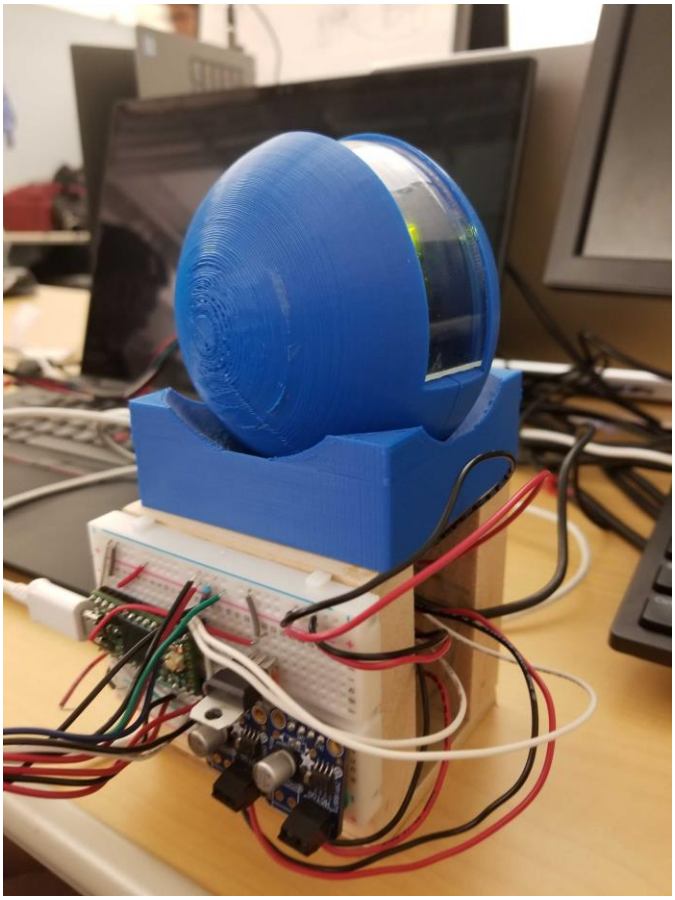

YawYaw

A robust Looky-Loo sensor module

Akash Harapanahalli
Andrew Hellrigel
Usman Jamal
Ethan Weinstock

HackGT6: Into the Rabbit Hole

Abstract

YawYaw is a "Looky-Loo" pan and tilt camera harness designed to address the National Security Innovation Networks's (NSIN) Look-Loo Challenge. The main goal of the challenge was to develop a system that would provide soldiers with hands-free interaction with localized autonomous sensor arrays. YawYaw was created through integrating four subsystems, mechanical, electrical, controls, and machine learning, to create a solution that enables a soldier to control a camera through head movement only. Through having limited user interaction with the camera module outside of head movement, YawYaw provides a simple interface for soldiers which would enable them to affectively utilize the additional data. YawYaw, despite the limitations of being developed in a thirty-six-hour hackathon, presents an effective solution to NSIN's challenge which can be improved through access to enhanced materials and manufacturing methods

Design Challenge

YawYaw was designed to address the challenges of NSIN's Looky-Loo challenge. The challenge entailed designing a pan-tilt multi-sensor system that would mount on unmanned ground vehicles (UGVs) and be controlled by a wearable headset. Currently, sensor array sensor used for autonomous function on UGVs are static and the field of view (FOV) cannot be adjusted with physically turning the vehicle. The two components of the challenge involved: 1) designing a control unit for the goggles that the user would wear to change the direction of the camera; 2) designing a sensor system that would be modular in nature and able to pan and tilt the sensor in the direction desired by the user.

The core requirements for the Looky-Loo challenge are detailed below in **Table I**. A listing of the special options for the Looky-Loo challenge is detailed in **Table II**.

Table I: A summary of the core requirements for the Looky-Loo challenge.

Left/right rotation: 180° (minimum); 360° (preferred)
Up/down rotation: 60° (minimum); 90°-120° (preferred)
Size: 8" by 5" by 3" box
Weight: 6 lbs (maximum)
Robust design enabling multiple users
Modular for different platforms

Table II: A summary of optional design constraints for the Looky-Loo challenge.

Able to be mounted on Unmanned Aerial Vehicles (UAVs) in addition to UGVs
Featuring machine learning control through computer interfacing
Capability to withstand hot and cold environments, dust, and other material intrusions.
Standardized interface and attachment system
Utilization of COTS components for increased fabrication speed and reduced cost.

Design Considerations

Based on the design requirements of the challenge and the requirements published on hackgt2019.devpost.com, Team YawYaw analyzed the challenge elements and developed a hierarchy of design goals and plans for innovating a new solution. A summary of design considerations and constraints for YawYaw can be found in **Table III** below.

Table III: A summary of design considerations for YawYaw.

Wants	Constraints
Able to survive extreme environments (withstand ±6G's) Low latency for data transmission 360° rotation 90°-120° tilt Waterproof Less than 2 lbs in weight	180° left/right rotation (pan) 60° tilt 8" x 3" x 5" form factor Less than 6 lbs in weight
Need to get	**To do**
2 motors Wireless modules 2 motor drivers Sensors for position Wires	Figure out TBS Crossfire CAD hardware Figure out sensor payload

Following attending the information session for the challenge hosted by NSIN, Team YawYaw concluded the following additional information about the design which is detailed below in **Table IV**.

Table IV: Additional design considerations.

A FatShark goggle headset would not be provided to the team
Sensor payload would not be provided to the team
Wireless communication modules, namely the TBS Crossfire, would not be provided to the team

Following the clarification of design challenges and constraints, Team YawYaw began working on the design of their robust system.

