Collin Nilsen
Varun Rao
Matthew Walker

ME 102B Final Project Report/Manual: Robotic Cat Toy

1. Opportunity

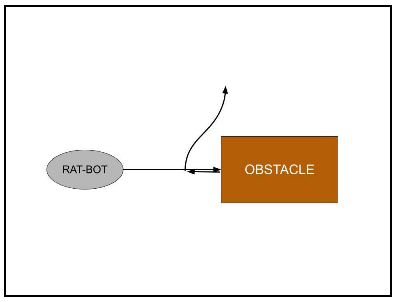
The opportunity that our product hoped to address is how we can keep our pet cats entertained and happy when we are away and they are alone. Now more than ever, time is money, and we just can't afford to take as much time to relax and take care of our pets as we want to. The idea behind our product was to create a robot that can interact with a cat and "play" with it, keeping it in a good mood and rewarding it for participating. The machine we built will be able to run away from a cat, detecting obstacles and moving out of the way to avoid collisions. When caught, the robot will drop a treat, rewarding the cat for good behavior and for winning the game. There are a few similar products that exist through a quick search on Amazon, however none of these products seem to hit all the key points we would like to. For example, some cat toys are remote controlled and thus still require human intervention, whereas other cat toys run on their own but don't drop treats to reward the cat.

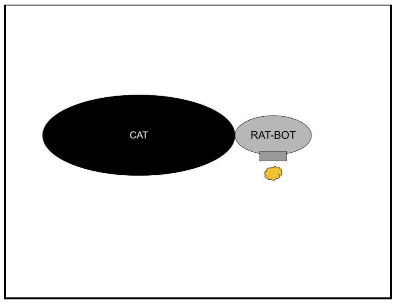
2. High Level Strategy

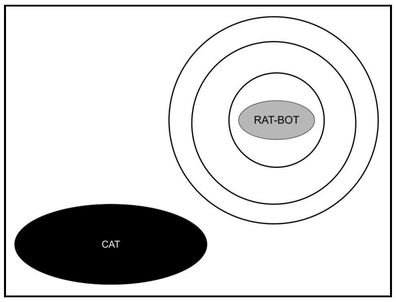
When turned on, the robot can drive around a room on two drive wheels and a third caster wheel, navigating obstacles. We would have liked to implement collision avoidance, but we found this quite difficult given our limitations, and were instead able to implement collision detection and rerouting such that the robot would not hit an obstacle after it has hit it the first time. While we initially assumed a speed of 3mph, we measured it move across an 8ft pool table in 2 seconds, resulting in a speed of about 2.7mph.

In order to distinguish between a collision and being "caught" by a cat, the robot uses a video game joystick, with two analog and one digital input, attached to a 3-D printed plastic dome that contains all the necessary parts of the robot. When the dome is bumped, the joystick is moved and the direction from which the impact came is known to the robot. When the button is clicked, the robot knows that it has been caught and activates a servo motor to move a trap door that drops a single treat to the cat. In order to prevent the cat from abusing the robot to get extra treats, the robot will go into a sleep state when it is caught, which does not allow the cat to "catch" it again until a certain time has passed.

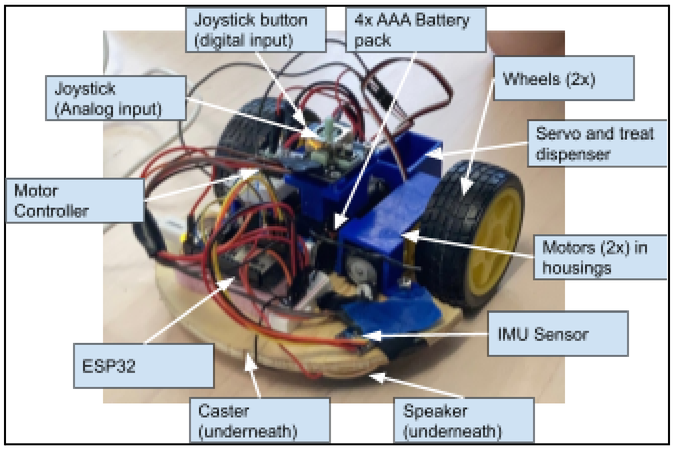
Finally, in the event that the robot is not caught by the cat in a certain amount of time, it will stop moving and play a noise in order to alert the cat to its location. If the cat does not find the robot, it will go to sleep.





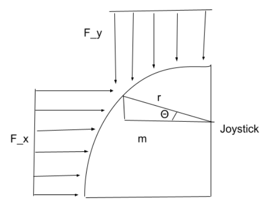


3. Integrated Physical Device + Subsystem Labels



4. Function-critical Decisions and Calculations

- Joystick calculations and observation:
    - The dome has about 20mm clearance in each horizontal direction. This means, if it hits an obstacle hard at an angle in any direction, the joystick will not shear off, as its total angular movement will be limited by the dome hitting the wooden base, not by the joystick itself bottoming out.
    - We want to find at what angle the joystick button will activate if a horizontal load is applied. This is complex because our “dome” isn’t a perfect hemisphere. We find that if a vertical force is applied around a 5cm diameter circle on top of the dome, the button will activate. Outside of that, the joystick will move.



