# An Intelligent Pick-and-Place Robot

# Abstract

3D printing provides a quick and affordable prototyping for designing new robots. Using this software this study will design and implement a pick and place robot with five degrees of freedom, controllable through keyboard key presses and with the ability to remember what instructions it has been given in order to repeat actions.

# Introduction

Within industry robots that can be programmed to automatically repeat and taught or pre-programmed action continuously, precisely and efficiently is important. These types of robots may have different degrees of freedom (DOF) depending on their purpose, with applications ranging from assembly and wielding to packaging.

Pick and place robots are the most popular as they can be highly accurate and have consistent movement making then the most suitable for production line and with the advancement of technology these automation robots are becoming more affordable.

This study will focus on the manoeuvring of pick and place task most similar to the industrial packaging robots. However unlike these robots that may have a limited DOF this study will present a 3D printed prototype with five DOF that can be controlled and taught by a human user.

The scope of this paper will look at a brief history and related research will be investigated in the Literature Review section. The hardware and software used to complete this project will be introduced in the Methodology section. A demonstration and explanation of how to implement this project will be presented in Implementation. The results of the project both from a hardware and software viewpoint will be Evaluation and Discussion section. Finally in Conclusion and Future Work will the final discussion of the robot and how it can be improved.

# Literature Review

In this section a brief overview of the history of industrial robot arms that began the pick and place designs and the related research that has been done to further advance these designs and increase the efficiency and accuracy.

To give a brief overview of the history the first industrial robot that fit the ISO definition was created by Griffith Taylor in 1937 (Meccano, 1938 and Taylor, 1995), which had five DOF with a grab and grab rotation, and would perform pre-programmed tasks.

From 1956 George Devol and Joseph Engelberger founded the first robot company Unimation and controlled the market until the 1970s when countries such as Japan and the USA started to create their own robots (History, 2012).

In 1974 the “first fully electric, microprocessor-controlled industrial robot” (History, 2012) that was created by the Japanese Robot Association (JIRA). This robot had memory storage and was able to be “programmed through 16 keys and a four digit LED display” (History 2012). Because of its design it was able to perform tasks similar to a human would and used to “wax and polish stainless steel tubes bent at 90 degree angles” (History, 2012) proving that a robot could perform intricate task that previously only a human could do.

In the late 1970s following the success of the JIRA robot, these anthropomorphic robots were being designed and created, by both companies and university researchers, to perform tasks of wielding and assembly.

These designs were then continuously improved upon, up until 2006 when a German company KUKA presented a “light weight robot” (Bischoff et al., 2010) that had “unique characteristics including a low mass-payload ratio and a programmable, active compliance” (Bischoff et al., 2010). The feature would allow academic engineers and researchers to further develop robot application with increased performance.

At the KUKA Company innovation is an important part of their business strategy and as a result of that they will aim to collaborate with academia for research projects and more often than not turn those collaborations into real world successful product.

Dahari and Tan aimed to “model the forward and inverse kinematics of a KUKA KR-16KS robotic arm in the application of a simple wielding process” (Dahari and Tan, 2011). They examined the current design, considering all the parameters of the joins and links, and produced a comprehensive analysis of the kinematics each can make.

With the new advancements in technology companies are beginning to change in how they view robots is industry. Previously as robots outperformed human workers they were considered the better and cheaper option on the manufactory floors however a new paradigm were humans and robots work together. Stadler et al. presented a paper in which they “explore general expectation of naïve users towards functional and humanoid robots within a special factory context” (Stadler et al., 2013). Their study concluded that results indicated these new orthomorphic robots would suitable functional within industry environments.

These robots such as the Baxter Robot (Guizzo and Ackerman, 2012. and Robotics, 2012) are general safer to work around due to sense ad software to understand the environment it is in. Furthermore unlike the current industrial robots that a fixed to a location these new design allow more flexibility in performing tasks as it can be moved to any location easily and only takes a short amount of time to train for the specific tasks (Robotics, 2012). This training, unlike current robots which “require sophisticated software programming” (Robotics, 2012), instead “can be trained just as you would teach a person” (Robotics, 2012) which could be provided by another human worker and taught in under 30 minutes. The downside to this is that could cost a minimum of $25,000 (rethink, 2016).

With these new designs research has begun shifting toward the interaction or interfaces available for robot controllers. Daniel et al. investigate current approached and introduce a “flexible user interface based on service oriented robot cell operation” (Daniel et al., 2013). Their study was to offer a link between the industrial robots and human cognition to “offer a solution for issues faced in creation of customized user interfaces by its flexibility” (Daniel et al., 2013) which they named FlexGui. They also performed an evaluation of this interface within industry demonstrating that “results show that a significant reduction in task execution time and a lower number of required interaction is achieved” (Daniel et al., 2014). This evaluation therefore within this context proved their FlexGui could achieve this due to “the intuitiveness of the system with human centred design” (Daniel et al., 2014).

# Methodology

In this section the hardware and software that will be used is split into two sections and presented to provide a clear understanding of the project and its functionality.

## Hardware

### 3D Printing

As current robots that perform similar tasks such as the Baxter (Robotics, 2012) can be expensive, a smaller 3D printed version can be created to meet the similar requirements as 3D printing provides the ability to easily prototype, customise or manufacture. This means a user can design, scale and assemble their robots affordably and quickly.

