Programmable Robotic Arm (ProgRArm)

Problem statement

Robotic arms are widely used nowadays for automating a variety of tasks in various industries. These arms can be programmed by a user to repeatedly perform a particular set of instructions. Our goal is to successfully create one such simple programmable arm using Arduino. The arm will also be able to detect obstacles in its rotation path so that any accidents are avoided. We will also add a functionality so that we can store some common instructions and then revoke them by a button press on the keypad instead of reprogramming the arm again and again. The arm can also be operated in a gesture-controlled mode, which would be very useful given there are many situations where humans would need to remotely accomplish certain tasks.

Team

Akash Gayakwad (190260004)
Gaurav Badre (190260012)
Hemant (190260023)

Description

Our goal is to make a robotic arm which can be programmed by the user to perform a specific task repeatedly.

The model: For simplicity and to ensure that the model actually turns out to be working without any physical problems, we would implement 3 degrees of freedom: one at the base for rotation, one hinge, and one for the claw at the tip of the arm so that it can catch and pick up objects. If this doesn't work out as expected, we can, for demonstration purposes, reduce the degree of freedom to 2: one at the base for rotation, and one hinge. The tip now won't have any claws. We could still use this to do a few tasks.

Mode A: Programming the arm to perform an instruction repeatedly: We are programming the movements by a set of potentiometers, one corresponding to each servo motor. The user would give an 'instruction' which would consist of N number of 'movements' (we have chosen N=5). These N movements form an instruction. This instruction or task is then carried out repeatedly by the arm.

Mode B: Storing a few common instructions: We will add a functionality so that we can store a few handy instructions in EEPROM, each labeled by a number so that they can be accessed later by pressing that number on the keypad. This reduces the effort of programming some very common tasks again and again.

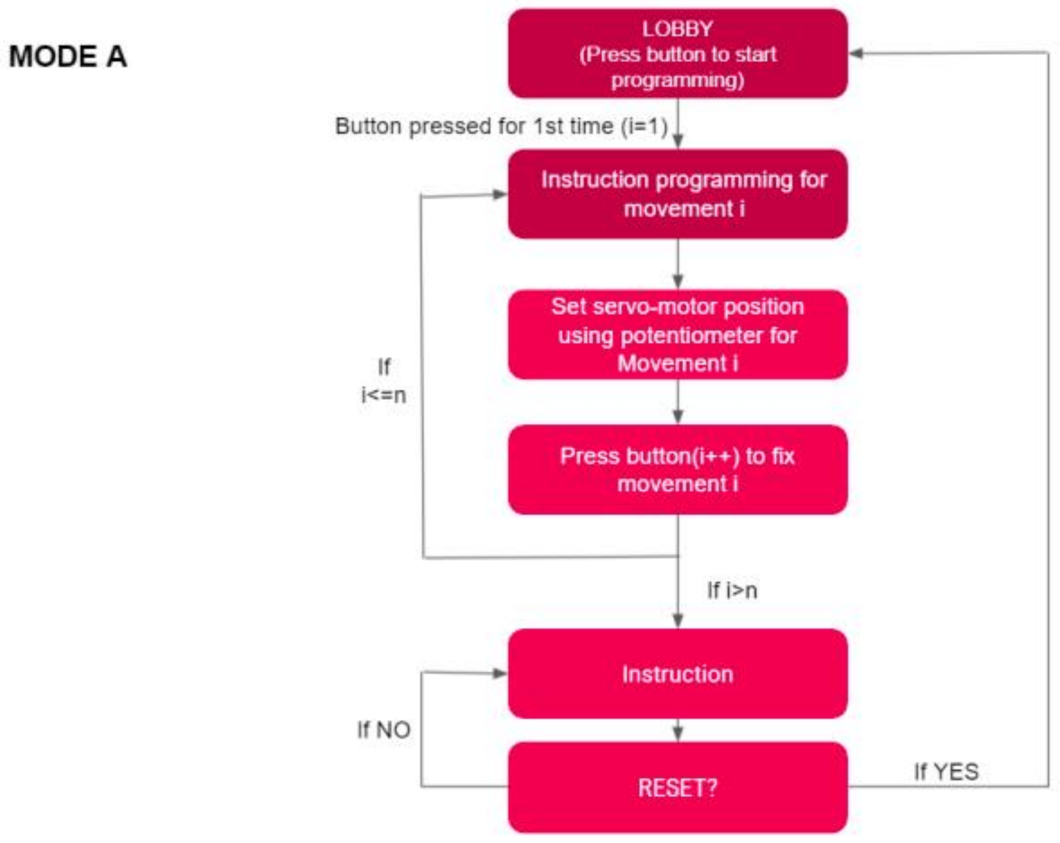
Mode C: Performing the stored instructions: Just invoke the stored instructions from the keypad button and ask the arm to perform that repeatedly.