T = Torque required to move joystick = 100g*2cm = 200gcm (measured with scale)

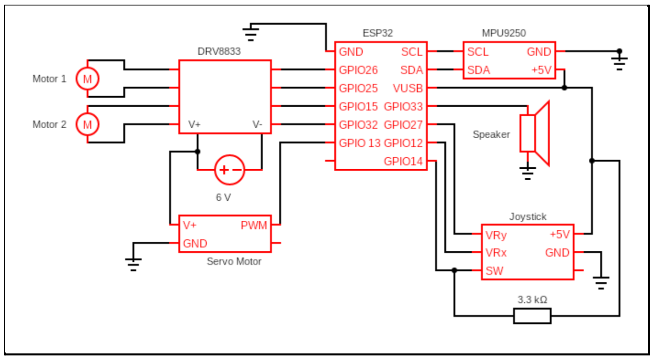
F_y (force to press button) = 761g (raw force measured with scale) - 100g (weight of dome) = 661g

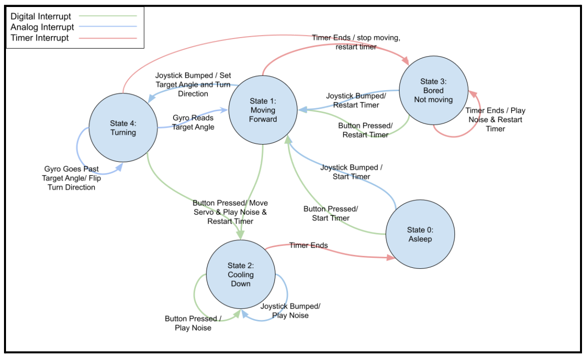
If F_y > 661g and T < 200gcm, button will be pressed. Otherwise, joystick will move.

    -
- Bearing calculations and observations:
    - Front casters are rated to 8kg, as they are meant for mounting underneath garbage cans
    - The caster consists of a steel bearing 10mm in diameter supported by 8 bearings 2mm in diameter rolling in a plastic cup.
    - Limiting factor is the plastic bearing cup, as it is less hard than the steel bearings, which would lead to pitting and poor rolling performance over time.
    - Even if a cat weighing 4 kg were to pounce on it, the bearings wouldn’t fail.
- Motor calculations:
    - Total Weight = 1kg
    - Wheel radius = 3.5cm
    - Approximate desired speed = 6km/hr = 1.66 m/s
    - Acceleration desired - get to top speed in 5 sec = 0.33m/s^2
    - T = F*a = 1kg * 3.5cm * 0.33 m/s^2 = 1162g*cm

  - Motor spec from P3: 800g*cm at 3V (We will be running the motors on 4xAAA batteries which will be 6V. This is within spec for the motors.
  - Using 2 motors will output 1600g*cm torque, giving ~1.37 FS
- Additional design considerations:
  - All holes in CAD were M3 to simplify hardware. Screws were used to attach motor mounts to the wood base and to attach joystick to joystick mount.
  - Wood base was cut with a Dremel router. Having a wood base allowed for faster and easier modifications than a 3D printed base.
  - The dome was designed with its center of mass to be exactly at the pivot, so that it would default to a resting position that would not activate the joystick. We set a threshold joystick value for the interrupt so the dome swaying slightly during acceleration would not set off the joystick interrupt.

5. Updated Circuit and State-transition Diagrams




Digital Interrupt
Analog Interrupt
Timer Interrupt
Timer Ends / stop moving, restart timer
Joystick Bumped / Set Target Angle and Turn Direction
State 4: Turning
State 1: Moving Forward
State 3: Bored Not moving
Gyro Reads Target Angle
Joystick Bumped/ Restart Timer
Button Pressed/ Restart Timer
Timer Ends / Play Noise & Restart Timer
Gyro Goes Past Target Angle/ Flip Turn Direction
Joystick Bumped / Start Timer
Button Pressed/ Move Servo & Play Noise & Restart Timer
Button Pressed/ Start Timer
State 0: Asleep
Timer Ends
State 2: Cooling Down
Joystick Bumped/ Play Noise
Button Pressed / Play Noise

6. Reflection

For us, a strategy that we would not have been able to complete our project without was having access to a 3-D printer. This allowed us to custom-fabricate several necessary parts on our project that were lighter and less labor-intensive to make than if we had machined them in the machine shop.

Another strategy that also worked well for us was learning how to solder, which was very useful in the final prototype because it allowed for smaller, more organized interior systems which fit under the shell. We tried to make our project as compact as possible, with the fewest extraneous wires. Rather than having multiple wires dangling around, we tried cutting each wire to the smallest possible length and soldering them directly, so they wouldn't come dislodged when going over a bump. This allowed for a robust construction that we (or a cat) could manhandle. During the project showcase, we encouraged the Dean of the College of Engineering to kick our robot, showing that it would keep working under a variety of loading conditions.

The main thing that we wish we would have done differently was work in a more stable way. Creating an organized timeline of design, manufacturing and testing during the first few weeks of the project would have greatly simplified our workflow. The final thing we wish we had done differently, which we greatly encourage other students to do, is to stay away from the IMU. We had so many problems with the MPU-9250 included in the kit because they were extremely easy to break, and when they were working, they were unreliable at best. We went through 4 IMU units in testing.