Using this technology this study will create a simple prototype for a five DOF robot that can be controlled via micro servos and an Arduino microcontroller. The chosen design (Thingiverse, 2013) has been decreased in size and redesigned in OpenScad, software for creating solid 3D CAD objects, so that it had a larger base and structurally stronger joints that cover the servo attachment preventing them from falling apart even without being screwed into the micro servo, with the exception of the second joint show in Figure 1 which requires the extra structural strength to ensure it can support the rest of the robot arm. The design as then converted into STL files that a 3D printer could use to print.

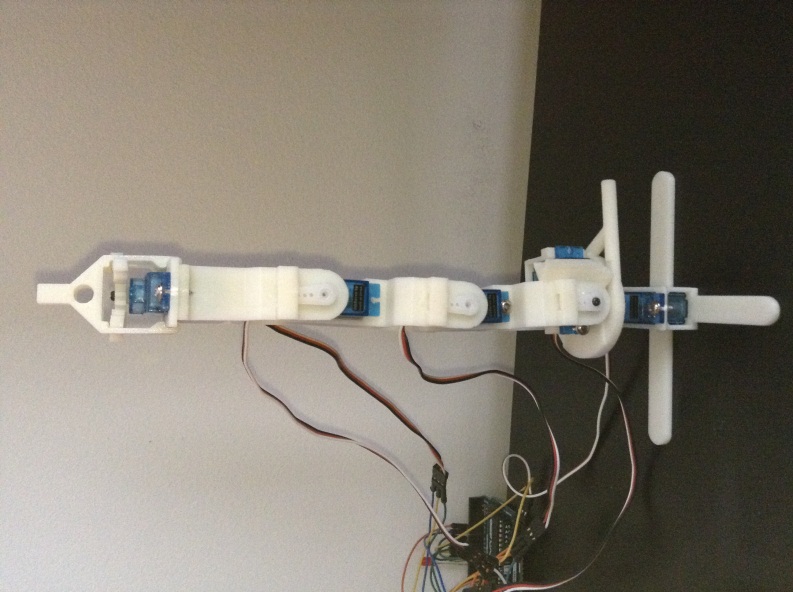
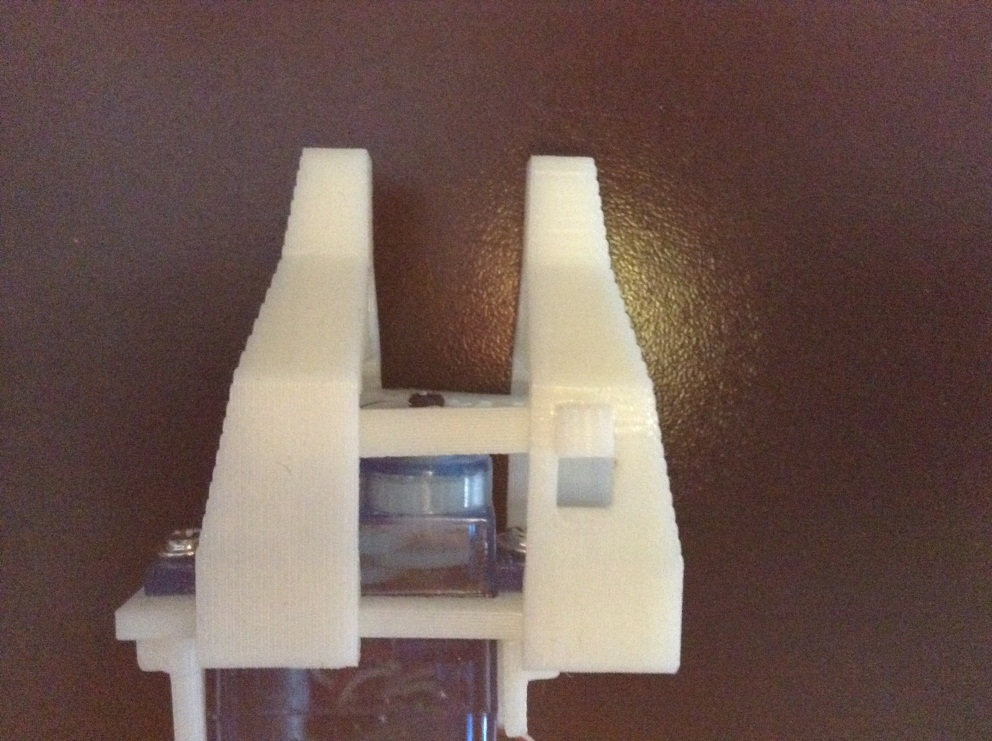
 

Figure 1 Finished Assembled 3D Print Figure 2 Alternate View of Gripper

Figure 1 shows the finished and assembled 3D print, all the positions discussed hereon will be from the position shown in Figure 1. Starting from the base is a servo allowing for the rotation of the entire robot, both clockwise and anti-clockwise directions. The second joint will turn 90 degrees left but will not move to the right as a structural design feature prevents it. The third joint can move 180 degrees in total, 90 degrees left and 90 degrees right, allowing the robot to reach positions close to the base if necessary. The fourth joint also moves 180 degrees but only 20 degrees left and 160 degrees right. This is a result of the third joint will provides sufficient movement towards the base already hence the fourth joint with a mirror design has more flexibility with lifting objects up instead. Finally the gripper on the top, shown in Figure 2, will open and close to the maximum and minimum the servo allows. These five parts in these positions allow for a five DOF robot capable of picking and placing objects within its 20cm range.

