BALANCING ROBOT

DAVE VE3OOI

JUNE-AUGUST 2022

I've never regretted anything I've done, even the things that I've failed at.

I've often regretted not trying something really big, because you'll never know.

Dean Kamen

EXPAND YOUR HORIZONS. KEEP LEARNING

Background

- I get bored easily
- I can operate a radio only so much
- I can build so many radios
- I tend to tackle complex projects to keep learning.

This is another log of one of my journeys.

- This is specifically for building a self balancing robot and learning about inertial measurement units (IMU)

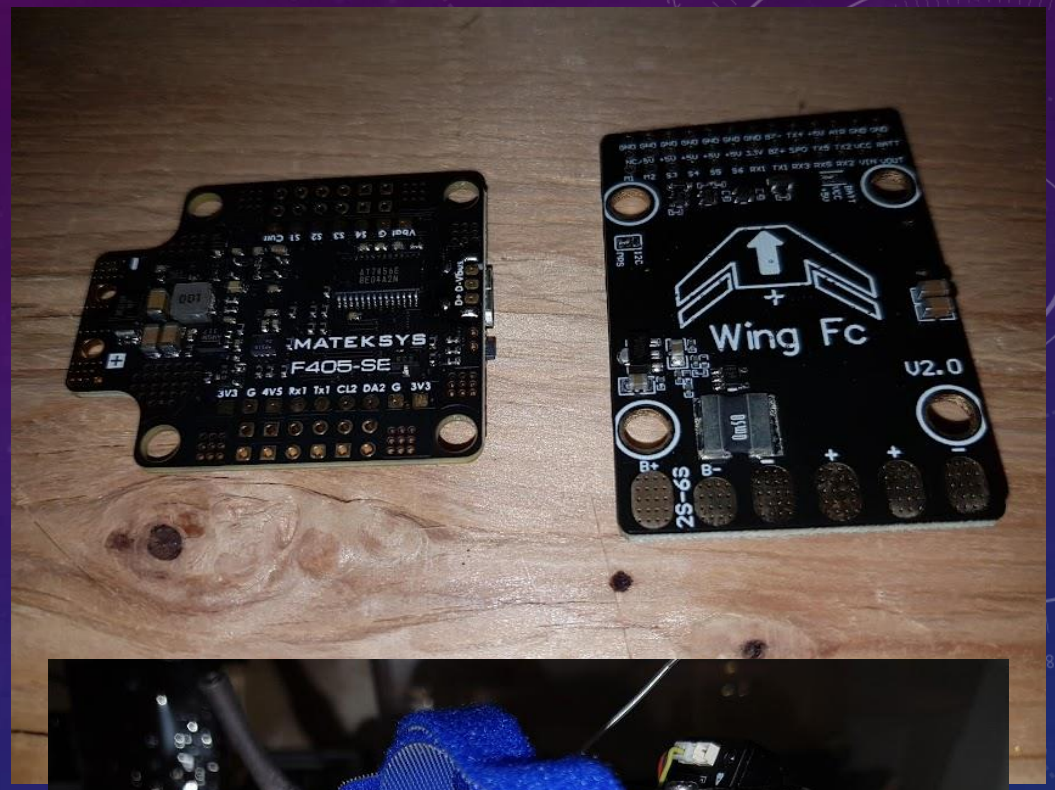
Let my journey encourage you to tackle something out of your comfort zone

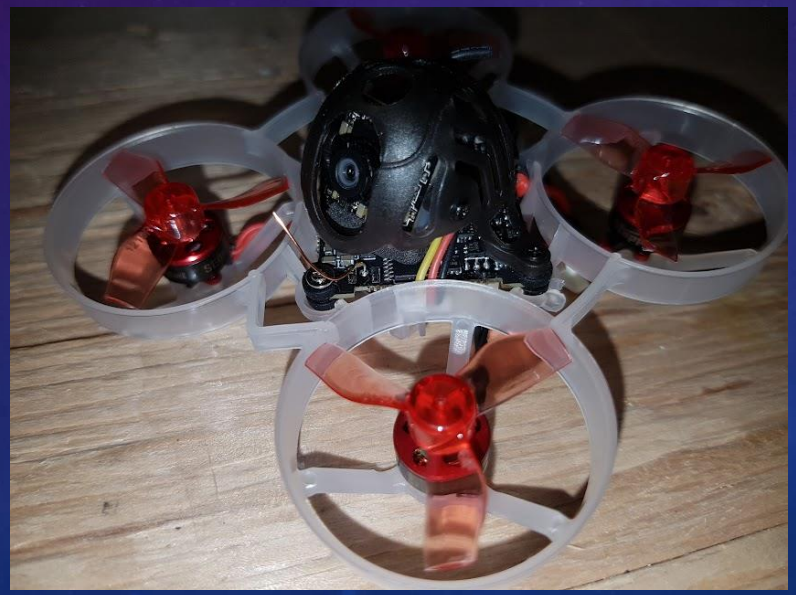
A BRIEF HISTORY…2009

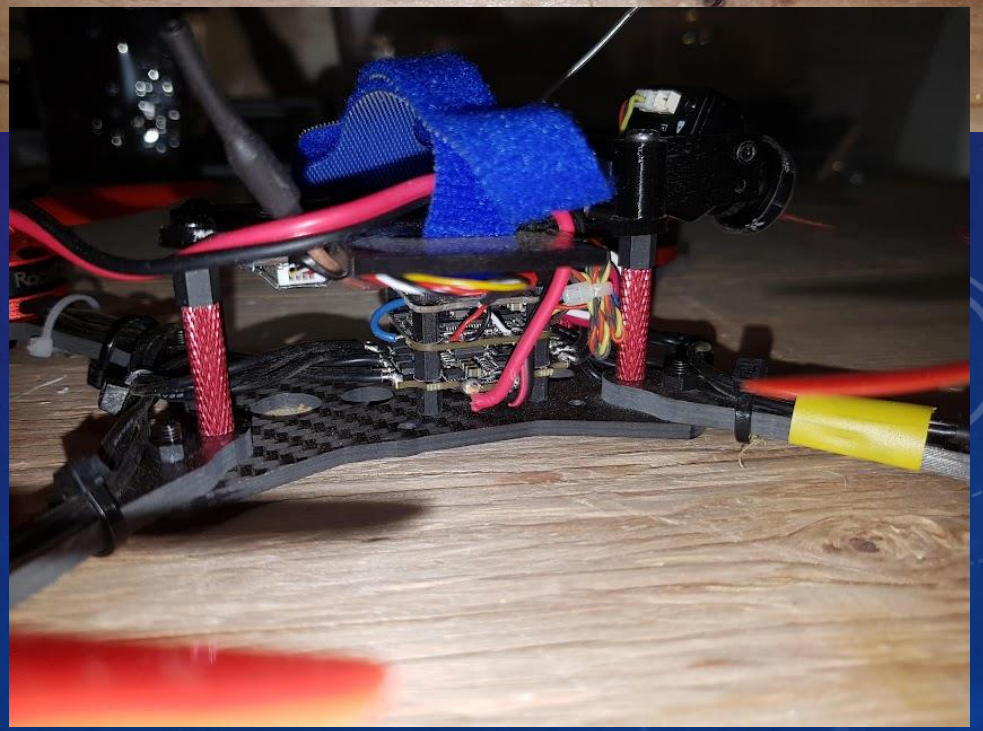
PIC 18F2620 (10 MIPS, 8 MHz, 10 bit ADCs@100 Ksps)

- Backup
- ADC v2.0.c
- ADC v2.0.h
- ADC v2.0.o
- DDR.mcw
- EEMemory v2.0.c
- EEMemory v2.0.h
- EEMemory v2.0.o
- FMS v2 bckup - Dec9.zip
- FMS v2 bckup - 10Feb15.zip
- FMS v2.cof
- FMS v2.hex
- FMS v2.map
- FMS v2.mcp
- FMS v2.mcs
- FMS v2.mcw
- ftoa.c
- ftoa.err
- ftoa.h
- ftoa.o
- GPS v2.0.c
- GPS v2.0.err
- GPS v2.0.h
- GPS v2.0.o
- i2c v2.0.c
- i2c v2.0.err
- i2c v2.0.h
- i2c v2.0.o
- Kalman v2.0.c
- Kalman v2.0.err
- Kalman v2.0.h
- Kalman v2.0.o
- Main System v2.0.c
- Main System v2.0.err
- Main System v2.0.h
- Main System v2.0.o
- PID v2.0.c
- PID v2.0.err
- PID v2.0.h
- PID v2.0.o
- PWM (CCM) v2.0.c
- PWM v1.h
- PWM v2.0.c
- PWM v2.0.h
- PWM v2.0.o
- Timer v2.0.c
- Timer v2.0.err
- Timer v2.0.h
- Timer v2.0.o
- UART v2.0.c
- UART v2.0.err
- Uart v2.0.h
- UART v2.0.o

