Robot Arm

Inverse Kinematics Robot Arm

KwangKyun Bruce Kim (201857200692), IlGyu Cho (201857200587)
YongWook Kim (201957300110), Jung Ho Song (201857300321)
Eunho Park (201957100222)

Objective

Household chores are tiresome tasks. However, one cannot say that the tasks are too complex for human beings to carry out. Most people have no difficulties in carrying out daily household chores, however, such labors tend to be mundane and routine that people often get exhausted. This brings questions how robotics would overcome this problem as the world is already full of robots that can build cars and ships. Folding pieces of handkerchiefs by robots appears to be an easier task than assembling an airplane. Surprisingly, however, this domain becomes a herculean field for robots. Contrary to car assembly in a production plant, calculating just amount of fluid and amount of force in each and every movement in an open environment has been far too complex. Robots were simply not good enough to locate a piece of handkerchief, fold it neatly, and stack it in a wardrobe, as we do not care to put it at a ‘designated’ location like components in a factory.

Recently, engineers in UC Berkeley created an autonomous household chores robot. The ‘Project Blue’ is about creating a robot by utilizing machine learning to learn complex human labor in an open environment. They designed a robot, durable enough to withstand an external intervention, and perform basic chores such as using a coffee machine and clean if there has been any spill. Engineers do ‘teach’ robots as it has been conventionally, but robots can ‘learn’ the best means and behaviors to successfully perform tasks. Such progress has opened infinite possibility in robotics that robots in near future will perform multiple functions in such natural fashion as humans would, but with more precision and higher performance.

This project aims to design an inverse kinematics robot arm to study basic technology in robotics, a foundation toward higher technologies including machine learning control. An inverse kinematic robot can relocate its arm to a designated coordinate in a three–dimensional space. Elementary maneuver of the arm is fundamental in controlling the robot. Coordinates can be provided by manual, but also possible to be provided autonomously via vision sensor, involving less human labor.

Introduction

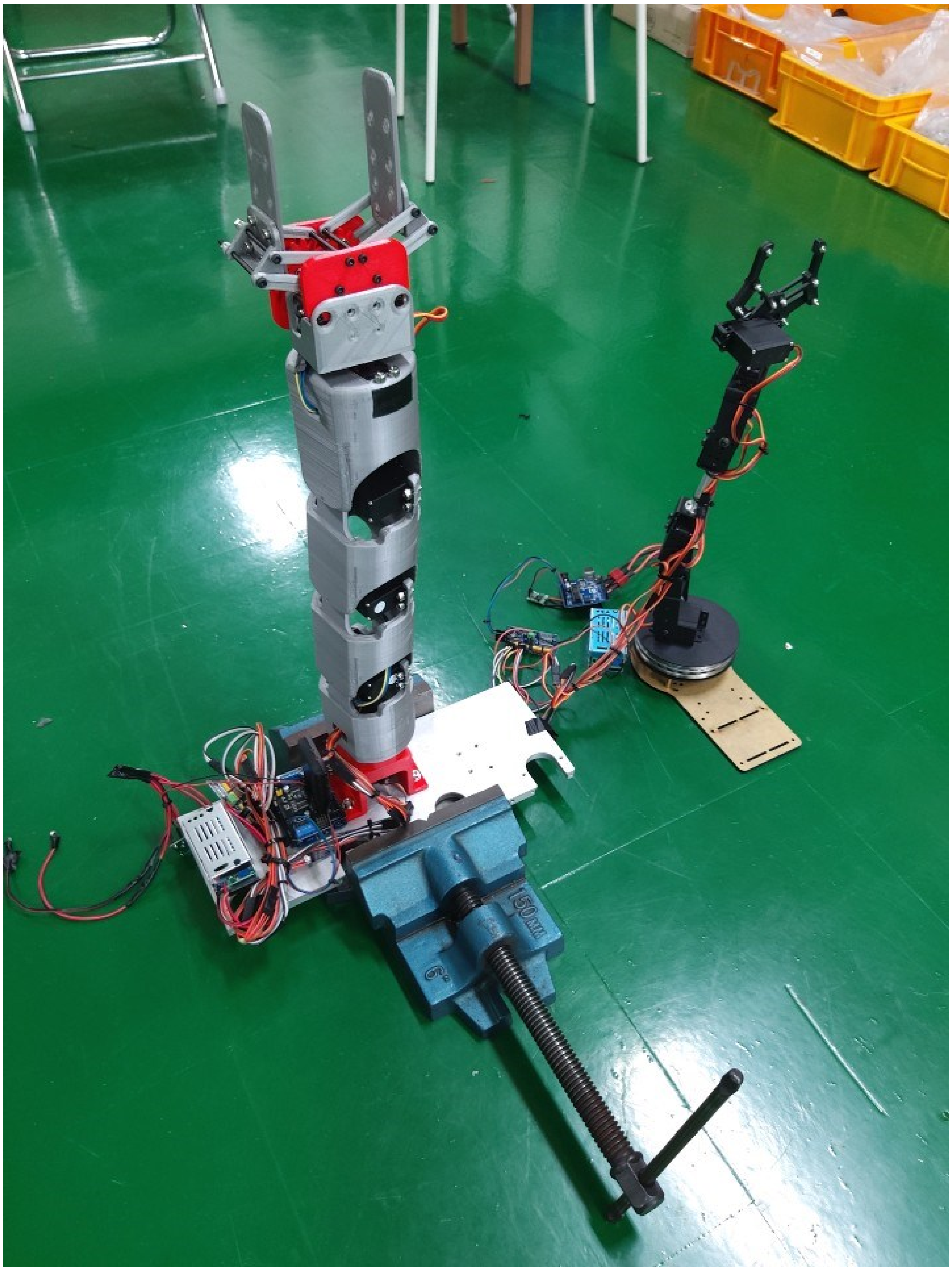


Figure 1. Assembled Robot Arm – "Waver" (left)
Test robot arm for software development (right)

The robot arm (left) is mainly designed to run on an Arduino Uno with a 16 channel PWM controller board attached to. While the board enables easier control of motors up to 16, this robot has 6 motors connected (6 Degrees of Freedom). The robot was programmed to make use of the kinematics equations to determine joint parameters, which would then relocate its end-effector (a gripper) to a desired position. While forward kinematics involves deciding each joint angle, then calculating Cartesian position and orientation of the end effector, our robot arm adopts inverse kinematics to input the Cartesian end-effector position and orientation, then compute all the joint trajectories.

Arduino Uno served as a main microcontroller for the entire system. The hardware for this project includes six controlled servo motors for each degree of freedom. The mechanical components constitute a base, bodies of arm, and a gripper, which are entirely 3D CAD modelled and 3D printed. The camera module for the robot is a Pixy2 module that learns to detect objects already taught. Several interfaces (SPI, I2C, UART, and USB) and simple communications of Pixy2 module grants the designer not to involve controller boards with an operation system.

The entire system was programmed and developed in C/C++ on Arduino IDE. With myriads of opensource libraries on the Internet, we were able to take full advantages of the libraries by merging them into our main software system. Such libraries combined include servo motor PWM control, I2C, and camera module, respectively, “PCA9685.h”, <Wire.h>, and “Pixy2.h”.

“PCA9685.h” is an Arduino library that enables PWM control of the motors connected to PCA9685 board, a 16-channel PWM Driver Module.

<Wire.h> is an Arduino library that enables I2C communication. This was required to maneuver the robot by controller, which we coded with a sample robot arm to check hardware its function. Also, this is needed for a Pixy2 module to communicate with the main controller.

The Pixy2 package (“Pixy2.h”) is an all-inclusive one that supports features to detect colors, lines, intersections and small barcodes. Among those, this project focuses on detecting color-based objects, and then to retrieve the position of the object in three-dimensional space.

