Development of an Arduino Robotic Arm Implementing a Graphical User Interface

…

Branton Ramsey and Ayoko Toluwanimi

School of Computer Science, Northeastern State University, OK, USA

Email: ramsey13@nsuok.edu, ayokot@nsuok.edu

Abstract: Robotics is the future of mankind and has been used in industries, and even our homes, for many years, doing many of the same skills as humans and are touted to be faster and more accurate than humans. Although manufacturing is at the top of the list of uses, with technology advancing almost daily, robotic arms are now being used in a wide variety of other fields, such as space exploration, prosthetics for humans, the food industry, and even performing certain procedures during surgery. The objective for my project was to develop a robotic arm that can perform assigned tasks through input given through a graphical user interface (GUI). The assigned tasks were for the arm to pivot, grip an object, relocate it and then release the object, all accomplished by using an open source electronics platform on a system chip called Arduino. The chip consists of a programmable circuit board that runs on a computer and is used to write and upload computer code. Designing a GUI requires you to understand the function your program is trying to produce so that the interface can allow the user to input the functions they desire to fulfill.

Key words: Arduino, Robotics, Servo, prosthetics, GUI, graphical user interface, computer, computer science, C#, C programming language, Visual Studio.

1. Introduction

The most commonly used robot configurations are articulated robots, SCARA robots, delta robots, and cartesian coordinate robots (gantry robots or x-y-z robots). In the context of general robotics, most types of robots would fall into the category of robotic arms. The word robot is from the Czech word robota meaning “forced labor.”

The earliest known industrial robot, conforming to the IO definition was completed by “Bill” Griffith P. Taylor in 1937 and published in Meccano Magazine, March 1938. [1][2] The crane-like device was built almost entirely using Meccano parts, and powered by a single electric motor. Five axes of movement were possible, including grab and grab rotation. Automation was achieved using punched paper tape to energize solenoids, which would facilitate the movement of the crane’s control levers. The robot could stack wooden blocks in pre-programmed patterns. [2] Our robotic arm will use a graphical user interface with sliders to control seven servo motors that will allow the arm to rotate 180 degrees (90 in each direction), pivot, and grab or relocate objects.

User interface design closely resembles the objectives of the system in which you are trying to design. An interface is a collection of mechanisms that will be used to interact with the system it is implementing. Most individuals have used a GUI in some form or another in their lifetime, for example your desktop computer or your smart phone both have these implementations of a GUI. These generally include windows, icons, and menus that can be manipulated by a mouse or your finger in the case of a smart phone. The advantages of a GUI make implementation more effortless, simplifying the process and making it easier for the user at hand. [3]

There are similar projects that have been performed recently. There is a specific one where a person created a robotic arm in order to assist Crescent Steel & Allied Products Ltd, which is a private organization, located in Nooraibad. The company manufactures large diameter spiral arc welded steel line pipes and is a pipe coating facility capable of applying multi and single layer of high-density polyethylene internal and external coatings. He designed an automated

system to achieve this without using human work force in order to show that automated technology is important. [31]

Another similar study was performed in order to create a robotic arm to assist in tasks which humans can not perform or in conditions too extreme for humans to work in. The main difference in this person's robotic arm is that his is programmed to allow you to input a specific degree and the robot servo motor will rotate to that exact degree. [32]

This can have its advantages and disadvantages. If you know the exact degree you need the arm to go to, then this is probably a faster and more exact implementation. If you do not know the exact degree you need the arm to move to, then this could cause problems. Our method will allow you to move a track bar slider until it has reached the desired position, and has a label telling you what degree the servo motor is on allowing you to visualize this for future reference.

2. Arduino Overview

Arduino is an open source electronics platform on an easy to use system chip. It can read inputs and follow a set of instructions to the microcontroller on the board. It consists of a programmable circuit board that runs on your computer. It is used to write and upload computer code to the physical board to perform programmed instructions. For this project we have chosen to use the Arduino Uno board specifically. [4] [16]

