NREIP RC Tank Jammer

Game of Drones

Grant Goulart

6/17/2024-8/23/2024

Contents

Project Summary:

The laser jammer tank is a Lynxmotion A4WD3 Wheeled Rover mounted with a pan and tilt system that houses a laser diode. The laser diode housing is modeled after a gatling gun that is utilized to modulate the laser beam into different frequencies. The laser system and the rover are operated remotely by two different personnel. Both operators have a live video feed of the drone cage through mounted baby monitors. The entire laser system, aside from the baby monitor, is controlled by Arduino microcontrollers that communicate via Bluetooth. The laser is modulated to 45 Hz, 95 Hz, and 210 Hz. These frequencies are selected on the controller with three switches sending Bluetooth signals to three relays. Due to the design, it wouldn't be good to have multiple relays thrown at once. In anticipation of people playing with the switches and turning multiple on at a time, there is a safety feature that disables all the relays if multiple switches are thrown. The accuracy of the pan/tilt system decreases the further away the target is but can move in 1-inch increments aiming for a target 10-feet away. For the photodiode receiver to get the best reading of the frequencies it must be within 20 feet of the jammer. The battery life should be around 4 hours, running the DC motor at max speed and moving the servo, so in reality it may be a little longer. The batteries are rechargeable too.

How to Use:

1. Charge batteries: There are two batteries necessary for this system to operate; a 12V NiMH (green) that powers the laser jammer and a 11.1V LiPo (black) that came with the rover. Charge the rover remote control with a USB-C cable, charge the baby monitor inside the jammer remote control using a micro-USB cable, and charge the rover baby monitor with a micro-USB cable.
2. Plug in everything:
    i. Power the rover by inserting the LiPo and plugging it into the receiving end connector (orange/yellow, looks just like the one on the battery).
    ii. Once the rover has power, the RF receiver has power. Connect the remote control by powering it on, it should beep indicating a connection.
    iii. The NiMH battery has a Tamiya connector, locate the receiving end Tamiya connector inside the rover and plug them in. All the buck converters will turn on and the Arduino Nano 33 Sense will receive power, the red LED indicates no Bluetooth connection has been made.
    iv. Power the jammer remote control Nano 33 Sense by inserting a micro-USB cable (powered by a battery, outlet, or computer). Once the Nano has power, the red LED on the jammer should turn blue, indicating Bluetooth connection success.
3. Check that all systems are operating properly.

Background:

In essence, this project is a remote-controlled car that has a laser capable of pseudo jamming mounted on top. The problems that inspired this concept were the need for an upgraded jammer to compete with the revamped sensors, and the need to increase the number of roles available in the game. An optional stretch goal was to modulate the laser to create more dynamic action in the game, allowing the laser operator to change between frequencies and match the frequency of the sensor. A concurrent project worked on a backdoor laser receiver on the sensors and programmed it to jam the sensors if it is hit with the laser at the proper frequency.

Required Specifications:

- Fully functioning RC car/tank that does not interfere with the drones' signal.

- Fine movement capability of camera/laser for an accurate laser spot of at least a one-inch tolerance at 10 feet.
- Camera is reasonably bore sighted with the laser.
- Laser wavelength must match laser receiver or within laser receiver bandwidth.
- Battery life of at least 30 minutes (rechargeable).
- Signal range of 40 feet minimum.

Gameplay Suggestions:

Tank – Jammer - Drones

- Sit tank and jammer operators on different sides of a table (force communication, not allowed to look at each other screens).
- The jammer is allowed to ask the tank driver to move slightly to assist with aiming until the jamming aim system is improved.
- If somehow the tank gets stuck or rolls, it is a crash. The jammer must not continue usage if the tank crashes.
- Communication between drone pilots and tank/jammer operators allowed.
- Tank cannot be sent on the first reconnaissance mission.
- Cannot use the tank to direct drones or jam while on the blue side. (tank must be in the field to direct them or do any jamming).
- Points for returning home safely.

Jammer - Battle manager

- Jammer must wait for permission from the battle manager to switch on the jammer and hit the target. (Prevents rapid changing of the frequency to eliminate them until they find the right one).
- Battle manager can only give permission when the EW operator has a lock?
- Points for every sensor that is jammed? Points for amount of time a sensor is jammed?

Jammer Miscellaneous

- The jammer CANNOT “jam” the visual observer by aiming at their eyes.
- Can “jam” the turret by aiming laser at camera module?
- It would be pretty cool to have a receiver on the turret, that if hit with the correct frequency shuts down the turret for a few seconds.
- Secret extra points for playing a lullaby.

Tank Miscellaneous

- Data collection from tank not valid for points. (ie: only asset tag numbers/locations valid if provided by the drones).
- Tank can take more than 1 hit from the turret (strong armor).
- Must be some sort of pathway for the tank in the field. At least half of the sensors are jammable from that pathway (the receivers, similar to asset tags, must be able to be hit by the jammer).
- Tank operator can try to move asset tags and can corner the high value asset tag to make drones mission easier.
- The tank is viable to be detected. Detecting the tank with the sensors is valid for points for the red team.

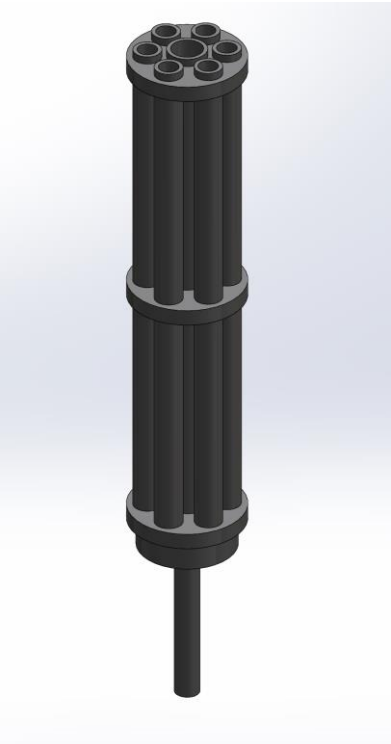
Design:

GitHub Link

Find all the CAD documents needed for this project in the repository. All the names shown here with the pictures relate to the CAD file names in the repository with the exception of “Intern 2024”. All the files will have that at the end of them, it was for my own organization to make it easier to find files I used for this project.

https://github.com/Always-Fresh/Laser-Jammer-Tank.git

Laser Holder



Overview:

- Arbitrary 6 barrels
- Barrel diameter must be larger than laser beam
- 50% duty cycle

- Keyway for motor

This concept began purely for visual appeal, based off some military vehicles. I realized I could modulate the laser by spinning the gatling gun, and everything started forming around this idea. The six-barrel design was chosen arbitrarily, but the rest of the design was shaped around the size of the laser diode and maintaining a compact design. The barrel holes are big enough to allow the laser beam through and relate exactly to the arc length between the barrel holes. As seen in *Figure 1*, r' and d are equal lengths. This ensures that the time spent on and off for the laser are equal, creating a 50% duty cycle. This part relates closely to the laser diode housing and the 12V motor. It has a long keyway that fits snugly around the motor, and that fits inside a ball bearing (which adds stability as a second point of contact).