Yet, in this project Pixy2 camera module was not fully integrated into the system. Image processing with the camera module was conducted only on a test level to recognize color balls. The next level is to calculate the distance between robot arm and a recognized target object to retrieve exact coordinates.

The Robot Arm Design

During an early stage of the robot development, our team members had gone through several enthusiastic discussions in deciding what kind of robot should we build. As much as technological aspects of a robot are important, we were fully aware that its purpose and our society’s familiarity with it are also crucial. It was then we had addressed a recent development of ‘Project Blue,’ in which we discovered an ‘intimacy toward robots’ as a key concept. This was well demonstrated by a concept letter written below (Figure 2).

The overall design for this project is divided into three parts: the hardware design/testing part, inverse kinematics design/testing part, and the integration and overall system assembly/testing part. By dividing up the development and design of this robot into three separate parts, we were able to validate on each component level, and then merge those on a system level.

To reduce damage risk of 3D printed robot, we managed to purchase a test robot that runs on a controller. After assembling it, we uploaded software to navigate robot by inverse kinematics equation.

Hardware Design

Robot Arm Design Concept

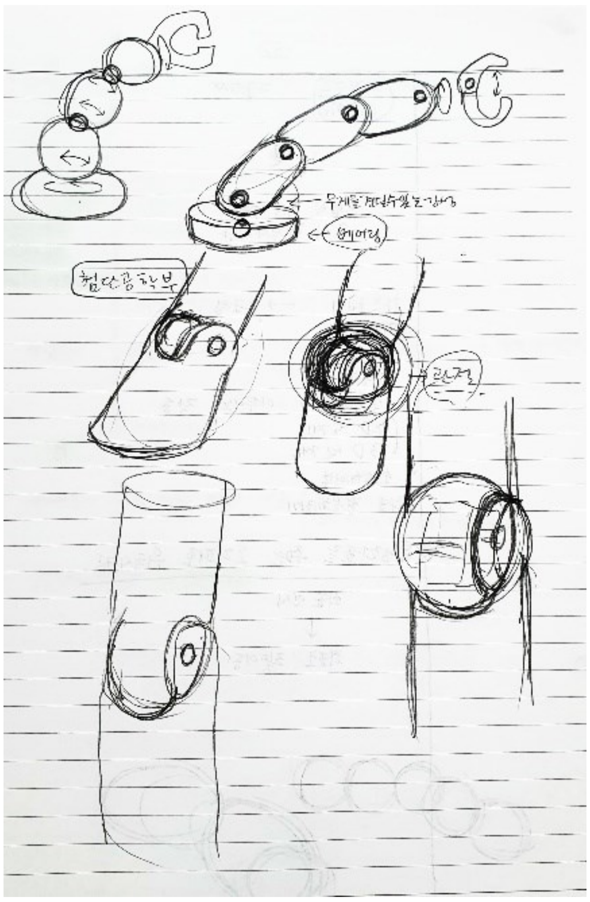


Figure 2. Robot Arm Design Concept

Our robot arm was designed after such contemplation and motivation.

"We are living in an age of transition when robot and man form a more intimate relationship. Our imagination of harmonization with robots is often portrayed in films and pictures; The reality, however, is a gap we realize when we meet robots existing today. To narrow the gap, an apposite design to formulate familiarity with us is essential.

We would like to find our concept of design from humanity.
Our calculation suggests linear structures in machine to have the most efficient form.
Our feelings, however, proves that straight lines do not exist in our forms.
Thus, linear forms will be minimized to necessary and curves will be adopted to produce an organic structure.

Shape of our robot will help us construct a closer and more natural relationship with robots.
We hope this design would be one of the pursuits that makes robots a part of our life"

- *Fireworks, Prime College Mechatronics Engineering Club*

Robot Arm Gripper, Bodies, and Base

The robot arm grabs and releases a target object by using a gripper as an end-effector. It has four body parts connecting each other. 6 motors located in each joint allow more degrees of freedom. Finally, it has a base with a bearing structure applied to withstand weight and moment of the upper part.

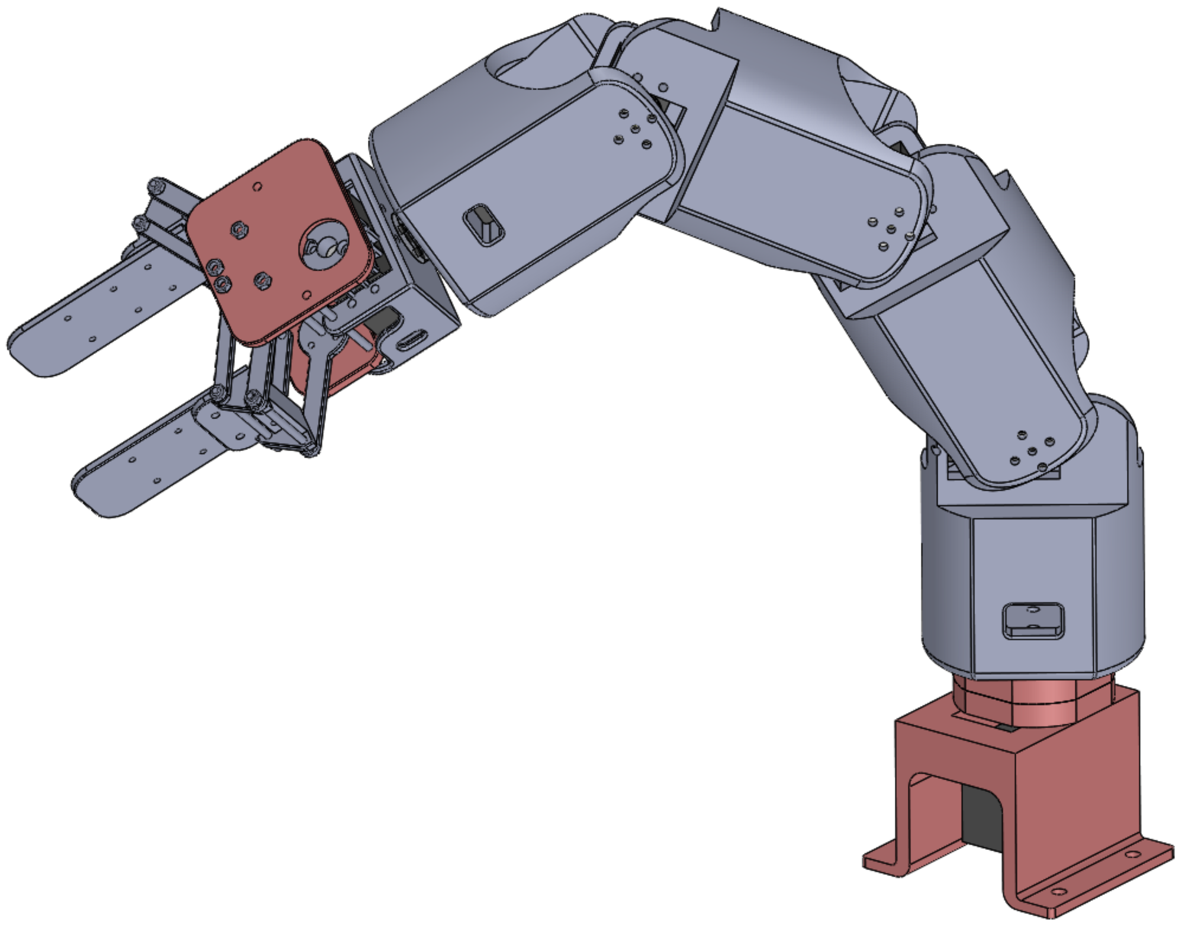


Figure 3. 3D CAD design of the Robot Arm

In order to grab a round object like a golf ball, the gripper has two flat plates that would close to hold an object. The kinematic design of the gripper enables a servo motor to open or close by rotating clockwise or counterclockwise, respectively.