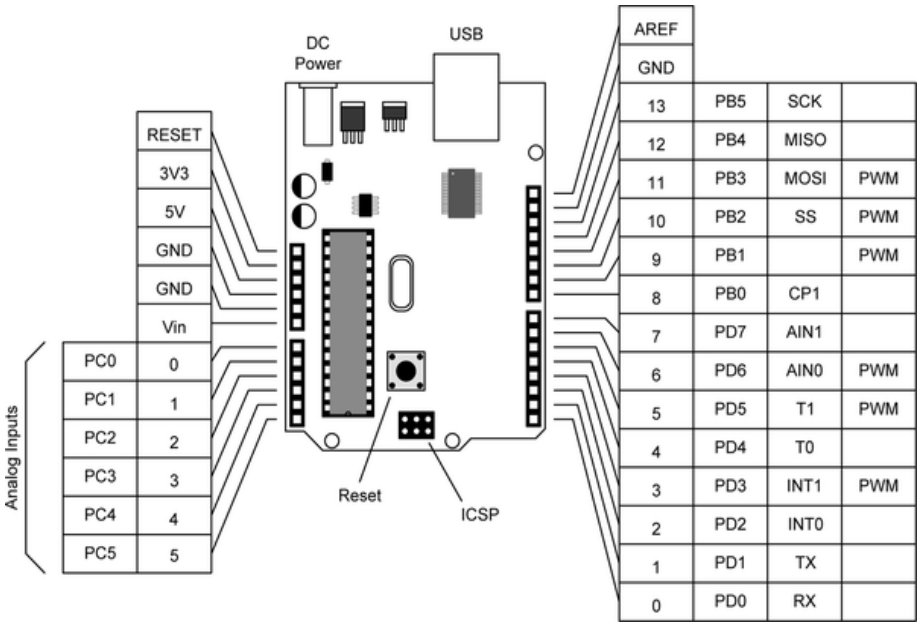

Fig. 1. Arduino Pinout Diagram

This diagram shows an overview of the specific Arduino used in this project. [10] [33] The board has pins to which you connect wires to send an electrical current to the circuit board. [4] In some circumstances you will need to use a power shield in order to get a large enough current to run through to your Arduino. [12] [16]

In some instances, your Arduino project may need more than the two-voltage supply that it comes with. In these situations, you will need the aforementioned Power Supply Shield. This allows a voltage of up to 12V in some cases. The power shield has a built-in voltage measuring function which displays the voltage output to Arduino pin 0. [12]

A Servo motor is a self-contained electrical device that rotates or pushes. Servos can be used for many things such as, but not limited to, helping your robots walk, moving remote controlled vehicles, or spinning objects. Servos range in size and there are many different variations of them. They are constructed from basic parts such as a motor, potentiometer that is connected to the output shaft, and a control board. The output shaft can be moved to specific degree-based positions by sending the servo an electrically coded signal. [13] [16]

We utilized the Arduino to power the arm and rotate all of the motors. The motors being used are Kuman KY620 Servo Motors. They spin in both a clockwise and counter-clockwise direction. The robot can rotate in a circular manner and move each joint successfully as well as open and close its grip to grab objects.

Thanks to the development of robotic arms, they can help make tasks easier and execute repetitive and precise movements with more accuracy than a human could achieve. They can even operate in conditions that may be deemed too hazardous for humans to operate under.

3. Pseudocode

Pseudocode was written to map out the process so that the team was on the same page and everything was laid out step by step. This lays out all of the possible scenarios of inputs that the user is capable of giving the GUI and what will result as the outcome.

1. User moves the interface slider.
2. Interface slider indicates the specified degree.
3. Robot arm moves to the corresponding position (in degrees).
4. User clicks reset button.
5. Servo motors reset to default position (90 degrees).

Pseudocode is a written list in human language that is a set of instructions that describe the executions a program will make when the code is finished in a programming language. The word “pseudo” means fake, so it means fake code. The algorithm of pseudocode is to set a subset of directions so that programmers and non-programmers alike can understand what the overall objective of the software is. [11]

An advantage is that most people cannot read programming languages so it allows for the software developers to communicate with their clients, such as business analysts, to review the steps to create the specifications of the program. Disadvantages of using pseudocode would be that, although it may be easy to read, it I not as helpful as a use case diagram or a flow chart would be to a programmer. Pseudocode can also cause nonprogrammers to misunderstand the complexity of a software project thus undervaluing it and not funding or allowing enough time as needed. [11]

4. Use Case Model

The team discussed and developed a use case diagram involving all of the initial input and output commands for our project. The objective is to develop a robotic arm that can perform assigned tasks through input given through a graphical user interface (GUI). These assigned tasks would be to pivot a joint, close its claw to grip an object, turn to move the object once it is within its grip, and drop

(relocate) the object once moved to its destination. The use case diagram shows that the user can give input to move the arm to a position, then lift the object. By doing this, the robot arm performs the task along with other tasks such as rotate or pivot the arm and exits the application.

This use case diagram below gives a visual representation of the potential inputs of the project and how it interacts with the user.

