UPM-115 - ATHENS-Program

João Lopes & Steven Huygens

Github project

POLITÉCNICA

November 27, 2023

November 27, 2023

Introduction

During this course we learned working with Arduino[1] and with the programming language Processing[2]. At the end we had to make a project with the things we learned during this course. Our idea was inspired by a bomb disposal robot[3]. We wanted to make a remote controlled car with an arm you would be able to control on top of the car.

First sketches

To begin the first sketches were made. The first sketches can be seen in Figure 5.

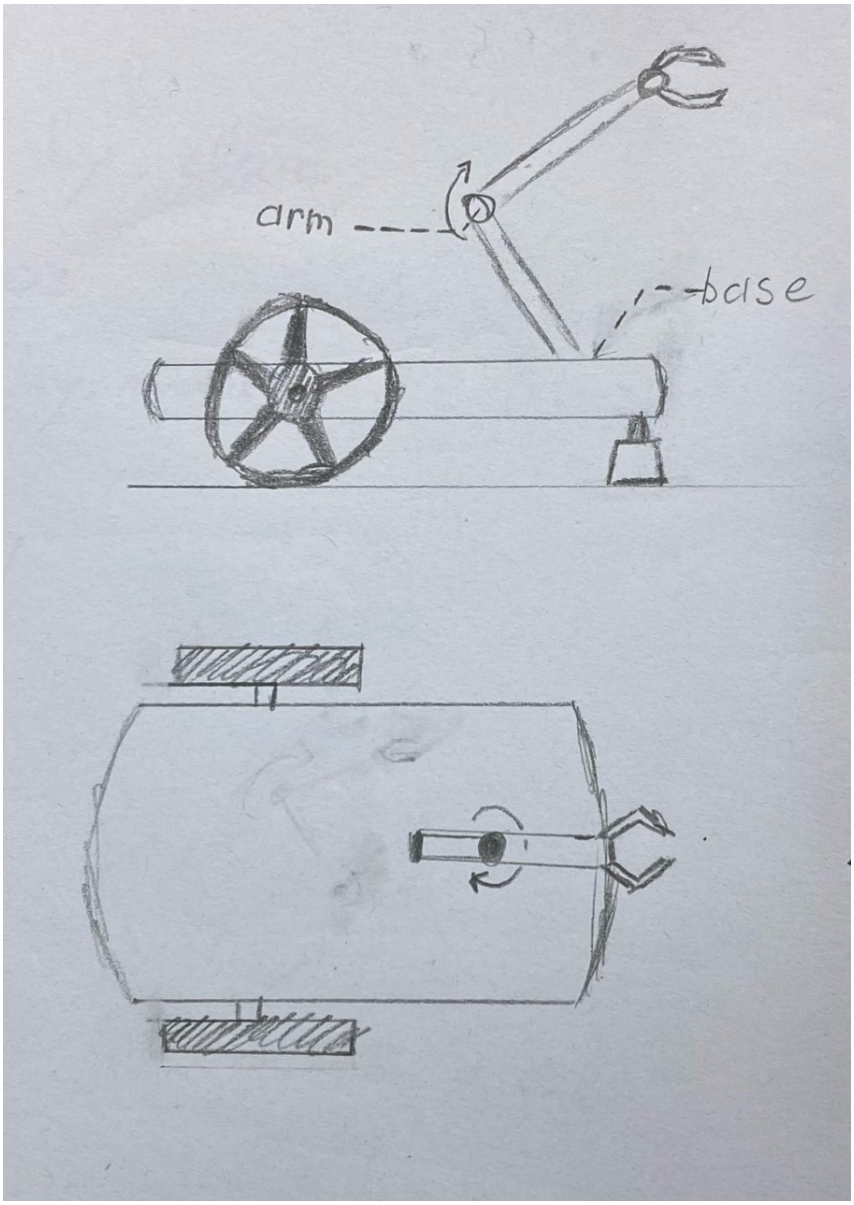

Figure 1: The first sketch of the Arduino project.

