```
// Gives access to respective libraries
#include <Servo.h>
#include <Stepper.h>
#include <math.h>

// Assigns variables of type 'int' - no decimal place (whole number)
int clockwisePosition = 10, anticlockwisePosition = 170, elbowPosition = 170, wristPosition = 90;
int step = 0, leftEdge, rightEdge, objectPosition, initialPosition;

// Assigns variables of type 'const int' - cannot be changed and has no decimal point (whole number)
const int stepsPerSweep = 1024, minDistance = 22, maxDistance = 40, limbLength = 20;
const int trigPin = 3, echoPin = 2;

// Assigns variables of type 'float' - has a decimal place
float time, distance, shoulderAngle, elbowAngle, wristAngle, gripperAngle = 10;

// Assigns variable 'objectFound' of type 'bool' - true or false
bool objectFound = false;

// Assigns variables to be servos
Servo clockwiseServo, anticlockwiseServo, elbowServo, wristServo, gripperServo;
// Initializes pins 9 through 12 for use by stepper motor with 4096 steps per revolution
Stepper baseStepper(4096, 9, 11, 10, 12);

// Runs once on start
void setup()
{
  // Configures 'trigPin' to behave as an output
  pinMode(trigPin, OUTPUT);
  // Configures 'echoPin' to behave as an input
  pinMode(echoPin, INPUT);
}

// Runs repeatedly
void loop()
{
  // Loops contents the value of 'stepsPerSweep' (1024) times - 180°
  for (step; step < stepsPerSweep; step++)
  {
    // Calls the 'ping' function to measure distance
    ping();

    // If the value of distance is less or equal to maxDistance (40), perform contents
    // If both of these conditions are met then an object has been found
    if (distance >= minDistance && distance <= maxDistance)
    {
      /* Repeats until the if statement runs through for the last time
      and so results in the step at the final (or right-most) edge of the object, assigning
      said value to 'rightEdge'*/
      rightEdge = step;

      // If 'objectFound' is false, perform contents
      // Runs once when the object is found to find the initial (or left-most) edge
      if (objectFound == false)
      {
        // Make object found true
        objectFound = true;
        // Assign 'leftEdge' to the current step
        leftEdge = step;
      }
    }

    // Turns stepper motor 1 step clockwise (top-down) each loop iteration
    baseStepper.step(-1);
    // Pauses for 10 milliseconds
    delay(10);

    /* If the object has been found and either distance is greater than 'maxDistance'
    or distance is equal to 0 (in the case that the distance was too great for the
    ultrasonic
    sensor to detect), perform contents.
    This is to detect when the stepper has passed the object so it can return.*/
    if (objectFound == true && (distance > maxDistance || distance == 0.00))
    {
      // Leaves the for loop - stops turning stepper
      break;
    }
  }

  // Finds the position of the object within the 40cm radius
  // '-30' to calibrate the position due to hardware inaccuracies
  objectPosition = (rightEdge - leftEdge) / 2.0 + leftEdge - 30;

  // Turns stepper ccw (top-down) to return to object
  for (step; step > objectPosition; step--)
  {
    // Turns stepper 1 step ccw (top-down) each loop iteration
    baseStepper.step(1);
    // Pauses for 10 milliseconds
    delay(10);
  }

  // Calls the 'ping' function to measure distance from centre of object
  ping();

  // If the value of distance is greater than or equal to 'minDistance' (21.5)
  // or less or equal to 'maxDistance' (40), perform contents
  // Makes sure the arm is actually pointing at object
  if (distance >= minDistance && distance <= maxDistance)
  {
    // Calls the 'calculations' function to calculate the servo angles
    calculations();

    // Calls the 'attach' function to attach servos
    attach();

    // Calls the 'extend' function to extend arm towards object
    extend();
    delay(1000);

    // While gripperAngle is less than 90°, perform contents - Grips object
    while (gripperAngle < 90)
    {
      // Turns 1° each loop iteration
      gripperServo.write(gripperAngle++);
      // Pauses for 5 milliseconds
      delay(5);
    }
    // Pauses for 1000 milliseconds
    delay(1000);

    // Calls the 'flex' function
    flex();