Appendix

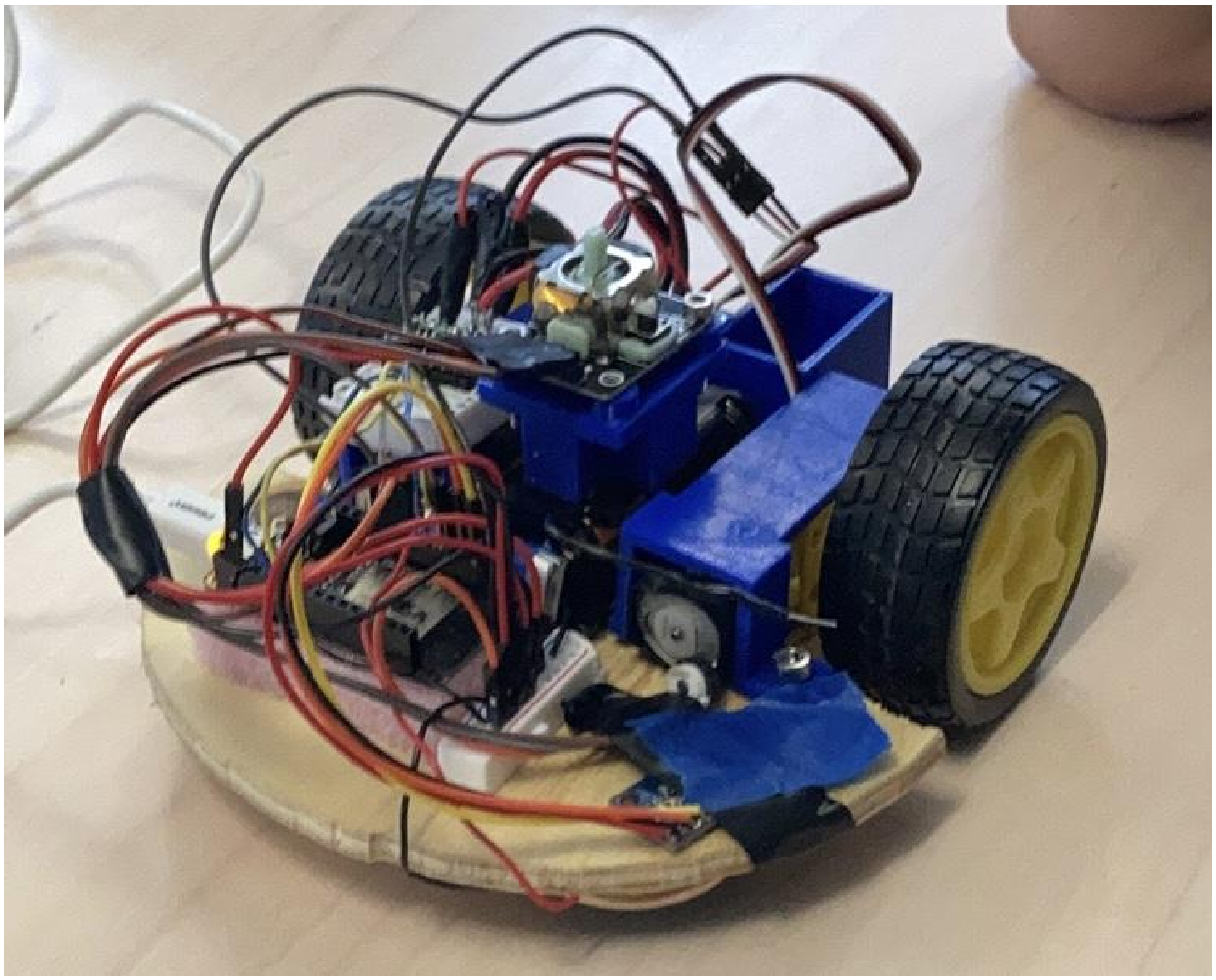
A. Complete Bill of Materials
B. Final Product Images
C. CAD Images
D. Event-Driven Programming Code Screenshots