Design Overview

YawYaw is designed as a bi-axial rotating camera that is capable of providing a 360° FOV panning and 120°FOV tilting. The system functions by rotating an ellipsoid that houses the tilting system to provide the full FOV. YawYaw is composed of four separate subsystems, mechanical, electrical, controls, and machine learning, that all work together to create the final product. **Table V** details all parts used in the creation of YawYaw and their costs.

Table V: A cost summary of YawYaw.

YawYaw Design Components			Total Cost
Mechanical Components			
Gearmotor	2	~$2.00	$4.00
3D printer components	5	~$2.00	$10.00
Total			$14.00
Electrical Components			
Teensy 4.0	1	~$20.00	$20.00
DRV 8871	2	~$7.50	$15.00
LSM9DS1 IMU	1	~$16.00	$16.00
Trimpot	2	~$1.00	$2.00
5V voltage regulator	1	~$1.00	$1.00
9V Battery	1	~$1.00	$1.00
Button	1	~$0.25	$0.25
Total			$54.00
Sensor Components			
USB camera	1	~$20.00	$20.00
Total Cost			**~$89.25**

Mechanical Design

When designing the mechanical system for the YawYaw, it was a priority to fulfill all of the design constraints to some extent. There were some issues that we couldn't work around because there was limited time and access to resources, but all of the design constraints were at least determined to have solutions that would have been possible given more time and access to resources.

Due to accessibility, the entire assembly was 3D printed with exception to the acrylic lens that was used for the camera to see through. These parts were designed, however, to be injection molded out of polycarbonate because it is virtually shatterproof, it provides a clear window for the camera to see out of while still protecting it, it is a good material to protect the sensitive electronics without any worries for accidental shortages, and it holds up well to harsh environments. It's ability to fully encase the camera and other electronics allows for the system to be water resistant with the potential to be fully waterproof given more time to create a more sophisticated design.

While it's difficult to say without formal testing, the design should be able to handle high acceleration given that the electronics could be mounted better in a custom designed PCB (see electrical section for details)

The top part of the design is somewhat complicated in its current version, but it is also believed that this piece could be injection molded with polycarbonate to create a housing for all of the electronics, and a cover could easily be designed to create complete water-resistance.

The camera we used was a Logitech webcam due to availability, ease of use, and low latency. However, the design was created to be versatile and by simply changing the design of the mount, nearly any small camera could be used in the device such as the Runcam Eagle 2.

For controlling the actual motion of the device, two small metal gear motors were used. These were used because they allowed for quick motion and were relatively cheap and lightweight compared to other alternatives such as stepper motors.

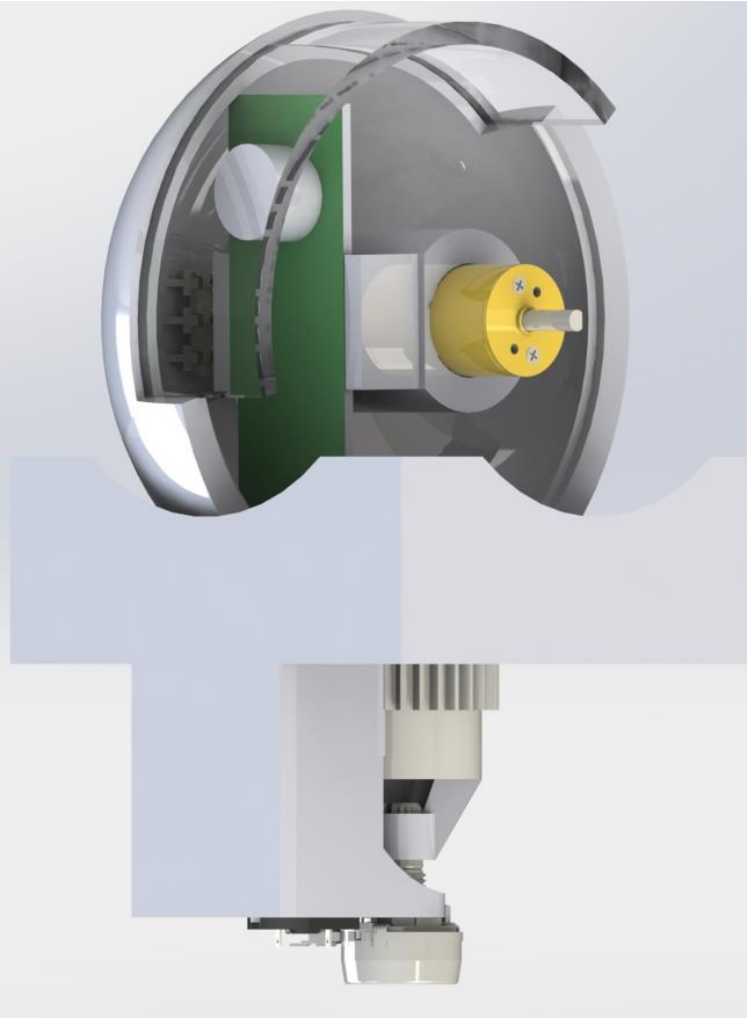

Figure 1: A view of the camera tilt component of the YawYaw.

