Project Deliverable #4: Icarus Intended Manual

Opportunity:

Our motivation lies in finding a way to maximize energy harvesting or minimize wasteful energy consumption using robotics.

Icarus's High Level Strategy:

The user will input the time of day, location, and date into Icarus Intended's code in order for the device to work correctly. The user must turn on Icarus by pressing two push buttons. The first push button will first zero Icarus' solar panel to 0 degrees using a limit switch then move to the desired tilt angle calculated based on date, time, and location. The tilt will only be adjusted if the button is pushed again. The second push button will wake Icarus up. Icarus will move depending on which side detects more sunlight. Icarus will continue moving until one of two conditions is met: 1) both the left and right side detect the same amount of light within a certain tolerance range that is above a set minimum threshold to guarantee solar power generation or 2) a timer has expired. If Icarus's timer has expired, the user must press the push button again to activate Icarus. However, if Icarus is in the sunlight collection stage and the measured sunlight has dropped below the threshold, Icarus will start moving again and the timer will be reset. The user can turn off Icarus anytime with the second push button.

We originally had wanted Icarus to use more sensors including an INA219 and a magnetometer. The INA219 was to measure the power output across the solar panel to compare to the minimum threshold. This proved to be much more challenging so instead we simplified our design and used the photoresistors to set a threshold of minimum light capture instead of minimum power generation. The magnetometer was to be used for another case for Icarus Intended to face the sun. Any time Icarus would stop moving in order to gather sunlight, the magnetometer would have been used to track Icarus' cardinal direction and make sure Icarus was pointing east or west depending on the time of the day to get the most sunlight intake as possible. This also proved to be very challenging so Icarus's final model does not take cardinal direction into account. We also wanted Icarus to be initialized with the same push button instead of two. We also had to change the motor drivers used for the DC motors for the drivetrain and for the stepper motor.

Function Critical Decisions and Calculations:

I. Right and Left Side Drive: Torque Needed to Drive

A. Max Desired speed of car: 1.0 m/s

B. Variables for worse conditions: over approximate the weight, driving on incline

1. Coefficient of friction for coarse-grained soils: **0.5**
2. Crude approximation for weight: Wheels: 0.40 * 4 = **1.6 Kg,** stepper motor = **0.2 Kg,** driving motors: 1 * 2 = **2 Kg**, solar cell, battery, housing, electronics, gears, lifting mechanism, etc: **3 Kg. Total approximate weight = ~7 Kg**
3. Consider an angle of incline: $\theta = 10^{o}$
4. Radius of drive wheel: r = **0.035 m**
5. Want to reach max speed in 2 seconds: a = 1.0/2 = **0.5 m/s²**

C. Solving for the necessary torque, angular velocity, and power supplied to the wheel:

1. $N = mgcos(\theta) = 7 * 9.81 * cos(10^{\circ}) = 67.627\ \text{N}$
2. $f = ma + mgsin(\theta) = 7(0.5 + 9.81 * sin(10^{\circ})) = 15.42\ \text{N}$

3. $u_s N = 67.627 * 0.5 = 33.8\,N \geq f = 16.19$ N - no slipping will occur
4. $\tau = f * R = 15.42 * 0.03183 = 0.54$ N*m
5. We are using two driving motors, so we can divide this by two: $\tau_{per\ motor} = 0.54/2 =$ **0.27 N*m**
6. $\omega = 1 / 2\pi(0.035) = 4.5$rev/s = **273 rpm** = 47.125 rad/s

D. Translating motor to gears to wheel:

1. $\tau_1/R_1 = \tau_2/R_2$, $\omega_1 R_1 = \omega_2 R_2$

R_1 = 0.02 m

1) Diametral Pitch = 24
2) Pitch angle: 20 degrees
3) T_1 = 0.27 N*m
4) $\omega_1 = 273$ rpm

R = 0.0114 m

(1) Diametral Pitch = 24
(2) Pitch angle: 20 degrees
(3) T_2 = 0.156
(4) $\omega_2 = 479$ rpm

E. Choosing the motor:

1. *Decision:* Motor needs to supply a torque of 0.156 N*m (1.59 kg*cm) and 479 rpm, hence we choose a motor that supplies 2.3 kg*cm of torque so that we had a factor of safety in case any of the other components are heavier than predicted.

II. Forces on Bearings and Motor Shafts

A. Torque from the motor will be translated through gears, so the only radial force on the motor shaft is the weight of the gear: 0.907 g, F = 0.0088N. This is well within the limit.

B. Each wheel will have a shaft supported by two bushings, hence the force on each bushing is W/8 (because there are four wheels): $F_{per\ bearing} = Mg/8$ **= 8.584 N**