    // Pauses for 1000 milliseconds
    delay(1000);

    // Returns stepper to initial position
    for (step; step > 0; step--)
    {
      // Turns stepper 1 step ccw (top-down) each loop iteration
      baseStepper.step(1);
      delay(10);
    }
    // Pauses for 1000 milliseconds
    delay(1000);

    // Calls the 'extend' function to drop object
    extend();
    // Pauses for 1000 milliseconds
    delay(1000);

    // While gripperAngle is greater than 10°, perform contents - Releases object
    while (gripperAngle > 10)
    {
      // Turns 1° back each loop iteration
      gripperServo.write(gripperAngle--);
      delay(5);
    }
    // Pauses for 1000 milliseconds
    delay(1000);

    // Calls the 'rest' function to return servos to rest positions
    rest();
    // Pauses for 1000 milliseconds
    delay(1000);
  }

  // Terminates the program
  exit(0);
}

// Declares 'ping' function - measures distance
void ping()
{
  // Writes a high value to 'trigPin' - emits sound
  digitalWrite(trigPin, HIGH);

  // Pauses for 10 microseconds
  delayMicroseconds (10);

  // Writes a low value to 'trigPin' - stops emiting sound
  digitalWrite(trigPin, LOW);

  /* pulseIn reads a high pulse from 'trigPin' and waits for a low pulse
  before stopping a timer and assigning said value to 'time'-
  recorded in microseconds */
  /* pulseIn is set so that it takes 1 second execute, thus it is necessary
  to decrease that time to 0.1 of second (hence '10000UL') so that each step
  of the stepper motor will not take a second to complete (since 'ping' is
  run each time the stepper takes a step)*/
  time = pulseIn(echoPin, HIGH, 10000UL);

  /* Converts time to distance by timesing itself by the speed of sound
  (0.0343 cm/µs) and halfs the result to remove the distance sound took
  to reflect back - distance is in cm.
  '-6' to remove the length of the wrist when extending to object*/
  distance = (time * 0.0343) / 2.0 - 6;
}

// Declares 'calculations' function to calculate servo angles
// All working for the calculations below can be found in my portfolio
void calculations()
{
  // Calculates 'shoulderAngle' using trigonometry
  shoulderAngle = acos((distance / 2.0) / limbLength);

  // Converts 'shoulderAngle' from radians to degrees
  shoulderAngle *= (180.0 / M_PI);

  // Calculates 'elbowAngle' using geometry
  elbowAngle = (90 - shoulderAngle) * 2.0;

  /* Since the elbow servo is pushing the forearm, the angle assigned to the servo
  is not the same as the angle calculated using trigonometry and geometry. Thus,
  the relationship used below corrects for this - evidence found in portfolio */
  elbowAngle = -16.0/17.0 * elbowAngle + 220;

  // Calculates 'wristAngle' using geometry
  wristAngle = 90 + shoulderAngle;

  /* At rest, the shoulder has a range of 125° that it can turn (that is to say that
  if the servo were to turn by 125° it would be parallel with the ground). Since the
  angle within this range has been calculated, we must substract it from 125° to find
  by how much the shoulder should be turned by */
  shoulderAngle = 125  - shoulderAngle;
```

```
}

// Declares 'attach' function to attach servos and set them to
// their rest positions to stop them from returning to 0
void attach()
{
  clockwiseServo.attach(8);
  clockwiseServo.write(10);

  anticlockwiseServo.attach(7);
  anticlockwiseServo.write(170);

  elbowServo.attach(6);
  elbowServo.write(170);

  wristServo.attach(5);
  wristServo.write(90);

  gripperServo.attach(4);
  gripperServo.write(10);
}

// Declares 'extend' function to extend arm
void extend()
{
  // As long as any of the conditions within the parentheses are met, loop contents - || means 'or'
  // Turns servos simultaneously, 1° each loop iteration
  while (clockwisePosition < shoulderAngle || elbowPosition > elbowAngle || wristPosition < wristAngle)
  {
    // If 'clockwisePosition' is less than 'shoulderAngle', perform contents
    // Rotates shoulder servos simultaneously in opposite directions
    if (clockwisePosition < shoulderAngle)
    {
      /* 'clockwiseServo' has a rest position at 10°. The varable 'clockwisePosition'
      is equal to 10° and is assigned to 'clockwiseServo' before having 1 added to it.