The gripper is then linked to a ‘wrist,’ a ‘wrist’ to an ‘elbow,’ then an ‘elbow’ to a ‘shoulder.’ Each body part has the same design; The joint part is an open-sphere structure, and bracket covers servo motors.

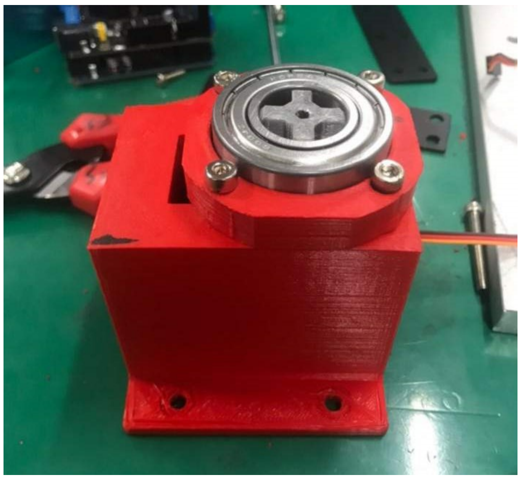
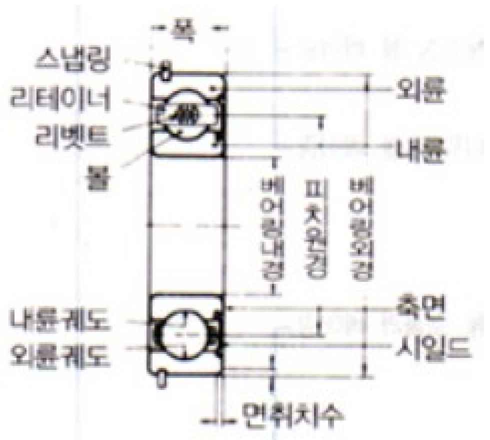
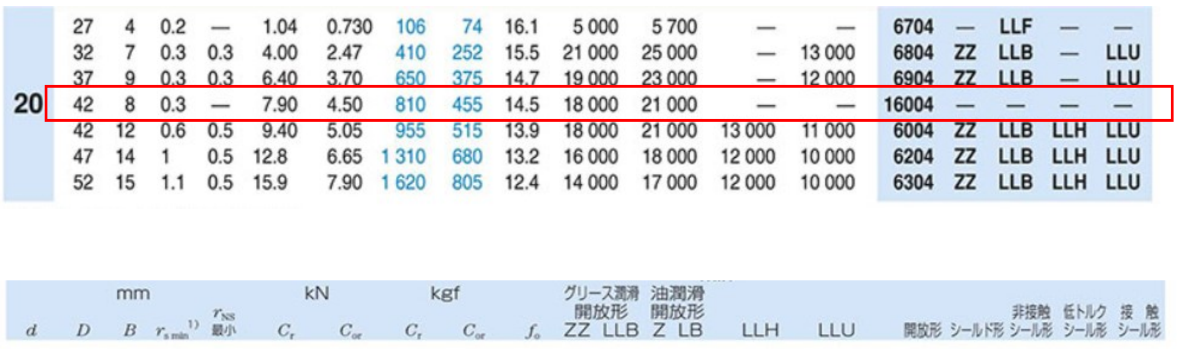


Figure 4. base bearing structure

The base part is comprised in three parts: a base which fixates the robot to the ground with a servo motor installed, a bearing structure that transfer axial loads of the robot arm, and a rotatory part linked to a ‘shoulder.





	mm				kN		kgf			グリース潤滑 開放形	油潤滑 開放形							
d	D	B	$r_{s\,min}$ 1)	r_{NS} 最小	C_r	C_{or}	C_r	C_{or}	f_o	ZZ LLB	Z LB	LLH	LLU	開放形	シールド形	非接触 シール形	低トルク シール形	接触 シール形
20	27	4	0.2	—	1.04	0.730	106	74	16.1	5 000	5 700	—	—	6704	—	LLF	—	—
	32	7	0.3	0.3	4.00	2.47	410	252	15.5	21 000	25 000	—	13 000	6804	ZZ	LLB	—	LLU
	37	9	0.3	0.3	6.40	3.70	650	375	14.7	19 000	23 000	—	12 000	6904	ZZ	LLB	—	LLU
	42	8	0.3	—	7.90	4.50	810	455	14.5	18 000	21 000	—	—	16004	—	—	—	—
	42	12	0.6	0.5	9.40	5.05	955	515	13.9	18 000	21 000	13 000	11 000	6004	ZZ	LLB	LLH	LLU
	47	14	1	0.5	12.8	6.65	1 310	680	13.2	16 000	18 000	12 000	10 000	6204	ZZ	LLB	LLH	LLU
	52	15	1.1	0.5	15.9	7.90	1 620	805	12.4	14 000	17 000	12 000	10 000	6304	ZZ	LLB	LLH	LLU

Figure 5. bearing type & dimension

Estimating the weight of the robot arm to be about 5kg, this was considered as a main factor, along with a maximum load-carrying capacity, to choose an apposite dimension of bearing. For the type of bearing, seal ZZ 6004zz was chosen. Inner diameter is 20mm, and outer diameter is 42mm. Permitted dynamic load is 455 kgf, and permitted static load is 810kfg.

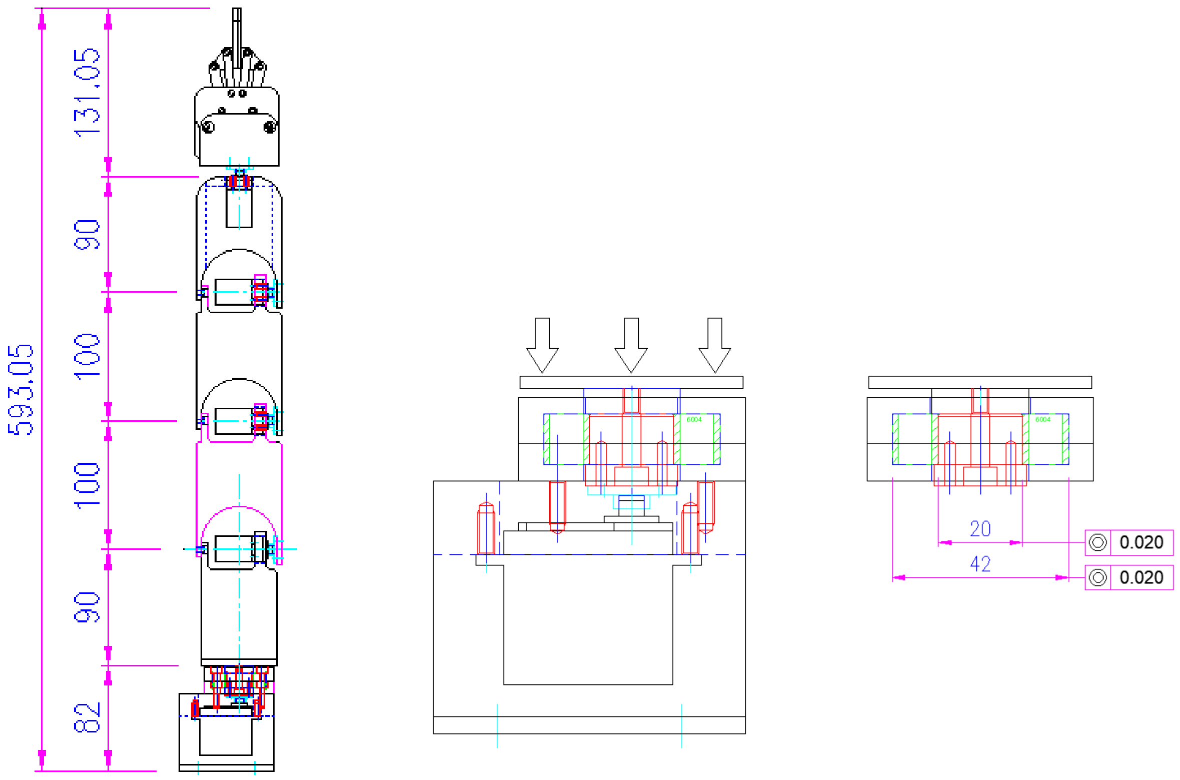


Figure 6. 2D CAD (left), Load direction (middle),
Bearing concentricity tolerance: 0.02mm (right)

As the height of the robot arm is 593mm, and its weight is 5kg, the base bearing structure was designed to withstand maximum load-carrying capacity of the base under 20kfg. During movement of the robot arm, its vibration and torsion are supported by the bearing's concentricity tolerance of 0.02mm. To avoid direct load on a base motor, which may damage the motor, the motor shaft is connected to the lower part of the bearing structure.