Gathering the parts

After the first sketches were completed, we could then start gathering the parts which were needed to complete this project. This is the list of the parts which were used in this project:

- 2x Metal chassis
- 2x Plastic/Rubber Wheel
- 2x DC Engine
- 1x L298N Motor Driver
- 1x NRF24L01 2.4GHz module
- 2x Servo Motor
- 1x 9V Battery
- 1x Mini Breadboard
- 1x L7805 5.0V Voltage Regulator
- 1x 5V Laser Module
- 1x Arduino Nano
- 1x Jumper Cables
- 1x Different Screws
- 1x 3D Print - Robot Arm

Wiring Diagrams

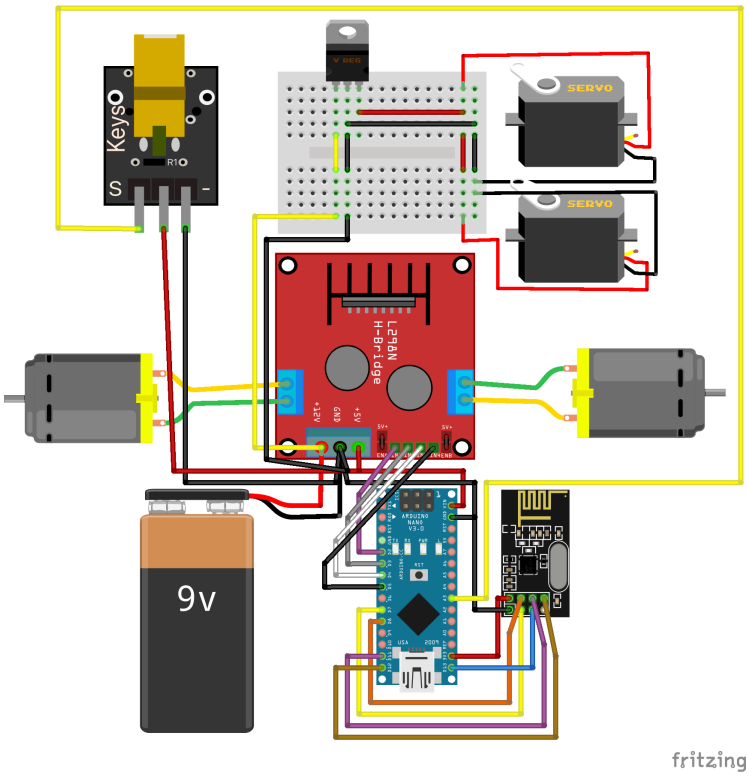

Figure 2: The wiring diagram for the robotic car.

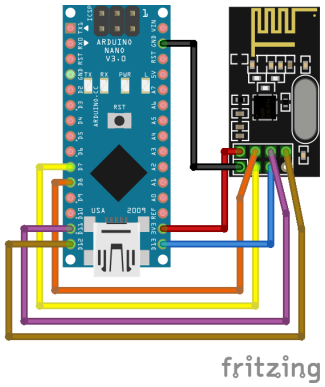

Figure 3: The wiring diagram for the transmitter/controller.

Results

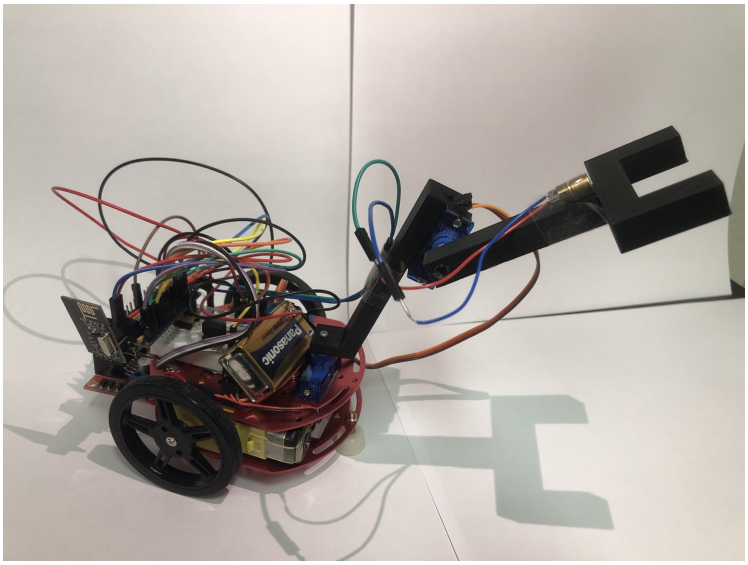

Figure 4: A picture of the final Robotic Car.