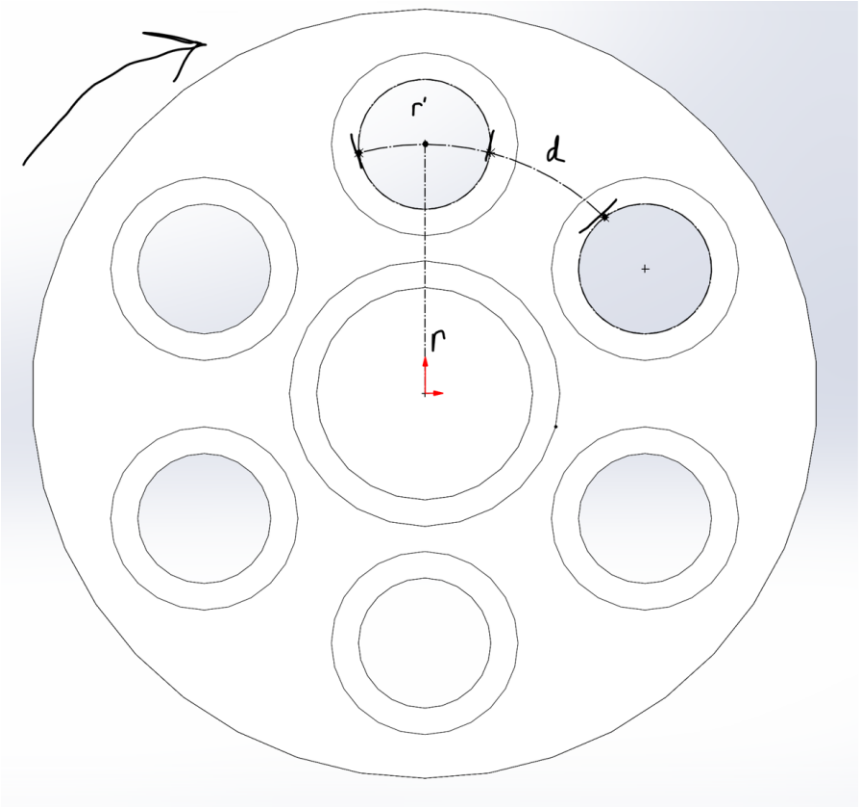


Figure 1:

The following derivation uses *Figure 1* as a reference and relates how the rotations per minute of the motor relate to the frequency of modulation.

Derivation	Equation Used
r = 9.5mm	
r' = d = 4.92mm	
d' = r'+d = 4.92mm x 2	
$V=\frac{d'}{t}$	Average Linear Velocity
$V=\frac{d'}{T}$	T(period) = t(time)
$\omega r = \frac{d'}{T}$	V=ωr (Avg Angular Velo)

$$T = \frac{d'}{\omega r}$$ Rearrange

$$f = \frac{\omega r}{d'}$$ (When ω is in Rad/s) $$f = \frac{1}{T}$$ (Frequency reciprocal)

$$f = \frac{r\pi}{30s \times d'}\omega$$ $$\frac{2\pi}{1\ \text{Rot}}$$ & $$\frac{1\ min}{60s}$$

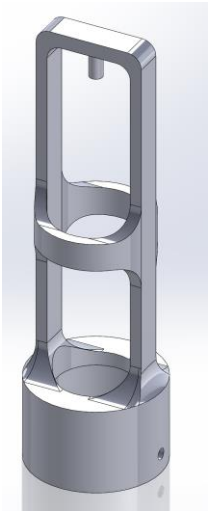
$$f = \frac{9.5\pi}{60s \times 4.92}\omega$$ (When ω is in RPM) Insert Values

Improvements:

- I am not sure how, but it may be possible to design a more stable model.
- The long keyway can be pretty weak, even though it does not have a lot of load on it.

  This could be improved by making it thicker or adding fillets.

Laser Modulator Stabilizer



Overview:

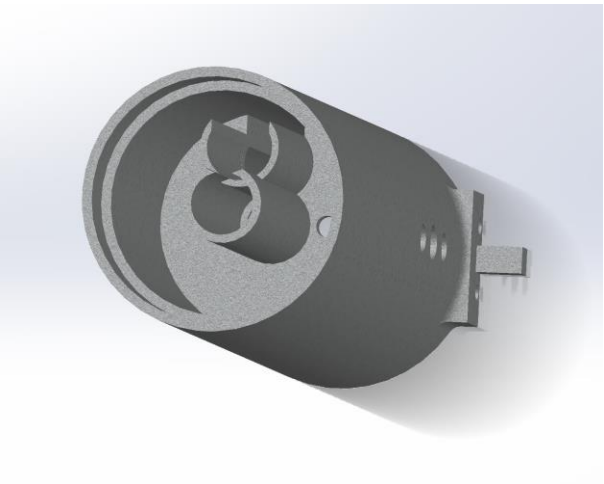
This attaches to the motor housing and stabilizes the gatling gun as it spins. This adds a second point of contact at the furthest lever arm and maximizes stability.

Improvements:

- Needs more holds for screws to go into the motor mount
- Needs a hole instead of a pillar to go into the gatling gun, where a pin goes through.

Laser Motor Mount

Zip



Overview:

- Fits 12V motor & laser diode
- Stabilizes gatling gun
- Baby camera mount compatible

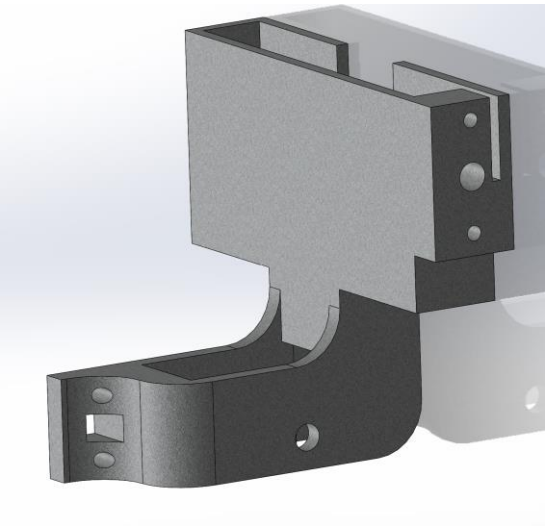
This design was shaped around the laser diode and the 12V motor, so changing it to fit another motor would be very involved. The length may seem long but it allows for some room so the laser diode is not stuck or rubbing against the gatling gun as it spins. Directly below the

diode holder is a extrude to hold the ball bearing that stabilizes the gatling gun. At the bottom on the outer edge there is a area that is compatible with the baby camera mount.

Improvements:

- Simplifying the CAD would be beneficial
- Adding holes for the laser stabilizer to connect.
- Holes for the laser stabilizer to connect to

Cam Mount Add On



Overview:

- Mounts a baby camera to laser diode housing
- Compatible with the laser diode housing

This design holds a gutted version of a baby camera. The body of the camera was necessary to remove due to its size, awkward shape, and unnecessary components. It has holes that fit the pin that comes with the camera to attach the camera to the mount. There is a path for the wires and a casing for the circuit board, as well as screw holes on the side to mount the

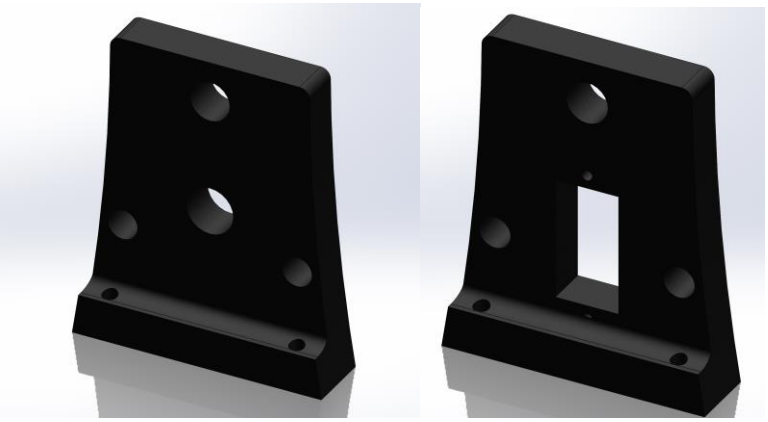
antenna. This design turned out very well, and only needed one test print and the second was nearly perfect. Compatible with the laser diode housing.