IMU MATURITY



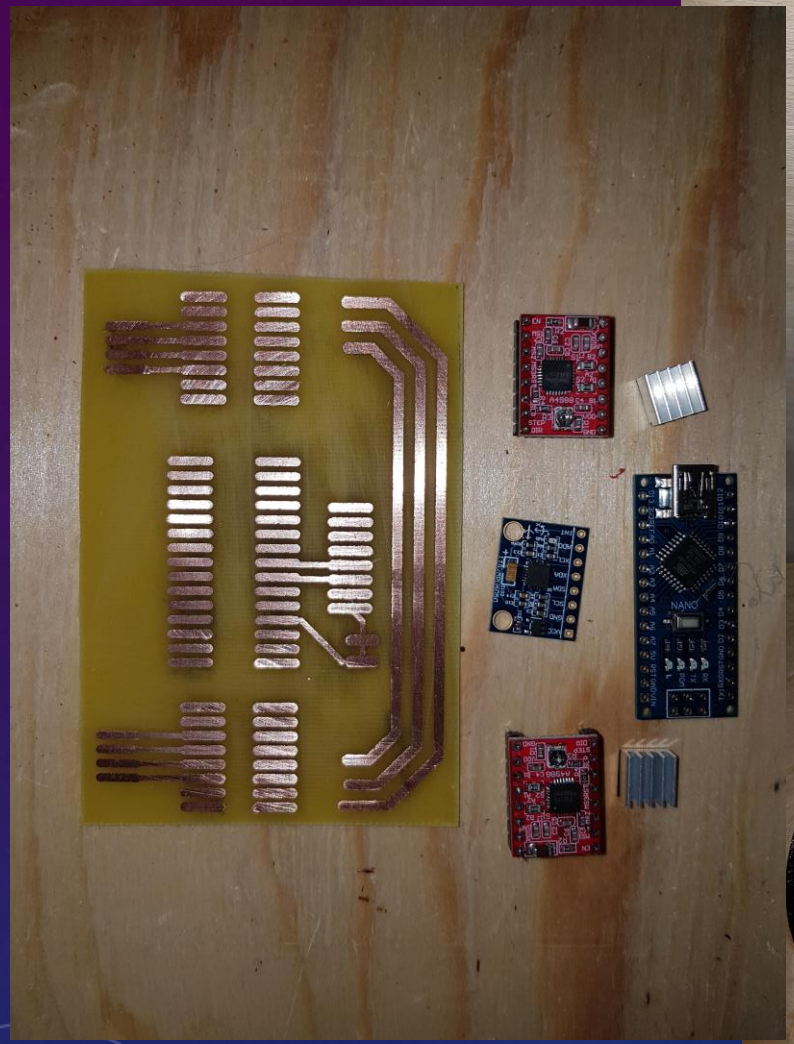


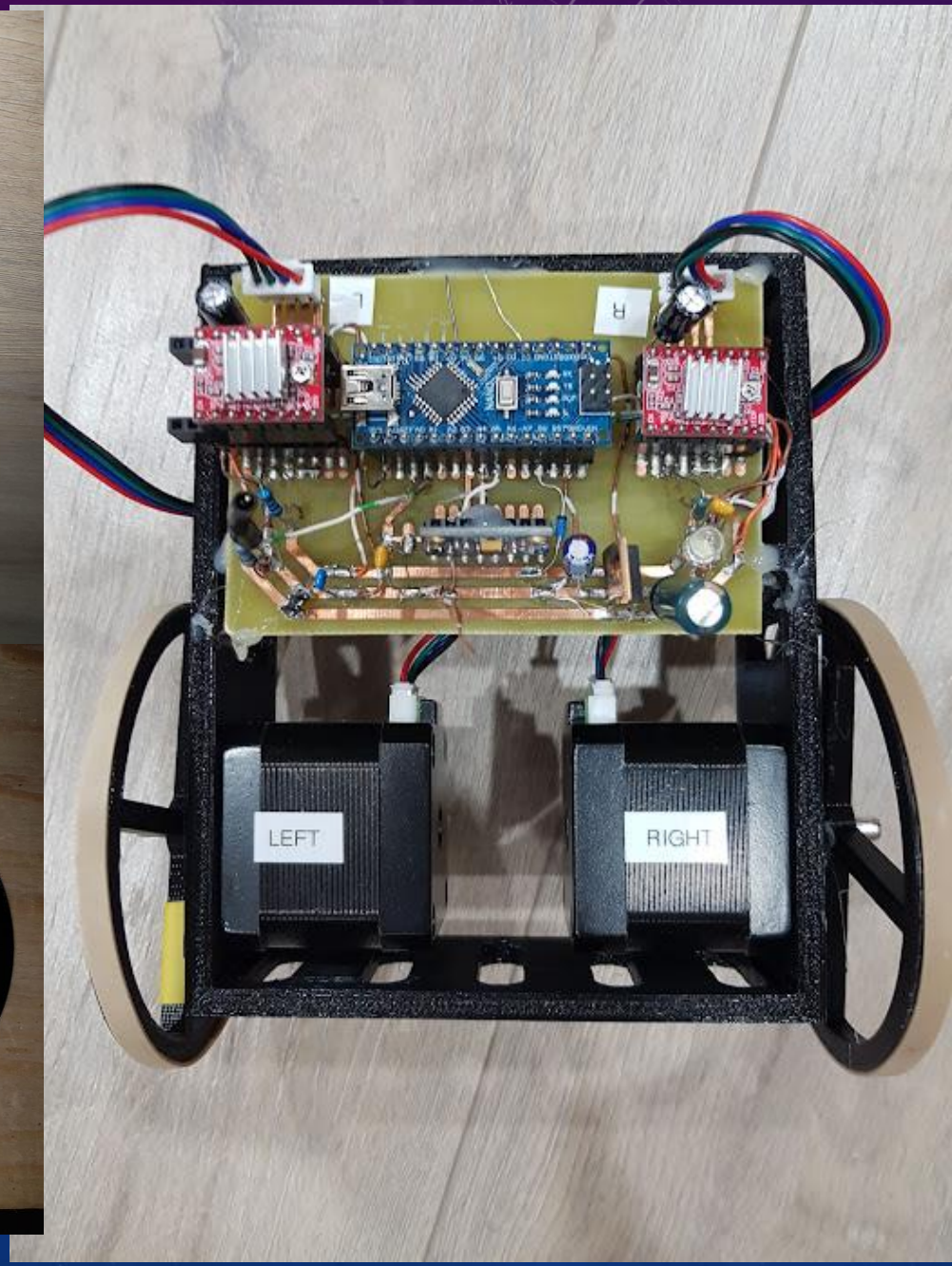


AMATEUR RADIO DISTRACTED ME UNTIL NOW….

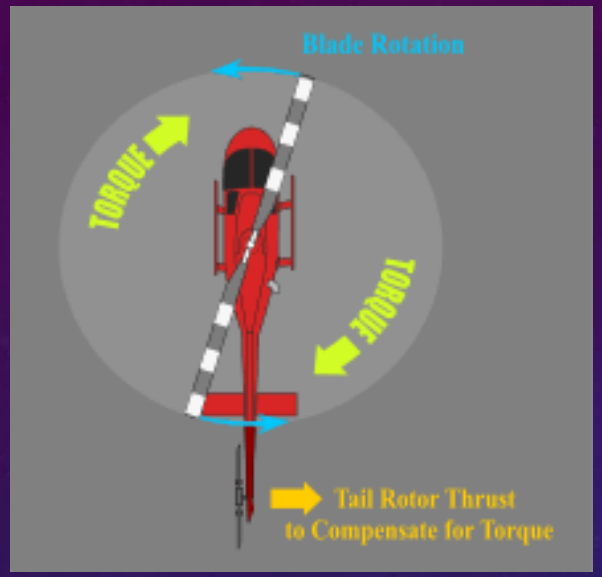
Radio

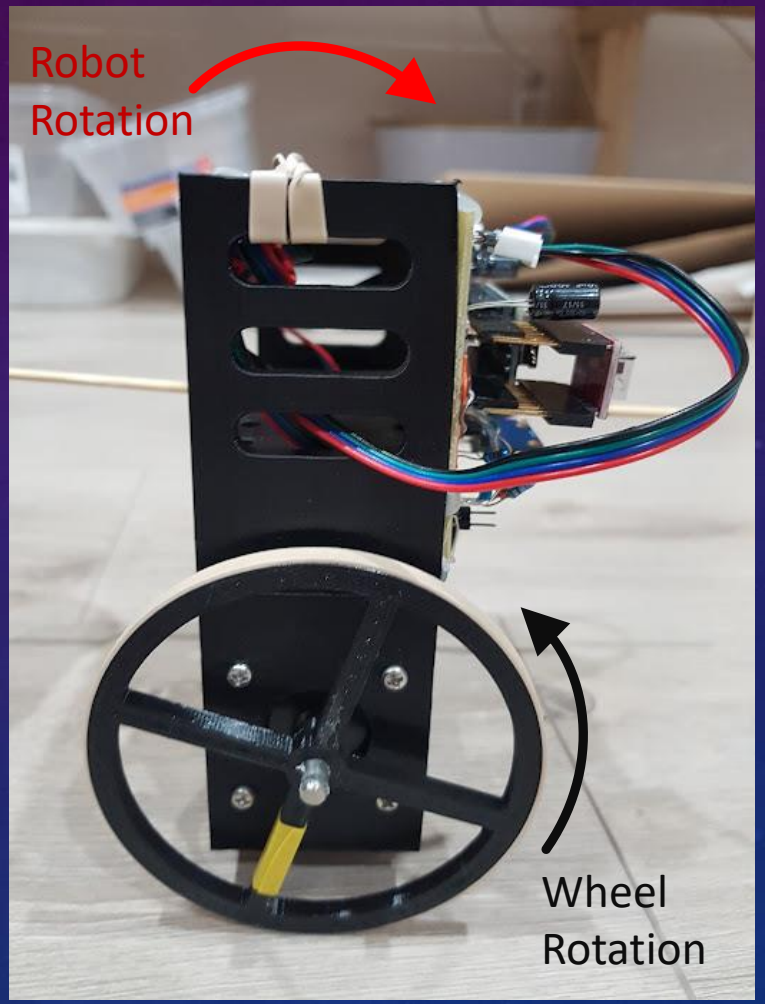
THE BUILD

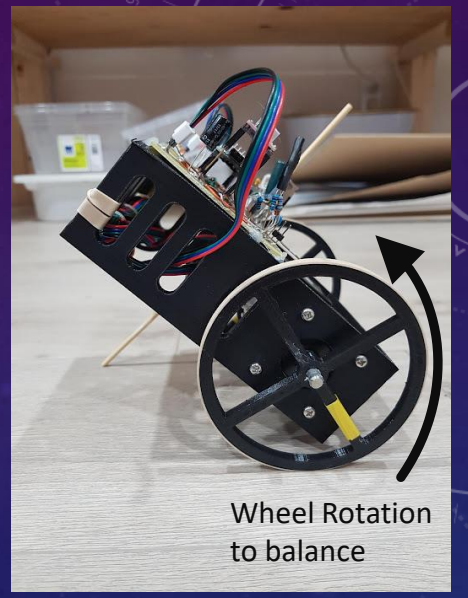




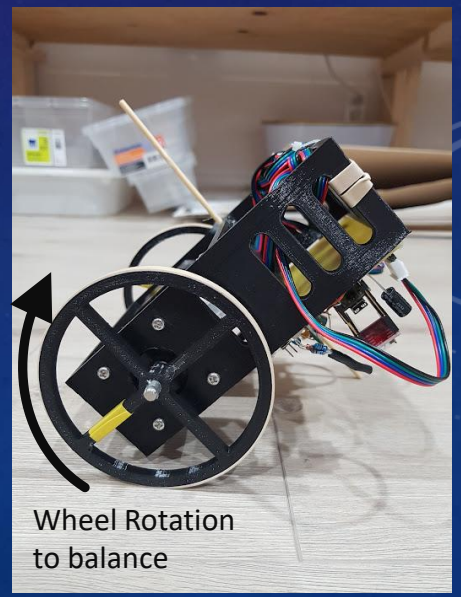
HOW DOES IT WORK?



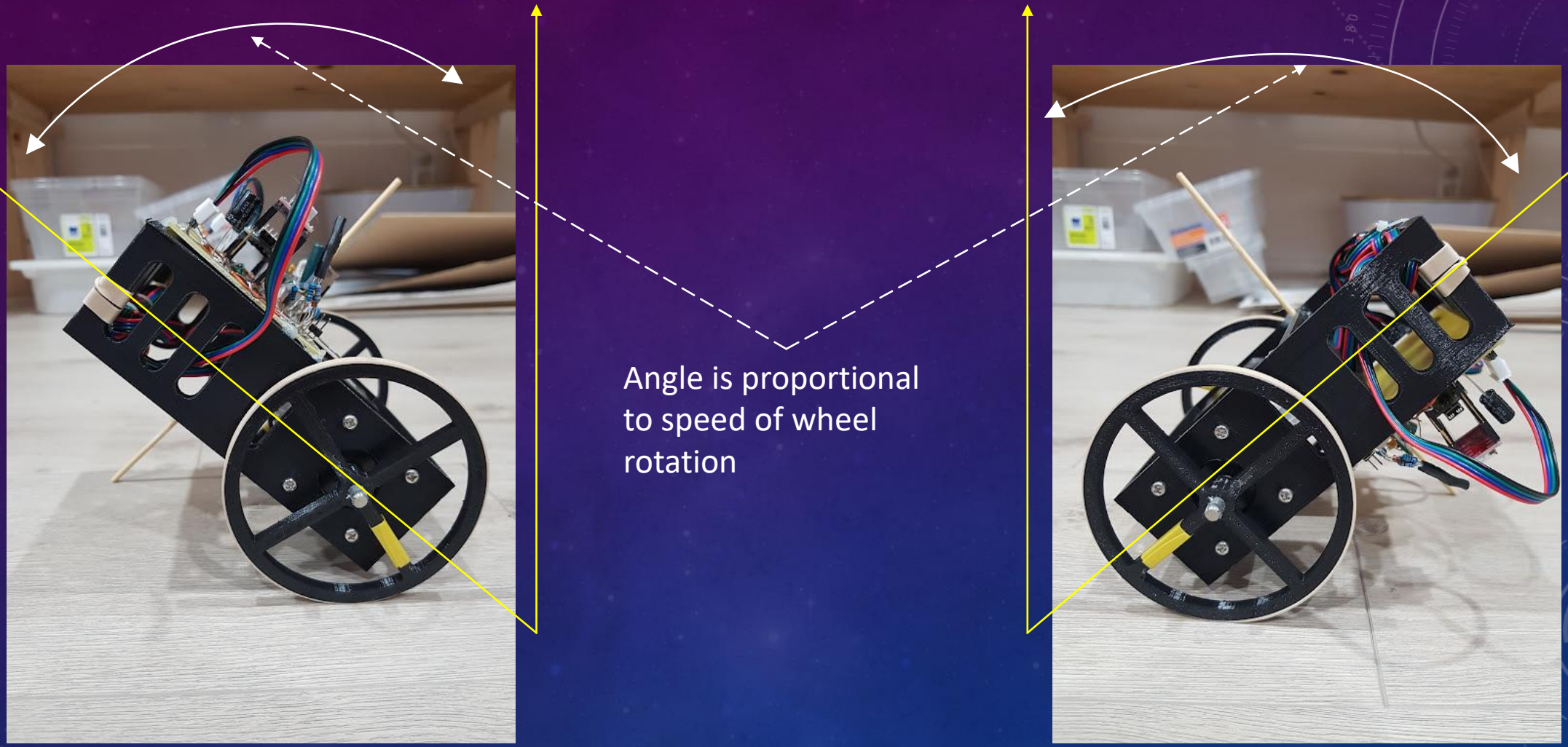




Drive robot into the tilt



HOW DOES IT WORK?

