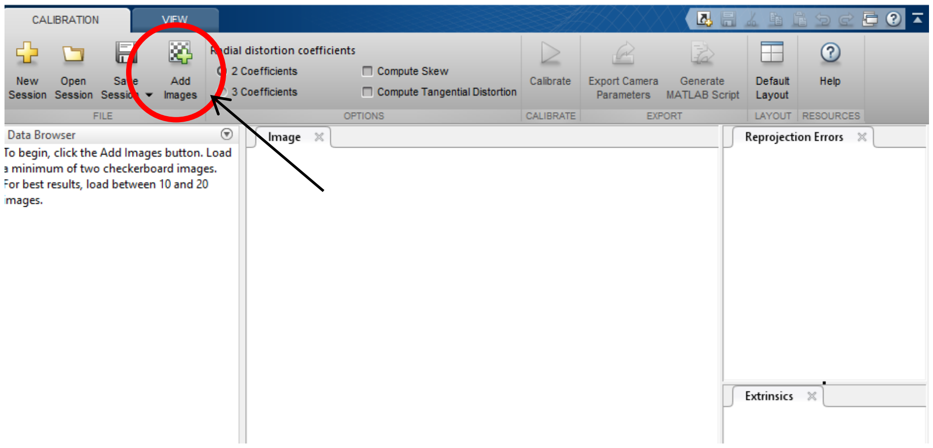
Figure 1. Types of lens distortions

Procedure to Remove Lens Distortion

1. First a checkerboard pattern must be created and the lengths of the squares must be known. The checkerboard should be mounted on a rigid surface and be perfectly flat. This research used a 7x10 square checkerboard with 8.5mm squares mounted on Plexiglass. This size checkerboard allowed for one side of the corners to be black squares and the other side be white squares. The checkerboard was printed on acid free paper and 3M super 77 adhesive was used to mount the checkerboard to the Plexiglass.
2. Next move the checkerboard around the workspace making sure to rotate it and take images with the camera you are trying to calibrate. It is important that the camera can always see all of the squares and, when taking the image, the checkerboard remains completely still. The Calibrator App requires at least three images to work, and Matlab suggests that 10-20 images work best. This research used 50 images since 20 images still produced significant errors. The tradeoff of using more images is the calibration takes longer. It is important to make sure that images are taken throughout the workspace especially near the edges.

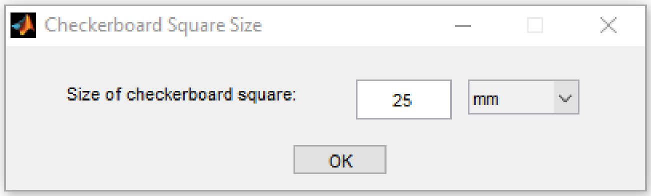


3. After the images have been obtained, start the Calibrator app by typing ‘cameraCalibrator’ in the Matlab command line.
4. Click ‘Add Images’ and select the images that you have just taken.

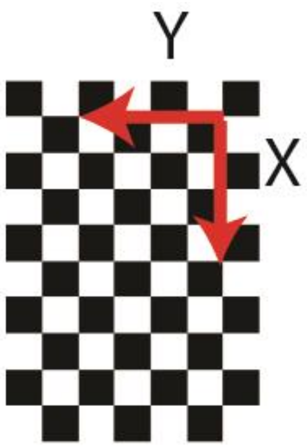


To begin, click the Add Images button. Load a minimum of two checkerboard images. For best results, load between 10 and 20 images.

5. Input the size of the checkerboard square. It is important that this is an accurate value since the calibration is dependent on it. After you hit enter, Matlab will automatically try to identify the checkerboard in each of the images. Some of the images will be discarded if checkerboard could not be found. It is also important to note that Matlab automatically defines the origin of each of the checkerboards at a black square (shown below) which is why it is important to have white squares on one side of the checkerboard and black on the other.