      Simply, since the statement is within a loop, it will turn 'clockwiseServo' 1°
      each loop iteration and will stop only when the servo has reached the desired
      postion, that being at the value of 'shoulderAngle'. */
      clockwiseServo.write(clockwisePosition++);
      // Same concept applied as above but in the anticlockwise direction
      anticlockwiseServo.write(anticlockwisePosition--);
      // Pauses for 10 milliseconds
      delay(10);
    }

    // If 'elbowPosition' is greater than 'elbowAngle', perform contents
    if (elbowPosition > elbowAngle)
    {
     // Turns 1° toward 'elbowAngle' each loop iteration
      elbowServo.write(elbowPosition--);
    }

    // If 'wristPosition' is less than 'wristAngle', perform contents
    if (wristPosition < wristAngle)
    {
      // Turns 1° toward 'wristAngle' each loop iteration
      wristServo.write(wristPosition++);
    }
  }
}

// Same concept as the 'extend' function but in the opposite direction - acts as an intermediate position
// while the arm is holding the object as the stepper returns the arm to the starting point
// (values are hard coded unlike above)
void flex()
{
  while (clockwisePosition > 35 || elbowPosition < 170 || wristPosition > 125)
  {
    if (clockwisePosition > 35)
    {
      clockwiseServo.write(clockwisePosition--);
      anticlockwiseServo.write(anticlockwisePosition++);
      delay(10);
    }

    if (elbowPosition < 170)
    {
      elbowServo.write(elbowPosition++);
    }

    if (wristPosition > 125)
    {
      wristServo.write(wristPosition--);
    }
  }
}

// Again, same concept as the 'extend' function but in the opposite direction - once the object has been
// dropped off the arm will return to a 'rest' (or neutral) position
// (values are again hard coded unlike in 'extend')
void rest()
{
  while (clockwisePosition > 10 || elbowPosition < 170 || wristPosition > 90)
  {
    if (clockwisePosition > 10)
    {
      clockwiseServo.write(clockwisePosition--);
      anticlockwiseServo.write(anticlockwisePosition++);
      delay(10);
    }

    if (elbowPosition < 170)
    {
      elbowServo.write(elbowPosition++);
    }

    if (wristPosition > 90)
    {
      wristServo.write(wristPosition--);
    }
  }
}
```

```
// Accesses respective libraries
#include <math.h>
#include <Servo.h>
#include <Stepper.h>

float time, distance; // Declares fraction variables

Servo cwShoulder, ccwShoulder, elbow, wrist, gripper; // Servo variables
Stepper stepper(4096, 5, 7, 6, 8); // Stepper motor using pins 9 - 12; 4096 SPR

// Constant integers
const int shoulderRest = 10, elbowRest = 170, wristRest = 90, gripperRest = 10;
const int minDist = 22, maxDist = 40, stepRange = 1024;
const int trigPin = 2, echoPin = 3, limbLength = 20;
const int clockwise = 1, counterclockwise = -1;

// Integer variables
int shoulderAngle, elbowAngle, wristAngle, gripperAngle = 90;
int step = 0, initial, final, position[5], i = 0;

bool obstruction; // True or false

void setup() { //   Runs once on start
  pinMode(trigPin, OUTPUT); // trigPin emits signal

  pinMode(echoPin, INPUT); // echoPin recieves signal
  attach(); // Attaches servos
}

void loop() { // Runs repeatedly
  controller();
}

float ping() { // Returns range
  digitalWrite(trigPin, HIGH); // Sends signal through trigPin

  delayMicroseconds (10); // Pauses for 10 µs while signal is emitted

  digitalWrite(trigPin, LOW); // Stops sending signal through trigPin

  time = pulseIn(echoPin, HIGH); // Reads duration of pulse from echoPin

  range = (time * 0.0343) / 2; // d = vt where v = 0.0343 cm/µs

  return range; // Returns range upon calling the function
}

void arithmetic(float dist) { // Calculates angles for servos from given distance
  // Proof earlier in digest
  shoulderAngle = acos((dist / 2) / limbLength);

  shoulderAngle *= (180 / M_PI); // Radians to degrees

  elbowAngle = -19/20 * 2 * (90 - shoulderAngle) + 220;

  wristAngle = 90 + shoulderAngle;

  shoulderAngle = 130 - shoulderAngle;
}

void attach() { // Attaches servos and sets them to rest values