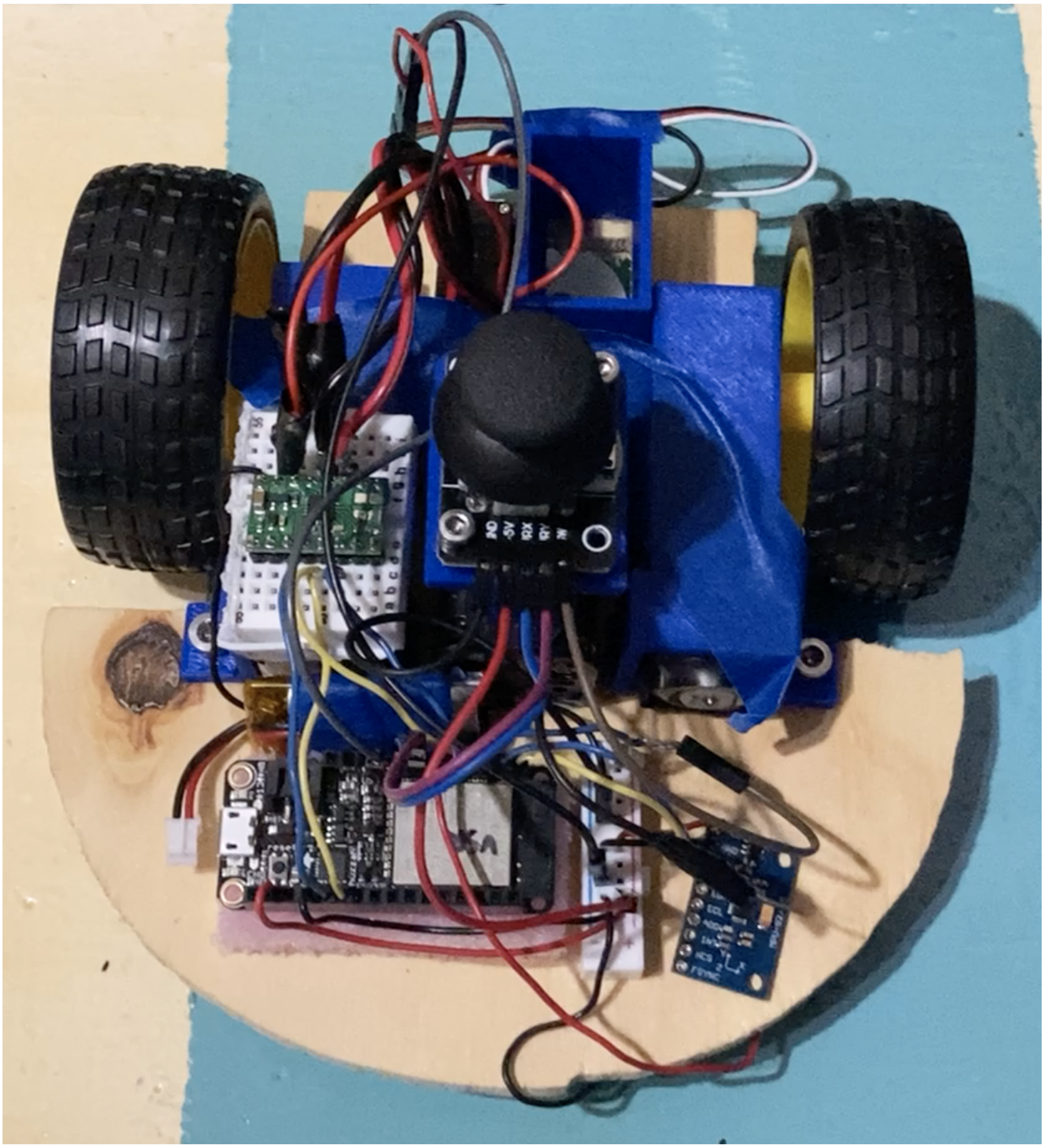
A. Bill of Materials

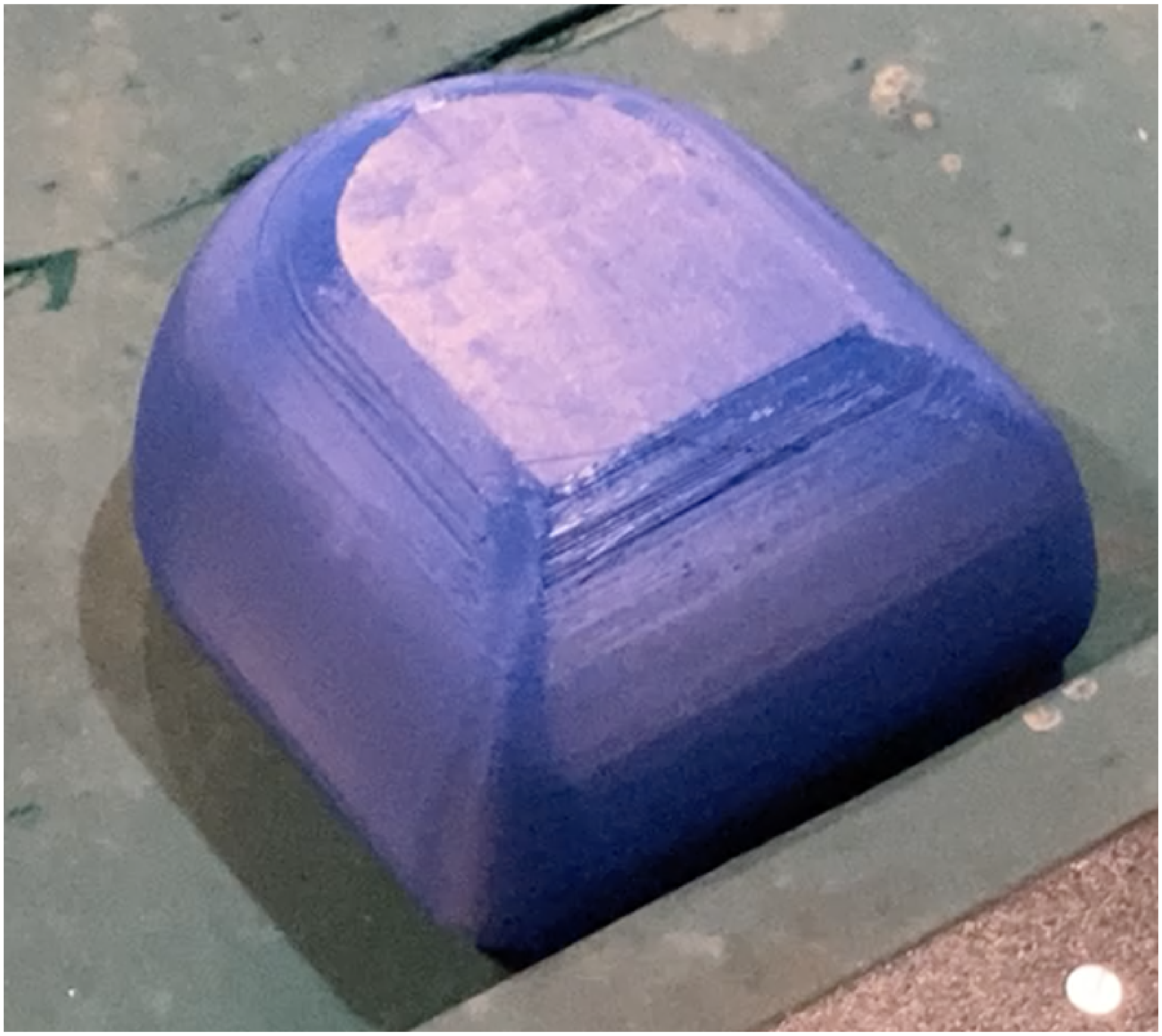
Item	Qty	Cost	Listing name/Source Link	CAD Link?
ESP32	1	Kit	HUZZAH32 Feather Microcontroller	link
Speaker	1	Kit	Mini Loudspeaker	28 x 5 mm circle (diameter x thickness)
Joystick	1	$10 for 6	PS2 Joystick game controller	Link
Motor Controller	1		DRV8833 Dual Motor Driver Carrier	
Motor	2	Already have	Perseids Chassis Encoder Wheels Battery	Wheel: (6 cm (diameter) x 2.7 cm (thickness)
Wheel	2	Already have		
Battery/ Battery Pack	1	Already have		
Encoder	2	Already have		
Caster Wheel	1	$6.49	Self Adhesive Caster Wheel	1.1 in x 1.8 inch baseplate, 0.55 inch from bottom of baseplate to tip of ball
MPU9250 Gyro Sensor	1	Kit	IMU	
Metric Hardware	many	$20	Metric Screw Set	
Servo		Got in Lab	Fitec FS90 Servo	