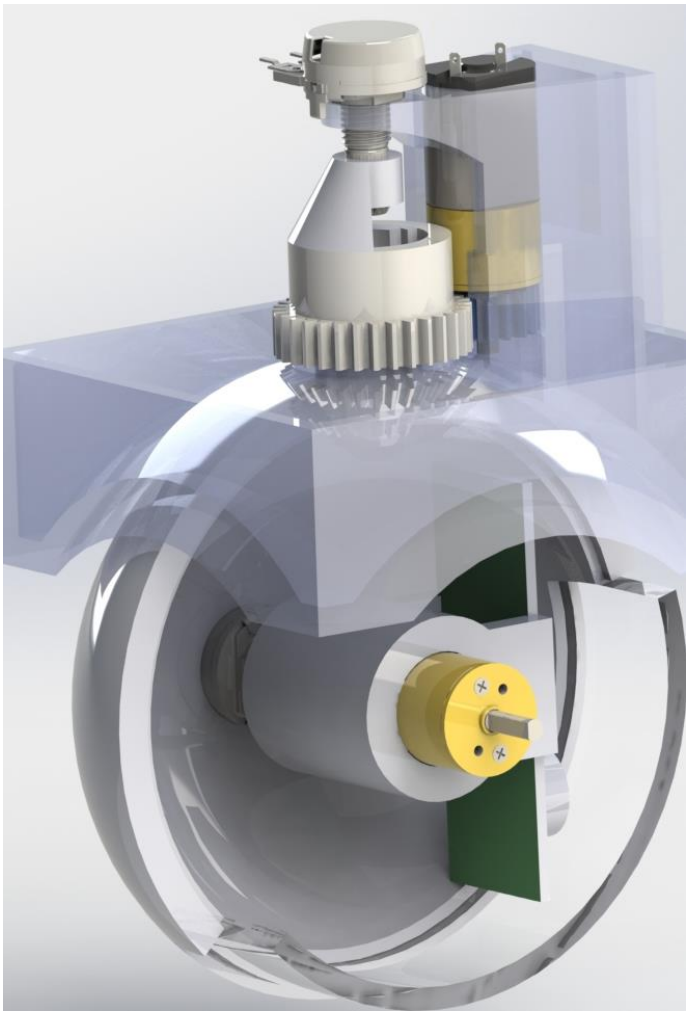

Figure 2: A view of the module rotation component of the YawYaw.

Electrical Design

The electrical subsystem of YawYaw is powered from a Teensy 4.0 microcontroller. The Teensy 4.0 was chosen to run the YawYaw as it is one of the fastest controllers on the market while also having a small form factor. These properties made the Teensy 4.0 advantageous for usage on the challenge as it would be able to handle the processing requirements of the task while also being able to fit into the required form factor.

An inertial measurement unit (IMU) was selected for YawYaw as it would enable the reading of an input from the movement of the user's head, one of the core requirements of the challenge. The LSM9DS1 IMU was specifically selected for the project as it was available for use by Team YawYaw and provided all the necessary data inputs needed to adjust the camera's FOV.

DRV8871 motor controllers were chosen for their compatibility with the gearmotors used for mechanical movement and their availability to Team YawYaw. The motor controllers are used to provide precise power to the motor and control the specific direction the sensor array faces.

Two trimpots were used on YawYaw to act as encoders for the two axes of rotation. The trimpots provide a measurement of the sensors' location and enable the software subsystem to correctly set the position of the sensor array.

A 5V voltage regulator was used to convert the output of a 9V battery to a voltage level safe for the Teensy 4.0.

A button was used to provide a rest function for the control subsystem.

Following the selection of the electrical components, a schematic was created in EAGLE. However, limitations in Team YawYaw's manufacturing capabilities prohibited the team from fabricating the PCB for YawYaw and a breadboard was created to address the team's electrical needs.