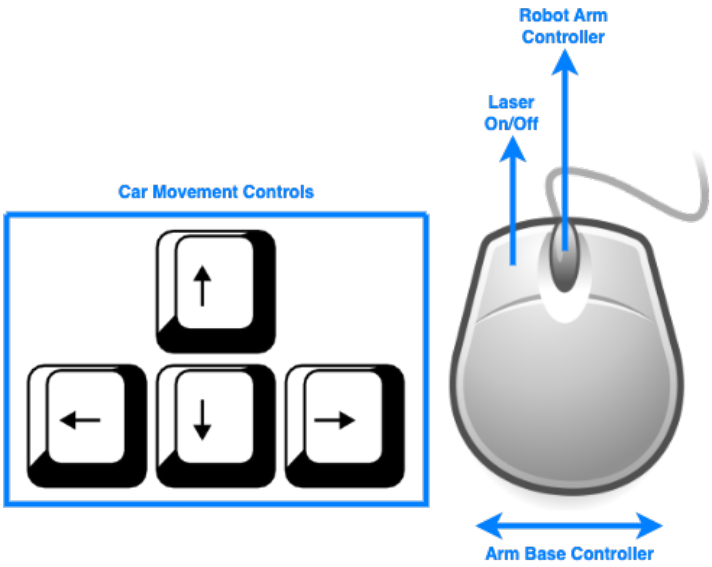

Figure 5: The controls of the car.

Source Code

This code can also be found on Github[4].

Arduino Source Code - Car robot

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Servo.h>

#define ENGINE_TUNING 2

struct dataPackage {
  byte forward = 0;
  byte backward = 0;
  byte left = 0;
  byte right = 0;
  byte baseServo = 90;
  byte armServo = 90;
  byte armLaser = 0;
};

Servo baseServo, armServo;
int motor1pin1 = 2;
int motor1pin2 = 3;
int motor2pin1 = 4;
int motor2pin2 = 5;

dataPackage datarx;
RF24 radio(7, 8); //CE, CSN
const byte address[6] = "00099";

void safetyReset(){
  datarx.forward = 0;
  datarx.backward = 0;
  datarx.left = 0;
  datarx.right = 0;
}

void stopEngines(){
  digitalWrite(motor1pin1, LOW);
  digitalWrite(motor1pin2, LOW);
  digitalWrite(motor2pin1, LOW);
  digitalWrite(motor2pin2, LOW);
}

void moveForward(){
  digitalWrite(motor1pin1, LOW);
  digitalWrite(motor1pin2, HIGH);
  digitalWrite(motor2pin1, LOW);
  digitalWrite(motor2pin2, HIGH);

  delay(ENGINE_TUNING);
```

```
49    stopEngines();
50  }
51
52  void moveBackward(){
53    digitalWrite(motor1pin1, HIGH);
54    digitalWrite(motor1pin2, LOW);
55    digitalWrite(motor2pin1, HIGH);
56    digitalWrite(motor2pin2, LOW);
57
58    delay(ENGINE_TUNING);
59    stopEngines();
60  }
61
62  void rotateLeft(){
63    digitalWrite(motor1pin1, HIGH);
64    digitalWrite(motor1pin2, LOW);
65    digitalWrite(motor2pin1, LOW);
66    digitalWrite(motor2pin2, HIGH);
67
68    delay(ENGINE_TUNING);
69    stopEngines();
70  }
71
72  void rotateRight(){
73    digitalWrite(motor1pin1, LOW);
74    digitalWrite(motor1pin2, HIGH);
75    digitalWrite(motor2pin1, HIGH);
76    digitalWrite(motor2pin2, LOW);
77
78    delay(ENGINE_TUNING);
79    stopEngines();
80  }
81
82  void turnLeft(){
83    digitalWrite(motor1pin1, HIGH);
84    digitalWrite(motor1pin2, LOW);
85    digitalWrite(motor2pin1, LOW);
86    digitalWrite(motor2pin2, HIGH);
87
88    delay(ENGINE_TUNING);
89    stopEngines();
90  }
91
92  void turnRight(){
93    digitalWrite(motor1pin1, LOW);
94    digitalWrite(motor1pin2, HIGH);
95    digitalWrite(motor2pin1, HIGH);
96    digitalWrite(motor2pin2, LOW);
97
98    delay(ENGINE_TUNING);
99    stopEngines();
100 }
101
102 void setup() {
```

```
  Serial.begin(115200);

  //Engine
  pinMode(motor1pin1, OUTPUT);
  pinMode(motor1pin2, OUTPUT);
  pinMode(motor2pin1, OUTPUT);
  pinMode(motor2pin2, OUTPUT);
  stopEngines();

  //Servos
  baseServo.attach(6);
  armServo.attach(9);

  //Laser
  pinMode(A3, OUTPUT);