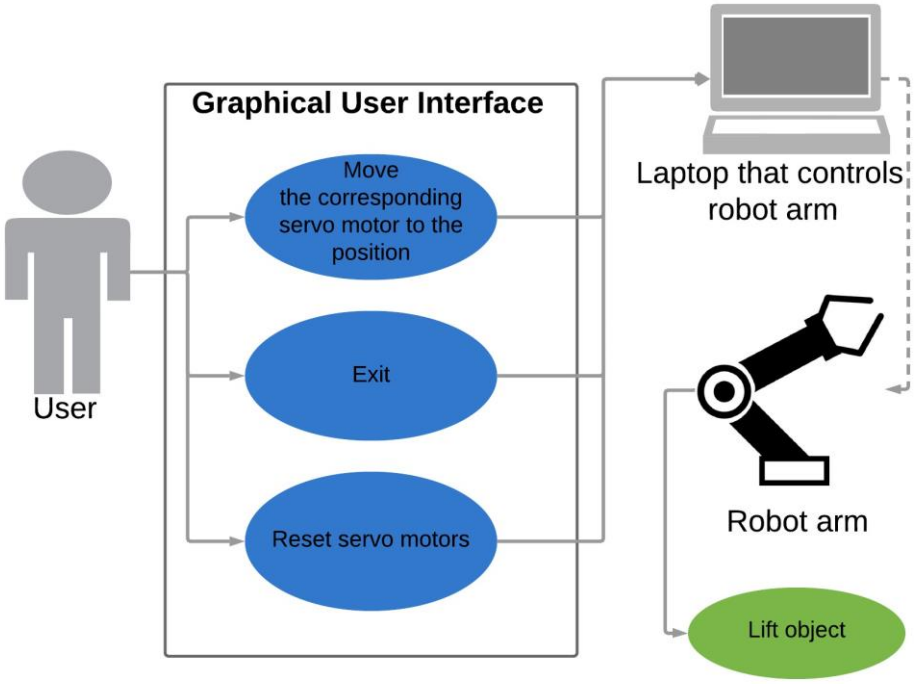

Fig. 2. Use Case Model

Use case diagrams are sometimes referred to as behavior diagrams used to describe a set of actions (use cases) that the software should be able to perform once completed. They generally include one or more external users of the system (known as actors). Each use case should include some observable result to the actors of the system. [14] [29]

5. Assembly of the Arm

We used a laser cutter in order to cut the needed shapes out of a piece of wood. A 3D printer was used in order to make the plastic joints that fasten onto the Servo motors which allows the motors to attach to the wooden pieces so that they will rotate. [28]

The image below is a diagram of the blueprint that was used to allow the laser cutter to cut out the pieces properly from a piece of wood.

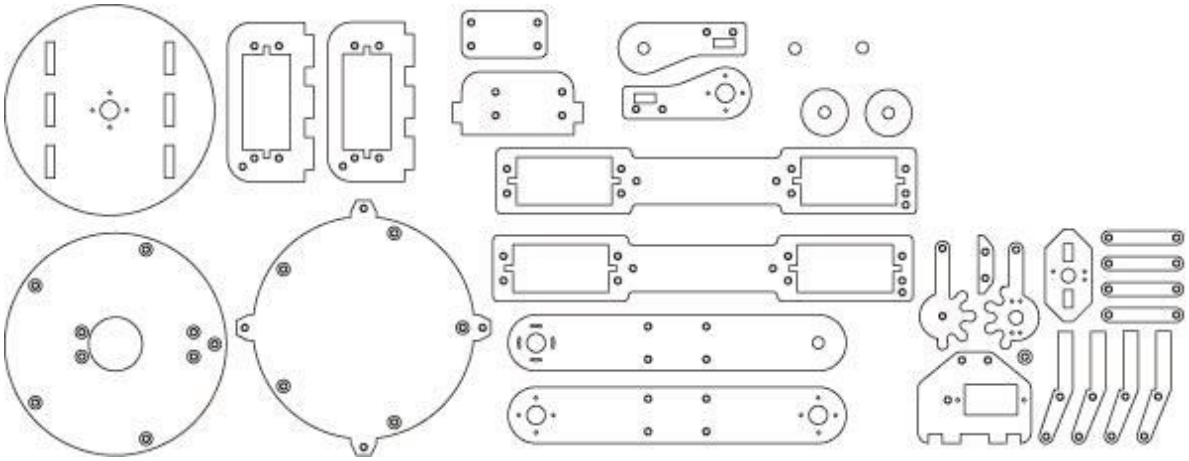

Fig. 3. Arm Cutout Diagram

A 3D printer was used to print out circular pieces so which set on the joints to allow smooth rotations of the motors with as little friction as possible. Next Gorilla Glue and screws were used to fasten most of the pieces together so that they would be durable enough and not break easily. [23]