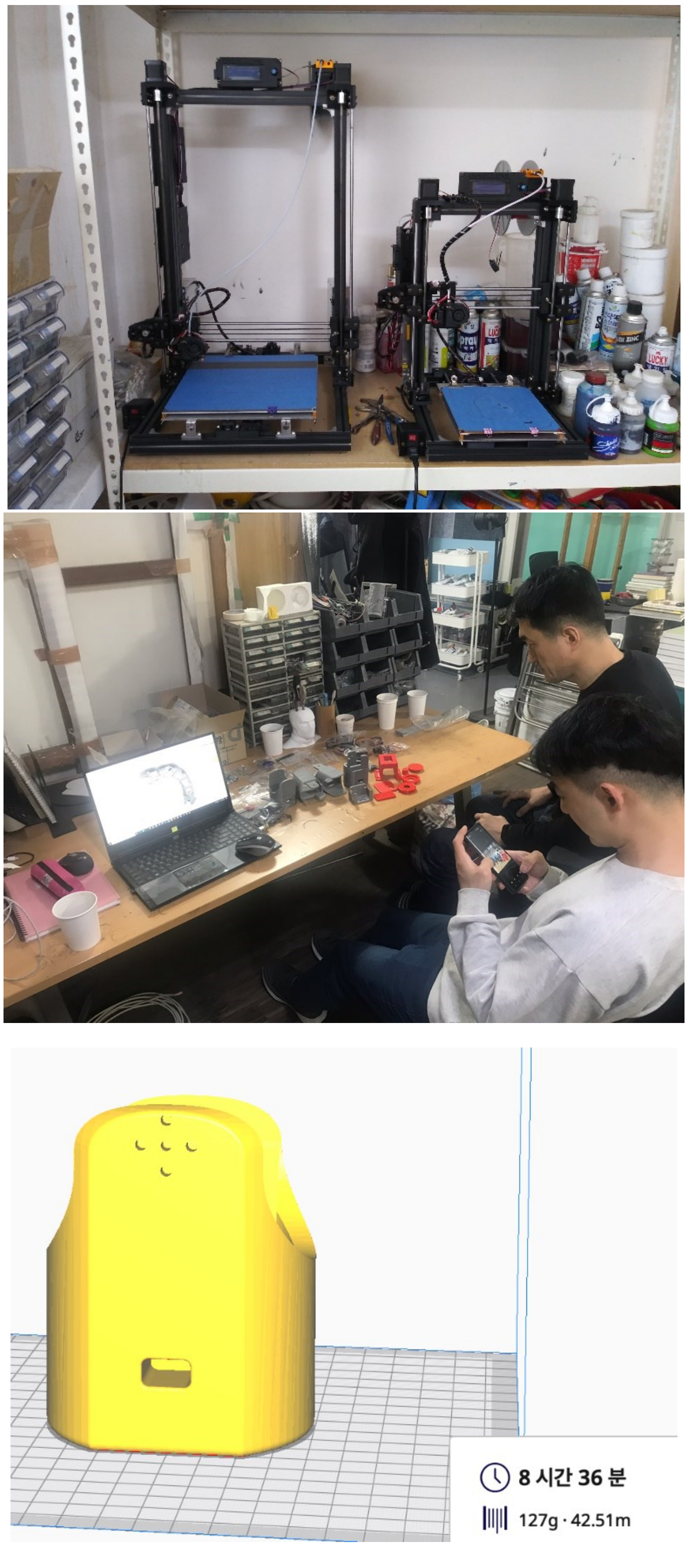


Figure 7. 3D printers and parts

The robot arm was designed on a CAD software. Our team chose Fused Deposition Modeling (FDM) as a printing method for 3D printing. Such method uses a thermoplastic filament, which is heated to its melting point and then extruded, layer by layer, to create a three-dimensional object. We used PETG, polyethylene terephthalate glycol, for parts in grey, with nozzle temperature 230°C and bed temperature 80°C. For parts in red, we used PLA, polylactic acid, with nozzle temperature 205°C and bed temperature 65°C. The printing took 73 hours in total.

Software Design

As mentioned above, there was risk in implementing inverse kinematics software on the 3D printed robot arm without enough validation. Thus, we adopted a test robot arm that has a different configuration in terms of length of each body parts and base heights. The test robot, however, maintained the same degrees of freedom and similar end-effector. Our goal was to check if the robot moves to a designated coordinate by utilizing an inverse kinematics system. After fully understanding inverse kinematics system, we searched for codes on the Internet, which we could modify and implement. We discovered several motor PWM control libraries and some inverse kinematics codes. We carefully chose codes that could be applied with our system configuration, namely PCA9685 motor driver module.

Inverse Kinematics System

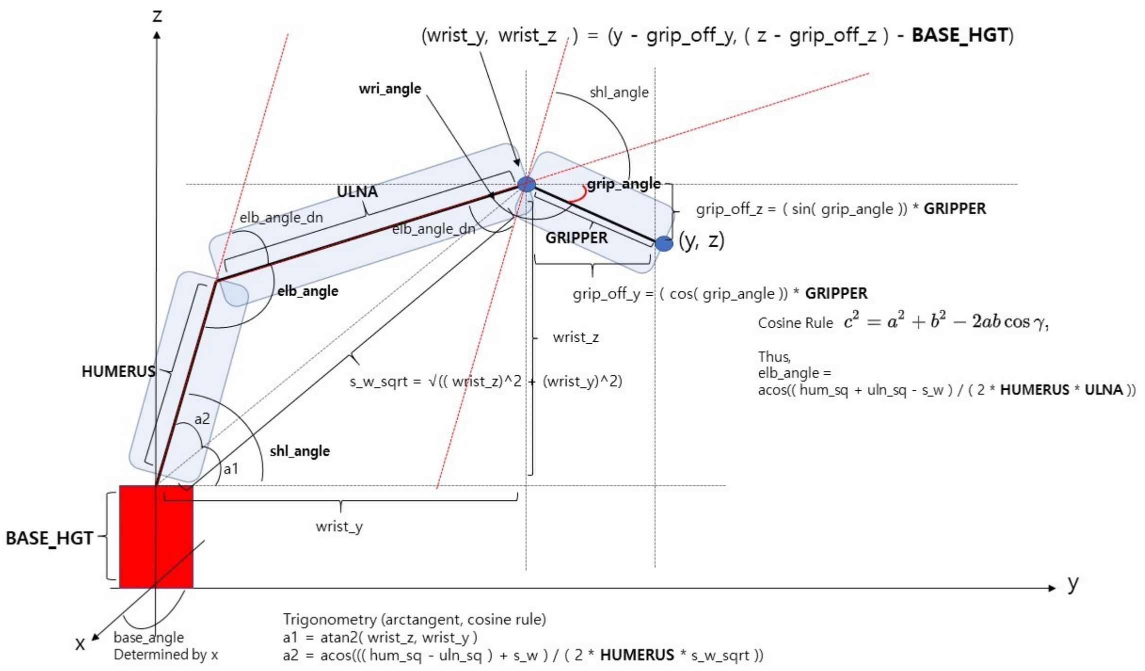

z
(wrist_y, wrist_z) = (y - grip_off_y, (z - grip_off_z) - BASE_HGT)
wri_angle
shl_angle
ULNA
grip_angle
grip_off_z = (sin(grip_angle)) * GRIPPER
elb_angle_dn
elb_angle_dn
GRIPPER
(y, z)
elb_angle
grip_off_y = (cos(grip_angle)) * GRIPPER
Cosine Rule $c^2 = a^2 + b^2 - 2ab\cos\gamma,$
wrist_z
Thus,
elb_angle =
acos((hum_sq + uln_sq - s_w) / (2 * HUMERUS * ULNA))
HUMERUS
s_w_sqrt = √((wrist_z)^2 + (wrist_y)^2)
a2
shl_angle
a1
BASE_HGT
wrist_y
y
x
base_angle
Determined by x
Trigonometry (arctangent, cosine rule)
a1 = atan2(wrist_z, wrist_y)
a2 = acos(((hum_sq - uln_sq) + s_w) / (2 * HUMERUS * s_w_sqrt))