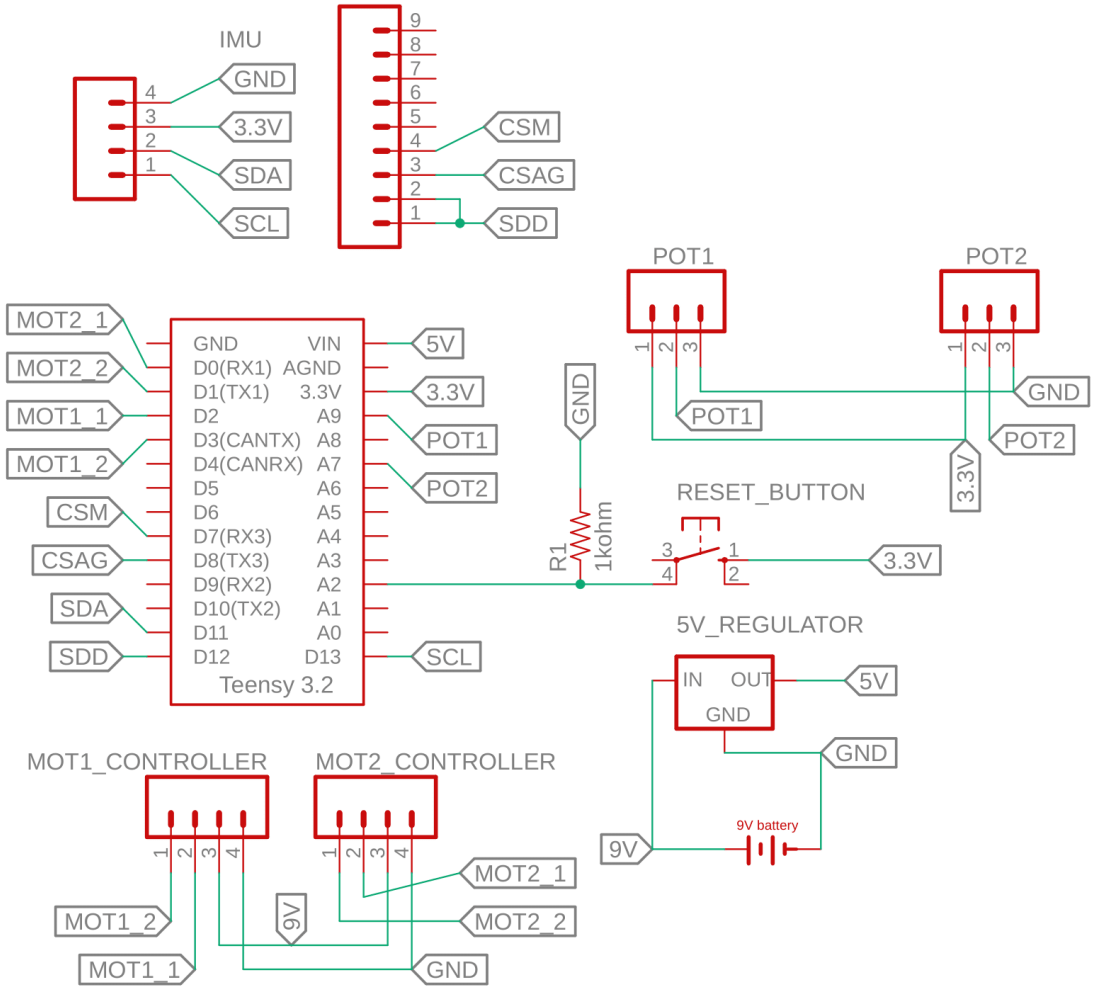

Figure 3: The schematic of the electrical subsystem.

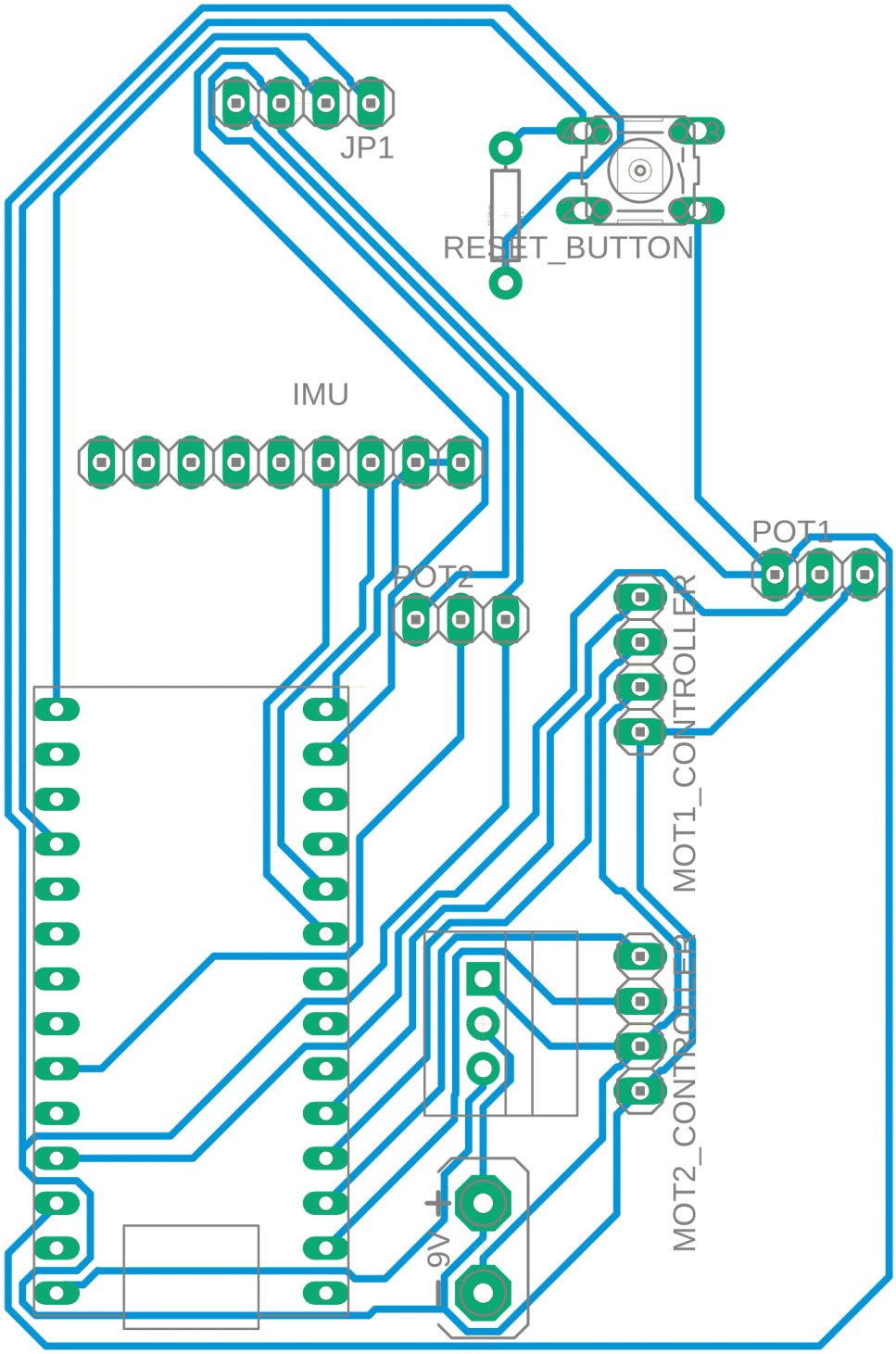

Figure 4: The PCB that was designed in EAGLE for YawYaw's electrical system.