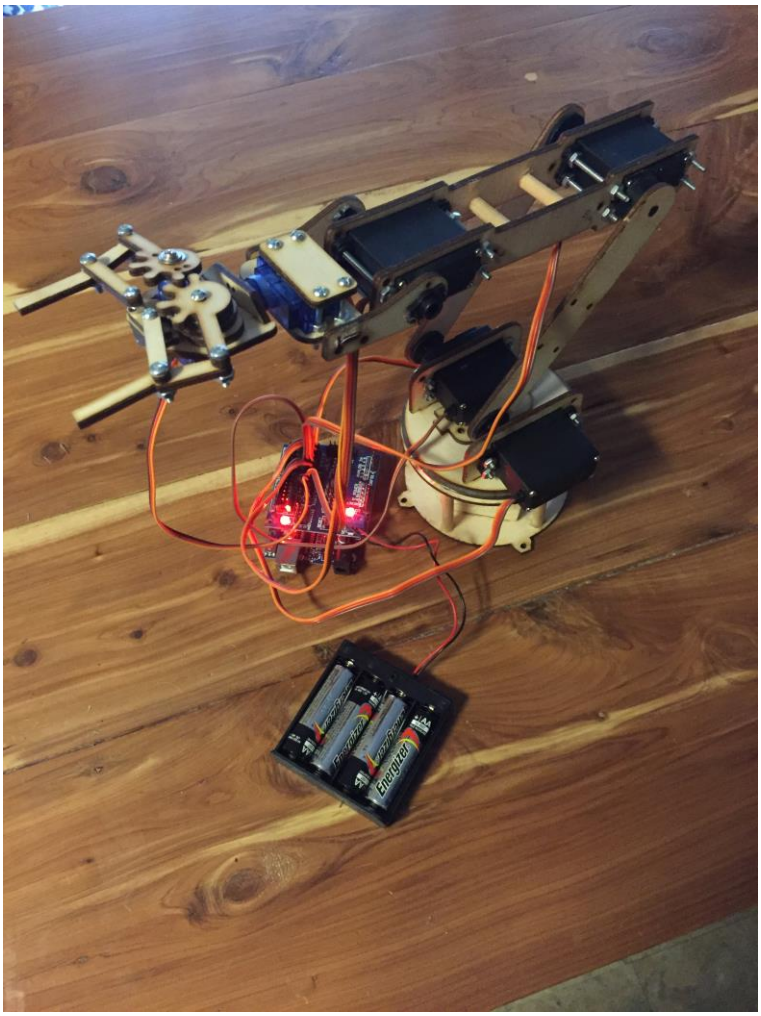

Fig. 4. Servo Arm

We used cylindrical pieces of wood for support beams at the base of the arm in which it will rotate on. These had to be cut to size, sanded, and then glued into place. A plastic lid was cut into a circle in order to create less friction for the base when it spins on top of it. Later testing proved that was not necessary because due to the motor lifting the joint high enough that it does not rub against the base. This caused a gap between the platform the motor set on and the actual base of the robotic arm. This gap caused the arm to shake when it rotated resulting in the arm falling completely off of the base because it was very top

heavy. The predicted solution was to find some sort of flat circular surface that is smooth enough that it does not cause friction but thick enough to fill the gap. We took an old computer that we had and removed the hard disk out of it and used it for this purpose.

The forearm also needed thin cylindrical pieces of wood cut out and placed in between in order to give it extra durability and support. Screws were also used to pinch the two pieces of wood around the motors for support. These screws were too long and blocked the wrist of the robotic arm from rotating a full 180 degrees. The team used a hack saw in order to trim these bolts since they were too long. This allowed the wrist to properly function as intended.

The servo hand uses a gear system to open and close the grip of the hand. The teeth of the gears lock together and move cohesively to close and release the grip which allows it to pick up and move objects when needed.

The wires on the servo motors are fairly short and work best if the Arduino is secured to the base underneath the arm.

6. Design of Graphical User Interface

The team has made the decision to use Visual Studio in conjunction with the Arduino Nightly IDE (integrated development environment) in order to complete a GUI (Graphical User Interface) for our robotic arm. In order to allow Visual Studio and Arduino Nightly to communicate amongst one another, the following line of code has to be imported for the library to function properly. [24]

Using System.IO.Ports;

Our GUI will allow the user to move sliders on a graphical interface to control how far the servo motor will move the robotic arm. [27] Each slider will have a label beneath it displaying the specific degree at which the corresponding servo motor is currently on allowing for more precise movements from the user. It will also have a reset button that will permit the user to reset the servo motors back to

the default 90 degrees, allowing it to rotate 90 degrees each direction totalling to the aforementioned 180 degrees combined.