Figure 8. Inverse kinematics equation

The kinematic system utilized several different packages in Arduino with open-source libraries for motor PWM control and math functions of inverse kinematics.

Inside software, a set_arm (float x, float y, float z, float grip_angle_d) is a main function with the object coordinate as an input and joint parameter as outputs. Input-coordinates of (x, y, z) belong to a XYZ plane. The software first calculates a base_angle from which is calculated by arctangent of (x, y) on a XY plane. The rest angles and corresponding parameters are all determined on a YZ plane. (y, z) refers a distance from base to the object on a Y axis, and the height of the object, respectively. Grip_angle, a gripper angle, is a constant value of an angle between a parallel line and a gripper. (see figure 9)

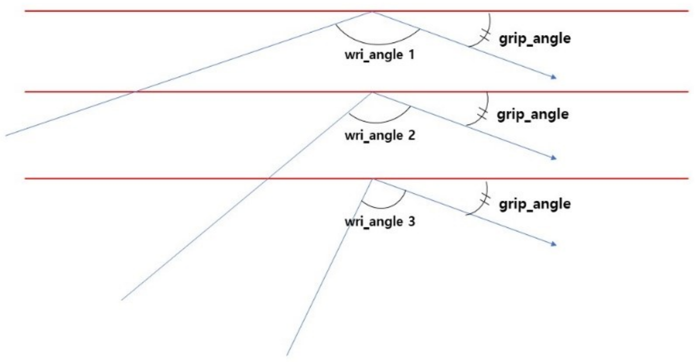


Figure 9. Grip_angle (a gripper angle)

On the other hand, wrist angle, an inner angle between ‘ULNA’ and ‘GRIPPER’ is a variable. Another to consider is that grip_angle is a negative value when it rotates from the imaginary parallel line (see red lines on figure 9) towards the surface (the gripper faces downward). Thus, we are able to calculate coordinates of the wrist (wrist_y, wrist_z) with the following equations.

grip_off_y = (cos(grip_angle)) * GRIPPER
grip_off_z = (sin(grip_angle)) * GRIPPER
(wrist_y, wrist_z) = (y - grip_off_y, (z - grip_off_z) - BASE_HGT)

With wrist coordinates provided, the distance between the origin (base) and wrist is then calculated.

s_w_sqrt = √((wrist_z)^2 + (wrist_y)^2)

By using law of cosine, we can calculate elb_angle, and ‘a2’ (Figure 8). Arctangent (wrist_y, wirst_z) gives the value of ‘a1’ (Figure 8). The sum of ‘a1’ and ‘a2’ is shl_angle. The following is the formula.

a1 = atan2(wrist_z, wrist_y)
a2 = acos(((hum_sq - uln_sq) + s_w) / (2 * HUMERUS * s_w_sqrt))
shl_angle = a1 + a2
elb_angle = acos((hum_sq + uln_sq - s_w) / (2 * HUMERUS * ULNA))

Wrist_angle is calculated by the angle required to rotate in a clockwise from its 0-degree starting from ‘ULNA’ direction. (Figure 8)

elb_angel > 0, shl_angle > 0, grip_angle <0,
elb_angle_dn = - (180 – elb_angle) < 0
wri_angle > 0
wri_angle = - 180 – (shl_angle + elb_angle_dn) + grip_angle
= - 180 – shl_angle - elb_angle_dn + grip_angle
= - 180 – shl_angle + 180 - elb_angle + grip_angle
= grip_angle_d - shl_angle_d - elb_angle_d

Joint parameters in degrees are then transformed to pulse signals to conduct PWM. To do so, we used “PCA9685.h,” an Arduino library for PCA9685 16-Channel PWM Driver Module. This project applies 180 degree controlled digital servo motors that run on a 20ms pulse width, equivalent to 50Hz frequency, to maneuver the robot arm. Once adequate frequency is defined in a setup function, each motor object is assigned to the connected Pin number on PCA9685 board. Then PWM controller function provides corresponding PWM signals to rotate motors.

```
pwmController.setChannelPWM(6, pwmServo1.pwmForAngle(bas_angle_d));
pwmController.setChannelPWM(0, pwmServo2.pwmForAngle(shl_angle_d - 105));
pwmController.setChannelPWM(1, pwmServo3.pwmForAngle(elb_angle_d + 22));
pwmController.setChannelPWM(2, pwmServo4.pwmForAngle(wri_angle_d + 20));
pwmController.setChannelPWM(4, pwmServo5.pwmForAngle(wro_angle_d));
pwmController.setChannelPWM(5, pwmServo6.pwmForAngle(grip_angle_d));
```

The following is a ‘pwmForAngle’ function included in “PCA9685.h” library.

```
uint16_t PCA9685_ServoEvaluator::pwmForAngle(float angle) {
    float retVal;
    angle = constrain(angle + 90, 0, 180);
    if (!_isCSpline) {
        retVal = _coeff[0] + (_coeff[1] * angle);
    }
    else {
        if (angle <= 90) {
            retVal = _coeff[0] + (_coeff[1] * angle) + (_coeff[2] * angle * an
gle) + (_coeff[3] * angle * angle * angle);
        }
        else {
            angle -= 90;
            retVal = _coeff[4] + (_coeff[5] * angle) + (_coeff[6] * angle * an
gle) + (_coeff[7] * angle * angle * angle);
        }
    }
    return (uint16_t)constrain((int)roundf(retVal), 0, PCA9685_PWM_FULL);
}
```

Loop function includes a simple ‘set_arm()’ function with coordinates provided manually.

Overall System Integration

Six servo motors, located in each body joint, are connected to PCA9685 16-Channel Motor Driver Module. GPIO pins 26 and 20 with fixed frequency of 50Hz. Software PWM was used to control the two motors as they do not require a great deal of precision. The motor driver was connected to a DC-DC convertor that converts 11.1V of battery voltage to 5V. Power is provided to V+ and GND on PCA9685. The motor driver was then connected to Arduino Uno in the following way: SCL - Analog pin 5, SDA – Analog pin4, GND – GND, VCC, 5V. The battery is also connected to another DC-DC convertor that converts 11.1 to 5V for signaling Arduino Uno.