### Arduino

Arduino is an open-source computer hardware called a microcontroller that is essentially a micro-computer, comprised of an internal CPU, RAM and IO interface and used for data analysis or control projects. In this project the Arduino UNO will be the chosen microcontroller shown in Figure 3. It provides all the power necessary to control the robot.

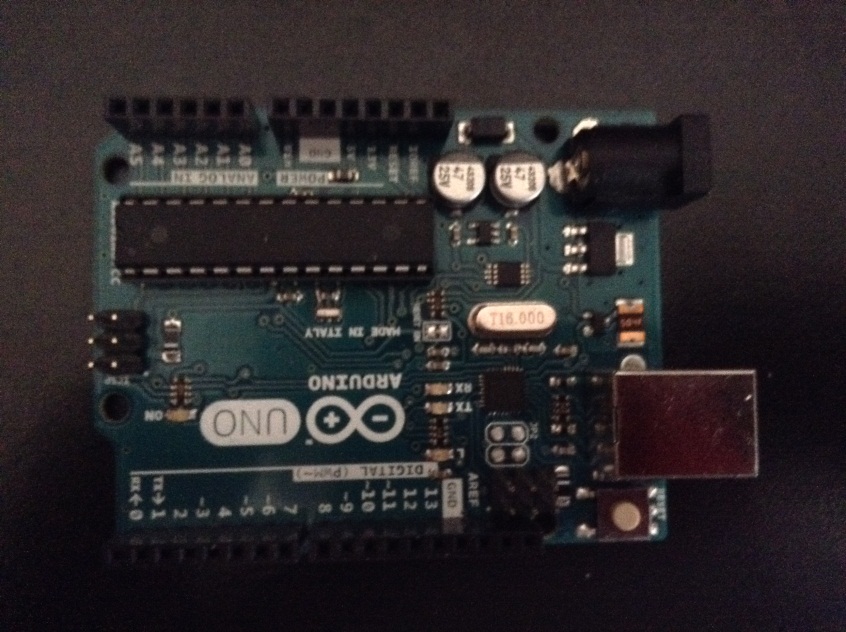


Figure 3 Arduino UNO

#### Additional Arduino Parts

Additionally to the 3D print and the Arduino microcontroller this project requires a few other components to work. They are as follows:

* 5 x Micro Servo 9g with accessories
* 14 x Jumper Wires (Male)
* Type-B USB Cable
* Breadboard

For this project the Micro Servos 9g were chosen for their size allowing for a smaller design overall of the project, saving printer material and producing a quicker printed prototype. Larger servos could still be used with the code that has been created however it would require a redesign of the 3D print and would possible need more than the 5 voltage power the Arduino UNO can supply via the Type-B USB cable that connect it to a PC.

Each servo requires three male jumper wires and as the UNO can only support power and grounds to one a Breadboard will also be required. Depending on the breadboard chosen additional wires or in the case of this project bent wires were used to supply more power to other areas of the breadboard.

## Software

### Arduino Software

In this section the basic structure of Arduino code will be presented and discussed followed by the illustration of the key areas of this project’s code to show how it works, explaining the functions and requirements for it to run.

Arduino Software is an open-source Integrated Development Environment (IDE), which is written in Java and based on software called Processing. The Arduino IDE can provide an easy way for writing code, called a sketch, and uploading the code to any Arduino board.

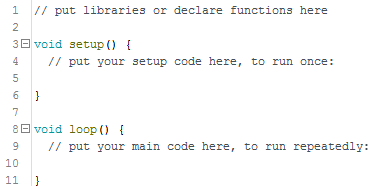


Figure 4 Basic Structure

Figure 4 shows the key parts of a sketch. A typical sketch can be split into three key sections. Section one is where the program will include the necessary libraries for the functions that will be used in the program or that the program will use for example servos or the on board memory. Here the global variables are also declared so that they can be used in the next two sections. Section two is ‘void setup()’ which is used to indicate the initial values of the system when the program is first started and will run only once. This can be things like attaching hardware to the board pin for instance an LED. Section three is ‘void loop()’ which is the main body of code of what the program is supposed to do, containing the statements that will run after setup and will run repeatedly so long as there is power.

# Implementation

In the following sections the code for the project will be presented and discussed using the key three parts previously presented.

#### Section One

##### Libraries

The only library used is Servo, shown in Figure 5, allows the code to declare servos will be used and to assign a variable name, attach those variables to the pin numbers on the Arduino UNO and finally to provide function between the servos in order to control them as shown in Section Three: Loop.



Figure 5 Library Code

##### Declarations

The declarations, shown in Figure 6, are global variable that all other section of code will have access to and be able to alter the values of. Firstly we create a char variable that will hold the user input from the serial monitor that can then be used by other section of code to perform task. A char array variable that will store the entire user input, concatenating it into a single line and an integer variable that will be used to tell the servo how many degrees to move in either direction. Finally the servos are declared starting with the base of the 3D print being ‘one’ counting up to ‘five’ which is the gripper.