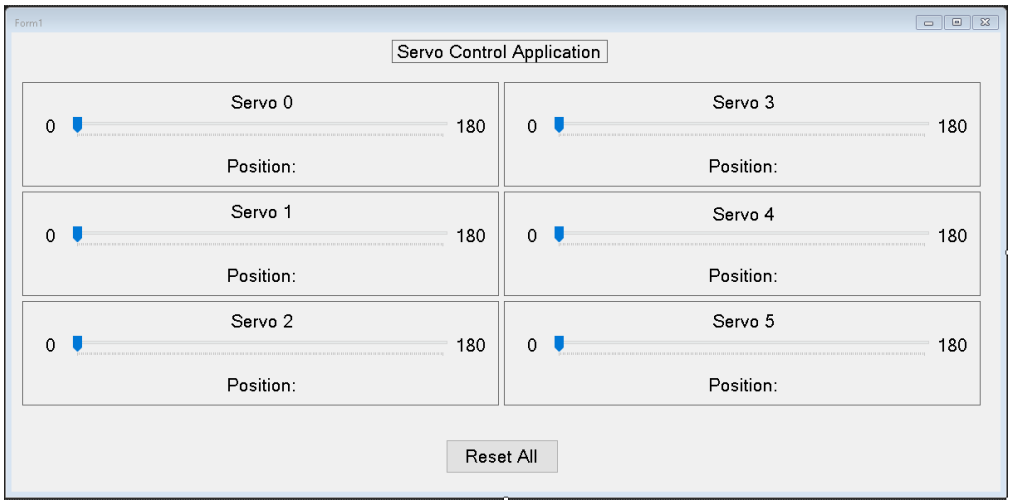

Fig. 5. Graphical User Interface

A GUI is an interface that uses icons or images to allow the user (generally a nonprogrammer) to interact with electronic devices or programs, rather than doing so by using text-based commands via a command prompt. They are also more visually pleasing closely resembling the image depicted above instead of multiple lines of text, thus also resulting in faster execution times removing the need to type lines of command, instead just using a mouse to click or interact with buttons (or in this case sliders) on the screen. Another example would be that a Microsoft Windows desktop is a GUI, whereas MS-DOS is a command line. They work by using icons and menus to carry out commands, such as opening, deleting, and moving files. They are primarily navigated using a mouse but a keyboard can be used in some cases. [15]

7. Code Implementation

The team decided that, due to familiarity with Visual Studio, that it would be the IDE of choice for the project. It will be used in conjunction with Arduino Nightly IDE in order to communicate with the Arduino and construct a GUI for the project. The following code will be split up between Arduino and Visual Studio code segments for their corresponding IDE.

7.1 Arduino Code

#include <Servo.h>

This imports the *Servo* library so that it can be utilized.

Int servoCount = 7;

Int servoPins[] = {2, 3, 4, 5, 6, 7, 8};

Servo servos [7];

The snippet of code allows you to get the number of servos you wish to implement into your code, set which pins you wish to use them on, and the last line puts the servos into an array.

AttachServos();

This is the method call for the function that attaches the servo to the pin within the array.

int channel;

int pos;

```
channel = Serial.readStringUntil(':').toInt();
pos = Serial.readStringUntil('*').toInt();
servos[channel].write(pos);
```

This is the serial event method which declares the channel and position as integers. The *readStringUntil()* reads characters from the serial buffer into a string. The function will end up terminating if the terminator character is detected or it times out. This specific one searched for ":" and will terminate as soon as it finds it. It then converts it into an integer and sets it to the channel. The last line writes the current position of the servo motor and holds that in storage within an array named "*servos*" to be used later.

```
void AttachServos()
{
  for(int i = 0; i < servoCount; i ++)
      {
     servos[i].attach(servoPins[i]);
      }
}
```

This is the method called *AttachServos()*. Within the for loop, it sets an integer named "*i*" and sets it to zero. When "*i*" is less than the servo count, it increments the integer up by one. The last line pulls servos from the array that is created within the Arduino code section. It then attaches them to the corresponding pins listed in the Arduino in ascending order.

7.2 Visual Studio Code

```
private void SendServoInfo(int channel, int pos)
        {
            string message = channel.ToString() + ':' +
pos.ToString() + '*';


            try
            {
                serialPort.Write(message);
            }
            catch
            {
  }
     }
```

The above is a method for *SendServoInfo()* which will be used numerous times later in the code so we will go ahead and establish it here. It is passed two integer variables in channel and pos. The first line within the method converts the pos and channel into a string and adds a char of either '*' or ':'and declares it equal to a string called message. The last part of the code creates a try catch and within it writes the message to the serial port. Previously we stated that within the *serialEvent()* method of the Arduino side of the code that there was a *Serial.readStringUntil()* which looked for the specific char within it's parenthesis, which this does the same. As soon as it reads the specified char it stops. [5]

```
private void ResetServos()
        {
            int centrePosition = 90;
            trackBarServo_0.Value = centrePosition;

            labelServoPos_0.Text = "Position: " +
centrePosition.ToString();

            for (int channel = 0; channel < 6; channel++)
            {
                SendServoInfo(channel, centrePosition);
            }
        }
```