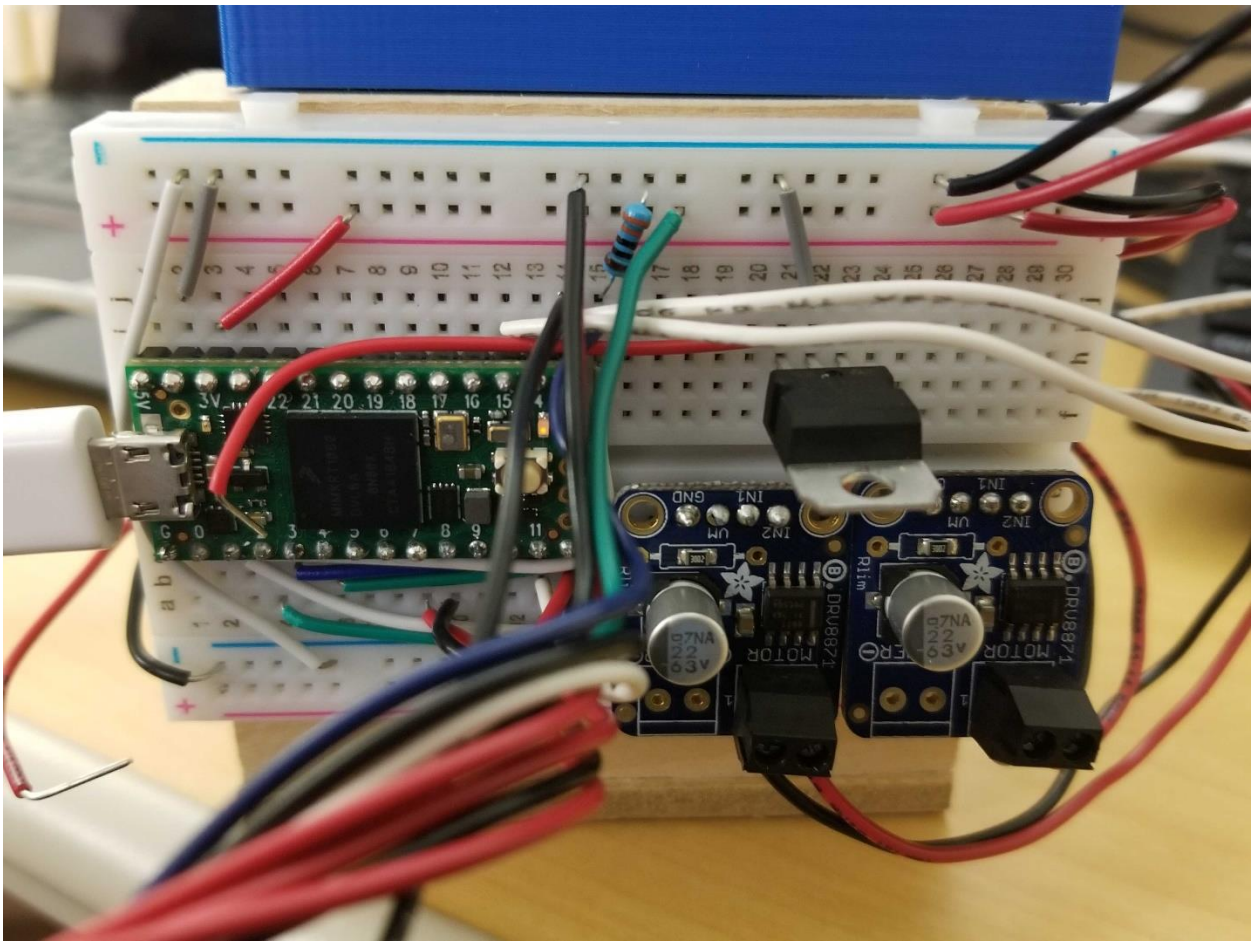

Figure 5: The breadboard that was created as an alternative to a PCB.

Controls Subsystem

YawYaw uses a Teensy 4.0 microprocessor to communicate with the LSM9DS1 IMU over Serial Peripheral Interface (SPI). To pan and tilt the sensor array, YawYaw utilized two Pololu 100:1 Micro Metal Gearmotor HPCB 6V together with two 10K ohm linear potentiometers to track their position.

Like mentioned before, the Teensy reads sensor data off of the the IMU using SPI, a standard communication protocol. However, while the IMU had a library, its actual usage ends up taking a multitude of function calls. As a result, Team YawYaw decided to make a wrapper class for all future calls to the IMU.

```cpp
#ifndef __IMU_H__
#define __IMU_H__

/**
 * imu.hpp
 * This header creates the imu class that wraps all future calls to the SparkFunLSM9DS1 API.
 * The IMU used for this project is the LSM9DS1 from sparkfun. Refer to their data sheet for more information.
 * While this is specifically built for use in this project, it can easily be scaled to work in other settings.
 *
 * The IMU plugs the magnetometer and the accelerometer/gyroscope into the MISO pin on the Teensy.
 * It plugs into the MOSI pin on the Teensy, as well as the CLK pin.
 * Finally, it plugs the M_CS into a digital port and the AG_CS into another digital port.
 */

#include <Wire.h>
#include <SPI.h>
#include <SparkFunLSM9DS1.h>

class imu {
private:
    // The SparkFunLSM9DS1 API is in use.
    LSM9DS1 sensor;

    // Need a non-temporary storage container to hold the accumulated gyro values.
    float acc_gz;

    int dt;

public:
    float angle_y, angle_z;

    /**
     * Sets up the new imu object with the M_CS and AG_CS pins specified.
     */
    void setup(int _M_CS, int _AG_CS, int _dt);

    /**
     * Calculates heading (angle_y) using the gyroscope and a simple threshold filter.
     * Calculates tilt (angle_z) using both gyroscope and accelerometer data.
     */
    void angles();
};

#endif
```

Figure 6: The header file for the imu class.