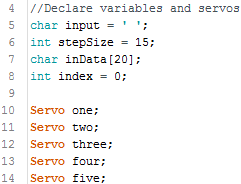


Figure 6 Declarations Code

#### Section Two: Setup

Figure 7 illustrated the setup function that runs once when the program is first run. Here the serial output beings and a length values or the string ‘inputString’ is given. The previously declared servos are attached to the digital pins on the UNO and then using one of the Servo library functions are given positions in degrees to centre the robot as in Figure 1. Finally a delay is given ensuring all setup has been processed and a message is output to the serial monitor to prompt the user that they are now able to provide input and control the robot’s movements.

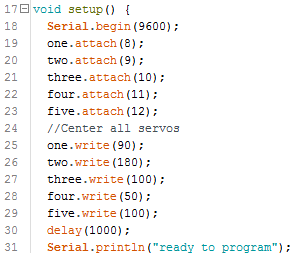


Figure 7 Void Setup Code

#### Section Three: Loop

In the loop() function is the main code for servo movement and defining what user input means. Figure 8 shows that if the user input is empty or a space character to ignore it. Next is the creation of local variable that will be used for the alteration of servo position, when initialised they are whatever the current position the servo is allowing for them to be overwritten in each loop.

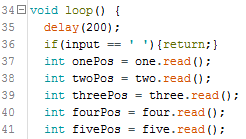


Figure 8 Local Variables Code

In order to process the user input a switch case (Figure 9) has been used that will take the user input character and depedning on the character change the degrees value of a servo that will be used in Figure 12. As shown in Figure 9 each letter will use the previously declared integer to increase or decrease by 15 degrees.

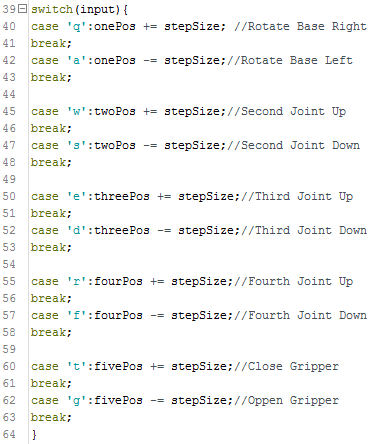
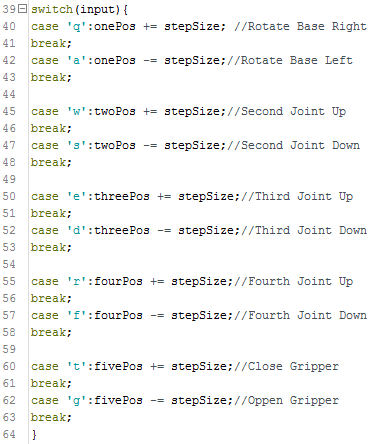


Figure 9 Switch Case Codes

To prevent the user for moving the servos beyong a certain possition the code in Figure 10 is used. This limites the movements of each servo from between 0 and 180 meaning if the fourth join is a 180 degrees and the user continues to input the character ‘r’ then nothing will happen.

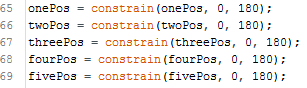
 

Figure 10 Constrained Positions Code Figure 11 Clear Character Code

Figure 12 shows the code that moves the servos uses the local variables. Essentially if the new servo positions such as ‘onePos’ does not equal the current servo possition a message will be printed to the serial monitor to tell the user the new degrees and will move the servo to the new position. This code prevent movement in servos that have not received any change. In each iteration of loop() the local variable are reset to a space using Figure 11 otherwise the code will repeatedly use the same caharacter over and over until a new one is given, for example if ‘s’ is provided the servo will repeatedly keep reducing by 15 until it gets to 0 degrees.

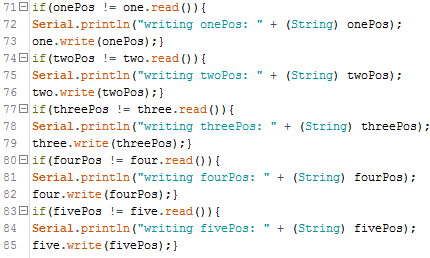


Figure 12 Servo Movements Code

To provide the user with additional commands two if statements have been created that will allow the user to quickly reset all of the servos back to their default positions (Figure 13) and to clear the char array of all characters (Figure 14).

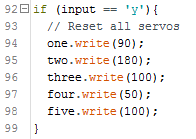


Figure 13 Reset Positions Code

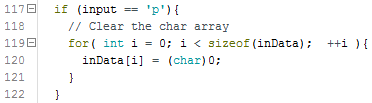


Figure 14 Clear Char Array

#### Section Four: Serial Event

This section of code shown in Figure 15 occurs whenever new data comes in the serial monitor and is run between each time loop() runs, therefore using a delay inside the loop can delay the response of the movement command in loop(). Any input from the user into the serial monitor will be processed by this method and have the character placed into the ‘input’ variable that will be used in Section Three before being erased at the end of the loop (Figure 11) whereas with the string ‘inputString’ that characters are continuously added to the string and will only be cleared with user input (Figure 14). Multiple bytes of data may be entering into the serial monitor at a time, for instance if the user wanted the robot’s second joint to move 45 degrees down left then they can input three characters ‘sss’ and the bytes will be processed one after the other. However due to there being no delay if the input is too long it is possible for the input character to be cut of before being passed over to loop().