1. Decision: This is not a sufficient enough weight to damage the bushings or cause excess friction.

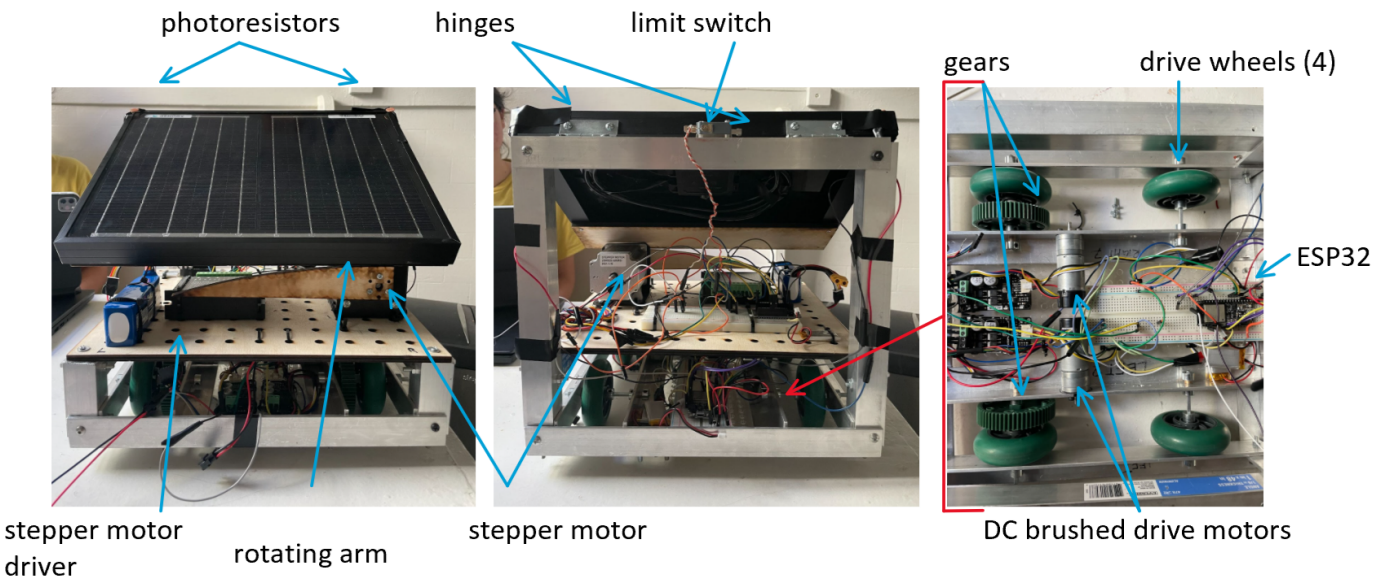
III. Motor Calculation for Rotating Arm

A. Force needed = (mass_panel + mass_fan)*g = (1.25kg) * (9.81) = 12.2625N

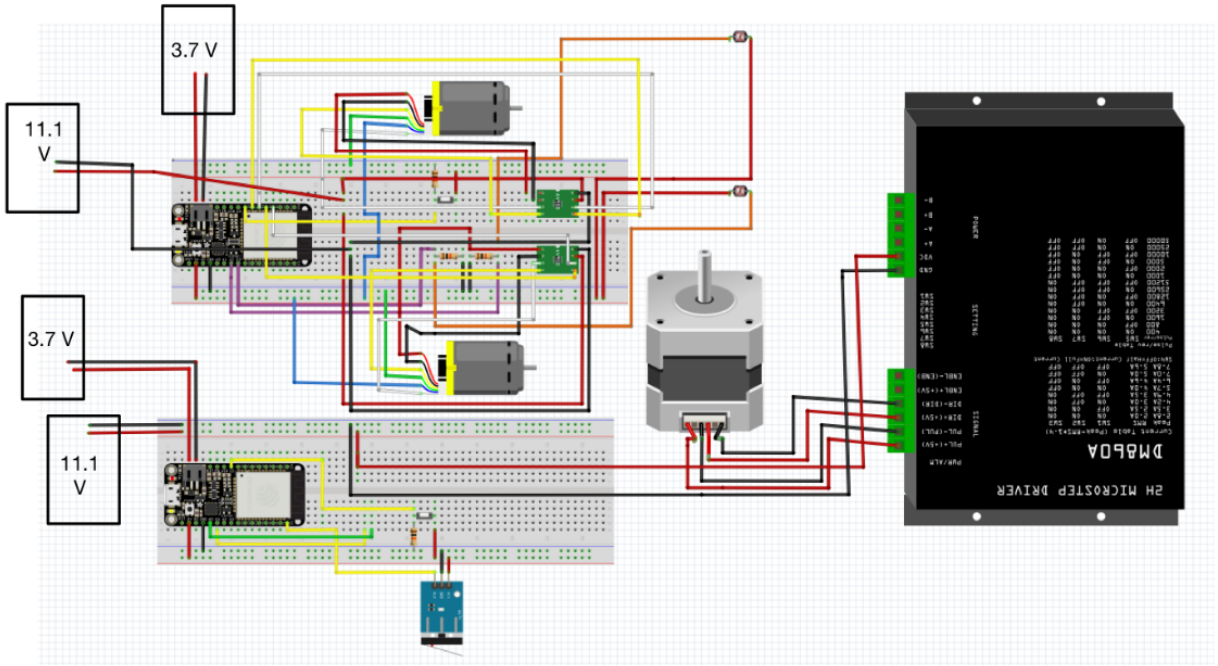
B. Torque = F* r*sin(theta) = 12.2625 * (.1905) * 1 = 2.34 Nm