This is a method call for *ResetServos()*. The first line within the method declares an integer called *centrePosition* and sets that equal to 90. This makes the default (starting) position 90 degrees whenever the program is first executed and when the reset button is pushed. Below the integer declaration it sets the *trackBarServo* value equal to the *centrePosition* which is 90. The following line resets the corresponding label at the same time. The for loop sets the channel to zero, then for each time the channel is less than six it increments it up by one. Within it, the *sendServo* method, which contains the values *channel* and *centrePosition*, are implemented again. [5]

```
private void trackBarServo_1_Scroll(object sender, EventArgs e)
        {
            int servoPos = trackBarServo_1.Value;



            if (serialPort.IsOpen)
            {
                labelServoPos_1.Text = "Position: " +
servoPos.ToString();
                SendServoInfo(1, servoPos);
            }


            if (serialPort.IsOpen)
            {
                labelServoPos_1.Text = "Position: " +
servoPos.ToString();
                SendServoInfo(2, 180 - servoPos);
            }
        }
```

We use the first line to create a method call which is used whenever the user moves the scroll bar on the screen within the GUI (Graphical User Interface). The *trackServoBar_1* indicates which trackbar slider you are wanting to correspond with this code. The second line makes the servo position equal to the distance you have moved the slider in single degree increments. The if statement then checks to see if the port is open, and if the port is open, then it executes the following code within the statement. *labelServoPos_1.Text* indicates which label will be changing to show the degree to which the servo has moved to. It takes the position from the corresponding channel established in the Arduino side of the code and converts it to a string. Using multiple channels allows us to set up

different pairs of motors so that others do not move when we are not trying to move them. The last line of the if statement then returns the servo info back. Within the parenthesis, the 1 is the servo motor number we are trying to move and it changes its servo position. The specific code above has two if statements which means that there are two motors working as a pair to accomplish the exact same goal, thus resulting in the second if statement containing the number 2 instead of 1. *TrackBarServo_1_Scroll* is repeated to total six times, one for each set of the servo motors on the arm, each with their own corresponding number to help differentiate them from one another. There is a total of seven motors, but two of them work as a pair, so they get two if statements within the same method instead of their own individual one. [6] [8]

If at any point there is a reason to make one servo motor work at opposite from the other in a pair, then you would make the last if statement look similar to the second one in the code snippet above. The crucial difference is the fact that it has 180 – *servoPos* and to get the opposite you just subtract the current servo position from 180 which will give you the remaining degrees, thus resulting in a mirrored effect. [6] [8] [25]

```
private void buttonResetServos_Click(object sender, EventArgs e)
        {
            ResetServos();


        }


        private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
        {
            ResetServos();


               }
```

Finally, these are the methods which reset the servos to default (90 degrees) when the form is opened, closed, or the reset button is clicked.

8. Conclusion and Future Work

Future endeavours will include attempting to move this to a mobile format through Wi-Fi or Bluetooth Arduino modules. Visual Studios has a strict requirement of having to be near a computer in order for the program to work. Allowing the arm to be used more remotely would give the project more mobility, make it less bulky, and allow it to be used in more ideal situations. Most everyone now has easy access to a mobile phone which would allow our program to be used by a person anywhere as long as the arm can be controlled by an app made for iOS or it could be created in Android Studio to be used on android devices have access to Wi-Fi or Bluetooth capabilities. [26]

The team could add wheels to the robotic arm so that it can have mobility in order to operate at a distance from the user in harsh or extreme conditions. A real-life scenario where this would come into play is if law enforcement or military personnel need to disarm a bomb. When it comes to the field of robotics, wheels are used when speed, accuracy, and stability are all a needed attribute so long as the goal is to move the robot from one location to the next. It would gain control, stability, and manoeuvrability with this development. While the wheels can make the robot move, a steering system would need to be implemented. [17] If the robotic arm is too far away from the person controlling it, obstacles in all directions may not be visible. In this scenario, the arm would also need a guiding system to detect objects in the way so it could avoid them. [18]

Bump sensors are a type of sensor that use whiskers or a plastic bumper to tell a robot when it has collided with an obstacle. Usually they work by having a sensor activate a mechanical switch, but in the case of the whisker the sensor can tell how far the whisker has been bent and therefore how far into the obstacle the robot drove. This allows the robot to then back up and find a different approach to

navigation. [19] This sensor has the perk of being extremely cost efficient. However, this type of sensor would not totally solve the problem. We would like to prevent the collision in the first place. [30]