Wire and Solder	N/A	Got in Lab		
Custom Fabricated Parts				
Base	1	$2	Cut with a router from ¼ in plywood	
Dome	1	$3	3D printed	
Treat holder	1	$1	3D printed	
Motor Mount	2	$1	3D printed	
Joystick Mount	1	$1	3D printed	
Servo arm	1	$1	3D printed	

<u>B. Final Product Images</u>

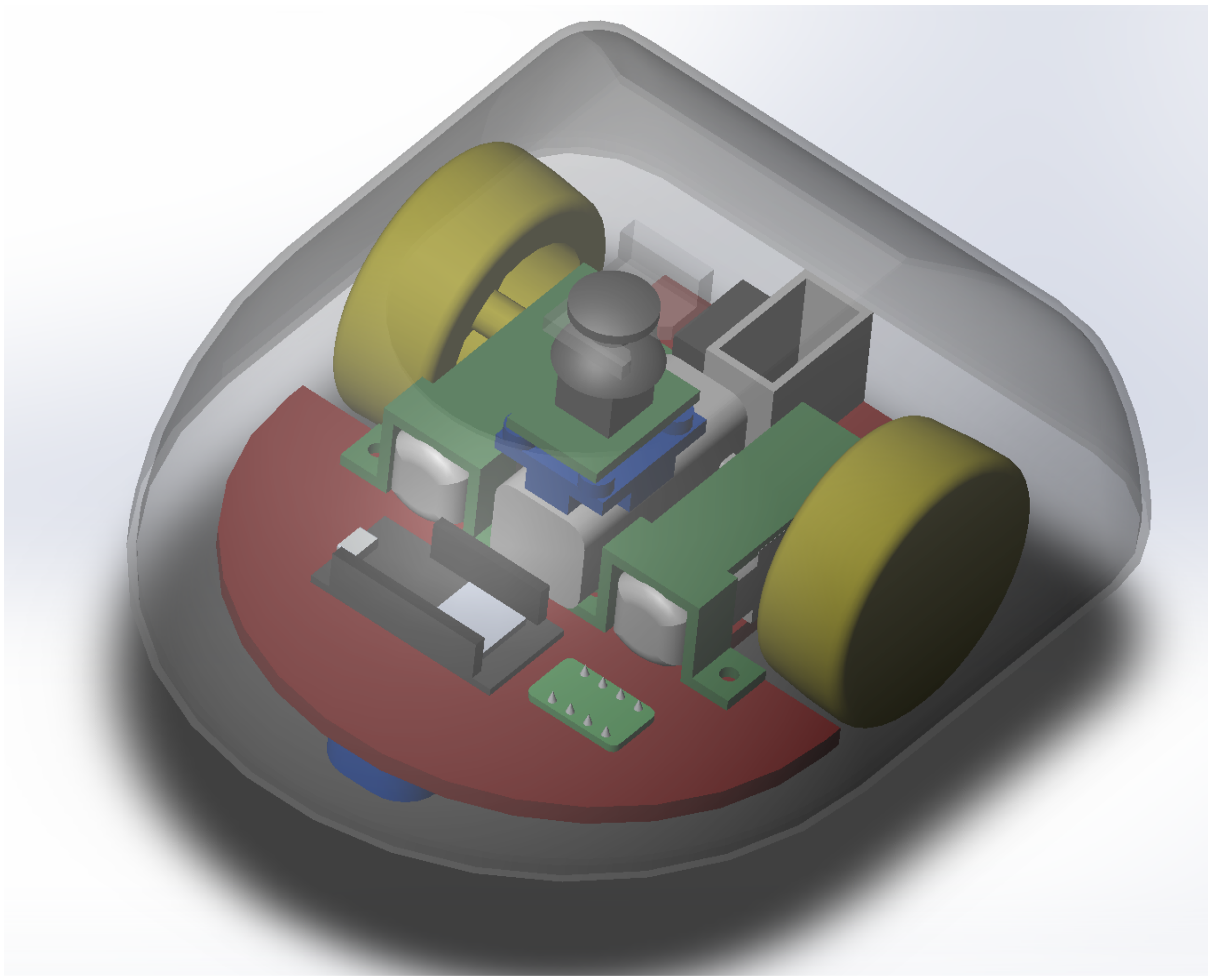




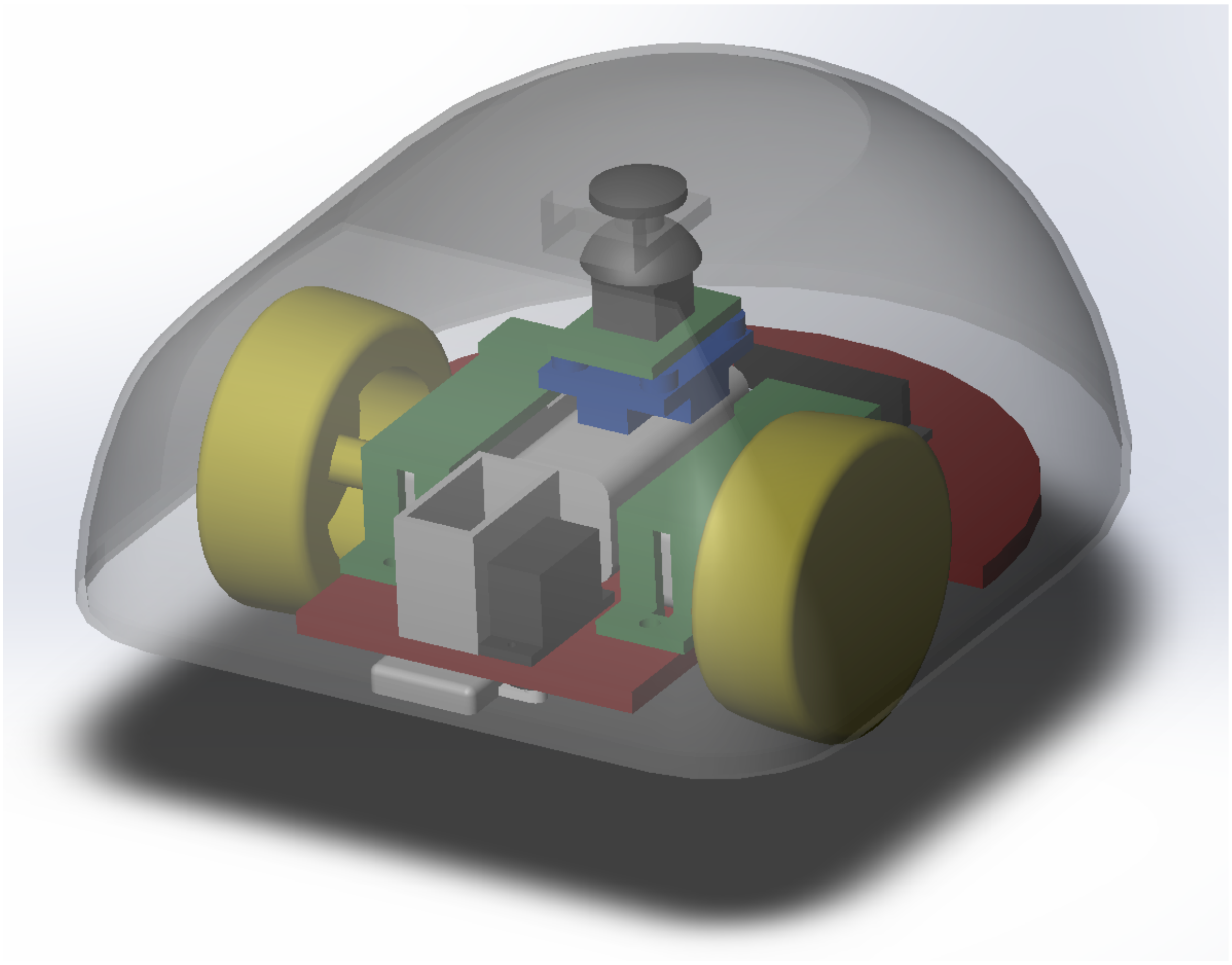