C. *Decision:* We chose the NEMA 23 motor which supplies 2.88 Nm.

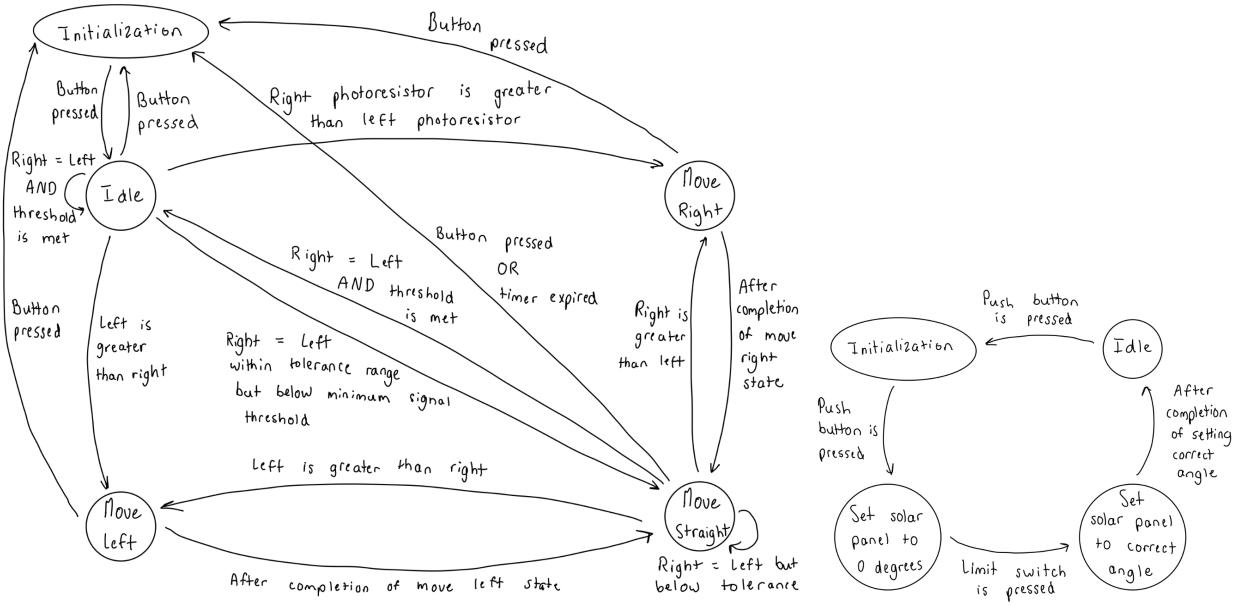
Photos of Integration:



Circuit Diagram:



State Diagram:



Reflection:

This project requires great communication and trust between teammates. We were able to delegate work well to one another and our communication was very good. In the future, we wish we would have started preparing for deliverables sooner, specifically the subsystem functionality check. Additionally, we wish we had looked more into power control for the motors as we had to change out all of our motor drivers in order to make Icarus functional.

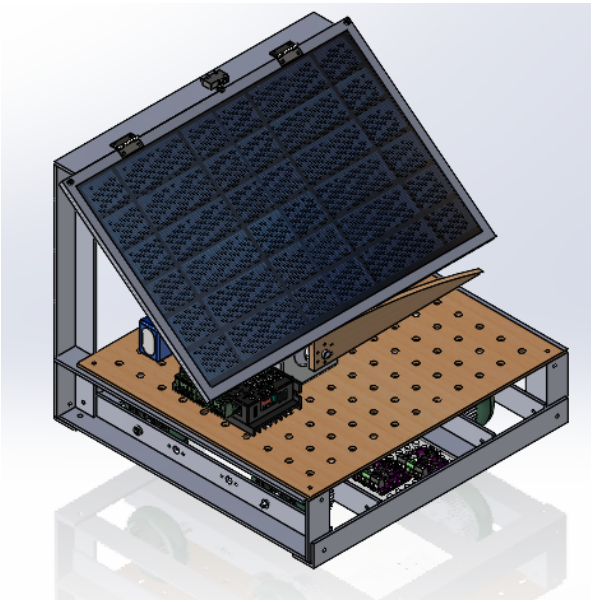
Bill Of Materials:

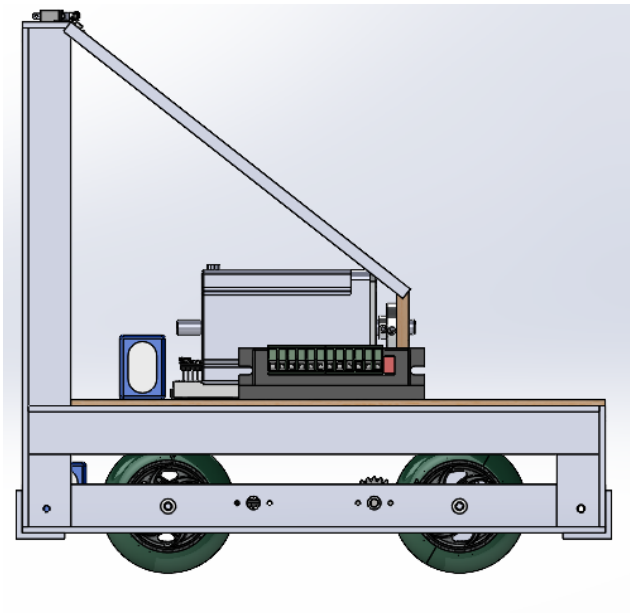
Icarus Intended's Purchase Portfolio					
Item Name	**Description**	**Purchase Justification**	**Price (ea.)**	**Quantity**	**Link to Item**
Pololu Universal Aluminum Mounting Hub for 4mm Shaft, #4-40 Holes	Motor Hub	For translating the motors motion to the gears	$6.95	2	https://www.pololu.com/product/1081
9.7:1 Metal Gear Motor 23Dx63L mm HP 12V with 48 CPR Encoder	Motor	For creating rotational motion to drive the device	$36.95	2	https://www.pololu.com/product/4802
48 in. x 1 in. x 1/8 in. Aluminum Angle Bar	Structure Channel	Material to be manufactured and used as the chassis	$16.67	3	https://www.homedepot.com/p/Everbilt-48-in-x-1-in-x-1-8-in-Aluminum-Angle-Bar-801707/204276130
M3-0.5 x 6 mm Zinc-Plated Plain Metric Socket Cap Screw (3-Piece per Bag)	Screw for motor mount	For securing the chasses	$0.83	4	https://www.homedepot.com/p/Everbilt-M3-0-5-x-10-mm-Plain-Metric-Socket-Cap-Screw-3-Piece-per-Bag-803188/204808024#product-overview
Everbilt #4-40 x 3/4 in. Combo Round Head Zinc Plated Machine Screw (8-Pack)	4-40 screws for mounting motor hub to gear	For securing the motor hub to the gear	$1.28	8	https://www.homedepot.com/p/4-40-x-3-4-in-Combo-Round-Head-Zinc-Plated-Machine-Screw-8-Pack-802961/204274600
10-Watt 12-Volt Monocrystalline Solar Panel	10 W Solar Panel	The bulk of the project -- to collect sunlight to charge a	$37.13	1	https://www.homedepot.com/p/Renogy-10-Watt-12-Volt-Monocrystalline-Solar-Panel-RNG-10D-SS/312531829?source=shoppingads&locale=en-US

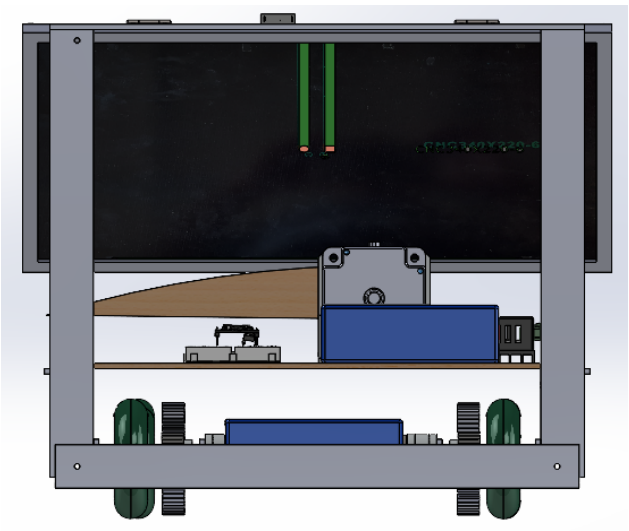
		battery			
STEPPERONLINE Nema 23 Stepper Motor 2.83Nm 4A 8-Wire 6.35mm Dual Shaft CNC Mill Lathe Router	Stepper Motor	To power the arm mechanism to tilt the solar array	$39.50	1	https://www.amazon.com/Stepper-2-83Nm-8-wire-6-35mm-Router/dp/B00Q62MBHQ/ref=pd_sbs_5/147-4300030-5093457?pd_rd_w=yNs60&pf_rd_p=690958f6-2825-419e-9c16-73ffd4055b65&pf_rd_r=NGQJQY0M3GQHMKC4Q4B5&pd_rd_r=66357da9-39df-4bc1-b853-594d9a18086c&pd_rd_wg=sqWwm&pd_rd_i=B00Q62MBHQ&psc=1
Surface-Mount Hinge with Holes, Polished 304 Stainless Steel, Non-removable Pin, 1.5" x 5/8" Leaves	Hinges	Needed to attach to the solar array so that it can tilt	$3.39	2	https://www.acehardware.com/departments/hardware/door-hardware/door-hinges/5005995
GIVEN	ESP32	Microcontrollers	NA	2	
	Stepper Motor Mount	To mount the stepper motor	$ 12.99	1	https://www.amazon.com/Stepper-Motor-Mounting-Bracket-Screws/dp/B073V77VLD/ref=sr_1_3?keywords=stepper+motor+nema+23+mount&qid=1637357155&sr=8-3
	Photoresistors	Signals to tell Icarus which way to drive and when to stop	N/A	2	
	Limit Switch	To zero the solar panel	N/A	1	
	Push Button	To turn on Icarus	N/A	2	
	3.7 V Lithium Ion Battery	To power the ESP32	N/A	2	
	Resistor (10 kOhms)	To pair with the push buttons and limit switch	N/A	4	
	11.1 V Battery	To power the actuators	N/A	2	