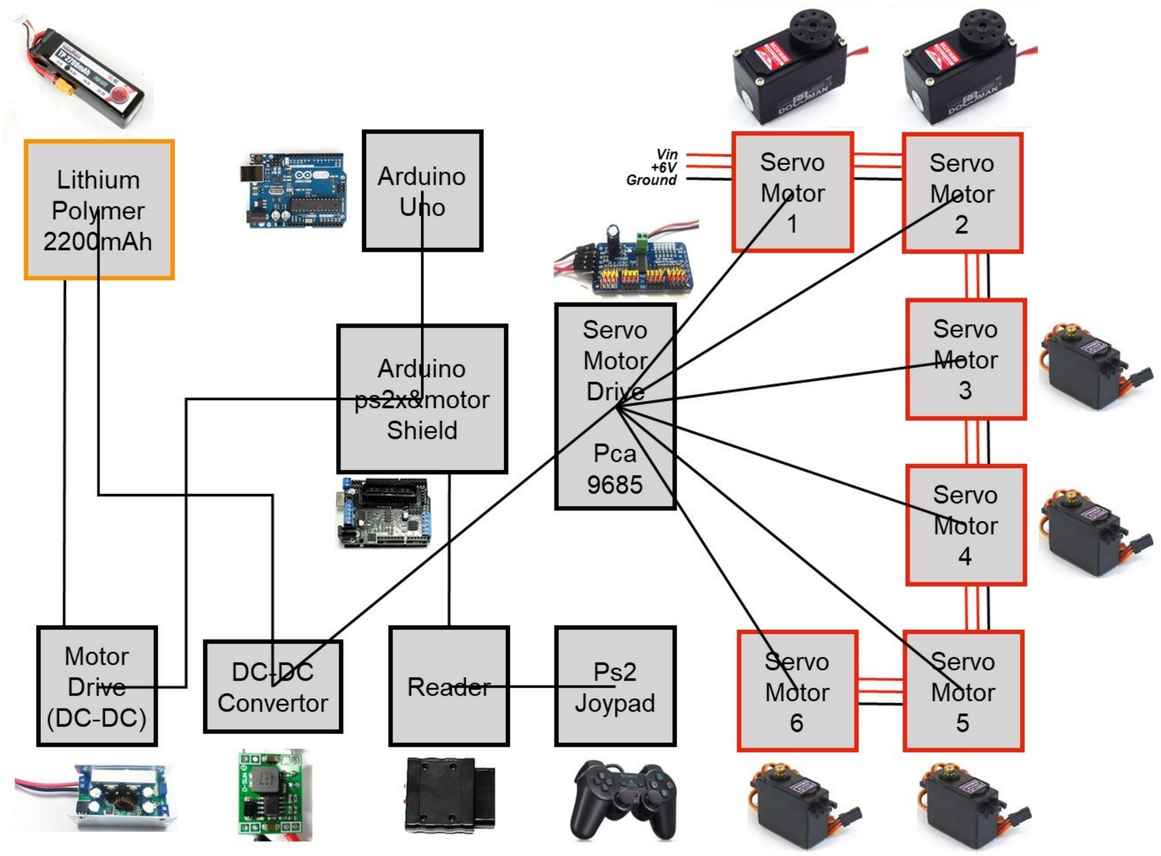


Figure 10. Schematic of the Overall System

Results and Future Work

Our robot performed as planned and met the goals we expected. Prior to testing our robot arm, we had conducted tests with existing robots to understand their dynamics, kinematics, and software. For the sake of practice, we created a 3D CAD model of an existing model.

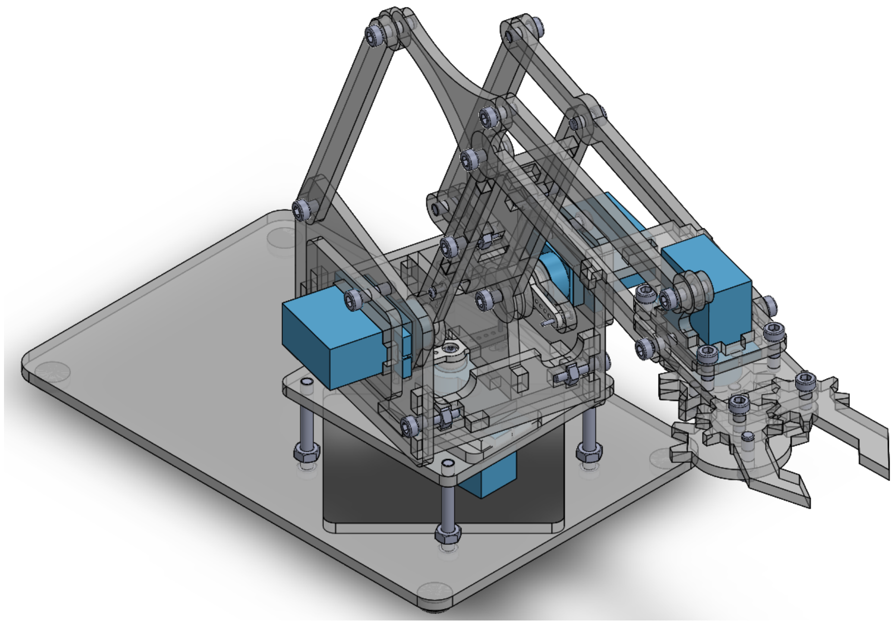


Figure 11. 3D CAD design practice

By observing movement of a comparison model, we could easily check whether the movement of the robot arm is good or not. It turned out that our robot arm was successful in its movement, and we checked mainly with a controller, before applying an inverse kinematics software.

Hardware Testing results

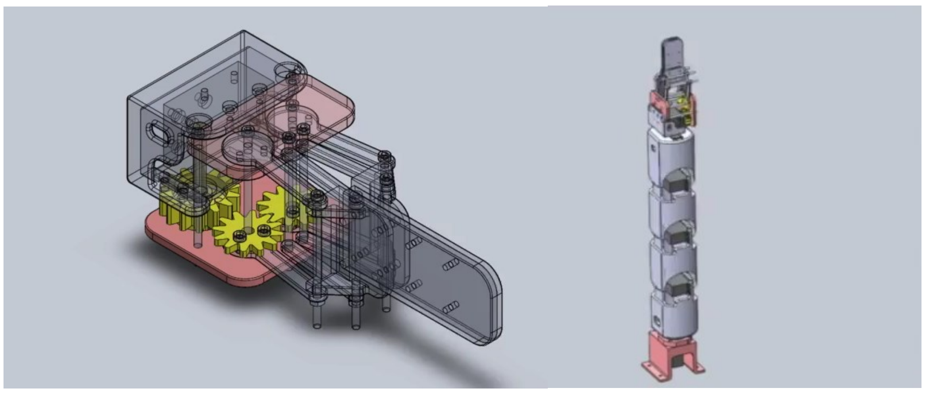


Figure 12. 3D CAD simulation Gripper (left), body (right)