Size of checkerboard square: 25 mm



6. After all of the images have been checked to make sure that the checkerboard was correctly found, choose the radial coefficients that you want to correct for (2 or 3) and choose if you want to compute the skew and the tangential distortion. It is generally recommended to start with the default setting. If the results are not accurate adjust the settings and try to recalibrate. When you are done choosing what coefficients, hit calculate. The calibration used for this dissertation used 3 values for radial distortion and did not compute the tangential distortion. This calibrator app adjusts for radial distortion using Equation (1) and tangential distortion using Equation (2).

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (1)$$

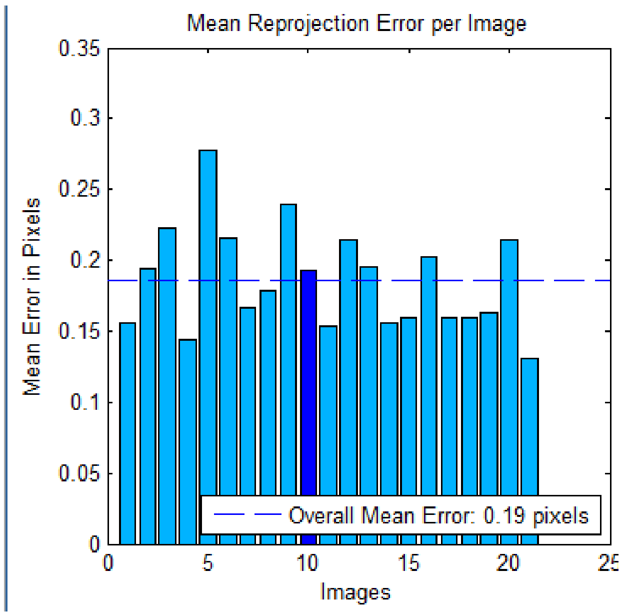
Where: x, y are undistorted pixel locations in normalized coordinates
k_n are the coefficients of radial distortion
and $r^2 = x^2 + y^2$

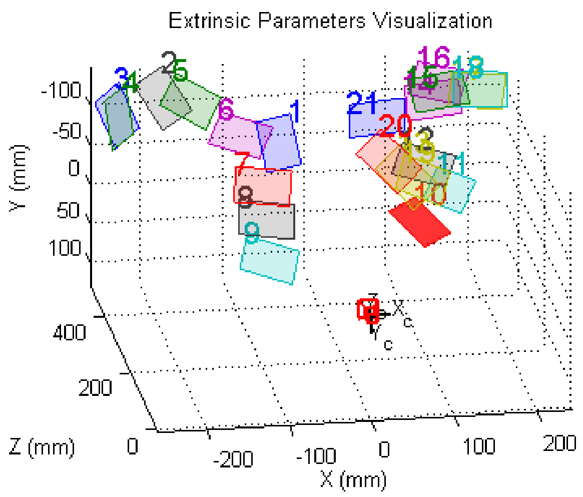
$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$

$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \quad (2)$$

Where: x,y are undistorted pixel location in normalized coordinates
$p_1$and p_2 are the coefficients of tangential distortion
and $r^2 = x^2 + y^2$

7. After the images have been calibrated, it is possible to look at the mean error of all the images by looking at the bar graph in the upper right corner. This error shows the re-projection error, or how far off the calculated corners of the checkerboard are from the actual corners. If a re-projection error of a certain image is very high, then that image should be discarded and the calibration should be run again. The extrinsic map shows an overview of where the checkerboard has been re-projected into the workspace. The camera is located at position Xc, Yc and this is a good way to check that a checkerboard has been placed in most of the locations of the workspace.





8. When the results are satisfactory, you can save the session and export the camera parameters to Matlab. You can then access the intrinsic matrix by *cameraParams.IntrinsicMatrix*. It should be noted that the Matlab inverts this matrix compared to other sources in the literature. You will need this matrix when using the *extrinsic* function in Matlab which is used to obtain the rotation and translation vectors that describe how the world is transformed relative to the camera.