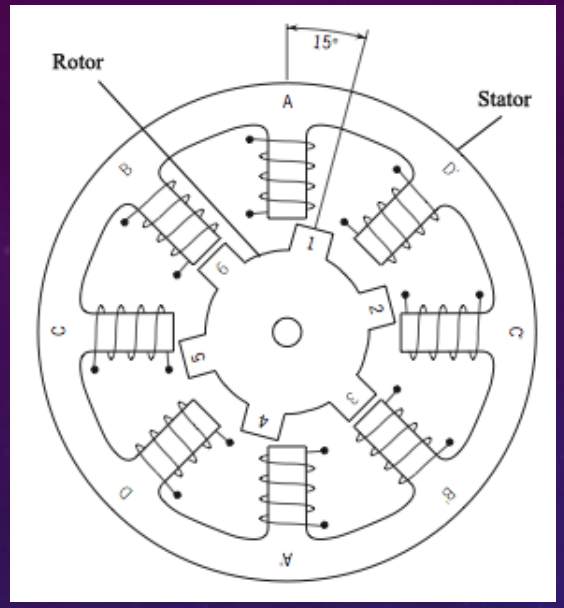


Angle is proportional to speed of wheel rotation

REQUIREMENTS

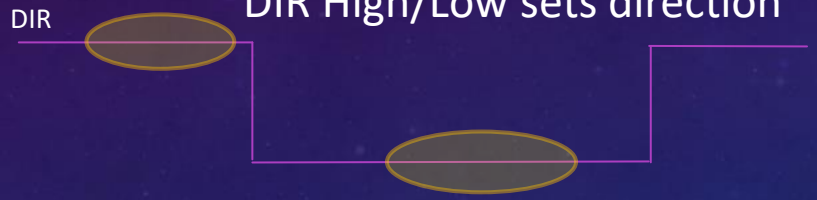
- Need to spin wheels
  - Set Direction
  - Set Speed
- Need to measure tilt of robot (Pitch) from the vertical (and “turn” around vertical)
  - Must set direction and speed of wheels based on robot pitch
- Use Arduino Nano and LiPo
- NO LIBRARIES. 100% HOME GROWN!!
- Continue to use Visual Studio and PlatformIO instead of Arduino IDE
- Monitor voltage to not over drain LiPo
- Troubleshoot (e.g., use LEDs and Scope)
- Protect robot from falling over
- Rubber on wheels
- Tune PID

1. STEPPER MOTORS

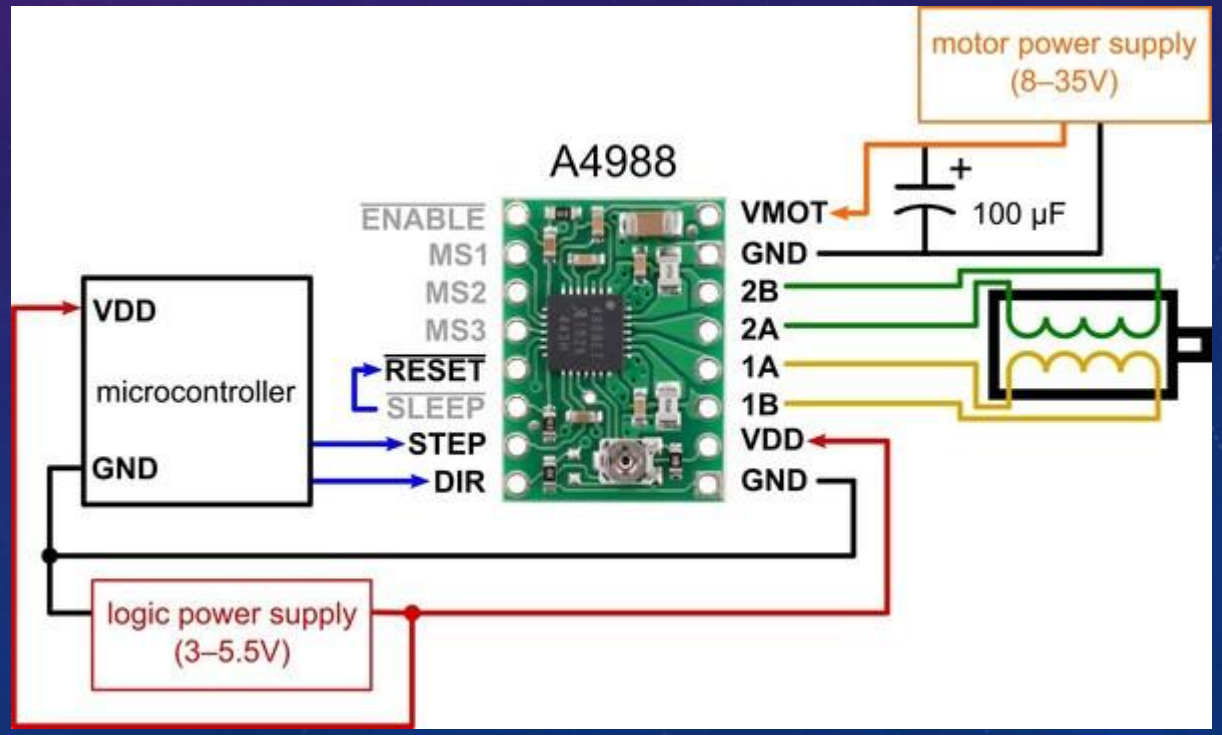


- Uses stator with many coils that can be switched on/off
- Rotor has alternating permanent magnets
- Energize a 2 or more coils caused motor to move small “step” in on direction
- Motors usually have 4 wires, 2 connected to a set of coils
- Use a driver to control direction and speed