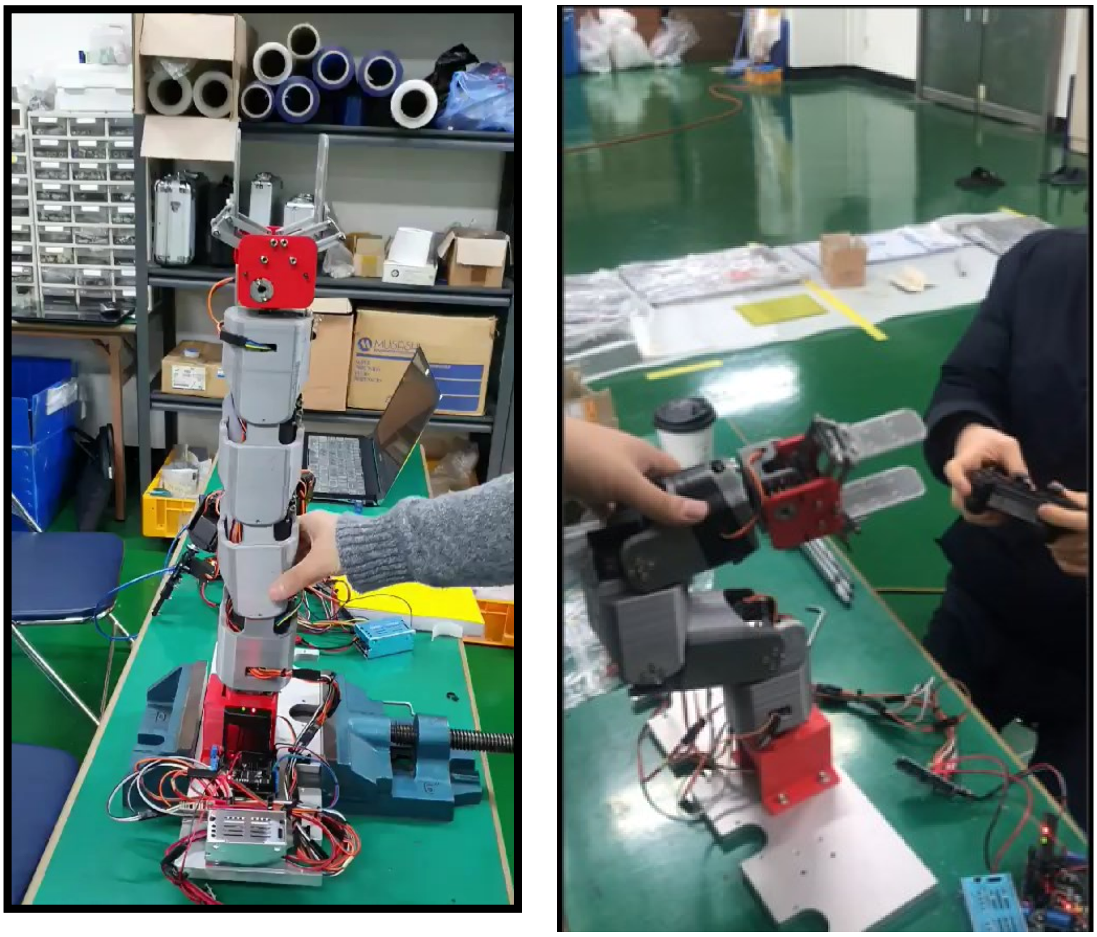


Figure 13. The Robot Arm actuation test

3D CAD simulation was successful that it well represented our design. Geometry was good that there was no assembly issue. While the bearing base structure did support axial loads of the robot arm, we realized that a base motor with a larger torque would have been better to support moment of the robot arm. Sometimes, we had to hold the bottom part of the robot arm to stabilize its posture.

Further discussion on buttressing lower body has followed, and we purchased motors with stronger torque to supplement the bottom part.

Software Testing results

Our main goal was to check whether the robot moves to a designated coordinate. As our team was confident about analysis on inverse kinematics and its code, we coded the robot arm to stand erect initially, then to move to a designated location. We expected the robot to move according to plan, however, the robot arm did stand vertically, but mobilized in somewhat random direction thereafter. We had to verify what the problem was. It turned out that we had skipped motor calibration and omitted to have the same direction for every motor implementation. As the initial position of each motor was different, each provided coordinate resulted in a different position. This implies that every motor direction should be aligned with the inverse kinematics analysis.

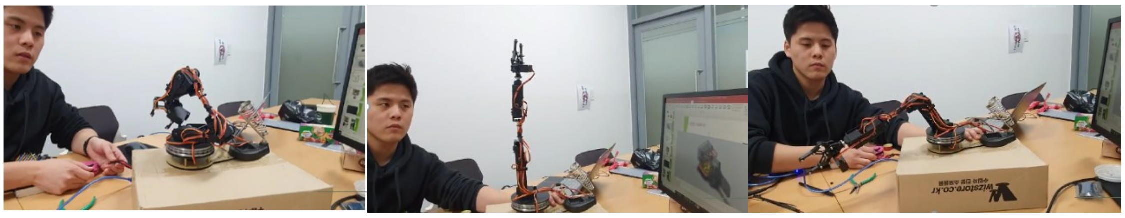


Figure 14. Inverse Kinematics test begin (left)
an initial position (middle), move to a designated coordinate (right)

For the future work, we plan to include a function to open and close gripper after moving to a designated position. The current code gives direct instruction to PWM signals to open and close the gripper. This needs to be integrated as a function, such as ‘grip_open()’ and ‘grip_close()’ for better use.

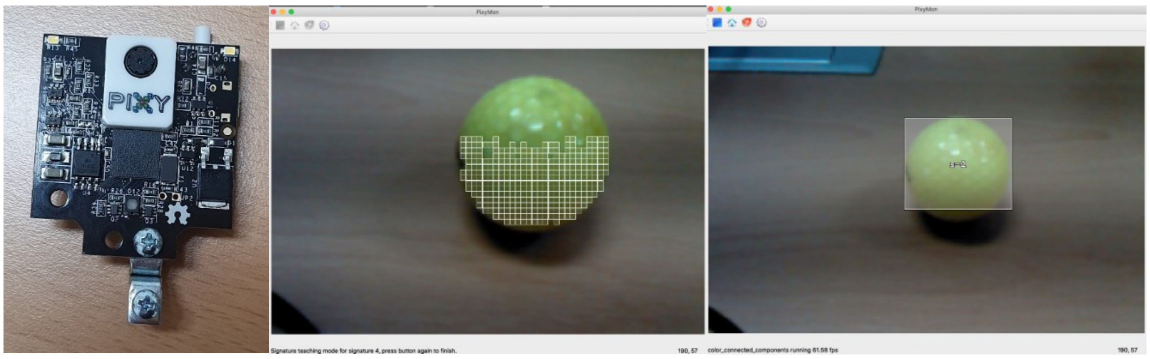


Figure 15. Pixy2 camera module(left), image detecting by hue (middle),
image tracking by hue (right)

Another goal in this project was to implement vision sensor to detect an object and calculate its coordinate. As for detecting and tracking an object by color, we applied Pixy2 camera module and a relevant software. While we were successful in detecting and tracking an object by pixy, we were unclear in calculating a distance between Pixy2 and a target object. We had to decide installation position of the camera module, mobility of the camera module, prevention of any visual hinderance, etc. All of these had to be decided before further software was implemented, but with limited time available, we discussed to postpone vision sensor implementation to the future works.

```
#include <Pixy2.h> //Include Pixy2 for visual recognition that would be inputs
Pixy2 Pixy;
#define X_CENTER ((PIXY_MAX_X - PIXY_MIN_X)/2)
#define Y_CENTER ((PIXY_MAX_Y - PIXY_MIN_Y)/2)
boolean FoundBlock = false;
uint16_t blocks, j;
blocks = pixy.getBlocks(); //Check for objects.
```

Conclusion

The overall final design, development, and demonstration were successful. The robot was able to navigate to a designated coordinate in an open environment. Hardware design was successful, and 3D printed materials were solid enough that the robot could withstand against external forces. Our future works include detecting a yellow golf ball target, approaching and retrieving to another detected position by, for example, detecting a basket in a random position. The robot will also be enhanced in its bottom part to maintain a more stable position.

After completing the project, we learned many lessons. First, we realized the importance of motor calibration and proper initial positions. Before realizing these problems, we were in agony trying to verify problems in our codes. Second, we realized that more delicate and precise movement is required for robots. Control theories, signal processing, and accompanying tools must be utilized in our future works. In conclusion, we were able to address hardware, software, and system aspects to produce an inverse kinematics robot arm. Our future works involve a more stability, vision-sensing function, and more precise movement.