C. CAD Images

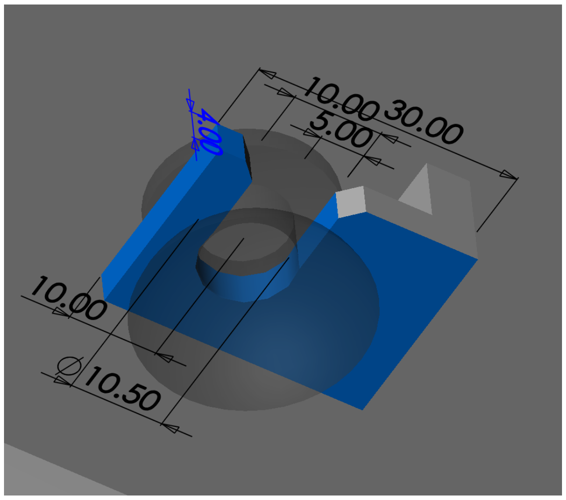
Full Assembly Front:



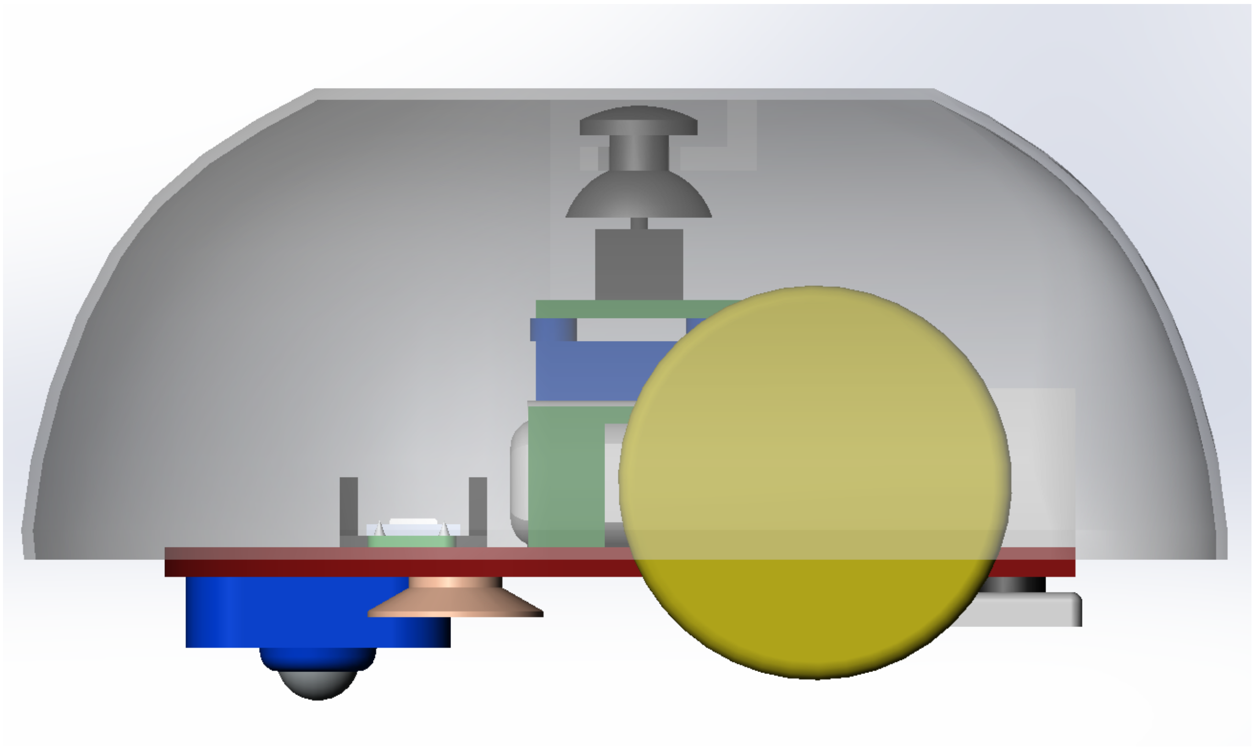
Full Assembly Rear:



Updated mechanical connection between joystick and dome. The new mount is a secure press fit that reassuringly snaps into place. The dimensions of the snap fit are shown in the image below.