Improvements:

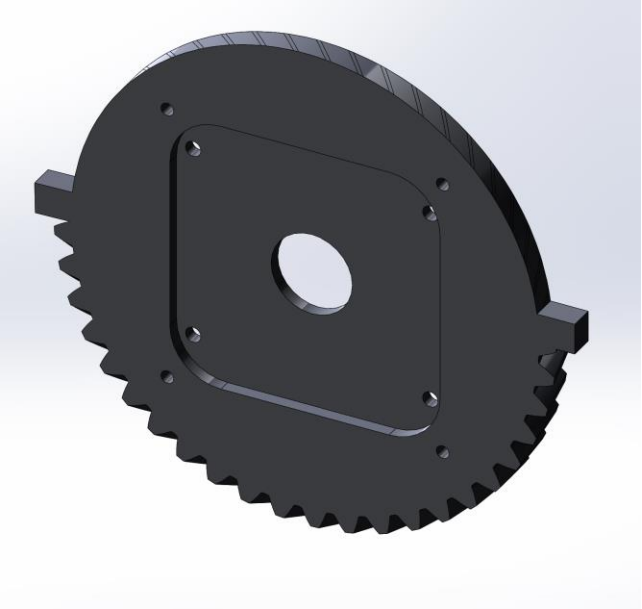
- There is a hole for the antenna in between the screw holes, and it needs to be moved slightly so that the antenna is not being bent anymore. It has not caused any problems but would just be a better design.
- Could be adjusted to be more in-line with the center axis of the barrel.
- Could be changed to allow for greater range of motion in the upward direction.

Pan/Tilt System

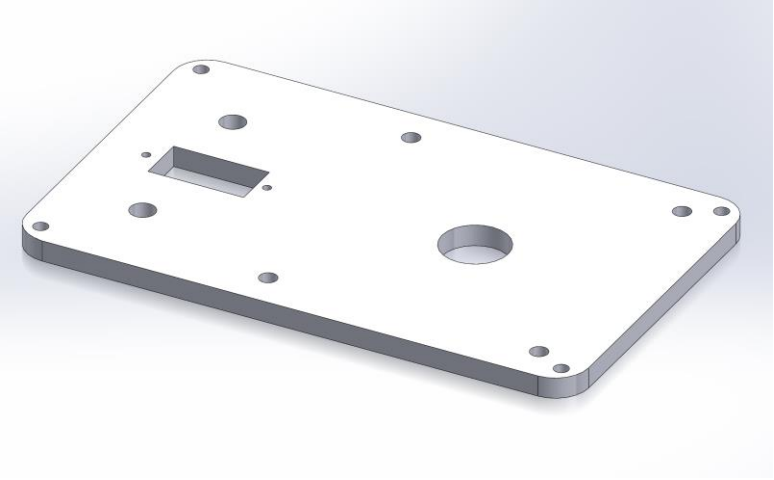
Tilt Holder



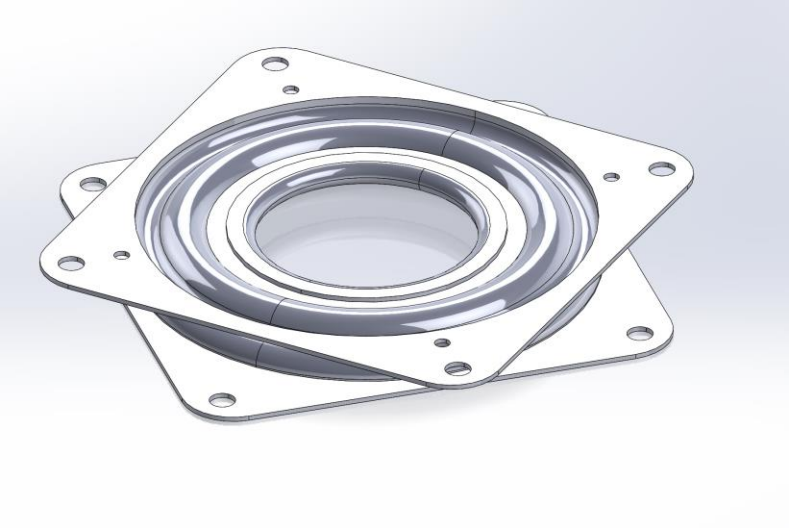
Pan Tilt Holder Gear



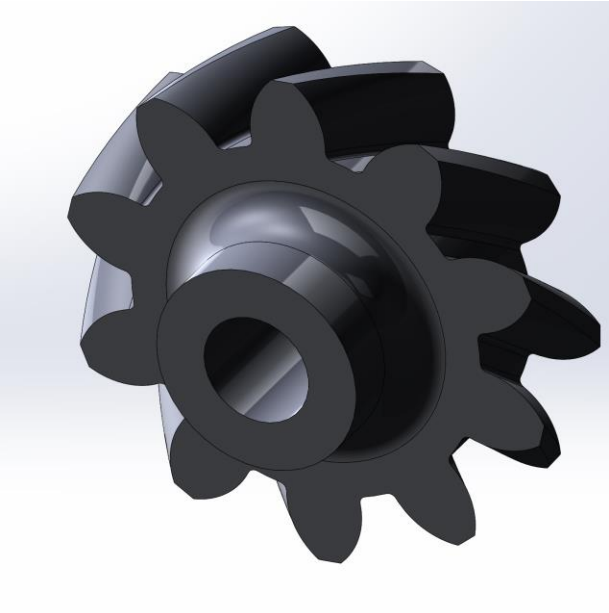
RC body Adapter



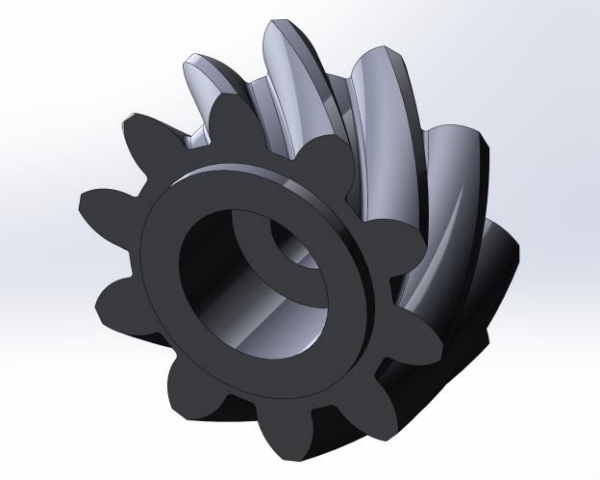
6031K16 3x3 ball bearing turn table



Pan Mover Gear

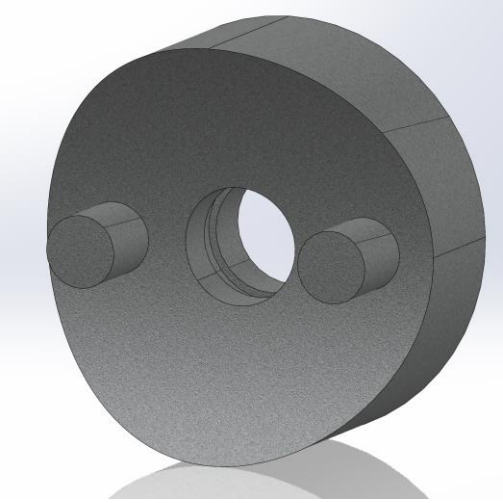


Small Helical Gear



Large Helical Gear

Spacer



90156A569_Flat-Head Quick-Release Pins

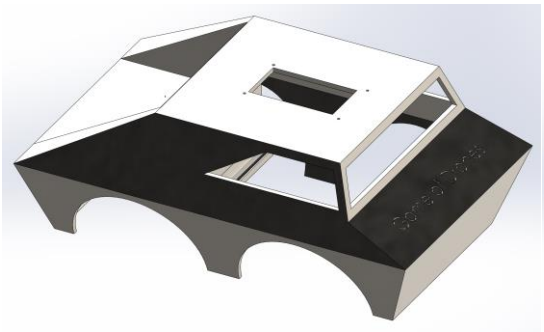
57155K209_Stainless Steel Ball Bearing

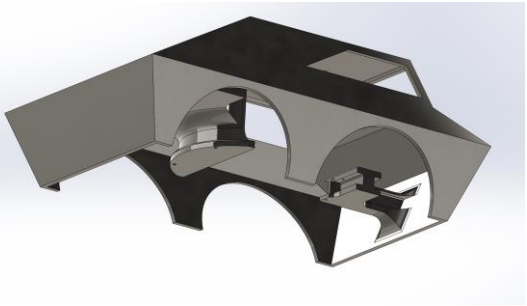
This system is based off another projects pan/tilt system (the nerf turret). It consists of two holders for the laser mount, a spacer, and two sets of helical gears. There are two metal pins that click into place to connect the holders, spacer, gears, and laser mount. The bottom base helical gear is connected to a lazy Susan that is then attached to the top of the RC tank body. The entire system is controlled by two 28BYJ-48 5V DC stepper motors. The gear ratios allow for precise movement and accuracy for the laser operator to be able to hit the small target sensor.