The LSM9DS1 is a $15 IMU, and as a result does not have perfect sensor readings. In fact, when the team first watched the sensor readings off the gyroscope (in terms of degrees/sec), they were surprised to see that there was drift in the range of +/- 5 deg/sec! Gyroscopes return an angular velocity, so the integral accumulates this slight error to make a decent effect over time. After some research, they came across this link: https://cache.freescale.com/files/sensors/doc/app_note/AN3461.pdf that explained how to use the three axis accelerometer data to filter the gyroscope data.

```
// Calculating the pitch using the accelerometers and a little bit of Linear Algebra :)
// https://cache.freescale.com/files/sensors/doc/app_note/AN3461.pdf
// In a nutshell, it compares the gravity vector (0, 0, 1) to the accelerometer vector.
// It uses rotation matrices to find the relationship and the angles that would cause the change.
// While this is not reliable in the short term, when added with the gyro's value, it can act as a filter and makes the long term value more precise.
float pitch = atan2(-this->sensor.ax, sqrt(this->sensor.az * this->sensor.az + this->sensor.ay * this->sensor.ay));

// Simple accumulation
this->acc_gz += 1 * ((abs(calc_gz) > 10 ? calc_gz : 0) * this->dt / 1000);

// Adding the accumulated angle with the pitch in the ratio indicated, acting as a filter for the gyroscope's angle (which is subject to drift over time).
this->angle_z = 0.95 * acc_gz + 0.05 * pitch;
```

Figure 7: The usage of the gravity filter to improve the pitch sensor reading

By comparing the gravity vector to the accelerometer vector, the pitch can be found. While data may be slightly erratic in the short term, its long term readings are much more reliable than the gyroscope. The ratio of 0.95:0.05 ensures that the accelerometer only affects the readings in the long term.

To control the motors, YawYaw uses the DRV8871 3.6-A Brushed DC Motor Driver. To control the speed of the motor, Pulse Width Modulation (PWM) is used on the signal pins of the motor driver. To avoid having to manually control the pins, a MotorDriver class was created.

```
#ifndef __MOTORDRIVER_H__
#define __MOTORDRIVER_H__

/**
 * MotorDriver.h
 * A wrapper to work with the DRV8871 Motor Driver.
 */

#include "PID.h"

class MotorDriver {
private:
    /**
     * The two leads that take input signal to power the motor.
     */
    int in1;
    int in2;
    PID pid;

public:
    /**
     * Sets up the digital out pins and the PID if specified.
     */
    void setup(int _in1, int _in2);
    void setup(int _in1, int _in2, float _kP, float _kI, float _kD, int _dt);

    /**
     * Power is taken in the range from -255 to 255 (dual 8-bit)
     */
    void setPower(int power);

    /**
     * Steps the internal PID and powers the motor.
     */
    void step(float error);
};

#endif
```

Figure 8: The header file for the MotorDriver class.

```
/**
 * Power is taken in the range from -255 to 255 (dual 8-bit)
 * Powering the motors is based on a potential difference between the in1 and in2 leads on the driver.
 * When in1 is LOW, the motor is powered "forward" and when in2 is LOW, the motor is powered "backward".
 */
void MotorDriver::setPower(int power){
    if(power > 0){
        analogWrite(this->in1, LOW);
        analogWrite(this->in2, power);
    } else if(power < 0){
        analogWrite(this->in1, abs(power));
        analogWrite(this->in2, LOW);
    } else {
        analogWrite(this->in1, LOW);
        analogWrite(this->in2, LOW);
    }
}
```

Figure 9: The setPower function within the MotorDriver class.

Finally, the motors' positions are tracked using the 10K ohm linear potentiometers. Their analog output is converted into degrees through a conversion formula, and used in conjunction with the degree output from the IMU. A PID (Proportion, Integral, Derivative) Controller was used with each motor/potentiometer combination to move motors to the desired angle.

```
#ifndef __PID_H__
#define __PID_H__

/**
 * PID.h
 * PID's are used for converting error into usable power output.
 * P - Proportion --> adding power as a proportion of the error
 * I - Integral   --> adding power as a proportion of the integral of error, leveling off any error present over longer periods of time.
 * D - Derivative --> removing power as a proportion of the derivative of error, preventing possible overshoots and restricting too large changes in error.
 */

class PID {
private:
    float kP, kI, kD;
    int dt;
    float integral;
    float derivative, prev_error;

public:
    /**
     * The constructor defines the PID constants and tells the PID how long the loop is that it is called in.
     */
    PID();
    PID(float _kP, float _kI, float _kD, int _dt);

    void setConstants(float _kP, float _kI, float _kD, int _dt);

    /**
     * The step function is an instantaneous step that calculates and returns a new power value.
     * This needs to be called within a closed loop where the parameter error is constantly updated.
     * PID output is in the range from -255 to 255 (dual 8-bit).
     */
    int step(float error);
};


#endif
```