  //NRF
  radio.begin();
  radio.openReadingPipe(0, address); // 00002
  radio.setPALevel(RF24_PA_MAX);
  radio.startListening();
}

unsigned long resetTimer = 0, targetResetTime = 0;
void loop() {
  //Motors
  if (datarx.forward){
    if(datarx.left){
      turnLeft();
    }
    else if (datarx.right){
      turnRight();
    }
    else{
      moveForward();
    }
  }
  else if(datarx.backward){
    if(datarx.left){
      turnLeft();
    }
    else if (datarx.right){
      turnRight();
    }
    else{
      moveBackward();
    }
  }
  else if (datarx.left){
    rotateLeft();
  }
  else if (datarx.right){
    rotateRight();
  }
```

```
  else{
    stopEngines();
  }

  digitalWrite(A3, datarx.armLaser);

  //Servos
  baseServo.write(datarx.baseServo);
  armServo.write(datarx.armServo);

  if (radio.available()) {
    resetTimer = millis();
    targetResetTime = resetTimer + 25;
    radio.read(&datarx, sizeof(dataPackage));
  }
  else{
    if (resetTimer > targetResetTime){
      safetyReset();
    }
  }

  delay(5);
}
```

Arduino Source Code - PC transmitter/controller

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

String receivedDataFromPython;
const int parsedDataSize = 7;
int parsedPythonData[parsedDataSize];
const char charToSplitOn = ',';

struct dataPackage {
  byte forward = 0;
  byte backward = 0;
  byte left = 0;
  byte right = 0;
  byte baseServo = 90;
  byte armServo = 90;
  byte armLaser = 0;
};

dataPackage dataTx;
RF24 radio(7, 8); //CE, CSN
const byte address[6] = "00099";


/**
 * This method uses the String receivedDataFromPython to update the int array
    parsedPythonData.
 * This gets done by splitting the receivedDataFromPython by the char
    charToSplitOn.
```

```
 * This method expects the String receivedDataFromPython to have exactly a parsedDataSize amount of (charToSplitOn - 1) characters.
 */
void parsePythonData() {
  String currentBuildString = String("");
  int counter = 0;
  for (int i = 0; i < receivedDataFromPython.length(); i++) {
    if (receivedDataFromPython.charAt(i) == charToSplitOn) {
      parsedPythonData[counter] = currentBuildString.toInt();
      counter += 1;
      currentBuildString = "";
    } else {
      currentBuildString += receivedDataFromPython.charAt(i);
    }
  }
  parsedPythonData[counter] = currentBuildString.toInt();
}


/**
 * This method uses the int array parsedPythonData to update the dataPackage dataTx.
 */
void updateDataTx() {
  dataTx.forward = parsedPythonData[0];
  dataTx.backward = parsedPythonData[1];
  dataTx.left = parsedPythonData[2];
  dataTx.right = parsedPythonData[3];
  dataTx.baseServo = parsedPythonData[4];
  dataTx.armServo = parsedPythonData[5];
  dataTx.armLaser = parsedPythonData[6];
}

void sendRadioData() {
  radio.write(&dataTx, sizeof(dataPackage));
}


/**
 * This method first parses the String receivedDataFromPython to then update the dataPackage dataTx.
 */
void handleReceivedPythonData() {
  parsePythonData();
  updateDataTx();
  sendRadioData();
}

void sendCurrentDataPacketToPython() {
  Serial.println(dataTx.forward);
  Serial.println(dataTx.backward);
  Serial.println(dataTx.left);
  Serial.println(dataTx.right);
  Serial.println(dataTx.baseServo);
```

```
  Serial.println(dataTx.armServo);
  Serial.println(dataTx.armLaser);
}

void setup() {
  Serial.begin(115200);
  Serial.setTimeout(1);
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MAX);
  radio.stopListening();
}


void loop() {
  while (Serial.available() < (parsedDataSize + parsedDataSize));
  delay(5); // to get make sure the full data is received
  receivedDataFromPython = Serial.readString();
  handleReceivedPythonData();
  sendCurrentDataPacketToPython();  // received data getting send back to python
}
```