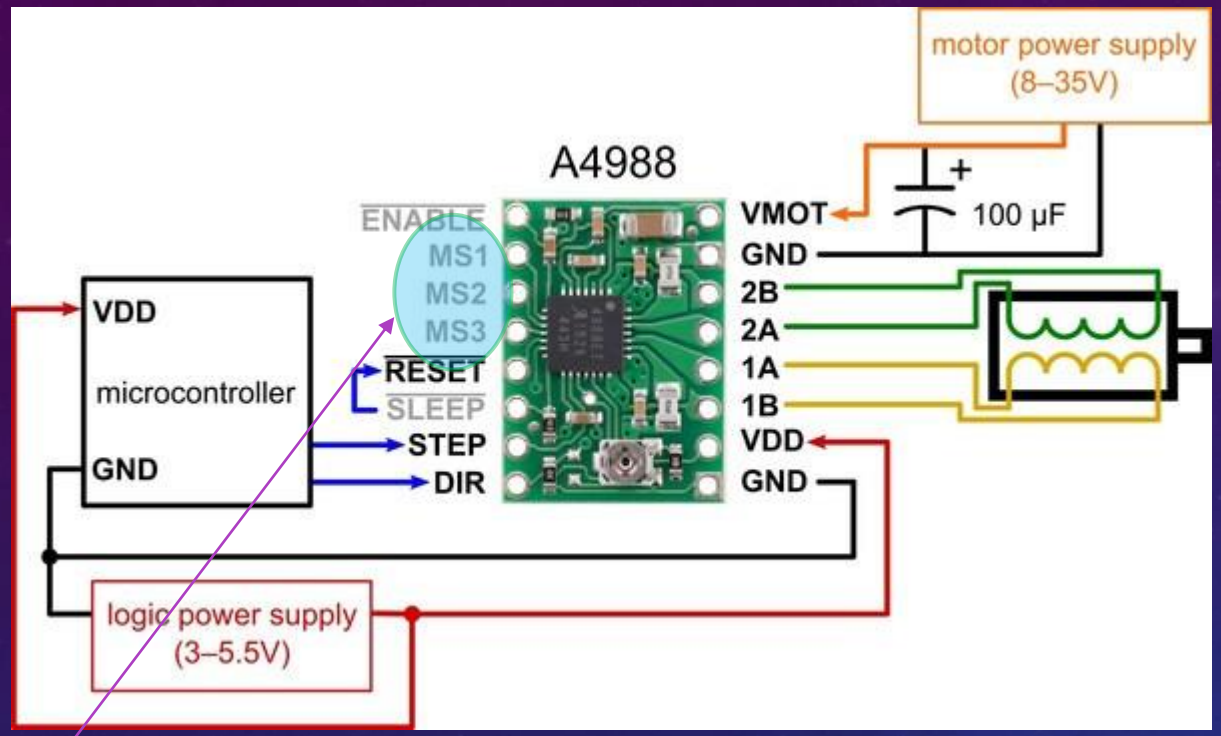
DIR High/Low sets direction



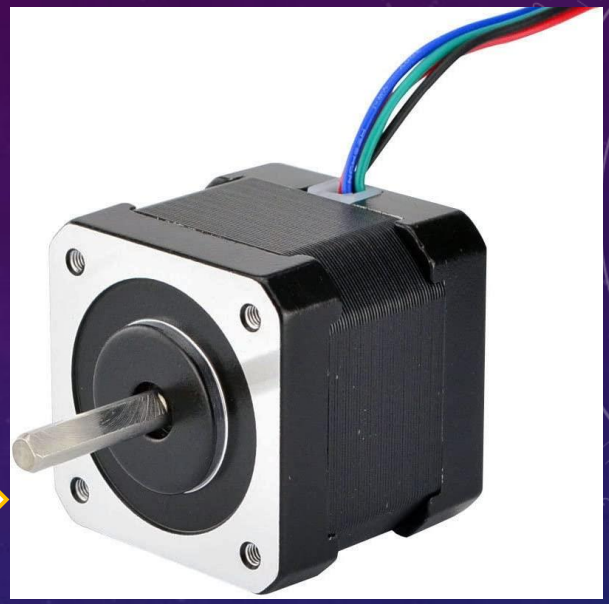
Falling edge of “Step” advances motor



1. STEPPER MOTORS



MS1/MS2/MS3 Set number of steps



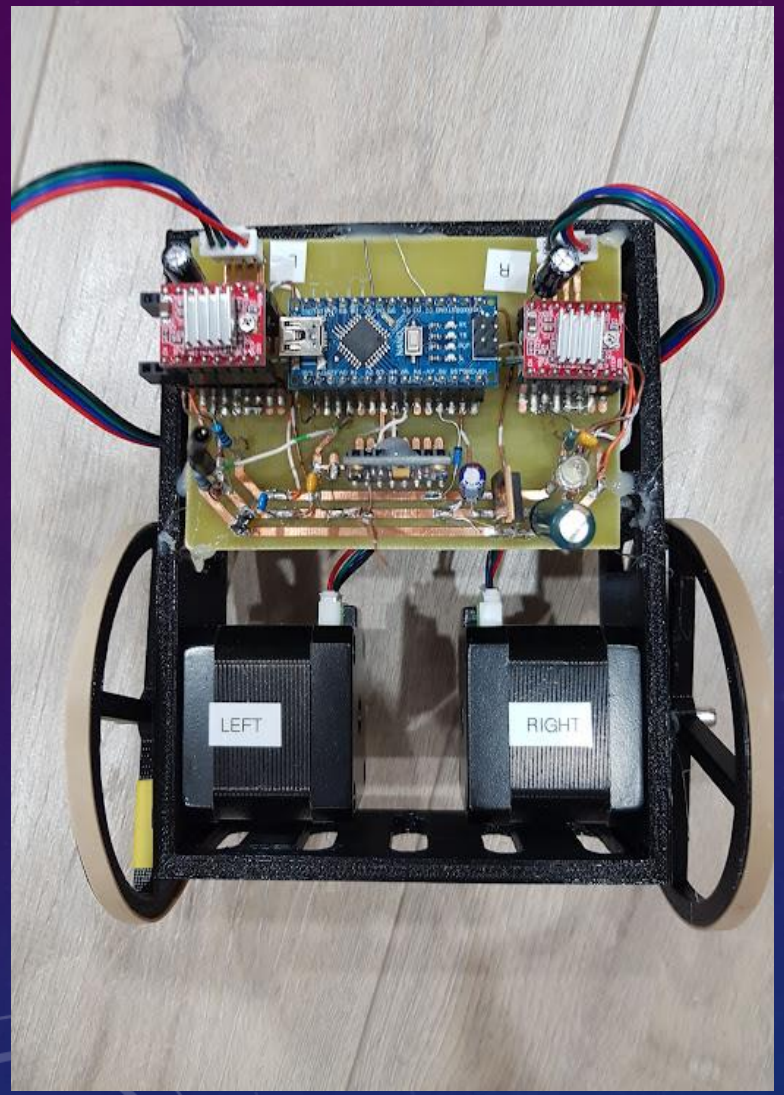
Nema 14 does 1.8 degrees per default step

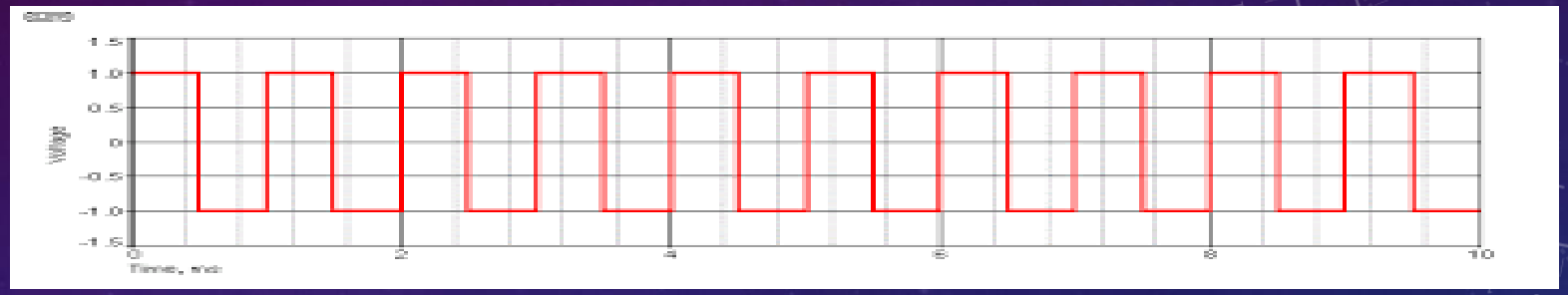
A4988 can Micro-step.

MS1-MS3:

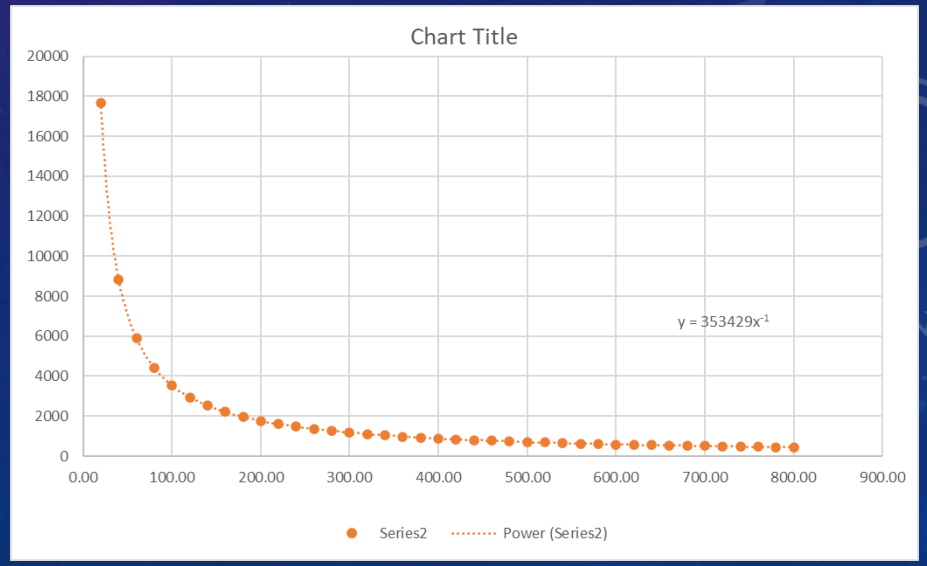
- 16 Steps for 1.8 degree
  - 0.112 deg/pulse. **8200** pulses for 1 rotation
- 8 Steps for 1.8 degree
  - 0.225 deg/pulse. **1600** pulses for 1 rotation
- 4 Steps for 1.8 degree
  - 0.45 deg/pulse. **800** pulses for 1 rotation
- 2 Steps for 1.8 degree
  - 0.9 deg/pulse. **400** pulses for 1 rotation
- 1 Steps for 1.8 degree
  - 1.8 deg/pulse. **200** pulses for 1 rotation

1. STEPPER MOTORS





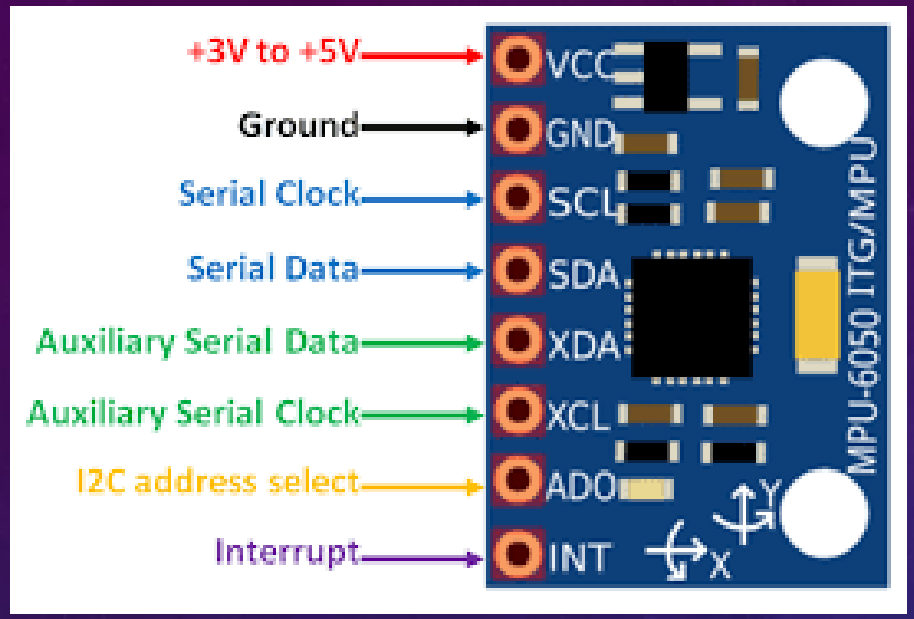
- Pulses (falling edges) sets speed of motor. i.e., period
- Nonlinear behaviour (period = 1/frequency)
- Motor speed = 1/frequency
- Causes grief for PID

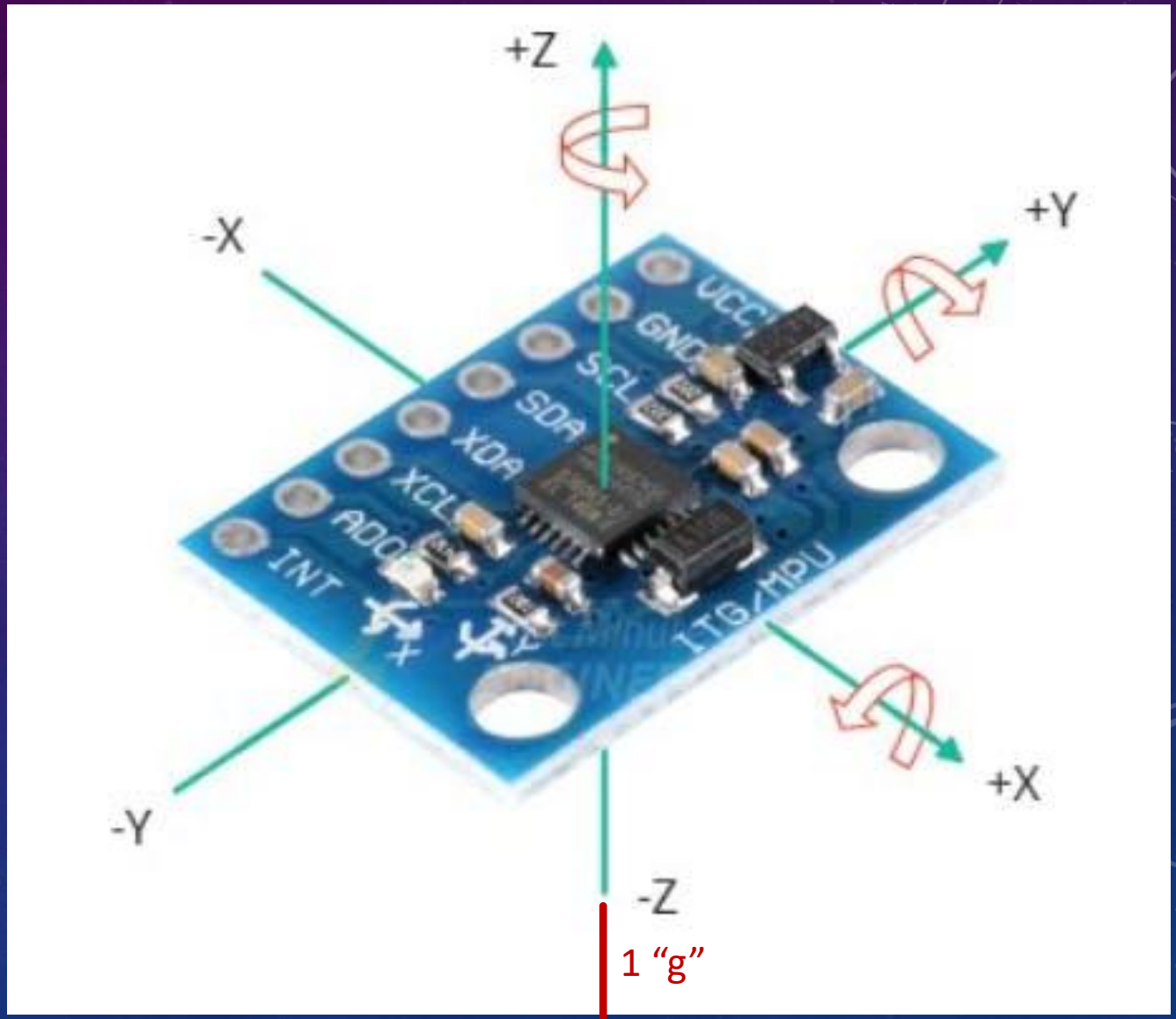


2. MEASURING PITCH

- Use an Accelerometer to measures force applied in X, Y, Z planes
  - E.g., acceleration due to gravity
- Use a Gyroscope to measure turn rate (angular rate) in X, Y, Z planes
  - Rate measure x time between measurements = angle
- Can get angle from Accelerometer and Gyroscope
  - Gyroscope drifts in time. No concept of flat and level.
  - Accelerometer give false reading for “real” acceleration and vibrations
- Combine both together (Complementary or Kalman Filter)