Radar would be another means that might solve this issue. Radar works by using a piece of equipment called a magnetron. This magnetron sends out radio waves which travel and bounce off objects. An antenna then receives these signals and gives an estimate on how far away the object is from the radar location. [22]

Lidar sensors are a form of radar sensor that allow robotic cars to navigate. These sensors allow cars to see valuable information about what the world looks like around them. [18] Lidar gives a continuous 360 degrees of visibility and highly accurate depth perception. This type of sensor works by firing off millions of beams of light per second, which enables a truly 3D visualization for the world around the sensor. You can practically make exact measurements of any object around it because it works almost like a sonar. [20] Below shows a diagram of what the output result of a Lidar sensor shows. [21]

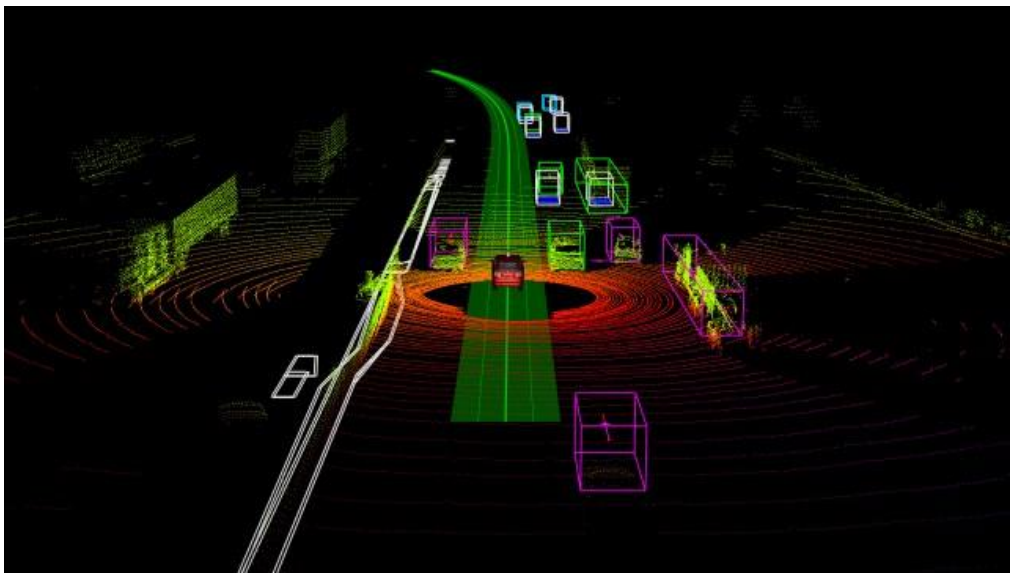

Fig. 6. Lidar Image

Infrared range sensors are another type that we would be interested in looking into. These sensors use infrared emitters and receivers. The receiver waits for the emitter to send out a pulse of infrared light, and if the same pulse is detected on the receiver within a set amount of time, then it knows there is an obstacle in the way ahead. A more advanced version of this transmitter can detect the amount of time it took to receive this pulse and know exactly how far away it is from the object so that it can make adjustments in advance to avoid.

References

[1] Simon, Mat. (2018) "The Wired Guide to Robots." https://www.wired.com/story/wired-guide-to-robots/

[2] ("Industrial Robot," n.d.) http://ses-engineering.biz/Robot/robot-industrial

[3] Desai, Karen. (2012) “Development of a Graphical User Interface for a Robotic Manipulator with Example Acquisition Capacity.” http://digitalcommons.ryerson.ca/dissertations

[4] (“Arduino,” n.d.) https://www.arduino.cc/

[5] (“Get Started with C# and Visual Studio Code,” 2018). https://docs.microsoft.com/en-us/dotnet/core/tutorials/with-visual-studio-code

[6] (“TrackBar Class, 2018”) https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.trackbar?view=netframework-4.7.2

[7] Lindsey, Jerry. (2016) “A Servo Library in C# for Raspberry Pi 3 Part #1, Implementing PWM.” https://jeremylindsayni.wordpress.com/2016/05/08/a-servo-library-in-c-for-raspberry-pi-3-part-1-implementing-pwm/

[8] (“Control Servo Motor from Windows Form Application using Serial Communication,” n.d.) https://code.msdn.microsoft.com/vstudio/Control-Servo-Motor-from-a61c06ac

[9] (“Control Two Servos from Windows C# Application with Arduino Uno,” 2018) https://stackoverflow.com/questions/52151044/control-two-servos-from-winfroms-c-sharp-application-with-arduino-uno