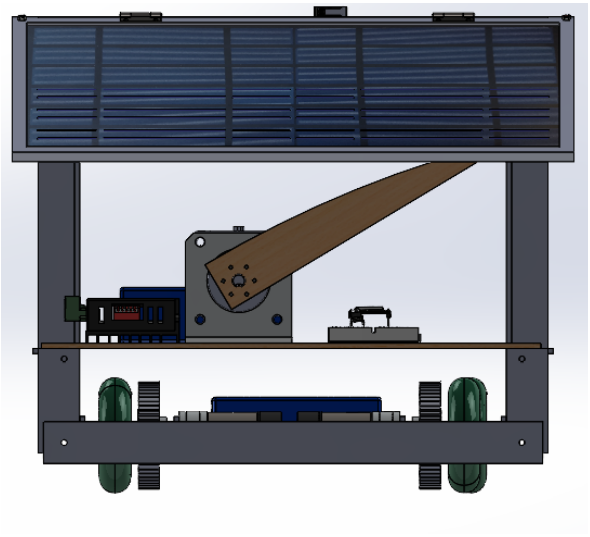
	Microstepper	Motor Driver for Stepper Motor		1	
	Cytron MD13S	Motor Driver for DC Drive Motors		2	

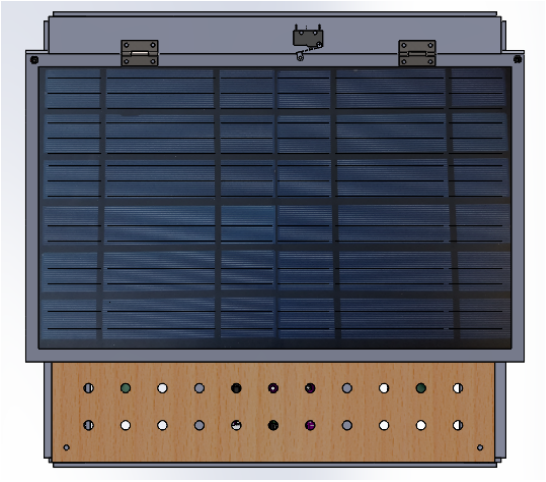
CAD:

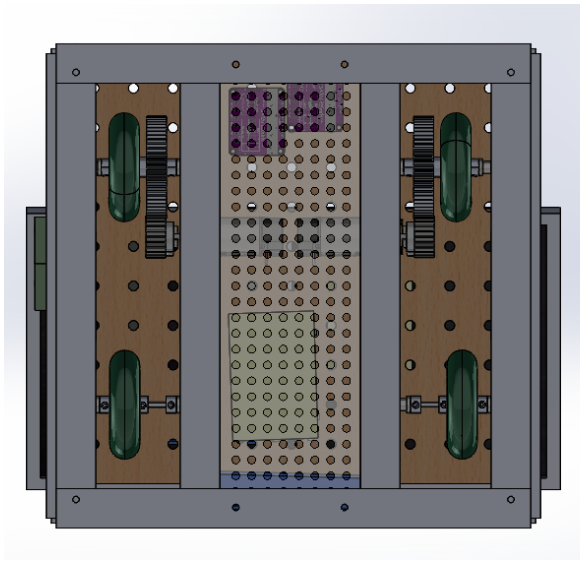


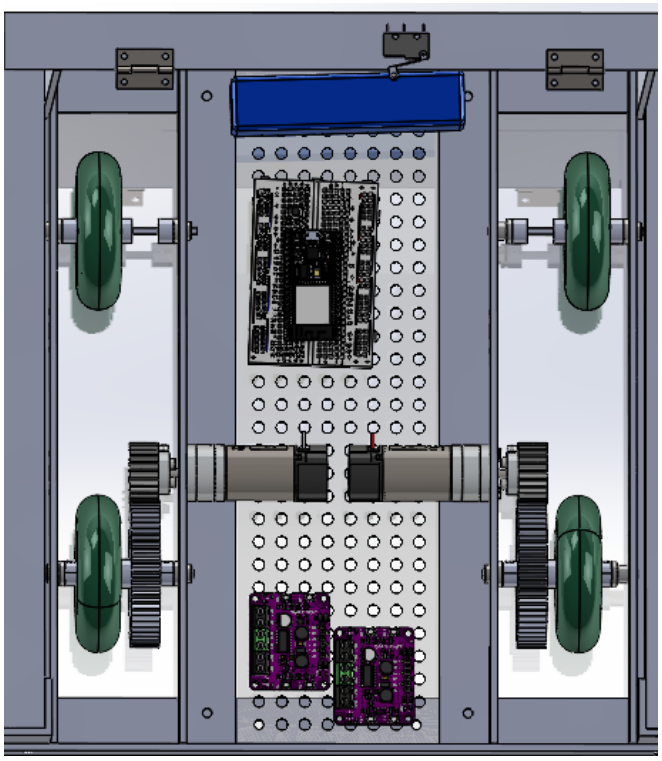


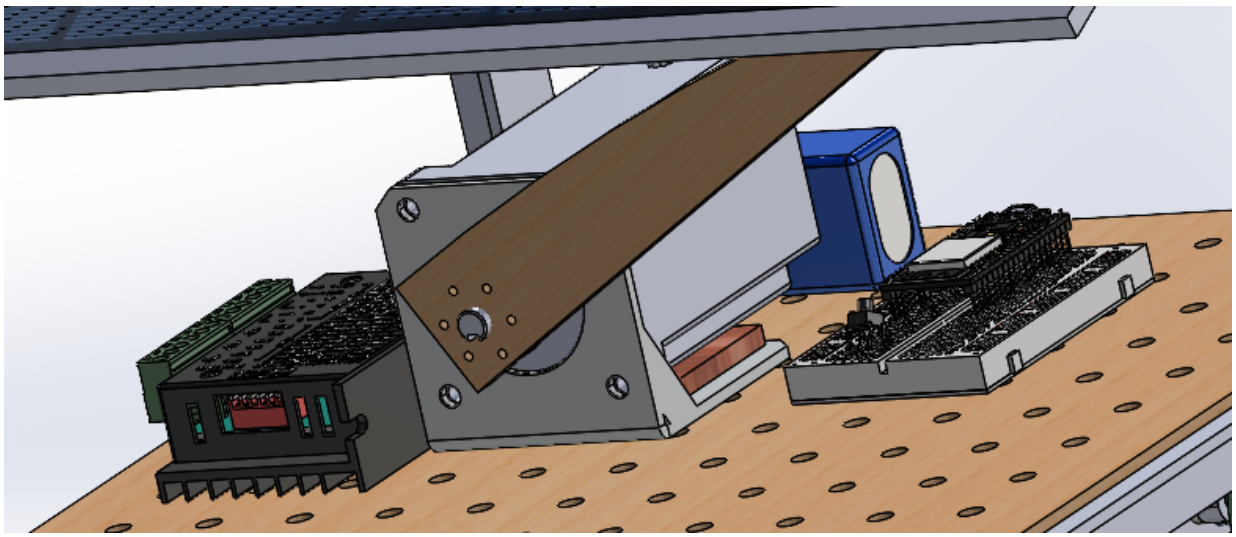












Code:

Upper Level ESP32 Code:

FINAL_102BProjectCodeUpper_V2 §

```
//Upper Code contains cods for the limit switch and stepper motor

//Libraries
#include <Arduino.h>
#include <Wire.h>
#define limitSwitch 21
#define dirPin 33
#define stepPin 27
#define BTN 12
#define ONBOARD_LED  2

double beta;
double tilt = 0;
byte state = 0;
volatile bool limitSwitchIsPressed = false;
volatile bool buttonIsPressed = false;
int switch_state = 0;

void IRAM_ATTR isr() {  // the function to be called when interrupt is triggered
  buttonIsPressed = true;
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(dirPin, OUTPUT);
  pinMode(stepPin, OUTPUT);
  pinMode(limitSwitch, INPUT);
  digitalWrite(dirPin, HIGH);
  pinMode(BTN, INPUT);
  attachInterrupt(BTN, isr, RISING);
  pinMode(ONBOARD_LED, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  limitSwitchIsPressed = digitalRead(limitSwitch);
  switch (state) {
    case 0:
      Serial.println("Platypus");
      if (buttonPressEvent()) {
        state = 1;
        tilt = 0;
        Serial.println("Initialize");
        digitalWrite(ONBOARD_LED, HIGH);
      }
      break;
    case 1: //setting the limit switch
      limitSwitchIsPressed = digitalRead(limitSwitch);
      if (limitSwitchEvent()) {
        Serial.print("Button pressed. Beta = ");
        tilt = 0;
        state = 2;
      }
      break;
    case 2:
      Serial.print("case 1");
      InitializeTilt();
      Serial.print("tilt found: ");
      Serial.println(tilt);
      state = 3;
      break;
    case 3:
      //do nothing
      if (buttonPressEvent()) {
        state = 0;
        TurnOff();
      }
      Serial.print("case 2");
      Serial.println("do nothing");
```

```
      break;
  }
}

void TurnOff() {
  beta = 36 - tilt;
  InitializeTilt_End();

  digitalWrite(dirPin, HIGH);
  digitalWrite(stepPin, LOW);
  Serial.println("turnoff");
}

void InitializeTilt_End() {
  Serial.println(beta);
  tilt = 0;
  if (tilt < beta) {
    digitalWrite(dirPin, LOW);
    while (tilt < beta) {
      // go down
      digitalWrite(stepPin, LOW);
      delayMicroseconds(2000);
      digitalWrite(stepPin, HIGH);
      delayMicroseconds(2000);
      tilt += .035; // 1.8;
      Serial.print(tilt);
      Serial.print("angle up");
    }
  } else {
    digitalWrite(dirPin, HIGH);