Updated dome shape. Created with lofts instead of extrude+fillet. Shallower angle at the base allows for better pivoting around the joystick when hitting obstacles at low heights.



D. Code

```
#include <math.h>
#include <Arduino.h>
#include <ESP32Servo.h>
#include <MPU9250_asukiaaa.h>

//motors
#define M1R 26
#define M1L 25
#define M2R 15
#define M2L 32
#define SERV 13
//joystick
#define JOYX 12
#define JOYY 27
#define JOYB 14
//gyro
#define SDA_PIN 23
#define SCL_PIN 22
//speaker
#define SPKR 33

MPU9250_asukiaaa mySensor;
Servo myservo;  // create servo object to control a servo

/////////////////////////////////////////////////////////////////////////////////VA
RIABLES/////////////////////////////////////////////////
int state = 0;

int turningThreshold = 10;
float gyroAngle = 0;
float gyroAngle0 = 0;
int targetAngle = 0;
//motor variables
const int freq = 5000;
const int M1R_Channel_1 = 2;
const int M1L_Channel_2 = 4;
const int M2R_Channel_3 = 6;
const int M2L_Channel_4 = 8;
const int Speaker_Channel = 10;
const int resolution = 8;
int MAX_PWM_VOLTAGE = 255;
int m = 0;
//servo variables
int pos = 0;    // variable to store the servo position
//joystick variables
double joyx = 0;
double joyy = 0;
double joyx0 = 0;
double joyy0 = 0;
int joyb = 0;
bool buttonWasPressed = false;
double tiltAngle = 0;
double joyAngle = 0;
double joyAngle0 = 0;
```

```
bool tilted = false;

//timer variables
bool timerInter = false;    // check timer interrupt
hw_timer_t * timer0 = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;



///////////////////////////////////////////////////////////////////////SETUP//////
////////////////////////////////////////////////////////////
//timer setup
void IRAM_ATTR timerTrigger() {
  portENTER_CRITICAL_ISR(&timerMux);
  timerInter = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux);
}
void startTimer(int secs) {  //The timer simply counts the number of Tic generated by
the quartz. With a quartz clocked at 80MHz, we will have 80,000,000 Tics.
  timer0 = timerBegin(0, 80, true); // divides the frequency by the prescaler:
80,000,000 / 80,000,000 = 1,000 tics / sec
  timerAttachInterrupt(timer0, &timerTrigger, true);    // sets which function do you
want to call when the interrupt is triggered
  timerAlarmWrite(timer0, secs, true);         // sets how many tics will you count to
trigger the interrupt
  timerAlarmEnable(timer0); // Enables timer
  Serial.println("timer started");
}
//button setup
void IRAM_ATTR buttonTriggered() {  // the function to be called when button interrupt
is triggered
  buttonWasPressed = true;
}

void setup() {  // put your setup code here, to run once:
  //Joystick and button inits
  pinMode(JOYB, INPUT); // configures the specified pin to behave either as an input
or an output
  pinMode(JOYX, INPUT);
  pinMode(JOYY, INPUT);
  joyx0 = analogRead(JOYX) - 2048;
  joyy0 = analogRead(JOYY) - 2048 ;
  attachInterrupt(JOYB, buttonTriggered, RISING);
  //servo setup
  myservo.setPeriodHertz(50);    // standard 50 hz servo
  myservo.attach(SERV, 500, 2400); // attaches servo to the servo object using default
min/max of 1000us and 2000us


  //motor setup
  ledcSetup(M1R_Channel_1, freq, resolution);
  ledcSetup(M1L_Channel_2, freq, resolution);
  ledcSetup(M2R_Channel_3, freq, resolution);
  ledcSetup(M2L_Channel_4, freq, resolution);
  ledcAttachPin(M1R, M1R_Channel_1);
  ledcAttachPin(M1L, M1L_Channel_2);
  ledcAttachPin(M2R, M2R_Channel_3);
  ledcAttachPin(M2L, M2L_Channel_4);
  setMotorPower(0, 0); //Stop both motors

  //gyro setup
  Wire.begin(SDA_PIN, SCL_PIN); //sda, scl
  mySensor.setWire(&Wire);
  mySensor.beginMag();
  mySensor.magUpdate();
  Serial.begin(115200);
  //speaker setup
  ledcSetup(Speaker_Channel, 1E5, 12);
  ledcAttachPin(SPKR, Speaker_Channel);
}

void loop() {
///////////////////////////////////////////////////////////////////////MAIN/////////
//////////////////////////////////////////////////////////////
  // put your main code here, to run repeatedly:
  if (buttonWasPressed) {
    buttonInterrupt();
  }
  updateJoyVals(); //Calculate joystick angle
  if (tiltAngle < 200) {
    tilted = false;
  } else if (tiltAngle >= 200 && tilted == false) { //200 arbitrary threshold
    JoystickInterrupt2();
  } else {

  }

  if (timerInter) { // intteruptCounter will be 'true' when timer interrupt is
triggered (every second)
    portENTER_CRITICAL(&timerMux);
    timerInter = false;
    portEXIT_CRITICAL(&timerMux);
    TimerInterrupt();
  }


  switch (state) {
    case 0: //Ready
      setMotorPower(0, 0); //Stop both motors
      break;
    case 1: //Moving straight
      setMotorPower(MAX_PWM_VOLTAGE, 0); // Move forward 100 % speed
      break;
    case 2: //cooldown
      setMotorPower(0, 0); //Stop both motors
      break;
    case 3: //bored
      setMotorPower(0, 0); //Stop both motors
      break;
    case 4:
      state4();
      break;
  }

}