Figure 10: The header file for the PID class.

```
/**
 * controller.ino
 * TARGET: Teensy 4.0
 * This is the main program to handle the funcitonality of YawYaw.
 */

#include "imu.h"
#include "MotorDriver.h"

#define LOOP_DELAY 10

imu main_imu;

MotorDriver cameraTilt;
MotorDriver cameraPan;

#define cameraTiltPot A0
#define cameraPanPot  A1

#define resetButton 0

void setup() {
  Serial.begin(115200);

  main_imu.setup(7, 8, LOOP_DELAY);

  cameraTilt.setup(1, 2, (float) 10.0, (float) 0.0, (float) 0.0, LOOP_DELAY);
  cameraPan .setup(4, 3, (float) 10.0, (float) 0.0, (float) 0.0, LOOP_DELAY);

  pinMode(resetButton, INPUT);
}

void loop() {
  main_imu.angles();

  if(digitalRead(resetButton) == HIGH){
    main_imu.angle_y = 0;
    main_imu.angle_z = 0;
  }

  int camTilt = analogRead(cameraTiltPot);
  int camPan = analogRead(cameraPanPot);

  cameraTilt.step( main_imu.angle_z - (270.0 / 1024.0) * (camTilt - 800) );
  cameraPan .step( main_imu.angle_y - (270.0 / 1024.0) * (camPan - 512) );

  delay(LOOP_DELAY);
}
```

Figure 11: “controller.ino”, the main file that is downloaded to the Teensy.

Machine Learning Subsystem

The YawYaw was designed with simplicity and the user in mind. It features a state-of-the-art convolutional neural network (CNN) model to classify objects in the field of vision. This aids the user in quickly assessing the surroundings in a high-pressure environment.

For the purposes of this demonstration, a database of cat and dog images was used to create a convolution neural network (CNN) model. This specific type of neural network is widely used for image classification, suiting this technology for the needs of the YawYaw. At the core of a CNN is a neural network, which is a learning framework that consists in multiple layers of artificial neurons, or nodes. Each node is randomly assigned a weighted input data value and passes it into to the activation function to determine an algorithm.

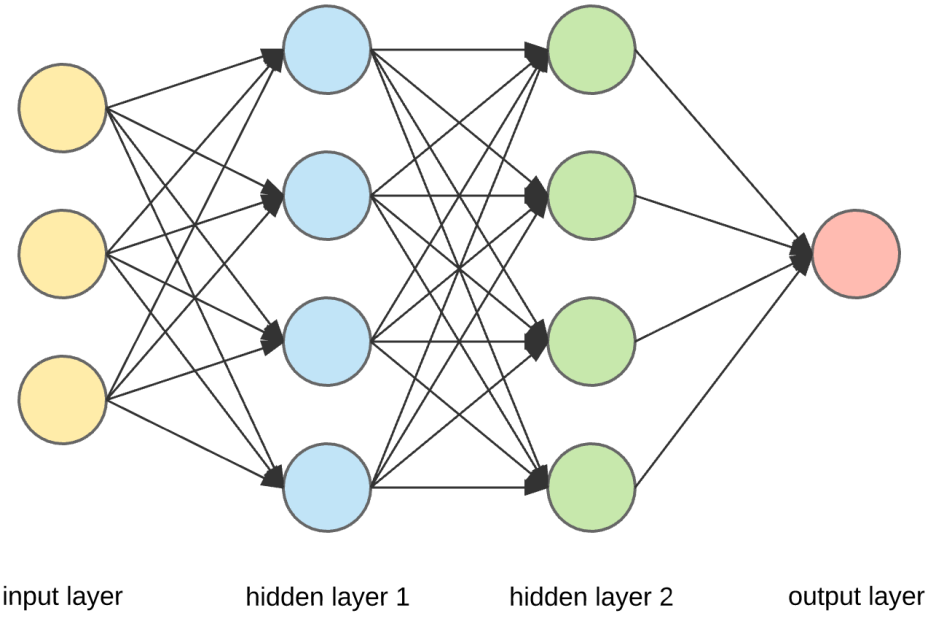

Figure 12: Architecture of a CNN

In [12]:
```
#Creating the Convolutional Neural Network(CNN) model-
#1.Begin by importing keras modules
#2.Create the network architecture (the way convolution layers are arranged) using the VGGnet structure.
from keras import layers
from keras import models
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import img_to_array, load_img

model = models.Sequential()
model.add(layers.Conv2D(32, (3,3),activation=('relu'), input_shape=(150, 150,3)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64,(3,3), activation= 'relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128,(5,5), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1,activation='sigmoid'))
```

Figure 13: Code for Network Architecture of CNN

The network architecture of this CNN was created following VGGnet structure that is widely used for object recognition. The convolutional layer is the building block of a CNN (which can be seen in the code as "...layers.Conv2D..."). The layers' consists of a set of learnable filters, or kernels, where in a forward pass each filter is convolved across the height and width of the input volume to produce a two-dimensional map of that filter. Thus, the neural network learns filters that activate when it detects some specific targeted feature of that input.

To push data into to the neural network, the data had to be sorted and standardized (see **Figure 14** below).

In [5]:
```
#This function iterates through the images, resizes them , and then appends them to the X and y list
#The X list will hold the training data set, and the y list will hold the training labels (1=dog, 0=cat)
def read_and_process_image(list_of_images):
    X= []
    y= []
    for image in train_imgs:
        X.append(cv2.resize(cv2.cvtColor(cv2.imread(image, cv2.IMREAD_COLOR), cv2.COLOR_BGR2RGB), (nrows,ncolumns),
                            interpolation=cv2.INTER_CUBIC))
        if 'dog.' in image:
            y.append(1)
        elif 'cat.' in image:
            y.append(0)
    return X, y
```

```
Epoch 63/64
100/100 [==============================] - 16s 157ms/step - loss: 0.4861 - acc: 0.7666 - val_loss: 0.2706 - val_acc: 0.7663
Epoch 64/64
100/100 [==============================] - 16s 156ms/step - loss: 0.5029 - acc: 0.7606 - val_loss: 0.4561 - val_acc: 0.7663
```

Figure 14: Image of Epoch outputs

After iterating through the epochs, the values of the training and validation accuracy were plotted to help visualize the precision of the algorithm.