  cwShoulder.attach(12); // Attaches servos to respective pins
  cwShoulder.write(shoulderRest); // Writes servos to rest values

  ccwShoulder.attach(13);
  ccwShoulder.write(180 - shoulderRest);

  elbow.attach(11);
  elbow.write(elbowRest);

  wrist.attach(10);
  wrist.write(wristRest);

  gripper.attach(9);
  gripper.write(10);
}

// Accepts integer values for desired servo positions and bearing
void gesture(int sPos, int ePos, int wPos, int bearing) {
  while (true) {
    increment(cwShoulder, sPos, bearing); // Calls increment function and inputs servo, position and bearing data

    increment(ccwShoulder, sPos, bearing * counterclockwise);

    increment(elbow, ePos, bearing * counterclockwise);

    increment(wrist, wPos, bearing);

    if (cwShoulder.read() == sPos) { // If servo position equals desired postion, perform contents
      if (elbow.read() == ePos) {
        if (wrist.read() == wPos) {
          break; } } } // Terminates the function
  }
}

// Integer function which accepts a Servo and two integer variables
void increment(Servo servo, int pos, int bearing) {
  int angle = servo.read(); // Initialises angle variable and assigns current servo position

  if (angle != pos) { // If angle doesn't equal given position, perform contents
    angle += bearing; // Adds bearing to angle
    servo.write(angle); // Returns angle upon calling the function
  }
}

void handle(int gPos, int bearing) { // Function for gripper control
  increment(gripper, gPos, bearing);
  delay(5); // Pauses for 5 ms
}

void revolve(int displacement, int bearing) { // Function for stepper morot control
  for (int s = 0; s < displacement; s++) { // For loop repeats until stepper reaches desired displacement
    stepper.step(bearing); // Steps once in the clockwise or counterclockwise direction (depending on input)
  }
}

void muscle() { // Grabs object
  revolve(position[i], counterclockwise); // Turns to object
  delay(500); // Pauses for 500 ms while statement is executed

  arithmetic(ping()); // Calculates angle for given distance
  delay(500);

  gesture(shoulderAngle, elbowAngle, wristAngle, clockwise); // Extends arm
  delay(500);

  handle(gripperAngle, clockwise); // Grips object
  delay(500);

  gesture(shoulderRest, elbowRest, wristRest, counterclockwise); // Retracts arm
  delay(500);

  revolve(position[i], clockwise); // Turns to the dropping zone (end of range)
  delay(500);

  handle(gripperRest, counterclockwise); // Releases object
  delay(500);
}

void controller() {
  stepper.step(1); // Steps clockwise
```

```
    distance = ping(); // Finds distance

    if (distance >= minDist && distance <= maxDist) { // Reduces noise
      if (!obstruction) { // If an obstruction has not yet been detected
        obstruction = true; // Set 'obstruction' to true
        initial = step; // Assigns the current step to the 'intial' edge of the object face
      }
      /* Continually re-assigns the current step to the 'final' integer variable until the
      obstruction is no longer detected and this way finds the 'final' edge of the object face  */
      final = step;

    } else { // If the above statement comparing distance is not true

      if (obstruction) { // If an obstruction was detected on the last step
        obstruction = false; // Set 'obstruction' to false
        /* Calculate the step which correlates to the middle of the object face and assigns this to the 'i'
        instance of the 'position' array */
        position[i] = stepRange - (final - initial) / 2 + initial;
        // Increments 'i' by 1 so the next instance of the 'position' array is assigned upon the next statment call
        i++;
      }
    }

    if (step == stepRange) { // If the stepper has completed the full range
      for (i; i >= 0; i--) { // Repeat for each object within the range
        muscle(); // Perform the 'muscle' function to grab the object
      }

      revolve(stepRange, 1); // Return to the starting point
      exit(0); // Terminate the program
    }

    step++; // Increments step by 1
  }
```