2. MPU 6050

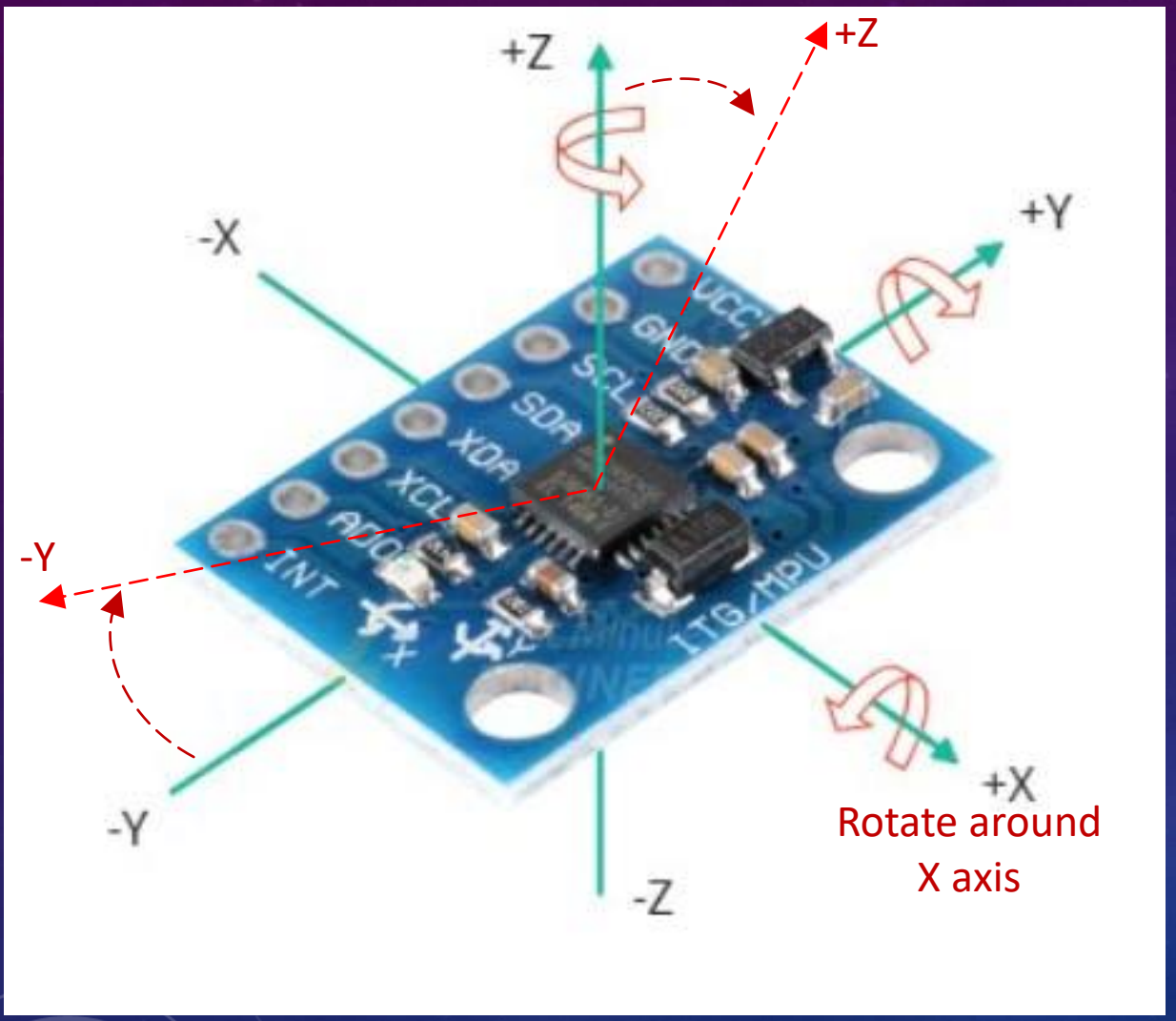




When flat

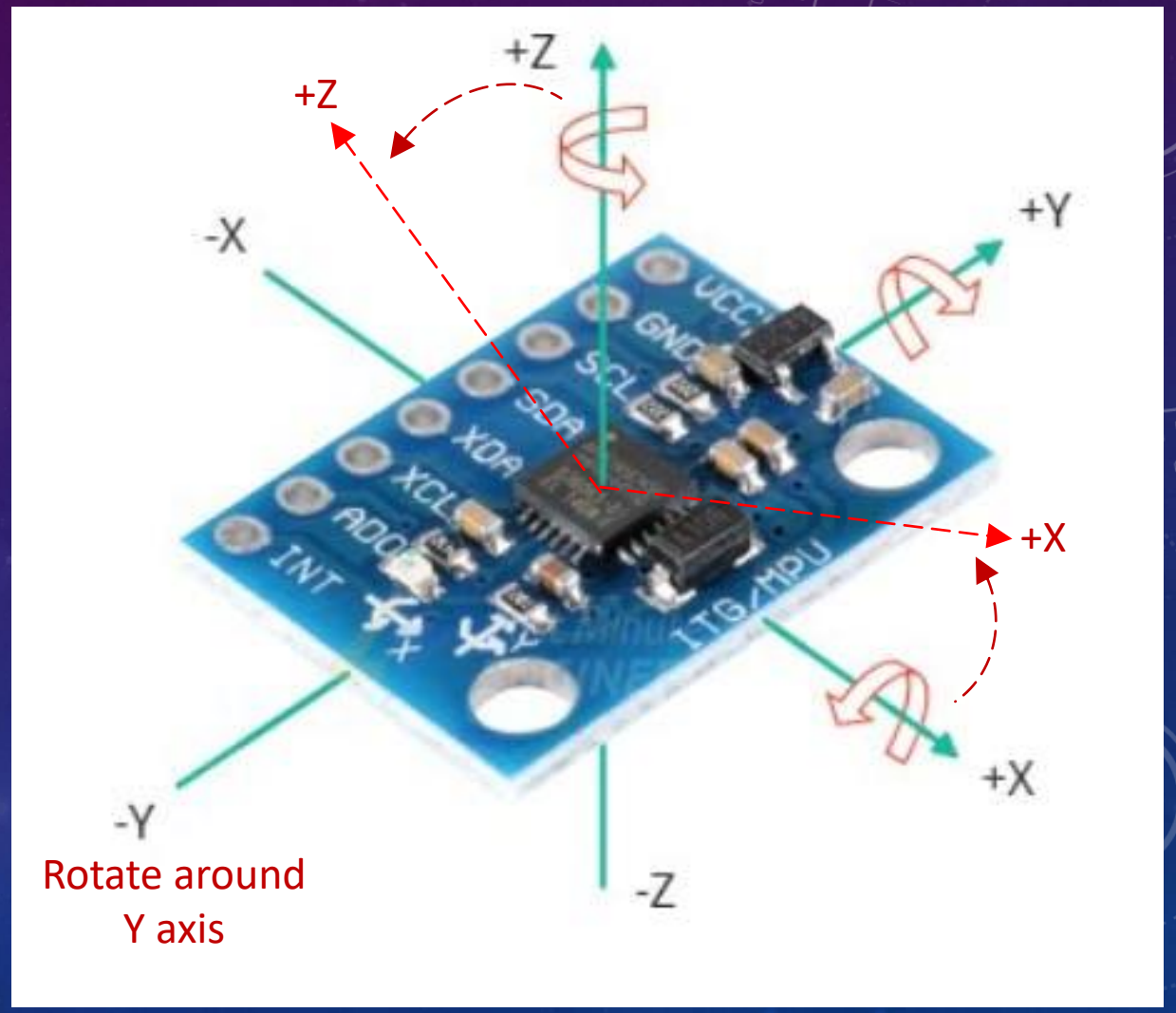
- Gyros output “0” rate
- Accel is 0 for Y, and X and -1g for Z

2. MPU 6050: ROTATIONS



Rotate around “X”

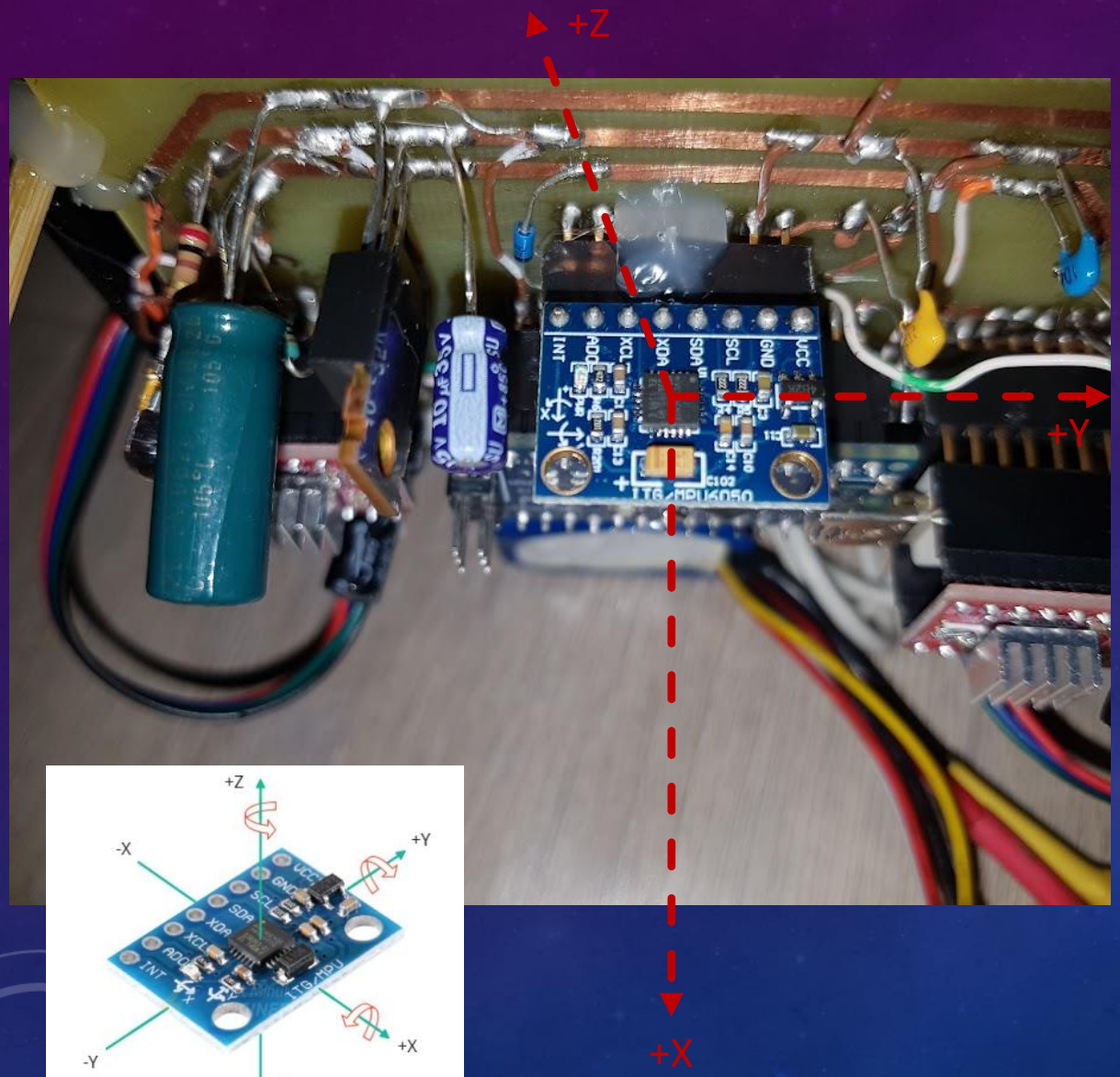
- X Gyro output a rate, Y and Z will be 0
- Accel is 0 for Z, and some number for Y and Z