Stretch goals-

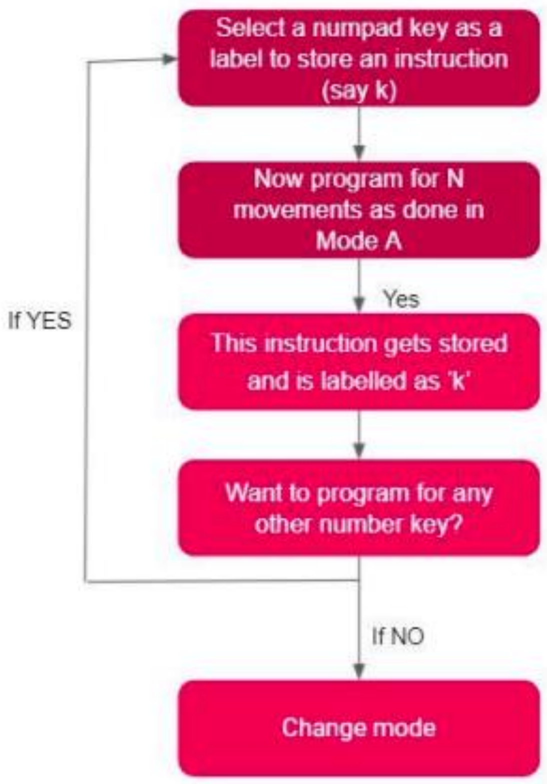
Obstacle detection: We can use 2 ultrasonic sensors (one on each side) on the arm to detect any possible obstacles in its rotation path. As soon as any obstacle is detected the arm stops its operation at a safe distance from the obstacle. To cut costs, we can remove this functionality.

Mode D- Gesture control: In this mode, the arm can be controlled by the gestures of our hand. We implement this simply by using flex resistors. For the 3 degree of freedom model, we would need one flex sensor at the index finger to control claw (because the index finger and the thumb somewhat forms a claw), one at the wrist to control the hinge, and one at some other finger (say middle finger) for controlling the rotation (up to 180 degrees). This mode is however a stretch goal, depending very much on the time constraints.

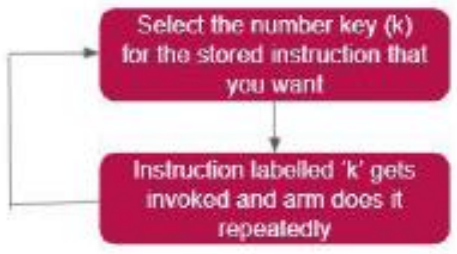
Block diagram


MODE A
LOBBY (Press button to start programming)
Button pressed for 1st time (i=1)
Instruction programming for movement i
Set servo-motor position using potentiometer for Movement i
If i<=n
Press button(i++) to fix movement i
If i>n
Instruction
If NO
RESET?
If YES

MODE B


Select a numpad key as a label to store an instruction (say k)
Now program for N movements as done in Mode A
Yes
This instruction gets stored and is labelled as 'k'
Want to program for any other number key?
If YES
If NO
Change mode

MODE C


Select the number key (k) for the stored instruction that you want
Instruction labelled 'k' gets invoked and arm does it repeatedly

Design details

There are three modes in which this project works.

When power is turned ON, the arm is in LOBBY state where it is waiting for user input to determine the mode of functioning. There are three modes (apart from LOBBY) in which this model works. There are two control buttons, SAVE and RESET. RESET sets the model in LOBBY state from wherever it is at the moment of pressing it. If the model is executing instructions repeatedly, pressing RESET takes it to LOBBY after finishing the ongoing instruction, so as to avoid abruptness. SAVE button is very handy while saving instructions. Each SAVE press saves a movement (or position). We can save upto 5 movements in an instruction.The detailed working of different modes is as follows-

MODE A:

Here, we control the servos with the potentiometer in real time. We save 5 positions/movements by pressing SAVE at each suitable/wanted positions of the three servos. As soon as SAVE is pressed for 5th time, i.e., after saving the final movement in the instruction, the model starts continuously executing the instruction repeatedly. It can be stopped by pressing RESET.