Improvements:

- Perhaps a better connection system instead of the pins.
- Better motor system. I messed around with stepper motors and servos, but the code and motor could definitely be improved for greater accuracy.
- Recommend: Iverntech Nema 11 Stepper Motor with 28mm Body. These should be small enough for the project, but much more capable than the 28BYJ-48 or servos. The code that is used to control the turret step motors could also be used for these.

RC Body





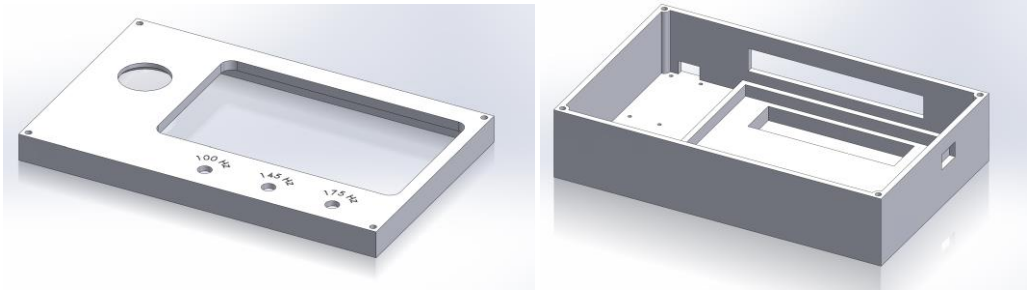
This part was based around a Humvee/cyber truck design visually, and practically to be compatible with the rover. On top it has holes to mount the lazy Susan to it, that holds the

jamming system. Inside there is room for wires, and mounts that attach to the rover. The model is huge, because the rover is much larger than anticipated. (Huge as in 2-3x larger than the print bed in our S3 3d printer and would take days to print).

Improvements:

- Sleeker/cleaner CAD design: it's messy because I was anticipating printing it in two separate parts, which called for creating different configurations and that got tricky.
- Size – if possible - to reduce it. Probably only through getting a smaller rc car or rover.

Arduino BT Controller Lid & Arduino BT Controller

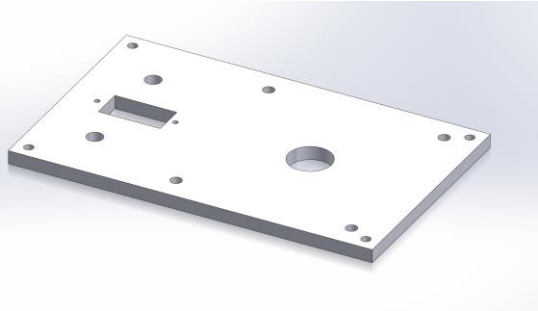


Designed to contain the baby moniter display, arduino nano 33 sense, an arduino analog joystick, and 3 toggle switches.

Improvements:

- More space for wires, the fit is currently a little tight.
- The hole on the right side of the bottom portion is for the charger for the baby monitor, and it is not in the correct position.
- The hole for the micro-USB needs to be bigger to fit the large rubber parts better.

RC Body Adapter

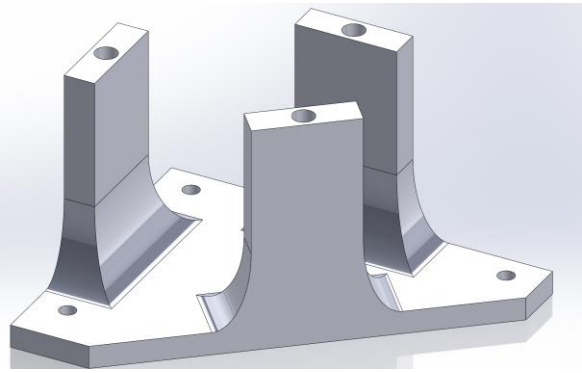


This is a adapter for the laser system on top of the rc car. Instead of needing to reprint the entire car to adapt to different motors, it is just this one piece that can be printed in a couple hours. The current model fits both a 28BYJ-48 step motor or a servo, and the turntable that the pan holder gear connects to.

Improvements:

- This needs holes added that are in the correct placement for the temporary rc body below.
- One of the holes for the servo needs to be moved, it is off slightly.

Temporary RC Body

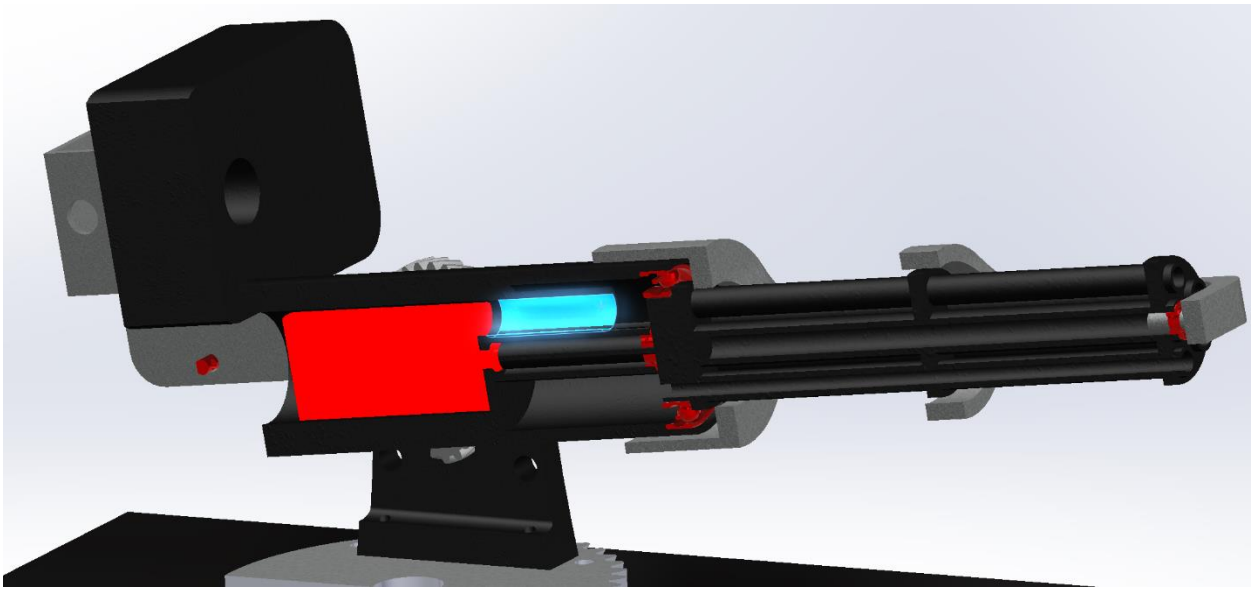


This is less of a temporary RC body and more of a replacement. This sits on the corner of the rover, using the included screws to attach to the top. Then the rc body adapter can be screwed into place on the top.