////////////////////////////////////////////////////////////////////////FUNCTIONS/////
///////////////////////////////////////////////////////
void updateJoyVals() {
  joyx = analogRead(JOYX) - 2048 - joyx0;
  joyy = analogRead(JOYY) - 2048 - joyy0;
  joyb = digitalRead(JOYB);
  joyAngle = (atan2(joyy, joyx) * 180 / 3.14);
  tiltAngle = sqrt(sq(joyx) + sq(joyy));
}

void updateGyroVals() {
  mySensor.magUpdate();
  gyroAngle = 2 * (mySensor.magHorizDirection() - gyroAngle0);

  if (gyroAngle > 180) {
    gyroAngle = gyroAngle - 360;
  } else if (gyroAngle < -180) {
    gyroAngle = gyroAngle + 360;
  }
}
void fixTargetangle() {
  if (targetAngle >= 180) {
    targetAngle = targetAngle - 360;
  } else if (gyroAngle <= -180) {
    targetAngle = targetAngle + 360;
  }
  Serial.print("targetAngle:");
  Serial.println(targetAngle);
}
//////////////////////////////////////////////////////////////////////////ACTIONS/////
//////////////////////////////////////////////////////////
void playSound() { //play speaker
  ledcWriteTone(Speaker_Channel, 1600);
  delay(700);
  ledcWriteTone(Speaker_Channel, 0);
  Serial.println("play sound");
}
void moveServo() {
  Serial.println("move servo");
  myservo.write(45);
  delay(1000);
  myservo.write(135);
}

void setMotorPower(int power, int direct) {

  switch (direct) {
    case 0:
      ledcWrite(M1R_Channel_1, 0);
      ledcWrite(M1L_Channel_2, power);
      ledcWrite(M2R_Channel_3, 0);
      ledcWrite(M2L_Channel_4, power);
      break;
    case 1:
      ledcWrite(M1R_Channel_1, power);
      ledcWrite(M1L_Channel_2, 0);
      ledcWrite(M2R_Channel_3, power);
      ledcWrite(M2L_Channel_4, 0);
      break;
    case 2: //turn left
      ledcWrite(M1R_Channel_1, 0);
      ledcWrite(M1L_Channel_2, power);
      ledcWrite(M2R_Channel_3, power);
      ledcWrite(M2L_Channel_4, 0);
      break;
    case 3://turn right
      ledcWrite(M1R_Channel_1, power);
      ledcWrite(M1L_Channel_2, 0);
      ledcWrite(M2R_Channel_3, 0);
      ledcWrite(M2L_Channel_4, power);
      break;
  }
}
////////////////////////////////////////////////////////////////////////INTERRUPTS/////
//////////////////////////////////////////////////////


void JoystickInterrupt2() {
  switch (state) {
    case 0:
      startTimer(30000000); //Start Bored Timer
      state = 1;
      break;
    case 1:
      updateJoyVals();//Calculate joystick angle
      mySensor.magUpdate();
      gyroAngle0 = mySensor.magHorizDirection();
      Serial.print("joyAngle:");
      Serial.println(joyAngle);
      Serial.print("gyroAngle0:");
      Serial.println(gyroAngle0);
      if (joyAngle >= 0) { // hit from right
        targetAngle = joyAngle - 135 ; //Set target angle
      } else if (joyAngle < 0) { //hit from left
        targetAngle = joyAngle + 135; //Set target angle
      }
      fixTargetangle();
      state = 4;
      break;
    case 2:
      playSound(); // same or different from “caught” sound
      break;
    case 3:
      startTimer(30000000); //Start Bored Timer
      state = 1;
      break;
    case 4:
      break;
  }
  tilted = true;
  Serial.print("state ");
  Serial.println(state);
}

void buttonInterrupt() {
  tilted = false;
  Serial.println("button press");
  switch (state) {
    case 0:
      startTimer(30000000);
      state = 1;
      break;
    case 1:
      timerStop(timer0);
      startTimer(10000000); //Start cooldown timer
      setMotorPower(0, 0); //Stop both motors
      playSound();
      moveServo(); //dispense treat
      state = 2;
      break;
    case 2:
      playSound(); // same or different from “caught” sound
      break;
    case 3:
      timerStop(timer0);
      startTimer(30000000);
      state = 1;
      break;
    case 4:
      timerStop(timer0);
      startTimer(10000000); //Start cooldown timer
      setMotorPower(0, 0); //Stop both motors
      playSound();
      //moveServo(); //dispense treat
      state = 2;
      break;
  }
  buttonWasPressed = false;
  Serial.print("state ");
  Serial.println(state);
}

void TimerInterrupt() {
  tilted = false;
  Serial.println("timeint");
  switch (state) {
    case 0:
      break;
    case 1:
      playSound();
      startTimer(30000000); //Restart Timer
      state = 3;
      break;
    case 2:
      timerStop(timer0);
      state = 0;
      break;
    case 3:
      playSound();
      startTimer(30000000); //Restart Timer
      break;
    case 4:
      playSound();
      startTimer(30000000); //Restart Timer
      state = 3;
      break;
      break;
  }

  Serial.print("state ");
  Serial.println(state);
}


void state4() {
  updateGyroVals();
  if (gyroAngle < targetAngle - turningThreshold) {
    setMotorPower(MAX_PWM_VOLTAGE, 2);
    Serial.println("10");
  } else {
    if (gyroAngle > targetAngle + turningThreshold) {
      setMotorPower(254, 3);
      Serial.println("20");
    }  else { //"gyro interrupt"
      state = 1;
      Serial.println("30");
    }
  }
}
```