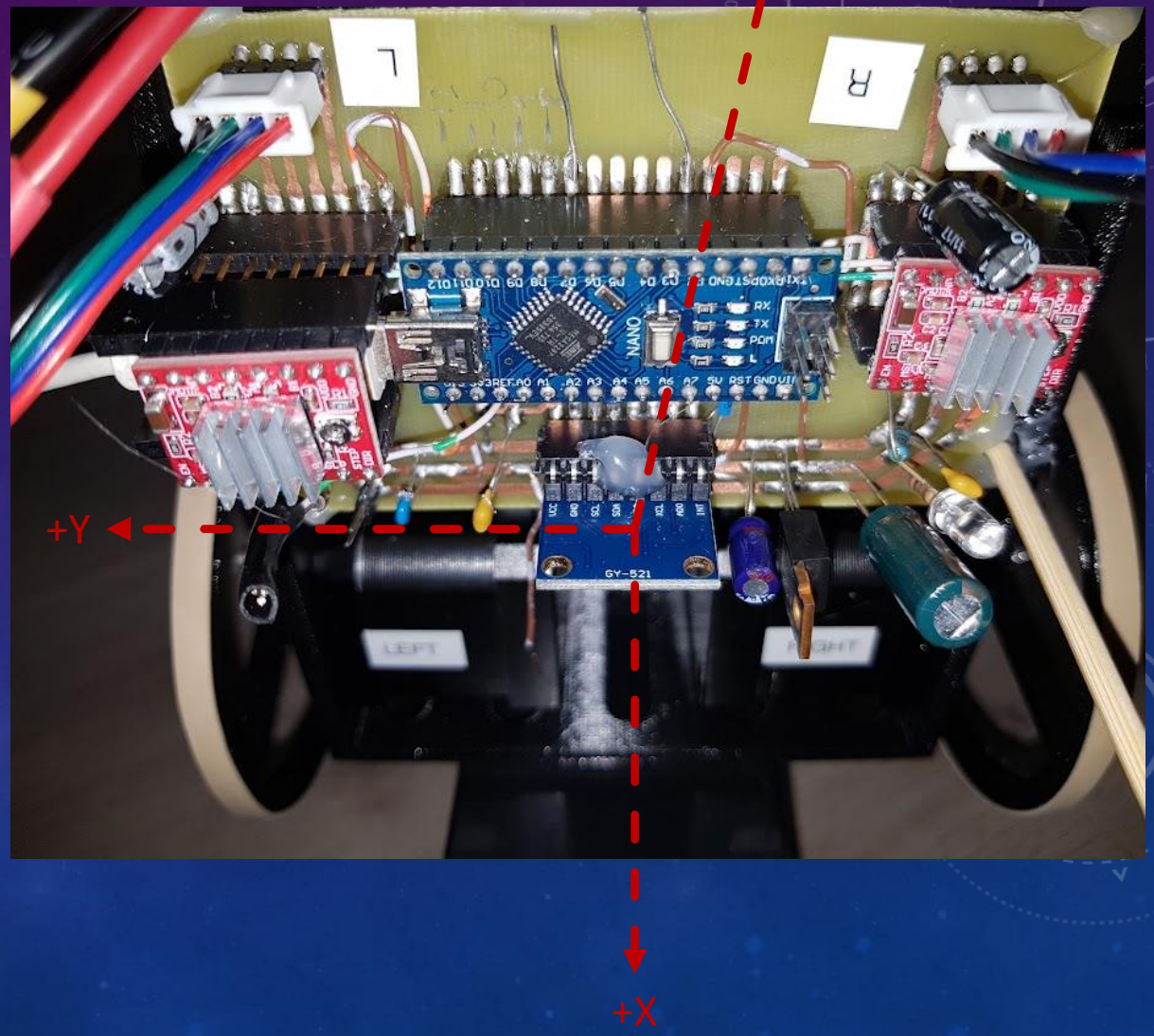


Rotate around “Y”

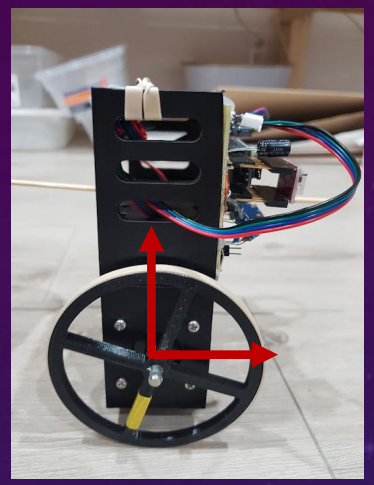
- Y Gyro output a rate, X and Z will be 0
- Accel is 0 for Y, and some number for X and Z

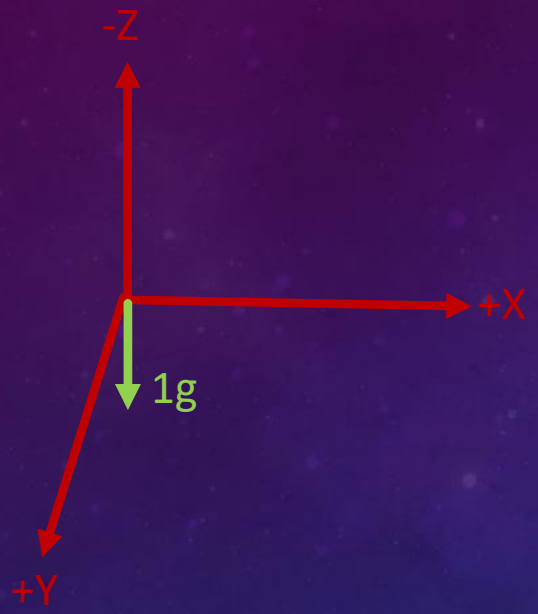
2. MPU 6050: MOUNTING ORIENTATION



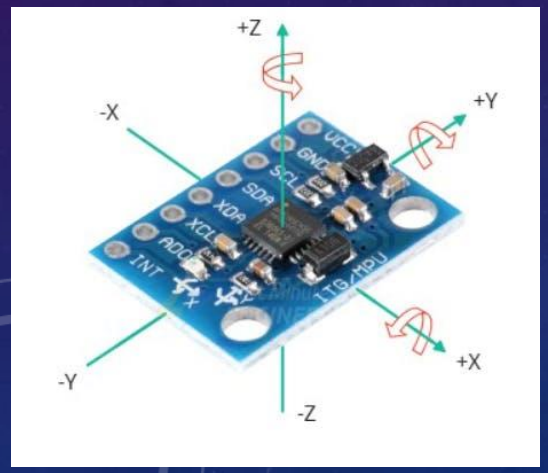


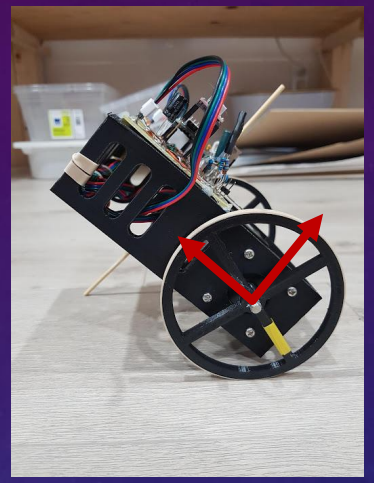
2. MPU 6050: MATH



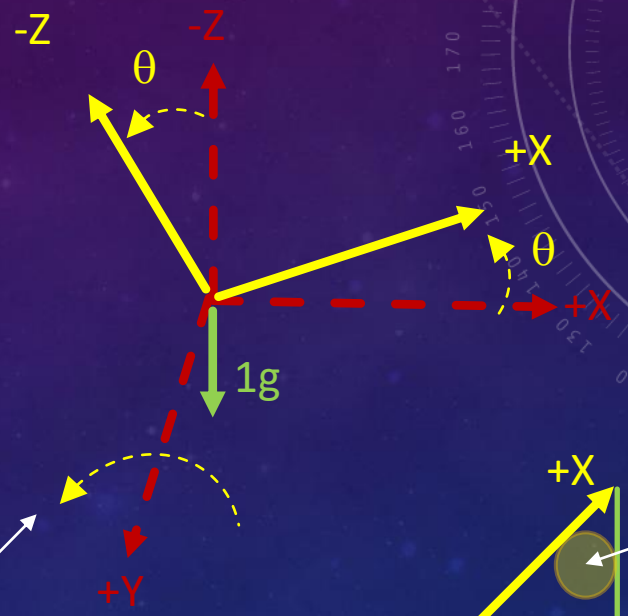


Robot Standing Straight Up





Robot "Pitches" Back
(i.e., rotation around Y axis)

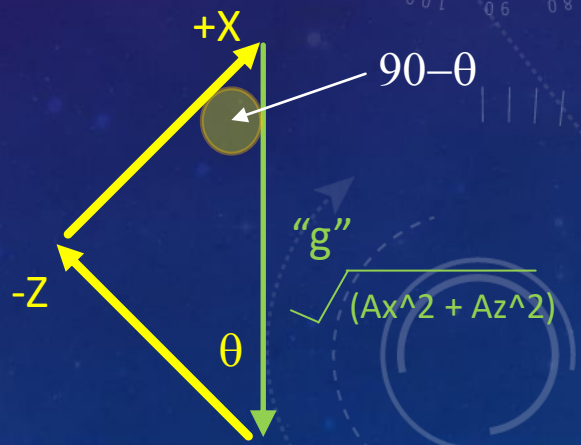


Gyroscope gives rotation rate around Y axis

θ = Gy x time

Gy Gyroscope Reading

e.g., measure every 4ms, time = 4ms



θ = atan(+X/-Z) = atan(Ax/Az)
θ = asin(Az/g)