Improvements:

- This could be spaced out more to allow easier access inside, if just for screwing it into and out of place. (Taller and wider/longer).

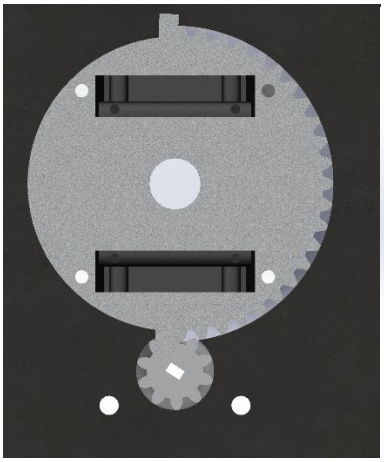
Assembly Steps (With some Pictures):

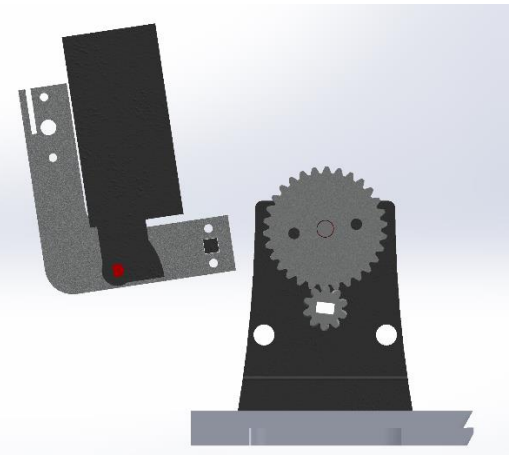


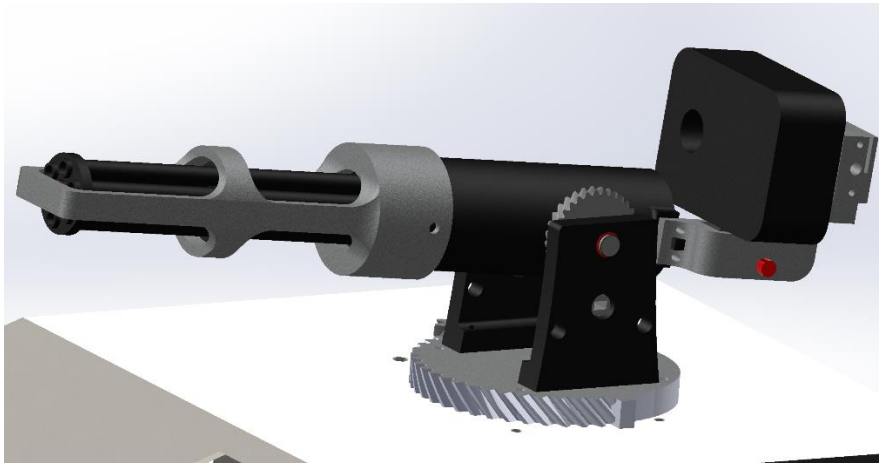
1. Starting with the motor housing put threaded inserts into the holes, and then screw the large helical gear to it.
2. Place slide the camera holder into its place and insert the screws.
3. In the blue highlighted area slide the laser diode in, making sure the wires go through out the back end.
4. The highlighted red is the 12V motor, slide this into the mount and screw into place, letting the laser diode wires sit above it.
5. The baby camera has to be dismantled to fit correctly. Everything should be stripped until all that is left is the camera face, the PCB that is needed to run it, and the antenna. (Note: because of this, the monitor will have an error showing up on the screen every few seconds, but this can be ignored).
6. The baby camera can now be placed into the holder.

7. The baby camera has a cord that powers it. It needs to be cut, stripped, and converted into headers that can go into a slicer. For this camera, the wire with the long gray stripes is power, and the other ground.
8. Place the small and large ball bearings (shown in dark red) in their respective slots in front of the laser diode. In addition, place the other small bearing into the front of the gatling gun.
9. Insert the gatling gun into the stabilizer, then insert the keyway onto the motor. Make sure it catches correctly.
10. Screw the stabilizer into the mount and insert the pin into through the stabilizer into the gatling gun.
11. On either side of the pan system holder, put ball bearings into the top hole. Then a pin goes through.
12. On one pan system holder, the spacer goes on the pin.
13. On the other put in threaded inserts into the two wide holes and screw the step motor into those.
14. Both pan system holders get screwed into the pan helical gear.

15. The pan helical gear goes into the turntable, which goes onto the RC car body.







16. The rover must be assembled following the manufacturer's instructions.

17. Attach the baby camera to the RC body directly.

18. Connect the replacement body to the car.

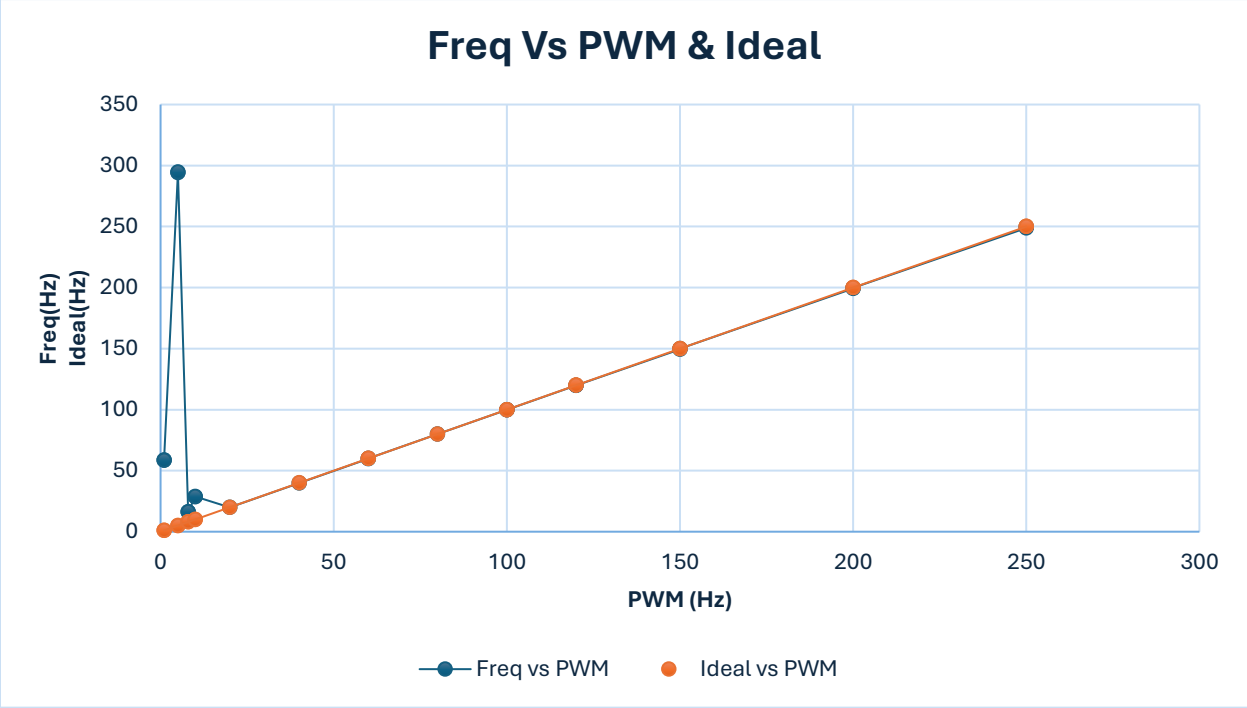
19. Connect the adapter to the replacement body.