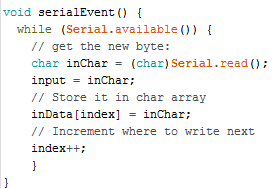


Figure 15 Serial Event Code

## Replay Function

The replay function will run whenever the user input the character ‘h’ into the serial monitor (Figure 16) and will parse each character. The delay is necessary between each character so as to not miss one when the loop iterates. The code to run this is essentially the same as the loop() method except it uses a char array ‘inData’ as the input and that it is separate set of case switch to the main loop(). The char array is created during the serial event (Figure 15) where each input is added to the next element of the array.

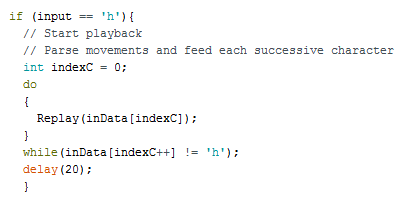


Figure 16 Replay function

## Controlling the Robot

Once the program is run and uploaded to the robot and the serial monitor has been opened, there is a small delay for the program to setup and once ready will present a ‘ready to program’ message in the serial monitor. At this point the robot can be controlled using the keys shown in Table 1. Due to the movement steps being 15 degrees the user has a good control over the action the robot can take.

Table 1 Key Controls

|  |  |
| --- | --- |
| **Key** | **Action** |
| q / a | Base Servo Rotate Left / Base Servo Rotate Right |
| w / s | Second Joint Up / Second Joint Down |
| e / d | Third Joint Up / Third Joint Down |
| r / f | Fourth Joint Up / Fourth Joint Down |
| t / g | Close Gripper / Open Gripper |
| h | Replay taught movements |
| y / p | Reset Servo to Default / Clear the taught moves |

# Evaluation and Discussion of Results

## Code

Controlling the movements of the robot is easily through inputting characters into the serial monitor. Through testing it was discovered that the maximum number of characters that could by input at once was four as a result of the loop() function looping back before all of the character had been fully parsed. This was fixed by adding a delay at the beginning of the loop() however this could slow down the speed on movements to commands if a longer command is input then the user must wait until the movements are completed before inputting new commands.

The replay function for the robot means that any input the user makes will be learnt. Using the reset position and clear input array function the user can restore the robot to default and then train the robot the motions it should make. This worked successfully making the same movement input.

In the first design of this code the step size was 30 degrees, meaning every character input would move the servo 30 degrees and would could a very erratic motion that could throw the robot of balance and fall over. To solve this issue the step size has been decreased to 15 degrees which provide a smother motion whilst still being quick.

## Design

Due to a design issue even with a decrease motion the robot is require to be weighted down so that it does not fall during testing. Reducing the step size again could prevent this however as it is a design feature this is something that will be fixed in future work rather than in this project as once weighted down the robot functions perfectly.

In the design used in this project the gripper, shown in Figure 17, could be improved. Currently it is controlled with a gear attached to the servo that moved the gripper through the edges. This design did not fail in moving or picking and placing objects but could be an improvement as due to the size of the gear the gripper can overextend; therefore an improvement would be to either increase the size of the gear or to change from a gear to another design for a better performance.

Another design feature it that gripper lacks is friction to hold objects. In the testing it had no issues with the weight or shape of the objects but over time the object would slip free. As a temporary fix two small elastic bands were attached to each prong to improve the grip. This improved the grip by a lot and allowed for various objects of different shapes, sizes and weights to be easily picked and placed.

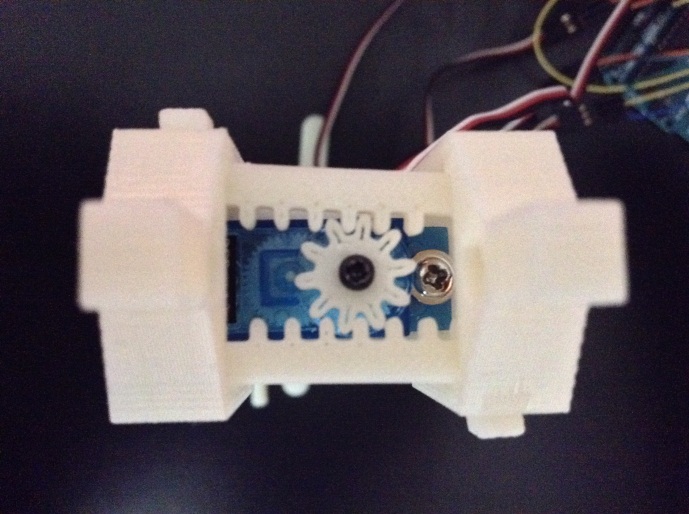


Figure 17 Close Up View of Grip

The last minor design feature that noticed was that on the second joint there is an excess of material (Figure 18) to prevent the servo from making a 180 degree movement, however in a better balanced design this could be removed allowing for more flexibility in the action it can take.