Python Source Code - main

```
import serial
import serial.tools.list_ports
import time
import CarController
import ArmController


def print_ports():
    ports = serial.tools.list_ports.comports()
    for port, desc, hwid in sorted(ports):
        print(f'{port}: {desc} [{hwid}]')


# this is to check which ports are available at the moment
print_ports()
arduino = serial.Serial(port='COM8', baudrate=115200, timeout=0.1)

SEND_RATE = 0.07  # the time in seconds between packets getting send

# set up the controllers
car_controller = CarController.CarController()
car_controller.start_listening()
arm_controller = ArmController.ArmController()
arm_controller.start_listening()


def get_all_states(car_controller: CarController.CarController, arm_controller:
    ArmController.ArmController) -> list[str]:
    """
    This method gets the states of the given controllers and puts them in a list
    of strings.
```

```
    """
    output = []
    car_controller_states = car_controller.get_states()
    arm_controller_states = arm_controller.get_states()
    for key in car_controller.available_keys:
        output.append(str(int(car_controller_states[key])))
    for key in arm_controller.available_keys:
        output.append(str(int(arm_controller_states[key])))
    return output


def encode_states(all_states: list[str]) -> bytes:
    """
    This method gets a list of strings and joins them separated by a ',' and converts this string to bytes.
    """
    return bytes(','.join(all_states), 'utf-8')


def write_to_arduino(data_to_send: bytes):
    """
    This method writes to the given data_to_send to the Arduino.
    The Arduino will normally send the received data back to check if the correct data has been received.
    """
    arduino.write(data_to_send)
    time.sleep(0.05)
    data_forward = str(arduino.readline()[:-2], 'utf-8')
    data_backward = str(arduino.readline()[:-2], 'utf-8')
    data_left = str(arduino.readline()[:-2], 'utf-8')
    data_right = str(arduino.readline()[:-2], 'utf-8')
    data_base = str(arduino.readline()[:-2], 'utf-8')
    data_arm = str(arduino.readline()[:-2], 'utf-8')
    data_laser = str(arduino.readline()[:-2], 'utf-8')
    print(f'Arduino data:\tforward: {data_forward}\tbackward: {data_backward}\tleft: {data_left}\tright: {data_right}\tbase: {data_base}\tarm: {data_arm}\tlaser: {data_laser}')


if __name__ == "__main__":
    while True:
        time.sleep(SEND_RATE)
        state_array = get_all_states(car_controller, arm_controller)
        encoded_states = encode_states(state_array)
        write_to_arduino(encoded_states)
```

Python Source Code - CarController

```
from pynput import keyboard
from pynput.keyboard import Key, KeyCode


class CarController:
    """
```

```
    This class is used to be able to control the car by using the arrow keys on the keyboard.
    """
    def __init__(self):
        self.available_keys = ['up', 'down', 'left', 'right']
        self.key_states = dict()
        for key in self.available_keys:  # initialize the values of the keys to False
            self.key_states[key] = False
        self.key_listener = keyboard.Listener(on_press=self.on_press,
                                              on_release=self.on_release)

    def start_listening(self):
        """
        This method starts the CarController object to listen to the arrow key inputs.
        """
        self.key_listener.start()

    def get_states(self):
        return self.key_states

    def on_press(self, key: Key | KeyCode):
        """
        This method is called whenever a key is pressed on the keyboard.
        If the key is an arrow key the corresponding value in the self.key_states will be changed to True.
        """
        try:
            key_name = key.char
        except AttributeError:
            key_name = key.name
        if key_name in self.available_keys:
            self.key_states[key_name] = True

    def on_release(self, key: Key | KeyCode):
        """
        This method is called whenever a key is pressed on the keyboard.
        If the key is an arrow key the corresponding value in the self.key_states will be changed to False.
        """
        try:
            key_name = key.char
        except AttributeError:
            key_name = key.name
        if key_name in self.available_keys:
            self.key_states[key_name] = False
```