20. Attached the preassembled jammer to the turntable, and the turntable to the adapter.

Data:

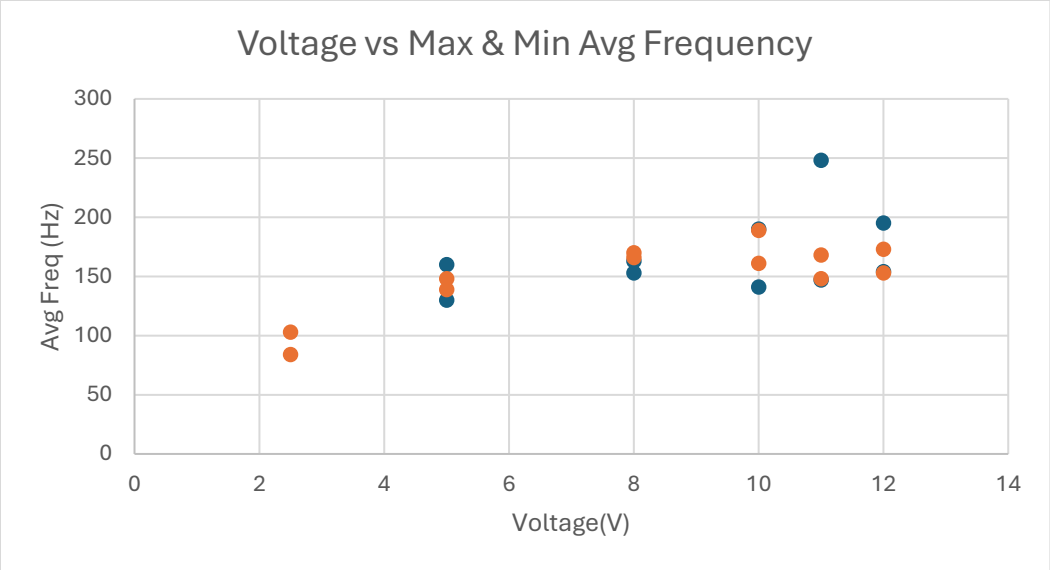
Data was collected to determine how the voltage given to the 12V planetary motor relates to the frequency generated by the gatling gun. First, the accuracy of the photodiode had to be

tested. The outputs of the photodiode were compared to a known pulse-width-modulation (PWM) of the laser diode. Ideally, the data would have a slope of 1, indicating that the photodiode is reading exactly what the frequency is.

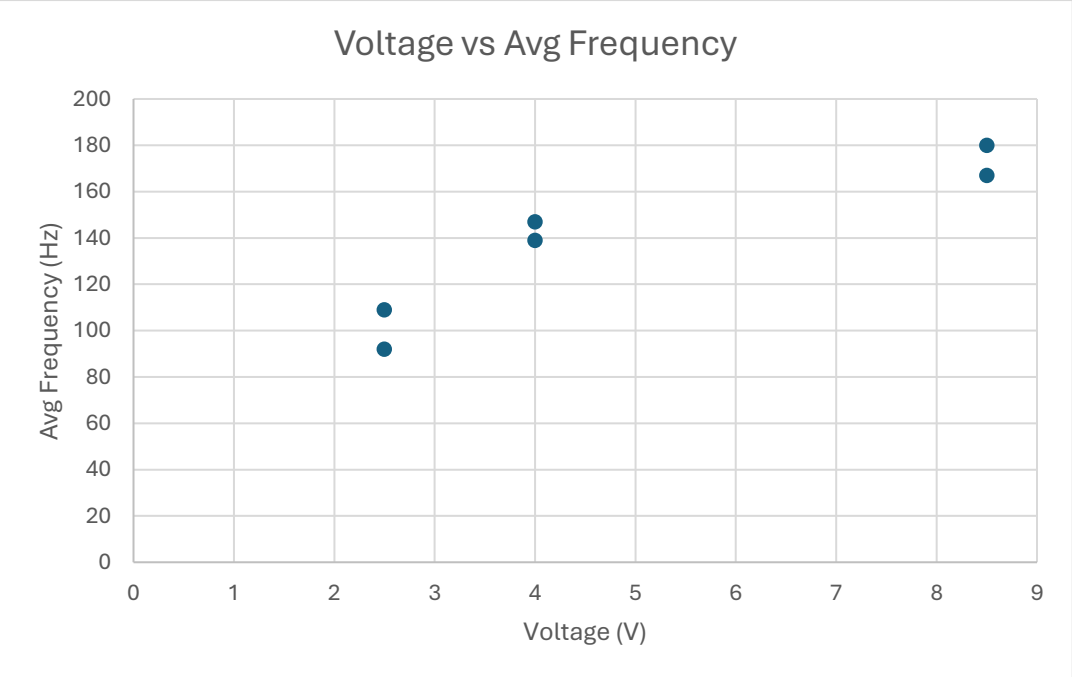


Graph 1

Based on this data, the photodiode is very accurate when the PWM is above 20 Hz because after that point, the data has a trend with a slope of nearly 1 and lines up well with the ideal.



Graph 2



Graph 3

Graph 2 and *3* shows the results from testing the frequency output of the gatling gun at different voltages (rpm). I took a few different approaches to make the outputs more generalized,

since they had some extreme outliers. The program collecting the frequencies was already taking averages, but I made it average the outputs it was producing from *Graph 2* and arrowed down the voltages I was likely to use and got *Graph 3*. *Graph 3* ranges in three voltages - 2.5, 4.0, and 8.5 volts--, producing 100 Hz, 140 Hz, and 175 Hz respectively. This range was chosen to minimize the number of volts (which minimizes how much the gatling gun vibrates) and to maximize the difference in the frequency outputs (to make it easier to differentiate).

As a result, at 2.5V the gatling gun will be outputting approximately 100Hz, at 4.0V - 145Hz , and at 8.5V - 175Hz . However!!! Taking the averages as I did is no longer viable. The sensor project has other limitations I was unaware of, and so it will need to be retested using the proper code to see what the photodiodes really detect with the actual program!!!

After quick analysis with the new program:

2.5V ➔20-60Hz

3.2V ➔60-160Hz

8.5V➔180-230Hz

This should be tested further and adjusted as needed, to differentiate between frequencies.

Wiring:

Due to the 12V DC motor chosen and designed around, the only way to change the speed (and thus the frequency) is by varying the voltage. To do this, I used buck converters and relays. The relays are powered by the Nano, then the 12V battery goes into the relay switch and through to the buck converters. The relays are programmed to stay closed even if two switches are thrown, for safety. Thus, the relays only close if one switch is thrown, and the voltage can be converted into the