Ax, Ay, Az Accelerometer Readings

CODE: MEASURE PITCH

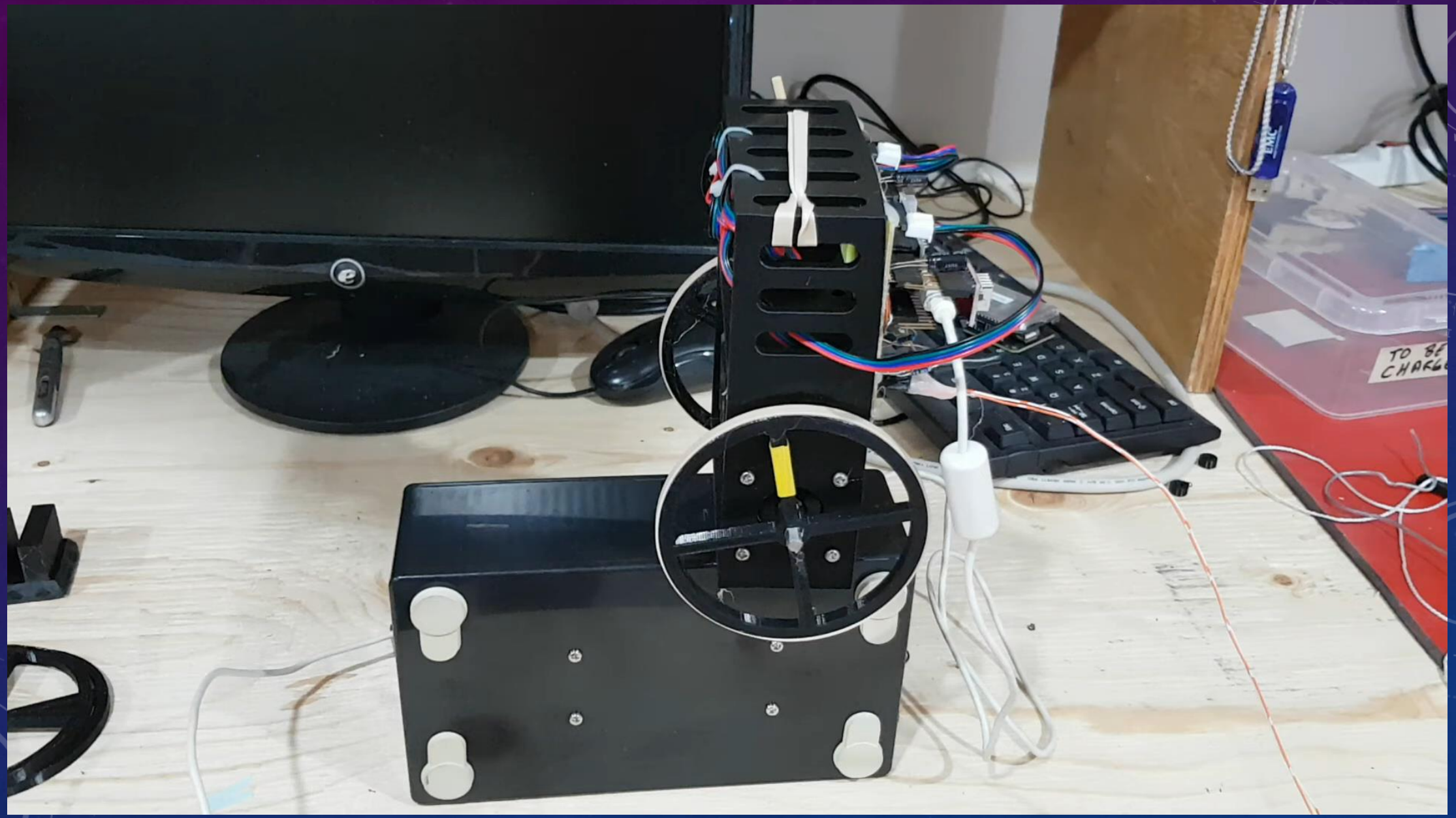
```
void stateMachineLoop(void) {
  if (flags & SAMPLE_MPU) {
    flags &= ~SAMPLE_MPU;
    readMPU6050(CALIBRATE);
    tel.timer = micros() - oldTimer;
    oldTimer = micros();
    tel.gyro_conversion = 131.0 / ((double)tel.timer / 1000000.0);
    if (tel.gyro_conversion < ((double)GYRO_CONVERSION_FACTOR / 2) ||
        tel.gyro_conversion > ((double)GYRO_CONVERSION_FACTOR * 2)) {
      Serial.println("T");
      tel.gyro_conversion = GYRO_CONVERSION_FACTOR;
    }

    computeTelemetry();
    computePID();
  }
}
```

Interrupt called ever 4ms and sets SAMPLE_MPU flag

```
// Rotations around Y is pitch.  I.e. X and Z moves
tel.gyro_ry = (double)tel.gyroy / tel.gyro_conversion;
tel.angle_ax = atan((double)tel.accelx / (double)tel.accelz);
tel.angle_ax *= DEGREES_CONVERSION_FACTOR;
```

```
tel.angle_gy += tel.gyro_ry;  // Pitch

tel.pitch = tel.angle_gy =
    tel.angle_gy * FILTER_GYRO_COEF + tel.angle_ax * FILTER_ACCEL_COEF;
```

PITCH MEASUREMENTS



CODE: CONTROL STEPPER

Interrupt called ever 20us to simulate frequency for STEP pulses

- Pulses “STEP” from High -> Low
- Waits i.e., counts 20us intervals
- Rinse and Repeat

E.g., Pulse Delay is 200

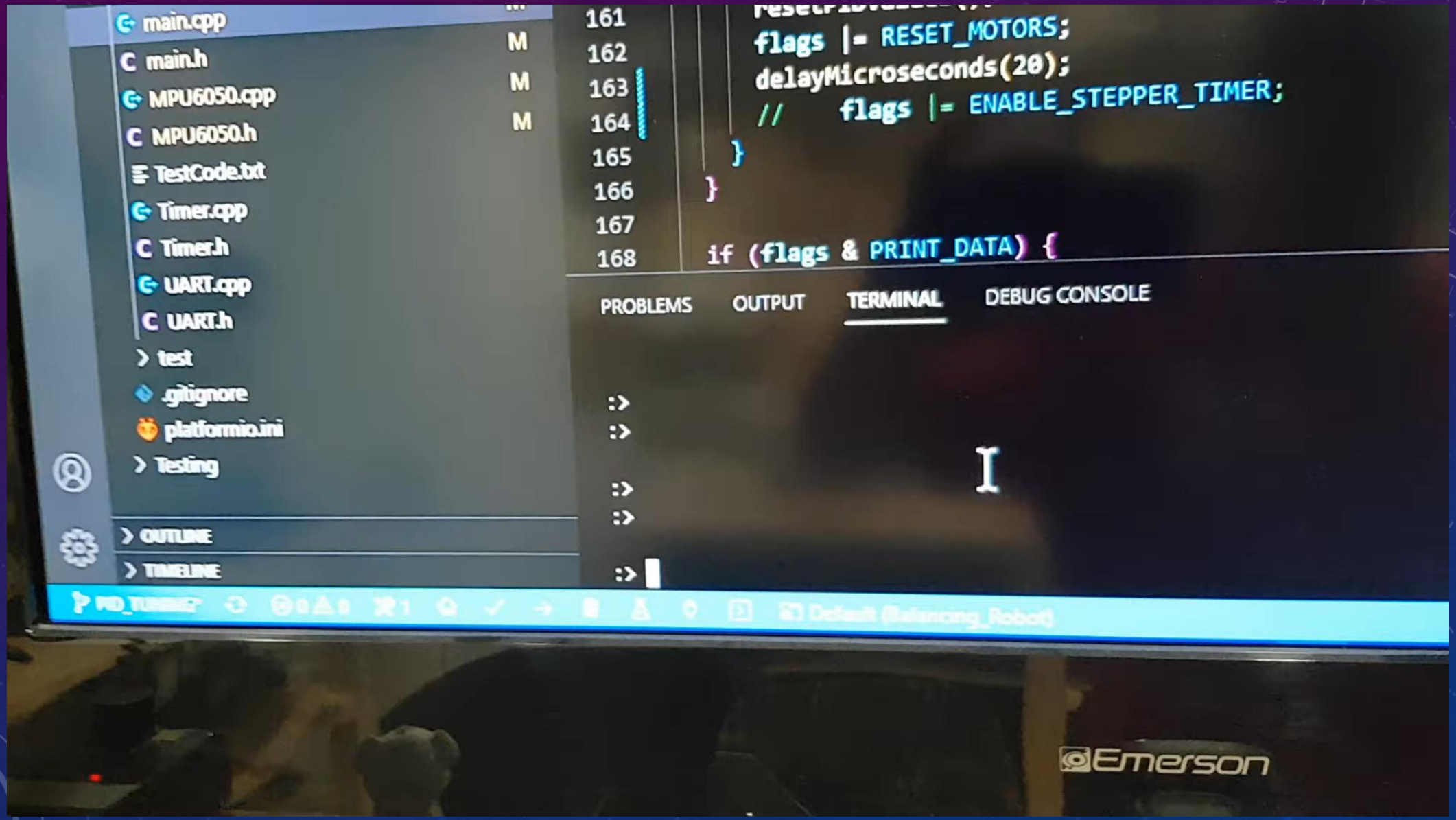
1. Pulses High to Low
2. Counts to 10

i.e., 10 x 20us or 200us or Frequency 5 KHz. Step pulses 5 KHz

```
if (flags & ENABLE_STEPPER_TIMER) {
  // Left motor pulse calculations
  tel.leftMotorCounter++;
  if (tel.leftMotorCounter > tel.leftMotorCurrent) {
    tel.leftMotorCounter = 0;
    tel.leftMotorCurrent = tel.leftMotorThrottle;
    if (tel.leftMotorCurrent < 0) {
      DIRL_C;
      tel.leftMotorCurrent *= -1;
    } else
      DIRL_CC;
  } else if (tel.leftMotorCounter == 1)
    STEPL_H;
  else if (tel.leftMotorCounter == 2)
    STEPL_L;

  // Right motor pulse calculations
  tel.rightMotorCounter++;
  if (tel.rightMotorCounter > tel.rightMotorCurrent) {
    tel.rightMotorCounter = 0;
    tel.rightMotorCurrent = tel.rightMotorThrottle;
    if (tel.rightMotorCurrent < 0) {
      DIRR_C;
      tel.rightMotorCurrent *= -1;
    } else
      DIRR_CC;
  } else if (tel.rightMotorCounter == 1)
    STEPR_H;
  else if (tel.rightMotorCounter == 2)
    STEPR_L;
}
```