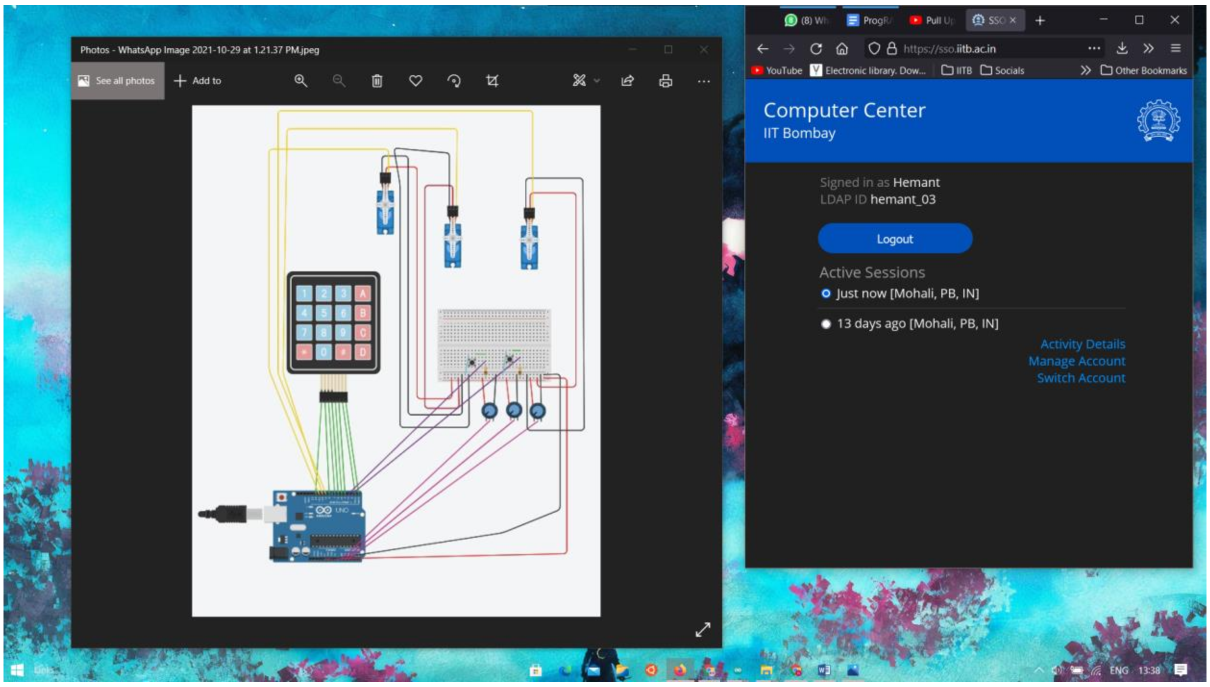
MODE B:

Here, we first need to input a label from the keypad (in our implementation, it can be either 1 or 2, other key presses would be ignored). After giving a label, we do the programming of 5 movements or an instruction as done in mode A. After 5th SAVE press, the arm won't start executing the instruction. Instead, this instruction would now be saved in the EEPROM of the AtMega chip. We can retrieve this instruction later, even after pressing the Arduino reset, or powering OFF and then ON. This comes handy for some very common instructions, This save the effort of programming again and again after powering OFF and On the Arduino as needs to be done in Mode A.

MODE C:

Here also, we first need to input a label from the keypad (again, only 1 or 2, others ignored). Then the Instruction that we saved under that label in mode B previously gets retrieved and the arm starts executing that instruction repeatedly. The execution stops when RESET is pressed.