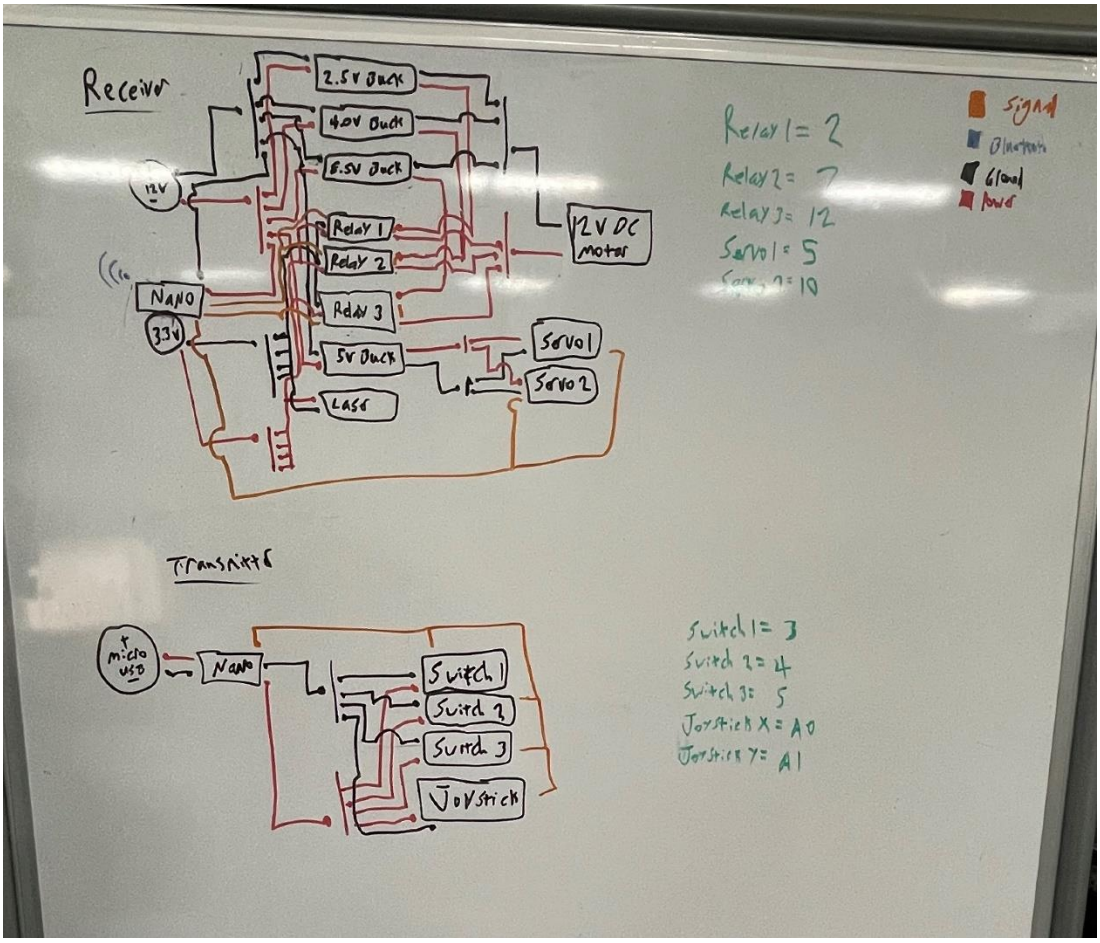
three voltages discussed in the data section. The Nano can be powered from any power supply, as it has its own built-in voltage regulator.

Improvements note: The pins on the Nano do not hold well, and there is an issue with the rover bumping around and jostling them loose. This will need to be fixed to improve reliability.

Code for Arduino IDE, utilizing the stepper.h, ArduinoBLE.h, and servo.h libraries and connects two Arduino Nano 33 Sense Boards via Bluetooth to control two stepper motors (or servos), 3 relays, and a DC motor.

Usage

Once the code is uploaded to the boards, they should remember it no matter what power source they are plugged into. The code implements the usage of the on board RGB lights to help troubleshoot connection. Once both boards are powered, the lights will initially be red, indicating that there is no connection. If the boards successfully connect, the lights will turn blue. Should they for any reason disconnect, they will turn red again. The reset button on the boards will not erase the code but will start the code from the beginning; try this if they have trouble connecting as it will reboot the signal they send out.



Code:

This code was written for use with Arduino IDE and Arduino Nano 33 Sense boards to communicate through Bluetooth. Once uploaded, the boards should remember the code until it is intentionally erased. When powered, the boards built-in RGB light will turn on red until they connect, at which point there will be a blue light. This is usually pretty fast.

The only possible issue that may show up is when the boards are connected, and the receiver loses power, but the transmitter remains active. When this happens and the receiver is powered again, they will not connect because the transmitter does not begin searching again. To

fix this issue, simply unplug the transmitter and repower it. This is a possible area of improvement.

Transmitter

```
#include <ArduinoBLE.h>


// UUIDs for the BLE service and characteristic
const char* deviceServiceUuid = "19b10000-e8f2-537e-4f6c-d104768a1214";
const char* deviceServiceCharacteristicUuid = "19b10001-e8f2-537e-4f6c-
d104768a1214";

int joystickX = 0;
int joystickY = 0;
int oldJoystickX = -1;
int oldJoystickY = -1;

const int switch1Pin = 3; // Defining switch pins
const int switch2Pin = 4;
const int switch3Pin = 5;

void setup() {
  Serial.begin(9600);

  if (!BLE.begin()) {
    Serial.println("* Starting Bluetooth® Low Energy module failed!");
    while (1);
  }

  BLE.setLocalName("Nano 33 BLE (Central)");
  BLE.advertise();

  Serial.println("Arduino Nano 33 BLE Sense (Central Device)");
  Serial.println(" ");

  pinMode(switch1Pin, INPUT_PULLUP); // Set switch 1 pin as input with internal
pull-up resistor
  pinMode(switch2Pin, INPUT_PULLUP); // Set switch 2 pin as input with internal
pull-up resistor
  pinMode(switch3Pin, INPUT_PULLUP); // Set switch 3 pin as input with internal
pull-up resistor

  // Initialize pins as outputs
```

```
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(LEDB, OUTPUT);

  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, HIGH);

  digitalWrite(LEDR, LOW);    //Red light means no connection
}

void loop() {
  connectToPeripheral();
}

void connectToPeripheral() {
  BLEDevice peripheral;

  Serial.println("- Discovering peripheral device...");
  do {
    BLE.scanForUuid(deviceServiceUuid);
    peripheral = BLE.available();
  } while (!peripheral);

  if (peripheral) {
    Serial.println("* Peripheral device found!");
    Serial.print("* Device MAC address: ");
    Serial.println(peripheral.address());
    Serial.print("* Device name: ");
    Serial.println(peripheral.localName());
    Serial.print("* Advertised service UUID: ");
    Serial.println(peripheral.advertisedServiceUuid());
    Serial.println(" ");
    BLE.stopScan();
    controlPeripheral(peripheral);
  }
}

void controlPeripheral(BLEDevice peripheral) {
  Serial.println("- Connecting to peripheral device...");

  if (peripheral.connect()) {
    Serial.println("* Connected to peripheral device!");
    Serial.println(" ");
    digitalWrite(LEDB, HIGH); //Turn off green light
```

```
    digitalWrite(LEDR, HIGH); //Turn off red light
    digitalWrite(LEDG,LOW); //Solid blue light indicates stable connection
  } else {
    Serial.println("* Connection to peripheral device failed!");
    Serial.println(" ");
    digitalWrite(LEDG, HIGH);   //Turn off green light
    digitalWrite(LEDB,HIGH);   //Disable blue light
    digitalWrite(LEDR,LOW);   //Red light indicates disconnection
    return;
  }

  Serial.println("- Discovering peripheral device attributes...");
  if (peripheral.discoverAttributes()) {
    Serial.println("* Peripheral device attributes discovered!");
    Serial.println(" ");
  } else {
    Serial.println("* Peripheral device attributes discovery failed!");
    Serial.println(" ");
    peripheral.disconnect();
    digitalWrite(LEDG, HIGH);   //Turn off green light
    digitalWrite(LEDB,HIGH);   //Disable blue light
    digitalWrite(LEDR,LOW);   //Red light indicates disconnection
    return;
  }

  BLECharacteristic joystickCharacteristic =
peripheral.characteristic(deviceServiceCharacteristicUuid);

  if (!joystickCharacteristic) {
    Serial.println("* Peripheral device does not have joystick characteristic!");
    peripheral.disconnect();
    digitalWrite(LEDR,LOW);
    return;
  } else if (!joystickCharacteristic.canWrite()) {
    Serial.println("* Peripheral does not have a writable joystick
characteristic!");
    peripheral.disconnect();
    digitalWrite(LEDG, HIGH);   //Turn off green light
    digitalWrite(LEDB,HIGH);   //Disable blue light
    digitalWrite(LEDR,LOW);   //Red light indicates disconnection
    return;
  }

  while (peripheral.connected()) {
    joystickX = analogRead(A0);  // Read X-axis
```

```
    joystickY = analogRead(A1);  // Read Y-axis

    // Read the switch states
    bool currentSwitch1State = digitalRead(switch1Pin);
    bool currentSwitch2State = digitalRead(switch2Pin);
    bool currentSwitch3State = digitalRead(switch3Pin);

    // Interpret the switch states:
    // LOW (pressed) = ON, HIGH (unpressed) = OFF
    bool switch1On = (currentSwitch1State == LOW);
    bool switch2On = (currentSwitch2State == LOW);
    bool switch3On = (currentSwitch3State == LOW);

    if (oldJoystickX != joystickX || oldJoystickY != joystickY) {
      oldJoystickX = joystickX;
      oldJoystickY = joystickY;

      Serial.print("* Writing X value to joystick characteristic: ");
      Serial.println(joystickX);
      Serial.print("* Writing Y value to joystick characteristic: ");
      Serial.println(joystickY);

      byte switchStates = 0;
      switchStates |= switch1On << 0; // Bit 0 for switch 1 (on = 1, off = 0)
      switchStates |= switch2On << 1; // Bit 1 for switch 2 (on = 1, off = 0)
      switchStates |= switch3On << 2; // Bit 2 for switch 3 (on = 1, off = 0)

      byte data[5];
      data[0] = (byte)(joystickX >> 8);  // High byte of X
      data[1] = (byte)(joystickX & 0xFF); // Low byte of X
      data[2] = (byte)(joystickY >> 8);  // High byte of Y
      data[3] = (byte)(joystickY & 0xFF); // Low byte of Y
      data[4] = switchStates;

      joystickCharacteristic.writeValue(data, 5);
    }
    delay(300); // Add a delay of 300 milliseconds
}

  Serial.println("- Peripheral device disconnected!");
  digitalWrite(LEDG, HIGH);   //Turn off green light
  digitalWrite(LEDB,HIGH);   //Disable blue light
  digitalWrite(LEDR,LOW);   //Red light indicates disconnection
}
```