Appendix

Code

```
#include <stdio.h>
#include <Wire.h>           //for using a controller or connecting Pixy2
#include "PCA9685.h"        //PWM motor control

#define BASE_HGT 65.00      //base height
#define HUMERUS 105.00      //shoulder-to-elbow "bone"
#define ULNA 150.00         //elbow-to-wrist "bone"
#define GRIPPER 180.00      //gripper
#define IN1 4               //wrist rotate
#define IN2 5               //gripper angle
#define IN3 6               //base angle
#define IN4 0               //shoulder angle
#define IN5 1               //elbow angle
#define IN6 2               //wrist angle

PCA9685 pwmController;
PCA9685_ServoEvaluator pwmServo1;
PCA9685_ServoEvaluator pwmServo2;
PCA9685_ServoEvaluator pwmServo3;
PCA9685_ServoEvaluator pwmServo4;
PCA9685_ServoEvaluator pwmServo5;
PCA9685_ServoEvaluator pwmServo6;

float hum_sq = HUMERUS*HUMERUS;
float uln_sq = ULNA*ULNA;

void circle();
void line();
void zero_x();
void set_arm( float x, float y, float z, float grip_angle_d );

void setup() {
    Serial.begin(57600);
    Wire.begin();
    Wire.setClock(400000);
    pwmController.resetDevices();
    pwmController.init(B000000);
    pwmController.setPWMFrequency(50);
    pwmController.setChannelPWM(6, pwmServo1.pwmForAngle(30));
       //base 20 degrees error
    pwmController.setChannelPWM(0, pwmServo2.pwmForAngle(-15));
       //shoulder -15 degrees error (+) CCW, (-) CW
    pwmController.setChannelPWM(1, pwmServo3.pwmForAngle(22));
```

```
       //elbow 22 degrees error (+) CCW
    pwmController.setChannelPWM(2, pwmServo4.pwmForAngle(20));
       //wrist 20 degrees error (+) CW
    pwmController.setChannelPWM(4, pwmServo5.pwmForAngle(0));
    pwmController.setChannelPWM(5, pwmServo6.pwmForAngle(30));
       //open 30, close -5
    delay(3000);
}

void loop()
{
    //zero_x();
    //line();
    //circle();
    delay(5000);
    set_arm(0, 330, 330, 30);
    delay(5000);
    pwmController.setChannelPWM(5, pwmServo6.pwmForAngle(-5));
    delay(5000);
    set_arm(100, 330, 330, 30);
    delay(2000);
    pwmController.setChannelPWM(5, pwmServo6.pwmForAngle(30));
}

/* z is height, y is distance from base center out, x is side to side. y,z can
 only be positive */
void set_arm( float x, float y, float z, float grip_angle_d ) {
    float grip_angle_r = radians( grip_angle_d );
       //grip angle in radians for use in calculations
       /* Base angle and radial distance from x,y coordinates */
    float bas_angle_r = atan2( x, y );
    float bas_angle_d = degrees(bas_angle_r);
       // transform to degree from radian
    float rdist = sqrt(( x * x ) + ( y * y ));
       /* rdist is y coordinate for the arm */
    y = rdist;
       /* Grip offsets calculated based on grip angle */
    float grip_off_z = ( sin( grip_angle_r )) * GRIPPER;
    float grip_off_y = ( cos( grip_angle_r )) * GRIPPER;
       /* Wrist position */
    float wrist_z = ( z - grip_off_z ) - BASE_HGT;
    float wrist_y = y - grip_off_y;
       /* Shoulder to wrist distance */
    float s_w = ( wrist_z * wrist_z ) + ( wrist_y * wrist_y );
    float s_w_sqrt = sqrt( s_w );
       /* s_w angle to ground */
       //float a1 = atan2( wrist_y, wrist_z );
    float a1 = atan2( wrist_z, wrist_y );
```

```
       /* s_w angle to humerus */
    float a2 = acos((( hum_sq - uln_sq ) + s_w ) /( 2 * HUMERUS * s_w_sqrt ));
       /* shoulder angle */
    float shl_angle_r = a1 + a2;
    float shl_angle_d = degrees( shl_angle_r );
       /* elbow angle */
    float elb_angle_r = acos((hum_sq + uln_sq - s_w )/( 2 * HUMERUS * ULNA ));
    float elb_angle_d = degrees( elb_angle_r );
    float elb_angle_dn = -( 180.0 - elb_angle_d );
       /* wrist angle */
    float wri_angle_d = ( grip_angle_d - elb_angle_dn ) - shl_angle_d;
       /* wrist rotation */
    float wro_angle_d = 90 ;
    grip_angle_d = constrain(grip_angle_d, -40, 40);

    //RB038 PCA9685 object use
    pwmController.setChannelPWM(6, pwmServo1.pwmForAngle(bas_angle_d));
    pwmController.setChannelPWM(0, pwmServo2.pwmForAngle(shl_angle_d - 105));
    pwmController.setChannelPWM(1, pwmServo3.pwmForAngle(elb_angle_d + 22));
    pwmController.setChannelPWM(2, pwmServo4.pwmForAngle(wri_angle_d + 20));
    pwmController.setChannelPWM(4, pwmServo5.pwmForAngle(wro_angle_d));
    pwmController.setChannelPWM(5, pwmServo6.pwmForAngle(30));

}

/* moves arm in y axis */
void zero_x() {
    for( double yaxis = 200.0; yaxis < 250.0; yaxis += 1 ) {
    set_arm( 0, yaxis, 300.0, 0 );
    delay( 10 );
    }
    for( double yaxis = 250.0; yaxis > 200.0; yaxis -= 1 ) {
    set_arm( 0, yaxis, 300.0, 0 );
    delay( 10 );
    }
}

/* moves arm in a straight line*/
void line() {
    for( double xaxis = -100.0; xaxis < 100.0; xaxis += 0.5 ) {
    set_arm( xaxis, 200, 100, 0 );
    delay( 10 );
    }
    for( float xaxis = 100.0; xaxis > -100.0; xaxis -= 0.5 ) {
    set_arm( xaxis, 200, 100, 0 );
    delay( 10 );
    }
}
```

```
/* draw circle on a y-z plane */
void circle()
{
    #define RADIUS 50.0
    float zaxis, yaxis;
    for(float angle = 0.0; angle < 360.0; angle += 1.0 ) {
    yaxis = RADIUS * sin( radians( angle )) + 200;
    zaxis = RADIUS * cos( radians( angle )) + 100;
    set_arm( 0, yaxis, zaxis, 0 );
    delay( 10 );
    }
}
```

Bill of Materials

Parts	Cost/Unit	Units	Total Cost
Arduino UNO	$25	1	$25 (Not Included)
Pixy 2 Camera Module	$70	1	$70
MG996R Servo Motor	$4	6	$24
ds5160 60kgf-cm torque Servo Motor	$30	2	$60
3D Print materials	$0	-	$0
Battery Pack	$0	1	$0
		Total Cost:	**$154**

Member Contributions

The overall work involved the collaboration of Bruce KwangKyun Kim, IlGyu Cho, YongWook Kim, Jung Ho Song, and Eunho Park. Bruce and IlGyu were mainly responsible for the software development. They analyzed inverse kinematics of the robot arm, discovered and modified open-source codes, and integrated them with other open-source algorithms for servo motor controls. Also, they managed to integrate software and hardware into an Arduino system by setting configuration of the overall system and wiring all the connections. YongWook mainly designed the robot arm with his concept article provided and conducted 3D printing. Jung Ho and Eunho modelled 2D & 3D CAD, respectively. Eunho made 3D simulations to check if the assembled model can motion with the given freedoms of degrees. The final overall system assembly and debugging required the combined efforts of all members to try and understand if the errors in the system were caused by a software or hardware issue or both.

References

Robotic Arm Inverse Kinematics on Arduino By Oleg Mazurov
Arduino Library for the PCA9685 16-Channel PWM Driver Module by Kasper Skårhøj
https://arduino.cc/
https://pixycam.com/
https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:hooking_up_pixy_to_a_microcontroller_-28like_an_arduino-29

Acknowledgements

We would like to thank Professor Seyoun Chung and professor Misoo Kim launching school's engineering contest and letting us acquire precious opportunity to develop a project. Both professors have assisted us in sharpening acumen regarding embedded systems and project designs in general.