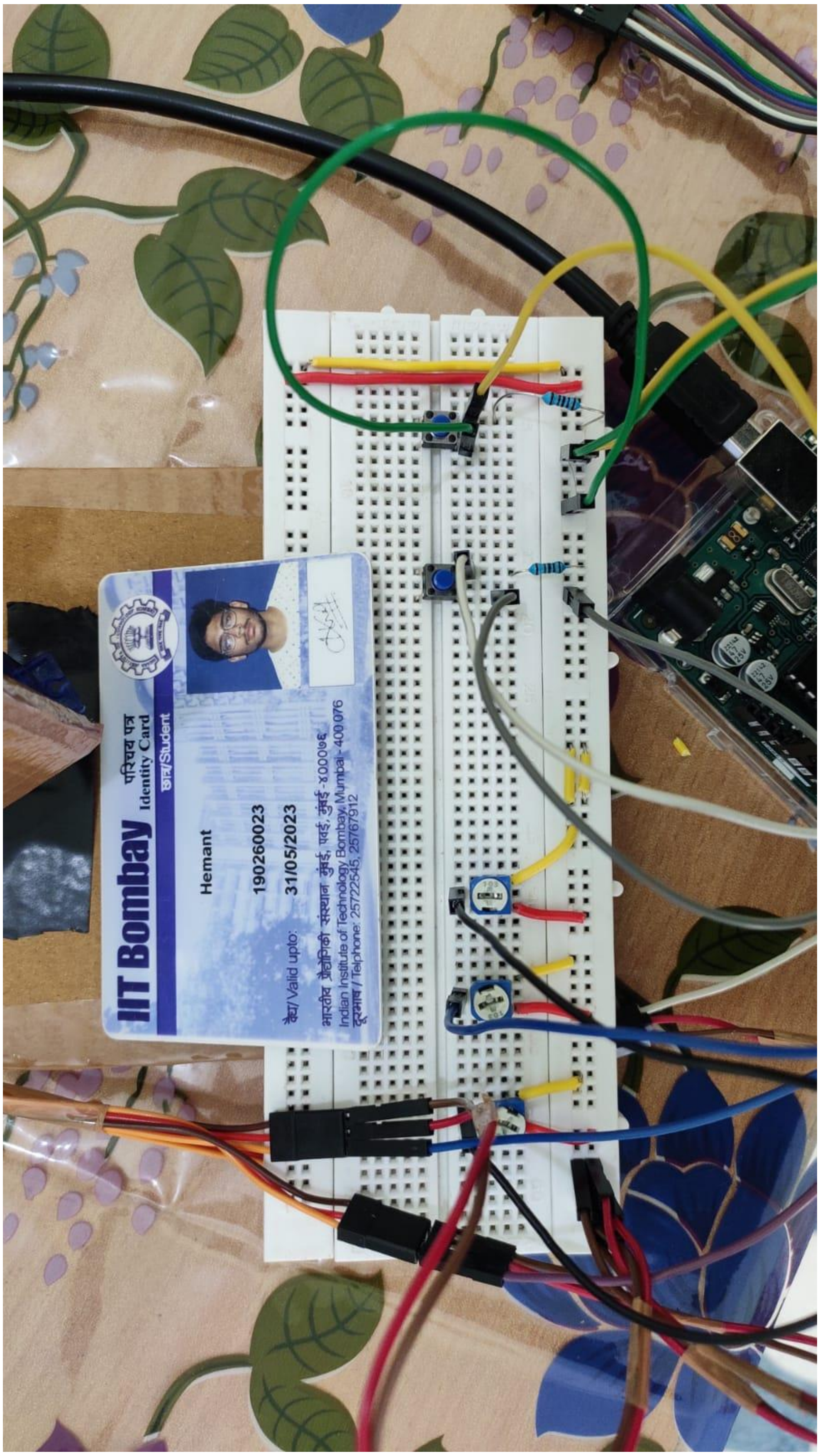
Circuit diagram and image

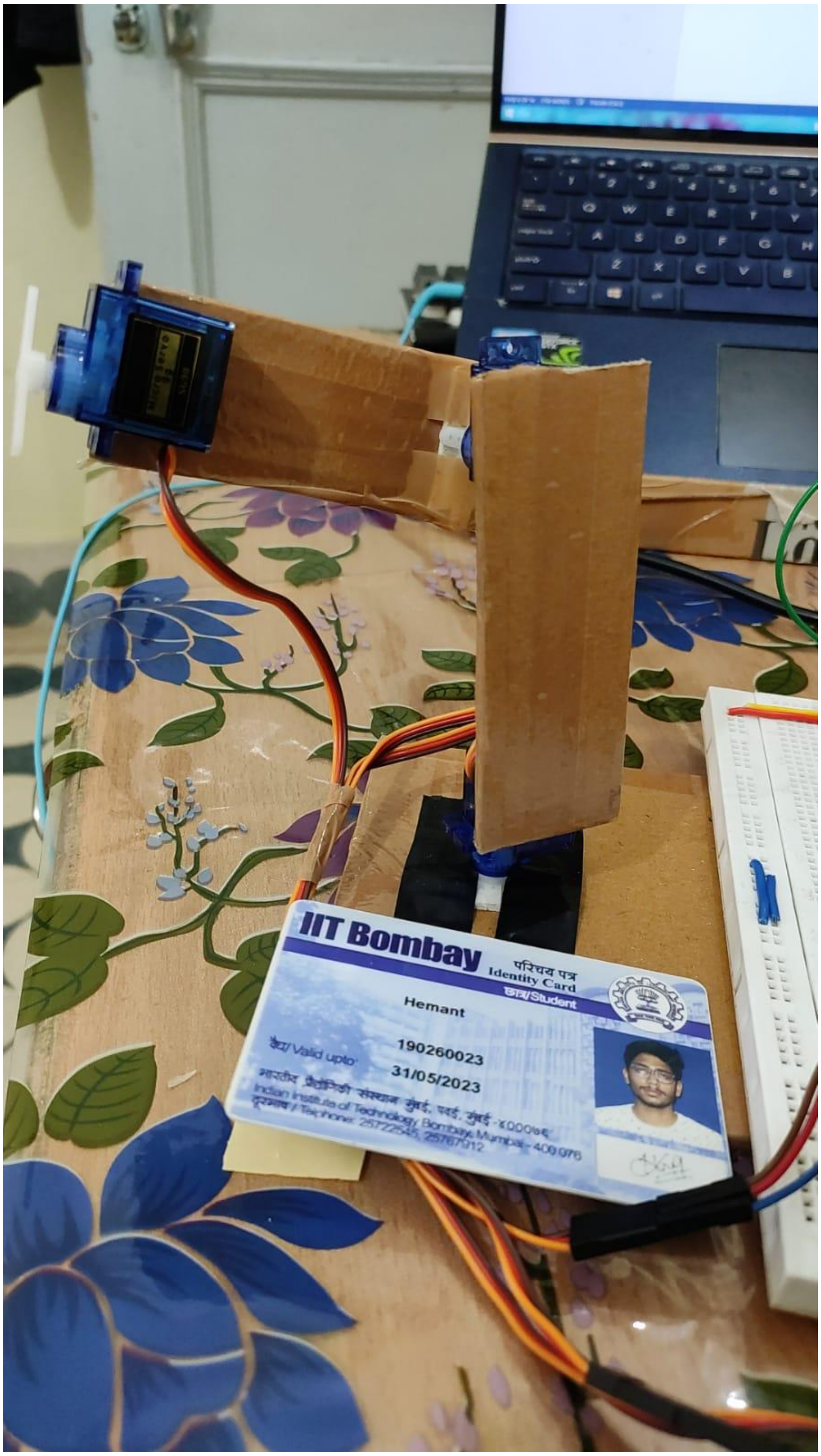


Signed in as Hemant
LDAP ID hemant_03

Active Sessions
Just now [Mohali, PB, IN]
13 days ago [Mohali, PB, IN]

Photos of model and breadboard circuit-


IIT Bombay
परिचय पत्र
Identity Card
छात्र/Student
Hemant
190260023
वैध/Valid upto:
31/05/2023
Indian Institute of Technology Bombay, Mumbai - 400 076
दूरभाष / Telephone: 25722545, 25767912


IIT Bombay
परिचय पत्र
Identity Card
छात्र/Student
Hemant
वैध/Valid upto
190260023
31/05/2023

Link Of Demo :

https://drive.google.com/drive/folders/14F8LzJUKNigh3SW4jxofbVYyYRVEFF0z?usp=sharing

The gears of the hinge motors of my design is slightly loose due to excessive demo/checking so it isnt picking up that much weight.

Include the final circuit diagrams and photographs of your project (with SSO / Icard as usual).Also include a link to your demo video. Note that while the rest of the report can be the same for all team members, **these images must be of your own implementation**! (More details in notes at the end of this document).