    while (tilt > beta) {
      // go up
      digitalWrite(stepPin, HIGH);
      delayMicroseconds(2000);
      digitalWrite(stepPin, LOW);
      delayMicroseconds(2000);
      tilt -= .035; //1.8
      Serial.print("angle down");
    }
  }
}

void InitializeTilt() {
  calculateTilt();
  Serial.println(beta);
  if (tilt < beta) {
    digitalWrite(dirPin, LOW);
    while (tilt < beta) {
      // go down
      digitalWrite(stepPin, LOW);
      delayMicroseconds(2000);
      digitalWrite(stepPin, HIGH);
      delayMicroseconds(2000);
      tilt += .035; // 1.8;
      Serial.print(tilt);
      Serial.print("angle up");
    }
  } else {
    digitalWrite(dirPin, HIGH);
    while (tilt > beta) {
      // go up
      digitalWrite(stepPin, HIGH);
      delayMicroseconds(2000);
      digitalWrite(stepPin, LOW);
      delayMicroseconds(2000);
      tilt -= .035; //1.8
      Serial.print("angle down");
    }
  }
}
```

```
141
142 double calculateTilt() {
143   double location = 37.8715 * (3.14 / 180); //INPUT LOCATON
144   double timeOfDay = 9; //INPUT TIME OF DAY
145   double H;
146
147   double dayOfMonth = 8; //INPUT DAY OF MONTH
148   double monthOfYear = 12; //INPUT MONTH OF YEAR
149   double daysInMonth[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
150   double declinationAngle;
151   H = 15 * (timeOfDay - 12) * (3.14 / 180);
152   double dayOfYear = 0;
153
154   for (int i = 0; i < monthOfYear - 1; i += 1) {
155     dayOfYear += daysInMonth[i];
156   }
157   dayOfYear += dayOfMonth;
158   declinationAngle = (23.45 * sin((360 * (dayOfYear - 81) / 365) * (3.14 / 180))) * (3.14 / 180);
159   beta = (180 * 3.14) * asin((cos(location) * cos(declinationAngle) * cos(H))
160     + (sin(location) *  sin(declinationAngle)));
161 }
162
163 bool limitSwitchEvent() {
164   if (limitSwitchIsPressed == true) {
165     limitSwitchIsPressed = false;
166     delayMicroseconds(2000);
167     digitalWrite(stepPin, HIGH);
168     delayMicroseconds(2000);
169     return true;
170   }
171   else {
172     //should be going up until it reaches correct tilt angle
173     digitalWrite(dirPin, HIGH);
174     digitalWrite(stepPin, HIGH);
175     delayMicroseconds(2000);
176     digitalWrite(stepPin, LOW);
177     delayMicroseconds(2000);
178     return false;
179   }
180 }
181
182 bool buttonPressEvent() {
183   if (buttonIsPressed == true) {
184     buttonIsPressed = false;
185     return true;
186   }
187 }
```