Receiver Code

```
#include <Servo.h>
#include <ArduinoBLE.h>

Servo PanServo;
Servo TiltServo;

const char* deviceServiceUuid = "19b10000-e8f2-537e-4f6c-d104768a1214";
const char* deviceServiceCharacteristicUuid = "19b10001-e8f2-537e-4f6c-
d104768a1214";

int joystickX = 0;
int joystickY = 0;
int deadzone = 20;
int TiltAngleIncrement = 20; // Adjust this value to change angle increment size
for tilting
int PanAngleIncrement = 20;   // Adjust this value to change angle increment size
for panning



#define RELAY1_PIN 2
#define RELAY2_PIN 7
#define RELAY3_PIN 12

BLEService joystickService(deviceServiceUuid);
BLECharacteristic joystickCharacteristic(deviceServiceCharacteristicUuid, BLERead
| BLEWrite, 5);

void setup() {
  Serial.begin(9600);

  if (!BLE.begin()) {
    Serial.println("- Starting Bluetooth® Low Energy module failed!");
    while (1);
  }

  BLE.setLocalName("Arduino Nano 33 BLE (Peripheral)");
  BLE.setAdvertisedService(joystickService);
  joystickService.addCharacteristic(joystickCharacteristic);
  BLE.addService(joystickService);

  byte initialValue[5] = {0, 0, 0, 0, 0};
  joystickCharacteristic.writeValue(initialValue, 5);
```

```
  BLE.advertise();

  Serial.println("Nano 33 BLE (Peripheral Device)");
  Serial.println(" ");

   pinMode(RELAY1_PIN, OUTPUT);
  pinMode(RELAY2_PIN, OUTPUT);
  pinMode(RELAY3_PIN, OUTPUT);

  PanServo.attach(10);
  TiltServo.attach(5);

  PanServo.write(90);
  TiltServo.write(90);

   // Initialize pins as outputs
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(LEDB, OUTPUT);

  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, HIGH);

  digitalWrite(LEDR, LOW);    //Red light means no connection
}

void loop() {
  BLEDevice central = BLE.central();
  Serial.println("- Discovering central device...");
  delay(100);

  if (central) {
    Serial.println("* Connected to central device!");
    Serial.print("* Device MAC address: ");
    Serial.println(central.address());
    Serial.println(" ");
    digitalWrite(LEDB, HIGH); //Turn off green light
    digitalWrite(LEDR, HIGH); //Turn off red light
    digitalWrite(LEDG,LOW); //Solid blue light indicates stable connection

    while (central.connected()) {
      if (joystickCharacteristic.written()) {
        byte data[5];
        joystickCharacteristic.readValue(data, 5);
```

```
        joystickX = (data[0] << 8) | data[1];
        joystickY = (data[2] << 8) | data[3];
        byte switchStates = data[4];

        bool switch1State = switchStates & (1 << 0);
        bool switch2State = switchStates & (1 << 1);
        bool switch3State = switchStates & (1 << 2);

        if (switch1State || switch2State || switch3State) {
          if (switch1State && switch2State) {
            Serial.println("Safety feature triggered: Switch 1 and 2 are both
ON!");
            digitalWrite(RELAY1_PIN, LOW);
            digitalWrite(RELAY2_PIN, LOW);
            digitalWrite(RELAY3_PIN, LOW);
          }
          else if (switch1State && switch3State) {
            Serial.println("Safety feature triggered: Switch 1 and 3 are both
ON!");
            digitalWrite(RELAY1_PIN, LOW);
            digitalWrite(RELAY2_PIN, LOW);
            digitalWrite(RELAY3_PIN, LOW);
          }
          else if (switch2State && switch3State) {
            Serial.println("Safety feature triggered: Switch 2 and 3 are both
ON!");
            digitalWrite(RELAY1_PIN, LOW);
            digitalWrite(RELAY2_PIN, LOW);
            digitalWrite(RELAY3_PIN, LOW);
          }else {
            if (switch1State) {
              Serial.println("Relay 1 ON");
              digitalWrite(RELAY1_PIN, HIGH);
            } else {
              Serial.println("Relay 1 OFF");
              digitalWrite(RELAY1_PIN, LOW);
            }

            if (switch2State) {
              Serial.println("Relay 2 ON");
              digitalWrite(RELAY2_PIN, HIGH);
            } else {
              Serial.println("Relay 2 OFF");
              digitalWrite(RELAY2_PIN, LOW);
```

```
          }

          if (switch3State) {
            Serial.println("Relay 3 ON");
            digitalWrite(RELAY3_PIN, HIGH);
          } else {
            Serial.println("Relay 3 OFF");
            digitalWrite(RELAY3_PIN, LOW);
          }
        }
      } else if (!switch1State || !switch2State || !switch3State) {
        digitalWrite(RELAY1_PIN, LOW);
        digitalWrite(RELAY2_PIN, LOW);
        digitalWrite(RELAY3_PIN, LOW);
      }

      int panAngle = PanServo.read();
      int tiltAngle = TiltServo.read();

     // Serial.println(panAngle);

     // Serial.println(joystickX);
     // Serial.println(joystickY);

      if (joystickX > 502 + deadzone) {
        PanServo.write(panAngle + PanAngleIncrement);
      } else if (joystickX < 502 - deadzone) {
        PanServo.write(panAngle - PanAngleIncrement);
      }

      if (joystickY < 518 - deadzone) {
        TiltServo.write(tiltAngle + TiltAngleIncrement);
        Serial.println(tiltAngle);
      } else if (joystickY > 518 + deadzone) {
        TiltServo.write(tiltAngle - TiltAngleIncrement);
      }

      delay(15);
    }
  }
  Serial.println("- Central device disconnected!");
  digitalWrite(LEDG, HIGH);   //Turn off green light
  digitalWrite(LEDB,HIGH);   //Disable blue light
  digitalWrite(LEDR,LOW);   //Red light indicates disconnection
}
```

```
}
```