Bill of materials

● Arduino Uno

● Servo motors (3+2)

● Keypad(1)

● Potentiometers (3)

Status and conclusion

Did you achieve all the goals you had defined? Did you achieve any bonus / stretch goals that you had set out? Any particular challenges that you had to overcome? What are the limitations of your project? What are the lessons learnt?

Yes all the basic goals are achieved.This constitutes MODE_A MODE_B MODE_C. As stated before this includes the basic constituents of Arm. It takes input from Keypad and potentiometer and changes the position of servo motors accordingly.
Due to time constraints we were unable to get **gesture contro**l. For **Obstacle detection** we have used 1 ultrasonic sensor on the arm to detect any possible obstacles in its rotation path. As soon as any obstacle is detected the arm stops its operation at a safe distance from the obstacle. To cut costs, we can remove this functionality.

We do have the code and implementation for the both gesture control and obstatcle dettion but the apart from time constraint there were some hardware issues also.
For **Gesture Control**:

1. We successfully made the code and homemade flex sensor but the data we were getting from it was random and non-consistent, it was working but wasn't upto the mark.
2. The fabrication of flex took a very high amount of time and the results was not great.

For **Obstacle Detection:**

1. We were planning to use 2 ultrasonic sensor which cost us 4 additional digital pins and we didn't have that, so we only used only 1 sensor.
2. We tried to remove the RESET pin in our circuit to replace it with the ultrasonic pin and implementing it.

3. We wrote the code and implemented that individually and that worked fine but when we added that function it didn't implement accordingly and due to time constraints we didn't have enough time to debug the whole code.

Servo Motors are very difficult to work with. Their functionality decreases with the use of time. Working with micro servo motors is difficult and requires a great amount of precision to work with them. After some time it gets completely nuisance.and starts malfunctioning. Over surfing over internet I got to know that the gear box of this motor gets loosen up. Thus it took most of our time. And we could not achieve more from this project.Secondly we dropped the idea of a flex sensor due to time constraints.

Actual programming is easier but implementing it on hardware requires a lot of attention. Here we were actually able to apply what we had learnt in our LABS. Using various components available in the market we were actually able to learn various attributes of Arduino.
New libraries such as **Keypad.h** made our project more user friendly.

My Contribution in this project:

1. Responsible for the implementation and fabrication of Homemade Flex sensor and Ultrasonic sensor.

2. Responsible for the design, coding and logic of the both sensors used in arduino.

3. Debugged and tinkered the code and made the hardware amendemants to ensure the proper working of model.

Source codes

You should list out your source codes here. Optionally, in addition to this, you can include a github link if you want.

```
#include<Servo.h>
#include<Keypad.h>
#include<EEPROM.h>

volatile char mode = 'L';  //L = lobby state
//byte  noMovements  =  5;     //noOfMovementsInAnInstruction
//remove this var at the end
volatile byte saveButton = 0;
volatile bool reset = true;
Servo servo1;  //floor
Servo servo2;  //hinge
Servo servo3;  //claw
int labelStore;
int labelExec;
```

```
volatile unsigned long int previousTime = 0;  //for save button
volatile unsigned long int currentTime;

//FOR INSTRUCTIONS

byte instruction[3][5];  //stores instruction

//FOR KEYPAD

const byte noRows = 4;
const byte noCols = 4;

char keyChar[noRows][noCols] = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};

byte rowPins[noRows] = {12, 8, 7, 6}; //{0, 1, 4, 5};
byte colPins[noCols] = {5, 4, 1, 0}; //{6, 7, 8, 12};

Keypad keypadIn(makeKeymap(keyChar), rowPins, colPins, noRows,
noCols);

void setup() {
  // put your setup code here, to run once:
  //Serial.begin(9600);
  attachInterrupt(digitalPinToInterrupt(2),
saveButtonIncrementISR, FALLING);
  attachInterrupt(digitalPinToInterrupt(3),    resetPressedISR,
FALLING);
  servo1.attach(9);  //floor
  servo2.attach(10);  //hinge
  servo3.attach(11);  //claw
}

void loop() {
  // put your main code here, to run repeatedly:
  switch(mode){
    case 'L':   //wait for keypad mode entry
      if(reset){
        initialPosition();
        reset = false;
      }
      while(true){
        mode = keypadIn.getKey();  //get keypad input for mode
        if(mode != NO_KEY){break;}
      }
      if(mode=='A' || mode=='B' || mode=='C'){   //mode= 'D'
when mode D gets implemented
      }
      else{
         mode = 'L';
      }
      break;
    case 'A':   //program and execute
      //Serial.println('A');
      program();   //program the arm
      execute();    //execte the programmed instruction
      break;
    case 'B':   //store instructions
      char keyB;
      while(!reset){
        //labelStore = (byte)atoi(keypadIn.getKey());
        keyB = keypadIn.getKey();
        //labelStore =  key - '0';
        if(keyB != NO_KEY){break;}
      }
      if(keyB=='1' || keyB=='2'){
        labelStore = keyB - '0';
        program();
        storeToEEPROM();
      }
      break;
    case 'C':   //retrieve stored instructions and execute
      char keyC;
      while(!reset){
          //labelExec = (byte)atoi(keypadIn.getKey());
          keyC = keypadIn.getKey();
          if(keyC != NO_KEY){break;}
        }
      if(keyC=='1' || keyC=='2'){
        labelExec = keyC - '0';
        retrieveFromEEPROM();
        execute();
      }
      break;

    //case 'D':   //gesture control
     // break;
  }

}

void initialPosition(){
    servo1.write(0);   //initial positions of all three servos,
depends on your model and choice
    servo2.write(70);
    servo3.write(130);
}

void program(){
    saveButton = 0;
    while((saveButton < 5) && !reset){
      byte servoPos1 = map(analogRead(A0), 0, 1023, 0, 180);
//value from pot 1 mapped to floor dof
      byte servoPos2 = map(analogRead(A1), 0, 1023, 80, 140);
//value from pot 2 mapped to hinge dof
      byte servoPos3 = map(analogRead(A2), 0, 1023, 130, 180);
//value from pot 3 mapped to claw dof
      servoPos1=constrain(servoPos1, 0, 180);
      servoPos2=constrain(servoPos2, 80, 140);
      servoPos3=constrain(servoPos3, 130, 180);
      instruction[0][saveButton] = servoPos1;
      instruction[1][saveButton] = servoPos2;
      instruction[2][saveButton] = servoPos3;
      servo1.write(servoPos1);
      servo2.write(servoPos2);
      servo3.write(servoPos3);
    }
}

void execute(){
  while(!reset){
   for(int j = 0; j < 5; j++){
      byte servoPos1=instruction[0][j];
      byte servoPos2=instruction[1][j];
      byte servoPos3=instruction[2][j];
      servoPos1=constrain(servoPos1, 0, 180);
      servoPos2=constrain(servoPos2, 80, 140);
      servoPos3=constrain(servoPos3, 130, 180);
      servo1.write(servoPos1);
      servo2.write(servoPos2);
      servo3.write(servoPos3);
      delay(1000);   //wait for some time after each movement
    }
   initialPosition();
  }
}

void storeToEEPROM(){
  if(!reset){
    int startIndex = 3*5*(labelStore-1);
    for(int i=0; i<3; i++){
      for(int j=0; j<5; j++){
        EEPROM.write(startIndex, instruction[i][j]);
        startIndex++;
      }
    }
  }
}

void retrieveFromEEPROM(){
  if(!reset){
    int startIndex = 3*5*(labelExec-1);
    for(int i=0; i<3; i++){
      for(int j=0; j<5; j++){
        instruction[i][j] = EEPROM.read(startIndex);
        startIndex++;
      }
    }
  }
}

void resetPressedISR(){
  reset = true;
  mode = 'L';
}

void saveButtonIncrementISR(){
  currentTime = millis();
  if((currentTime-previousTime)>50){
    saveButton++;
    previousTime = currentTime;
  }
}
```

Notes:

1. The first parts of the report should be same as your project proposal document
2. Length of the report:
   a. The written part of the report should be 3-4 pages
   b. Circuit diagram, photos, etc can take up more pages as needed
   c. Source code can take up more pages as needed
3. The circuit diagram of the project can be the same diagram for all team members, but take a screenshot with your own SSO screen (for logistical reasons).
4. Photos and video of the setup must be different for each team member! Other parts can be the same.
5. Of course, you can delete this “notes” section from your report !