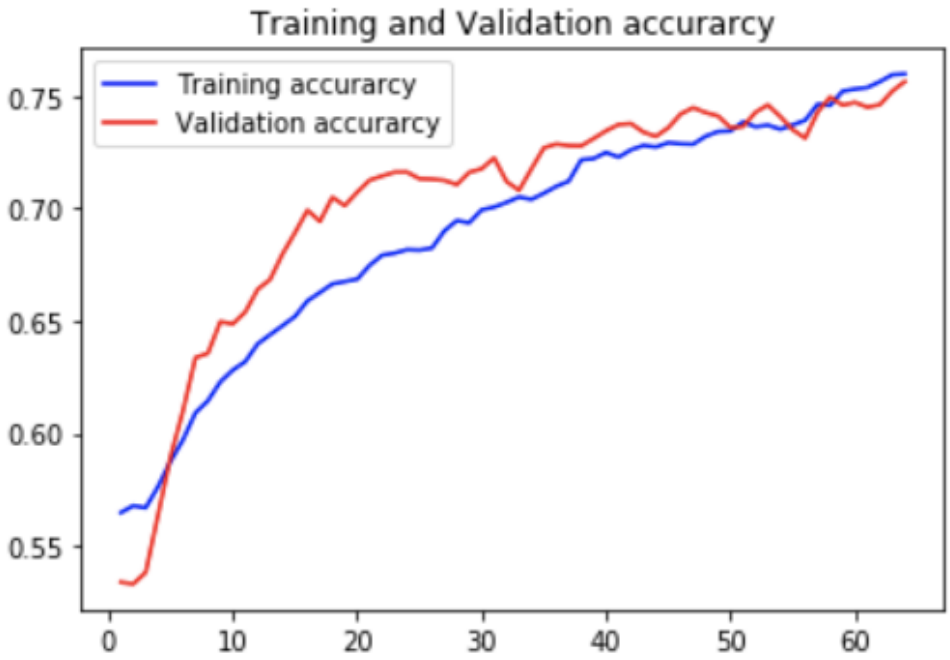

Figure 15: Graph of Training and Validation Accuracy

Challenges Faced and Future Improvements

Throughout the process of developing the YawYaw, the team faced challenges in battery technology, manufacturing capabilities and wireless modules that prevented YawYaw from reaching its maximum potential as a solution to the Looky-Loo challenge.

Issues with 9V batteries that function as the power supply for the YawYaw negatively impact the YawYaw's performance. The 9V batteries are not able to supply enough power for the YawYaw over a relatively brief period. As the battery voltage drops, the motors lose power and are not able to pan and tilt the camera assembly as effectively. As motor performance drops, the YawYaw's success as a solution is diminished. In the future, using a LiPo battery system would eliminate relatively rapid voltage drops and allow the YawYaw to function independent form a power source.

The team faced manufacturing limitations that impacted the functionality of the YawYaw. Lack of access to access to all desired manufacturing tools inhibited the YawYaw's functionality. While the design of YawYaw was based entirely on 3D-printed components, the team had limited access to 3D-printers. This limited access impacted the team's design flow as artificial deadlines were imposed on the team to get parts to the 3D printer before access was terminated for the day. In addition, lack of precision in the 3D printers available hindered the

team's ability to construct the YawYaw as parts did not mesh and interface as intended. Further, the team's inability to produce custom PCBs hindered electrical subsystem design effectiveness as the team had to spend time creating and debugging a bread-boarded setup. In the future, access to injection molding techniques would improve the functionality of the YawYaw. Injection molding would enable fast and accurate production of the YawYaw parts that would ease construction and create higher quality components.

Following the initial problem reading, Team YawYaw had concluded that wireless communication between the head module and the sensor array would be a key component to the success of the YawYaw. Initially, the xBee was considered as a wireless communication module. However, the inability to place the xBee available to the team in a breadboard and the lack of unfamiliarity caused to the team to remove beyond the xBee as a communications module. The team then tried the ESP32 WiFi/Bluetooth module. However, the team still failed to implement wireless communication with this module. The team finally switched to the ESP8266 wireless communication module embedded in the Node MCU Amica microcontroller. The team found success with using the ESP8266 as a communications module as communication was successfully established between ESP8266 units. However, the team ultimately abandoned wireless communications as numerous issues became apartment throughout the testing process, namely that wireless communication failed to be consistently effective. While it was possible to transmit data at rates over 25 Hz, sometimes data transmission would drop to a 0.33 Hz rate, equivalent to a 3 second pause in data transmission. This gap eliminated wireless communication as an effective solution as it would not be able to convey the sensor data from the headset to the sensor-array at a rate which would enable effective control. While it was possible to send data at high speeds, it was impossible to read data at 25 Hz. As sending speed would have needed to be reduced to be readable, this provided another reason to eliminate wireless communication from the system. Future work to improve YawYaw would involve creating a wireless communication system between the headset and the pan-tilt system. This communication means would most likely involve higher quality wireless communication processors that were not available to Team YawYaw over the duration of HackGT6.

Analysis and Conclusion

The mechanical design, electrical design, controls, and machine learning were all purposely chosen in a manner that best fits the design constraints, but are also the crucial to the YawYaw's versatile functionality.