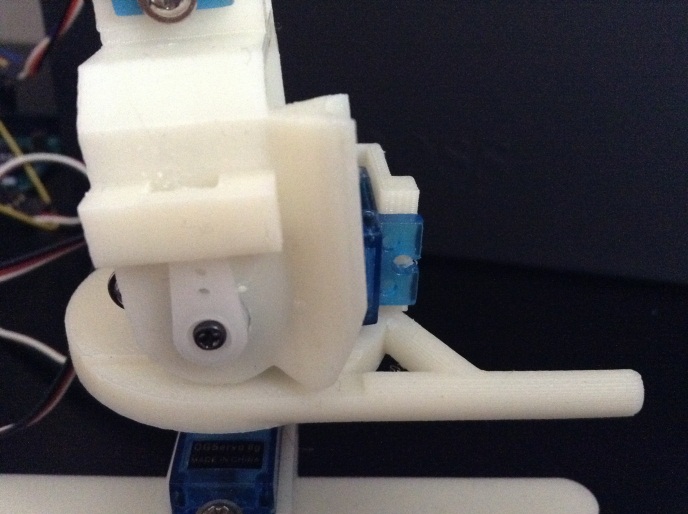


Figure Second Joint Material

# Conclusion and Future Work

## Conclusion

In this section a discussion about the project and the results will be presented. This project started out with the goal of designing and creating a small and cheap way to prototype pick and place robots that could be controlled via user input with a secondary aim to have the robot learn a replay any learnt actions. When testing the design and code a few issues became revealed, some which could be fixed straight away and some which would require future work to improve or fix. Overall the study has completed both the primary and secondary aim of this study have been completed creating a much cheaper design that is capable of lifting quite heavy objects within its 20cm range.

## Future Work

Future work for this project would involve the improvements to the design and scaling the design up so that alternative servos can be used. Due to the size of the prototype in this study the analogue feedback servos, that will further increase the user input for control, could not be used.

Creating a new prototype using these feedback servos would allow for a more precision movement and for the training or reprograming to occur much faster and you simpler. The new redesign removes the ability to control the robots movements through the serial monitor and key presses and replaces it with a way to physically alter the motions and for it to be dynamically trained and quickly replayed to view the precision efficient of those movements. The addition of finding the minimum and maximum values means that the movements are much smoother that the previous design.

This future work coding has already been completed and included in the disc along with this project code. The new code has been presented and discussed in Appendix A showing the additions of new code discussed briefly here. It should be mentions however that due to the lack of a prototype and analog feedback servos this new code has not been fully tested although should work perfectly.

The final design change is to have a more stable base (Figure 19). When the movements are quick such as in the setup where the maximum and minimum servo position are detected, the quick movements cause the robot to fall over if it is not properly weighted down and a larger or differently shaped base, for example a circle of material rather than a cross shown in Appendix B, could solve the balancing issues.

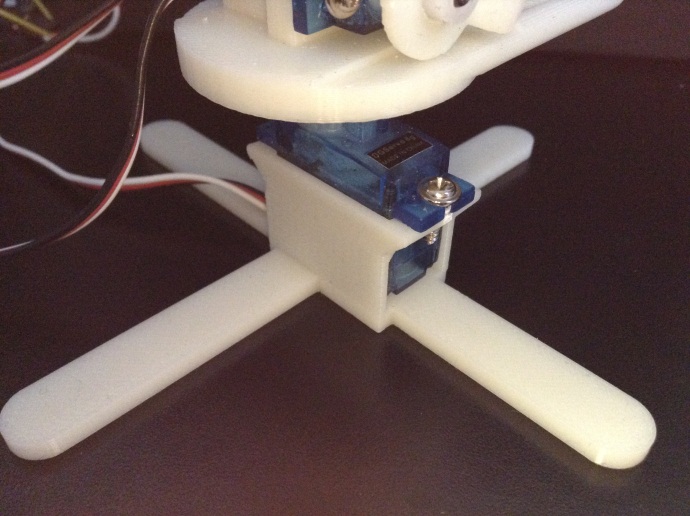


Figure 19 Close Up View

# Reference List

Meccano, 1938. "An Automatic Block-Setting Crane". *Meccano Magazine* (Liverpool UK: Meccano) **23** (3): 172. March 1938.

Taylor, Griffith P. (1995). Robin Johnson, ed. *The Robot Gargantua*. Gargantua: Constructor Quarterly

History, (2012) History of Industrial Robots. [online]. Available at: http://www.ifr.org/uploads/media/History\_of\_Industrial\_Robots\_online\_brochure\_by\_IFR\_2012.pdf [Accessed 16 Dec. 2015].

Bischoff, R., Kurth, J., Schreiber, G., Koeppe, R., Albu-Schäffer, A., Beyer, A., Eiberger, O., Haddadin, S., Stemmer, A., Grunwald, G. and Hirzinger, G., 2010, June. The KUKA-DLR Lightweight Robot arm-a new reference platform for robotics research and manufacturing. In Robotics (ISR), 2010 41st international symposium on and 2010 6th German conference on robotics (ROBOTIK) (pp. 1-8). VDE.

Dahari, M. and Tan, J.D., 2011, April. Forward and inverse kinematics model for robotic welding process using KR-16KS KUKA robot. In Modeling, Simulation and Applied Optimization (ICMSAO), 2011 4th International Conference on (pp. 1-6). IEEE.