Python Source Code - ArmController

```
from pynput import mouse
import pyautogui

# get the screen size to check for the right limit of the screen
print(f'screen size: {pyautogui.size()}')
```

```
LEFT_BORDER = 0 + 20
RIGHT_BORDER = pyautogui.size().width - 20


class ArmController:
    """
    This class is used to be able to control the arm of the car by using the horizontal movement of the mouse and
    the scroll wheel of the mouse and to toggle the laser on the car on or off by using the left mouse button.
    """
    # increments of the angles of the car
    ARM_ANGLE_INCREMENT = 1
    BASE_ANGLE_INCREMENT = 1.5
    MAX_ANGLE = 175
    MIN_ANGLE = 5

    def __init__(self):
        self.available_keys = ['base', 'arm', 'laser_on']
        self.mouse_listener = mouse.Listener(
            on_move=self.on_move,
            on_scroll=self.on_scroll,
            on_click=self.on_click)
        self.number_buffer_size = 2
        self.composite_buffer_size = 3
        self.number_buffer = []
        self.composite_buffer = []
        self.angle_states = {'base': 90, 'arm': 90, 'laser_on': False}

    def start_listening(self):
        """
        This method starts the ArmController object to listen to mouse changes.
        """
        self.mouse_listener.start()

    def get_states(self):
        return self.angle_states

    def on_move(self, x, y):
        """
        This method is called whenever the mouse is moved.
        If the mouse is moved horizontally the values self.angle_states['base'] will be changed accordingly.
        The number_buffer and composite_buffer are used to avoid jittery movement of the mouse.
        """
        self.number_buffer.append(x)
        if len(self.number_buffer) == self.number_buffer_size:
            average_of_numbers = sum(self.number_buffer) / self.number_buffer_size
            self.composite_buffer.append(average_of_numbers)
            self.number_buffer = []
            self.handle_added_composite()

    def handle_added_composite(self):
```

```
56          if len(self.composite_buffer) < 3:  # composite_buffer is not full yet
57              return
58          c1, c2, c3 = self.composite_buffer
59          if c1 >= RIGHT_BORDER and c2 >= RIGHT_BORDER and c3 >= RIGHT_BORDER:  #
    means the mouse moved right
60              self.move_base_angle(False)
61          elif c1 <= LEFT_BORDER and c2 <= RIGHT_BORDER and c3 <= LEFT_BORDER:  #
    means the mouse moved left
62              self.move_base_angle(True)
63          elif c1 <= c2 <= c3:  # means the mouse moved right
64              self.move_base_angle(False)
65          elif c1 >= c2 >= c3:  # means the mouse moved left
66              self.move_base_angle(True)
67          self.composite_buffer = []  # empty the composite_buffer
68
69      def move_base_angle(self, needs_to_increment: bool):
70          """
71          This method changes the value of self.angle_states['base'] by + or -
    BASE_ANGLE_INCREMENT.
72          """
73          if needs_to_increment:
74              new_angle = self.angle_states['base'] + ArmController.
    BASE_ANGLE_INCREMENT
75              self.angle_states['base'] = new_angle if new_angle < ArmController.
    MAX_ANGLE else ArmController.MAX_ANGLE
76          else:
77              new_angle = self.angle_states['base'] - ArmController.
    BASE_ANGLE_INCREMENT
78              self.angle_states['base'] = new_angle if new_angle > ArmController.
    MIN_ANGLE else ArmController.MIN_ANGLE
79
80      def on_scroll(self, x, y, dx, dy):
81          """
82          This method is called whenever the scroll wheel on the mouse is used.
83          The values self.angle_states['arm'] will be changed accordingly.
84          """
85          if dy < 0:  # means the mouse scrolled down
86              self.move_arm_angle(False)
87          else:
88              self.move_arm_angle(True)
89
90      def move_arm_angle(self, needs_to_increment: bool):
91          """
92          This method changes the value of self.angle_states['arm'] by + or -
    BASE_ANGLE_INCREMENT.
93          """
94          if needs_to_increment:
95              new_angle = self.angle_states['arm'] + ArmController.
    ARM_ANGLE_INCREMENT
96              self.angle_states['arm'] = new_angle if new_angle < ArmController.
    MAX_ANGLE else ArmController.MAX_ANGLE
97          else:
98              new_angle = self.angle_states['arm'] - ArmController.
    ARM_ANGLE_INCREMENT
```

```
99              self.angle_states['arm'] = new_angle if new_angle > ArmController.
    MIN_ANGLE else ArmController.MIN_ANGLE
100
101     def on_click(self, x, y, button, pressed):
102         """
103         This method is called whenever the left mouse button is clicked.
104         The values self.angle_states['laser_on'] will be changed accordingly.
105         """
106         try:
107             button_name = button.char
108         except AttributeError:
109             button_name = button.name
110         if button_name == 'left' and pressed:
111             self.angle_states['laser_on'] = not self.angle_states['laser_on']
```

References

[1] "Arduino." https://www.arduino.cc/. [Online; Accessed: 2022-11-18].

[2] "Processing." https://processing.org/. [Online; Accessed: 2022-11-18].

[3] "bomb-disposal-robot." https://en.wikipedia.org/wiki/ANDROS. [Online; Accessed: 2022-11-18].

[4] "Github." https://github.com/stevenhgs/Athens-Arduino-Project. [Online; Accessed: 2022-11-18].