Drivetrain ESP32 Code:

FINAL_Post_Prof_Code_v3_debounce

```
//Libraries
#include <Wire.h>

#define BIN_1 14 //pwm_left
#define BIN_2 13 //dir_left
#define AIN_1 25 //pwm_right
#define AIN_2 26 //dir_right
#define BTN 12
#define PHO_R 33
#define PHO_L 27
#define ONBOARD_LED  2

unsigned char state = 0;
int LeftPhotoresistorVal;
int RightPhotoresistorVal;
unsigned long Time;

//for driving motors
// setting PWM properties ----------------------------
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 127;
volatile bool buttonIsPressed = false;
int power = 0;
float Threshold = 6000;

//for timer
//Setup interrupt variables ----------------------------
volatile bool interruptCounter = false;    // check timer interrupt
volatile bool interruptCounter_Button = false;
int totalInterrupts = 0;   // counts the number of triggering of the alarm
hw_timer_t * timer0 = NULL;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

//Initialization -------------------------------------
void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  interruptCounter = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void TimerInterruptInit() {  //The timer simply counts the number of Tic generated by the quartz.
  timer0 = timerBegin(0, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 =
  timerAttachInterrupt(timer0, &onTime0, true);    // sets which function do you want to call whe
  timerAlarmWrite(timer0, 10000000, true);         // sets how many tics will you count to trigge
  timerAlarmEnable(timer0); // Enables timer
}

void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  interruptCounter_Button = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void TimerInterruptInit_Button() {  //The timer simply counts the number of Tic generated by the
  timer1 = timerBegin(1, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 =
  timerAttachInterrupt(timer1, &onTime1, true);    // sets which function do you want to call whe
  timerAlarmWrite(timer1, 500000, true);         // sets how many tics will you count to trigger
  timerAlarmEnable(timer1); // Enables timer
}

void IRAM_ATTR isr() {  // the function to be called when interrupt is triggered
  buttonIsPressed = true;
  TimerInterruptInit_Button();
}
```

```
71
72 void setup() {
73   // put your setup code here, to run once:
74   TimerInterruptInit(); //Initialize timer interrupt
75   Serial.begin(9600);
76   pinMode(BTN, INPUT); // configures the specified pin to behave either as an input or an output
77   pinMode(PHO_R, INPUT);
78   pinMode(PHO_L, INPUT);
79   pinMode(ONBOARD_LED, OUTPUT);
80   attachInterrupt(BTN, isr, RISING);
81
82   //FOR DRIVER MOTORS
83   /* configure LED PWM functionalitites */
84   ledcSetup(ledChannel_1, freq, resolution);
85
86   /* attach the channel to the GPIO to be controlled */
87   ledcAttachPin(BIN_1, ledChannel_1);
88   ledcAttachPin(AIN_1, ledChannel_2);
89   pinMode(BIN_2, OUTPUT);
90   pinMode(AIN_2, OUTPUT);
91 }
92
93 //service routines ------------------------------------------------------
94 void MoveLeft() {
95   Serial.println("Turn Left");
96   ledcWrite(ledChannel_2, 255);
97   digitalWrite(AIN_2, LOW);
98   ledcWrite(ledChannel_1, 255);
99   digitalWrite(BIN_2, LOW);
100
101   delay (250); //CHANGE THIS TO THE RIGHT DELAY FOR DESIRED AMOUNT OF TURN
102
103   ledcWrite(ledChannel_1, 0);
104   digitalWrite(BIN_2, LOW);
105   ledcWrite(ledChannel_2, 0);
106   digitalWrite(AIN_2, LOW);
107 }
108
109 void MoveRight() {
110   Serial.println("Turn Right");
111   ledcWrite(ledChannel_1, 255);
112   digitalWrite(BIN_2, HIGH);
113   ledcWrite(ledChannel_2, 255);
114   digitalWrite(AIN_2, HIGH);
115
116   delay(250); //CHANGE THIS TO THE RIGHT DELAY FOR DESIRED AMOUNT OF TURN
117
118   ledcWrite(ledChannel_1, 0);
119   digitalWrite(BIN_2, LOW);
120   ledcWrite(ledChannel_2, 0);
121   digitalWrite(AIN_2, LOW);
122 }
123
124 void Straight() {
125   Serial.println("Moving Straight");
126   ledcWrite(ledChannel_2, 125);
127   digitalWrite(AIN_2, LOW);
128   ledcWrite(ledChannel_1, 125);
129   digitalWrite(BIN_2, HIGH);
130   }
131
132 void Debounce() {
133   timerRestart(timer1);
134   timerStop(timer1);
135   buttonIsPressed = false;
136   interruptCounter_Button = false;
137 }
138
139 void Idle() {
140   Serial.println("Idle - At Rest");
```

```
  ledcWrite(ledChannel_1, 0);
  digitalWrite(BIN_2, LOW);
  ledcWrite(ledChannel_2, 0);
  digitalWrite(AIN_2, LOW);
}

//Switch Case ----------------------------------------------------------------
void loop() {
  // put your main code here, to run repeatedly:
  switch (state) {
    case 0 : // motor is stopped
      Serial.println("Platypus");
      Idle();
      if (buttonPressEvent()) {
        Debounce();
        state = 1;
        timerRestart(timer0);
        Serial.println("Initialize");
        digitalWrite(ONBOARD_LED, HIGH);
      }
      break;
    case 1:
      //Idle State
      Idle();
      Serial.println(analogRead(PHO_L));
      Serial.println(analogRead(PHO_R));
      LeftPhotoresistorVal = analogRead(PHO_L);
      RightPhotoresistorVal = analogRead(PHO_R);
      if (buttonPressEvent()) {
        Debounce();
        state = 0;
      } else if ( LeftPhotoresistorGreater()) {
        state = 2; //move left
        timerRestart(timer0);
      } else if (RightPhotoresistorGreater()) {
        state = 3; //move right
        timerRestart(timer0);
      } else if (PhotoResistorsEqual() && PhotoThresholdNotMet()) {
        state = 4; //move straight
        timerRestart(timer0);
      } else {
        Serial.println("Stay in State Idle");
        delay(1000);
        state = 1;
      }
      break;
    case 2: //move left
      if (buttonPressEvent()) {
        Debounce();
        state = 0;
      }
      MoveLeft();
      state = 4;
      break;
    case 3: //move right
      if (buttonPressEvent()) {
        Debounce();
        state = 0;
      }
      MoveRight();
      state = 4;
      break;
    case 4: //move straight
      Straight();
      if (buttonPressEvent()) {
        Debounce();
        state = 0;
      }
      else if (TimerExpired()) {
        state = 0;
```

```
        Serial.println("Timer Works");
        timerStop(timer0);
      }
      else if (LeftPhotoresistorGreater()) {
        state = 2; //move left
      } else if (RightPhotoresistorGreater()) {
        state = 3; //move right
      } else if (PhotoResistorsEqual() && PhotoThresholdNotMet()) {
        state = 4;
      } else {
        state = 1;
      }
  }
}


//event checkers -----------------------------------------------------------
bool buttonPressEvent() {
  if (buttonIsPressed == true) {
    if (interruptCounter_Button) {
      return true;
    }
  }
  return false;
}

bool LeftPhotoresistorGreater() {
  LeftPhotoresistorVal = analogRead(PHO_L); //whatever pin we're using
  RightPhotoresistorVal = analogRead(PHO_R);
  if (LeftPhotoresistorVal - RightPhotoresistorVal > 500 ) {
    return true;
  }
  return false;
}

bool RightPhotoresistorGreater() {
  LeftPhotoresistorVal = analogRead(PHO_L);
  RightPhotoresistorVal = analogRead(PHO_R);
  if (RightPhotoresistorVal - LeftPhotoresistorVal > 500) {
    return true;
  }
  return false;
}

bool PhotoResistorsEqual() {
  LeftPhotoresistorVal = analogRead(PHO_L);
  RightPhotoresistorVal = analogRead(PHO_R);
  if (abs(RightPhotoresistorVal - LeftPhotoresistorVal) < 500) {
    return true;
  }
  return false;
}

bool TimerExpired() {
  if (interruptCounter) { // intteruptCounter will be 'true' when timer interrupt is triggered
    portENTER_CRITICAL(&timerMux0);
    interruptCounter = false;
    portEXIT_CRITICAL(&timerMux0);
    return true; //IF timer is expired, we want this to return false
  }
  return false;
}

bool PhotoThresholdNotMet() {
  LeftPhotoresistorVal = analogRead(PHO_L);
  RightPhotoresistorVal = analogRead(PHO_R);
  if (LeftPhotoresistorVal + RightPhotoresistorVal < 6500) {
    return true;
  }
}
```