Guizzo, E. and Ackerman, E., 2012. How rethink robotics built its new baxter robot worker. IEEE Spectrum.

Robotics, (2012). Robotics News. [online]. Available at: http://www.robotics.org/content-detail.cfm/Industrial-Robotics-News/Rethink-Robotics-Revolutionizes-Manufacturing-with-Humanoid-Robot/content\_id/3658 [Accessed 3 Nov. 2015].

Stadler, S., Weiss, A., Mirnig, N. and Tscheligi, M., 2013, March. Anthropomorphism in the factory: a paradigm change?. In *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction* (pp. 231-232). IEEE Press.

Rethink, (2016) Build a Baxter Robot. [online]. Available at: http://www.rethinkrobotics.com/build-a-bot/baxter/ [Accessed 2 Jan. 2016].

Daniel, B., Korondi, P. and Thomessen, T., 2013, November. New approach for industrial robot controller user interface. In Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE (pp. 7831-7836). IEEE.

Dániel, B., Korondi, P., Sziebig, G. and Thomessen, T., 2014. Evaluation of flexible graphical user interface for intuitive human robot interactions. *Acta Polytechnica Hungarica*, *11*(1), pp.135-151.

Thingiverse. (2013) Micro Robot Design. [online]. Available at: http://www.thingiverse.com/thing:65081 [Accessed 3 Nov. 2015].

# Appendix A – Future Work Code

### Section One

#### Library

EEPROM is a memory which can read and write values based on the code of the sketch (Figure 20). In this new design it has been used to make the system more intelligent by storing the user input and allowing for these learnt movements to be replayed whenever a button is pressed. To allow for the user to know when they are allowed to make an action the LED attached to pin 13 will blink.



Figure 20 Library Code

#### Declarations

For the new variables all are declared globally (Figure 21, 22) which will be used for storage and calibration. Figure 21 shows the ‘addr’ variable that will store the EEPROM positions from manual manipulation of the robot’s positions. The boolean will alternated between true and false depending on whether the user records any manual positions

Calibration starts in section one of the code were calibration values that required the minimum and maximum values for both the degrees and feedback are placed (Figure 22). These are integers so it is necessary to declare all the minimum and maximum variables for each servo.



Figure 21 Storage Variables Code

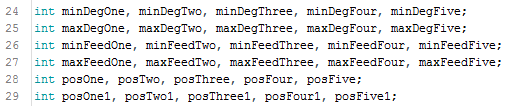


Figure 22 Calibration Code

### Section Two: Setup

In addition to starting a serial ouput and attaching servo the new code attached an LED as an output and two buttons and inputs (Figure 23). Although not in code servos will be attached to the analog pins on the UNO via feedback wires.

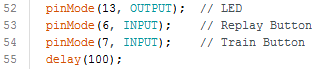


Figure 23 Attach LED and Buttons Code

The easiest method for calibrating the servos is to start them all out at 90 degrees then step through each servo’s motion starting from One and ending at Five to record the analog feedback values when the servos are positioned at their minimums and maximums (Figure 24).

When each servo is at its minimum, read the analog value from the respective servo analog pin and store it in the respective variable. After this is complete, place the servos back a 90 then detach them to both save power and to allow them to be moved by the users hands (Figure 25).

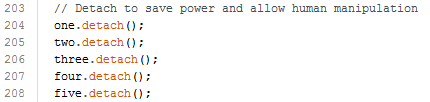
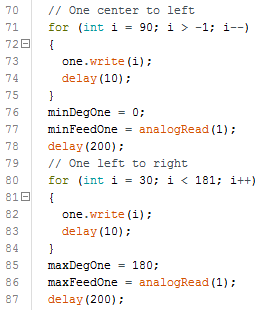


Figure 24 Min Max Feedback Sample Code Figure 25 Detach Servos Code

Finally blink the LED on and off to let the user know that the setup has finished and they can now start training new movements or replay the movements storing in the EEPROM (Figure 26).

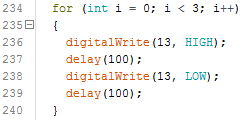


Figure 26 Blink LED Code

### Section Three: Loop

Unlike in the first design code which processes input via serial port due the lack of feedback wires, the new main code will poll the state using the record and replay buttons. After recording or replaying each servo is moved back to the default positions and detached.

#### Record

When the record button is pressed the recorded will change to true and the LED will light up to let the user know that they can move the robot into the desired positions. Whilst this is happening the minimum and maximum position of the user movements are being read through analog and converted in servo degrees. Finally these positions are constrained and the positions are then written into the EEPROM (Figure 27).