[10] (“Arduino Uno Pinout Diagram,” n.d.) https://www.jameco.com/jameco/workshop/circuitnotes/cn-arduino-uno.html

[11] Pogue, Linda. (n.d.) “What Are the Advantage & Limitations of Pseudocode?” https://www.techwalla.com/articles/what-are-the-advantages-limitations-of-pseudocode

[12] (“Power Shield,” n.d.)
https://www.dfrobot.com/wiki/index.php/Power_Shield_(SKU:DFR0105)
[13] (“What is a Servo Motor,” n.d.)
https://www.jameco.com/Jameco/content/servos.html
[14] (“UML Use Case Diagrams,” n.d.)
https://www.uml-diagrams.org/use-case-diagrams.html
[15] (“GUI,” n.d.)
https://www.computerhope.com/jargon/g/gui.htm
[16] (“Pololu Maestro Servo Controller User’s Guide.” n.d.)
https://www.servocity.com/files/index/download/id/1456873301/
[17] (“Automotive Machines and Design Principles for Robotics Teams and Future Engineers,” 2017)
https://www.thezebra.com/insurance-news/4451/automotive-machines-design-principles-robotics-teams-future-engineers/
[18] Said, Carolyn. (2017) “Building Smarter Sensors for Robot Cars.”
https://www.govtech.com/fs/transportation/Building-Smarter-Sensor-for-Robot-Cars.html
[19] Krause, Spencer. (2017) “Choosing the Best Sensors for a Mobile Robot, Part One.”
https://www.sensorsmag.com/components/choosing-best-sensors-for-a-mobile-robot-part-one
[20] Cameron, Oliver. (2017) “An Introduction to LIDAR: The Key Self-Driving Car Sensor.”
https://news.voyage.auto/an-introduction-to-lidar-the-key-self-driving-car-sensor-a7e405590cff
[21] Hewitt, John. (2015) “A Laser and a Raspberry Pi Can Disable a Self-Driving Car.”
https://www.extremetech.com/extreme/213517-a-laser-and-a-raspberry-pi-can-disable-a-self-driving-car
[22] Woodford, Chris. (2018) “Radar”
https://www.explainthatstuff.com/radar.html
[23] jjshortcut. (2010) “Robotic Arm with 7 Servos.”
https://www.thingiverse.com/thing:2433
[24] Guru and Kid. (2017) “Arduino Interface with Visual Studio 2012 Visual C# Easy.”
https://www.youtube.com/watch?v=GLzFWm7_U6k

[25] McKenzie, Ian. (2016) “Two Servos.” https://www.youtube.com/watch?v=e5bjH26fqDQ
[26] (“How to Use Your Android to Communicate with Your Arduino”) https://www.digikey.com/en/maker/projects/how-to-use-your-android-to-communicate-with-your-arduino/aed1f8e3fa044264a4310c6b3b2a4364
[27] (“Designing GUI Applications with Windows Forms”) http://www.informit.com/articles/article.aspx?p=101720&seqNum=17
[28] (“3D Printing Basics”) https://3dprintingindustry.com/3d-printing-basics-free-beginners-guide
[29] (“Lucid Charts”) https://www.lucidchart.com/
[30] (“How Does a Touch Sensor Work?”) https://www.teachengineering.org/lessons/view/umo_sensorswork_lesson03
[31] Allahyar, Arsalan. (2013) "Automation: A Robotic Arm." https://www.academia.edu/14933221/Automation_A_Robotic_Arm_FYP_Thesis
[32] Faravar, Arian. (2014) "Design, Implementation and Control of a Robotic Arm Using PIC 16F877A Microcontroller." https://pdfs.semanticscholar.org/10f8/ebe4038ba1363e813d4bb6a9004b1a99e88d.pdf
[33] Hughes, J. M. (2016) “Arduino: A Technical Reference.” https://www.oreilly.com/library/view/arduino-a-technical/9781491934319/ch04.html

Bios

Branton, Ramsey

Bachelor of Computer Science with Minor in Information Systems at Northeastern State University, Tahlequah, OK, USA. Associate of Arts in Computer Information Systems at Carl Albert State College. Graduated from Poteau High School.

Research: Arduino robotic arm and graphical user interface application development.

Email: ramsey13@nsuok.edu

Ayoko, Toluwanimi

Undergraduate student at Northeastern State University, Tahlequah, OK, USA.

Attended Victory Christian High School in Tulsa, OK. Studied Computer Engineering & Information Systems at Oral Roberts University for two years. Bachelor' Degree in Computer Science and minor in Mathematics.

Research: Arduino robotic arm and graphical user interface application development.

Email: ayokot@nsuok.edu