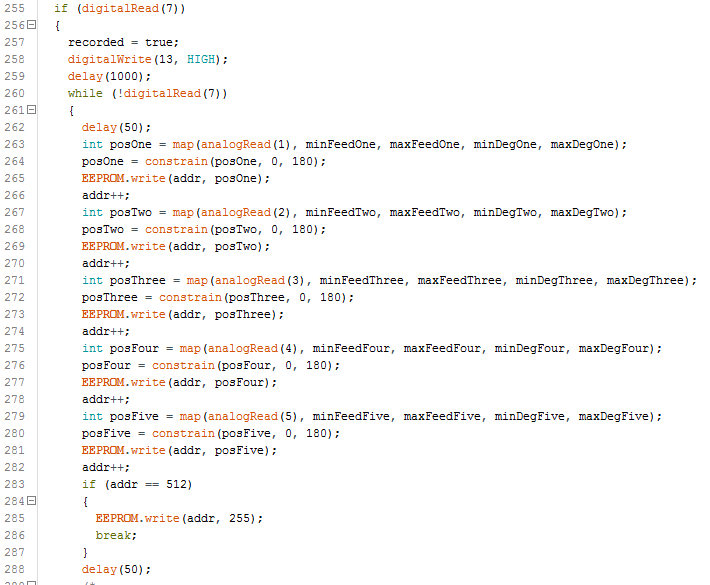


Figure 27 Record Motions Sample Code

#### Replay

Then the replay button is pressed the stored servo positions will be executed to allow the user to see how well the robot moves and if those are the correct, if not then the user can press the record button and redo the process and overwrite the EEPROM (Figure 28, 29).

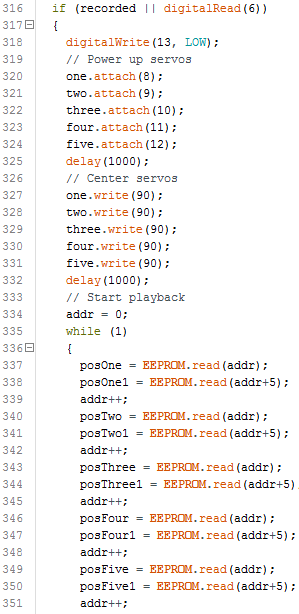
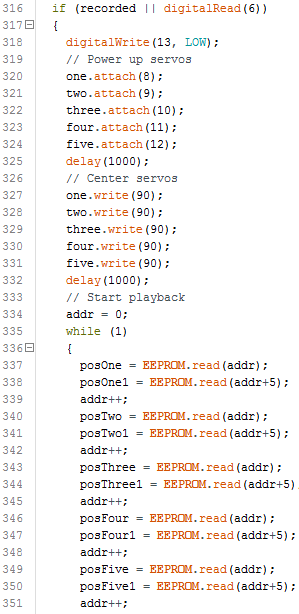


Figure 28 Replay Code 1

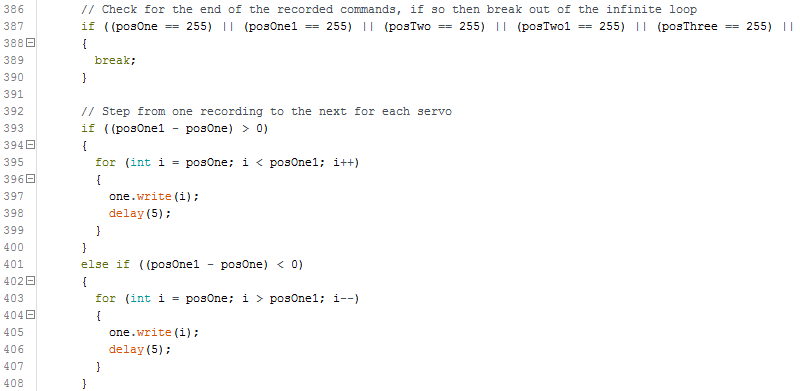
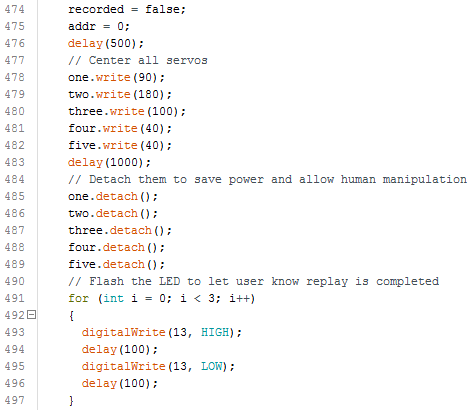


Figure 29 Replay Code 2

# Appendix B – Future Work Base Design

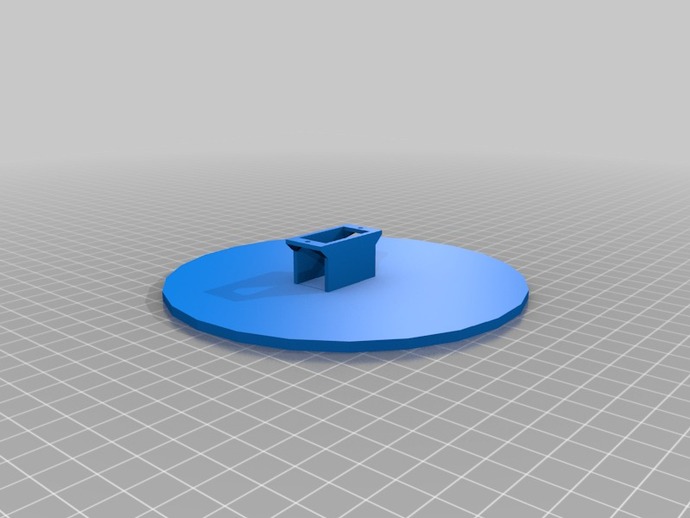


Figure 30 Alternate Base Design