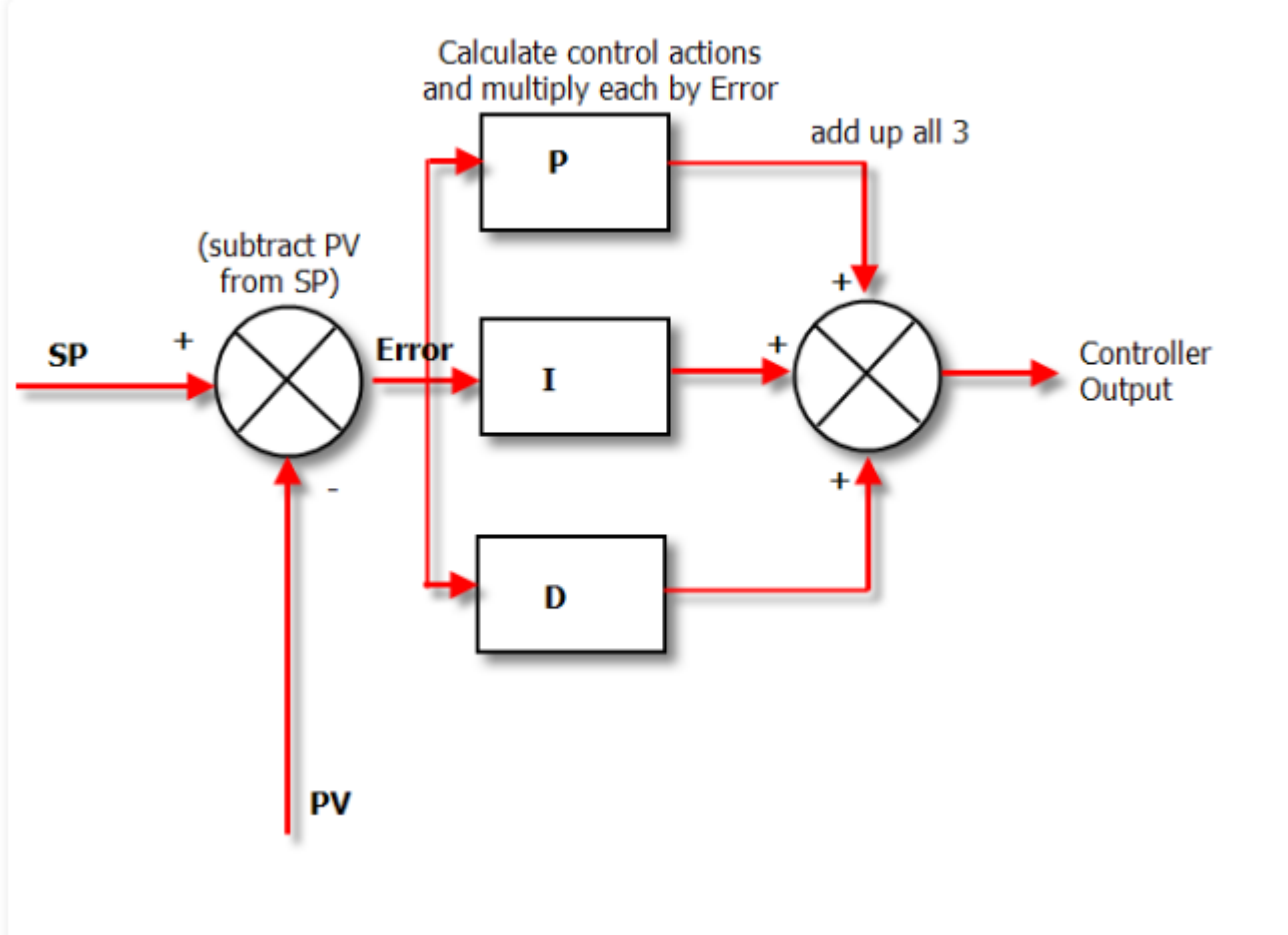
Had to replace my code with Joop Brokking’s code (http://brokking.net/)

STEPPER TESTING



PID - CLOSE LOOP CONTROL SYSTEM

A **proportional–integral–derivative controller** (**PID controller** or **three-term controller**) is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an *error value* $e(t)$ as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms (denoted *P*, *I*, and *D* respectively), hence the name.



(P) Proportional: Kp x Error

- If the robot tilts forward move wheels to balance.
- Get a “jerky” movement. Oscillation

(I) Integral: Ki x Sum of Errors

- Accelerates the wheel spin over time
- If robot does not seem to be centering itself quickly, spin wheels faster

(D) Derivative: Kd x Difference in Error (since last measurement)

- If the robot is moving to fast to correct, it slows down the motion.
- It smooths out the “jerky” movement. Damps oscillation

CODE: PID TUNING – THE FUN BEGINS

Core PID Calculations

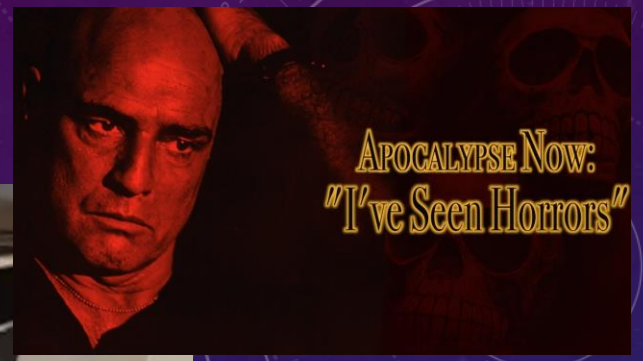
```
tel.pitchError = tel.pitch - tel.pitchUpright - tel.pitchSetPoint;

if (abs(tel.pitch) >= MAXIMUM_ANGLE || abs(tel.pitchError) > MAXIMUM_ANGLE) {
  flags &= ~MEASURE_DISTANCE;
  tel.msCounter = 0;
  digitalWrite(LED, HIGH);
  flags = 0;
  PIN_RESET;
  DISABLE_STEPPER;
  resetPIDValues();
  Serial.println("Fall Down Detected");
  return;
}

if (tel.pidTotal > 35 || tel.pidTotal < -35) {
  tel.pitchError += tel.pidTotal * PID_BRAKE;
  digitalWrite(LED, LOW);
}

tel.pid_p = tel.Kp * tel.pitchError;
tel.pid_i += (tel.Ki * tel.pitchError);
tel.pid_d = tel.Kd * (tel.pitchError - tel.priorPitchError);
tel.priorPitchError = tel.pitchError;

if (tel.pid_i > MAXIMUM_PID_VALUE) {
  tel.pid_i = MAXIMUM_PID_VALUE;
} else if (tel.pid_i < -MAXIMUM_PID_VALUE) {
  tel.pid_i = -MAXIMUM_PID_VALUE;
}

tel.pidTotal = tel.pid_p + tel.pid_i + tel.pid_d;

if (tel.pidTotal > MAXIMUM_PID_VALUE) {
  tel.pidTotal = MAXIMUM_PID_VALUE;
} else if (tel.pidTotal < -MAXIMUM_PID_VALUE) {
  tel.pidTotal = -MAXIMUM_PID_VALUE;
}

if (tel.pidTotal < PID_DEAD_ZONE && tel.pidTotal > -PID_DEAD_ZONE) {
  tel.pidTotal = 0;
}

tel.pidLeft = tel.pidTotal;
tel.pidRight = tel.pidTotal;
```

Compensate for nonlinear behaviour

```
if (tel.pidLeft > 0)
  tel.pidLeft = 405 - (1 / (tel.pidLeft + 9)) * 5500;
else if (tel.pidLeft < 0)
  tel.pidLeft = -405 - (1 / (tel.pidLeft - 9)) * 5500;

if (tel.pidRight > 0)
  tel.pidRight = 405 - (1 / (tel.pidRight + 9)) * 5500;
else if (tel.pidRight < 0)
  tel.pidRight = -405 - (1 / (tel.pidRight - 9)) * 5500;

// Calculate the needed pulse time for the left and right stepper motor
// controllers
if (tel.pidLeft > 0)
  tel.leftMotor = 400 - tel.pidLeft;
else if (tel.pidLeft < 0)
  tel.leftMotor = -400 - tel.pidLeft;
else
  tel.leftMotor = 0;

if (tel.pidRight > 0)
  tel.rightMotor = 400 - tel.pidRight;
else if (tel.pidRight < 0)
  tel.rightMotor = -400 - tel.pidRight;
else
  tel.rightMotor = 0;

// Copy the pulse time to the throttle variables so the interrupt subroutine
// can use them
tel.leftMotorThrottle = tel.leftMotor;
tel.rightMotorThrottle = tel.rightMotor;
```

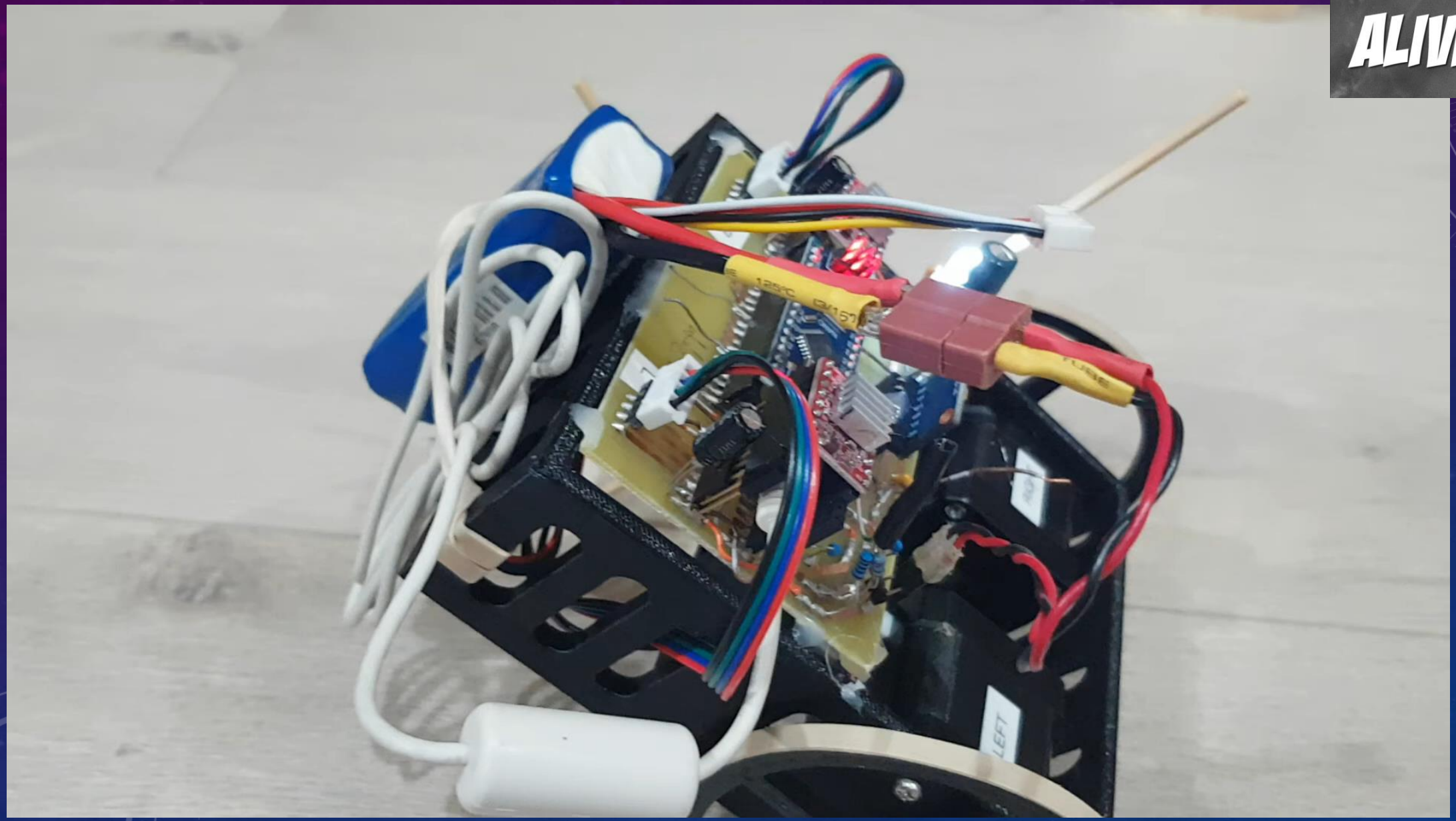
Had to replace my code with Joop Brokking's code (http://brokking.net/)

THE HORROR…



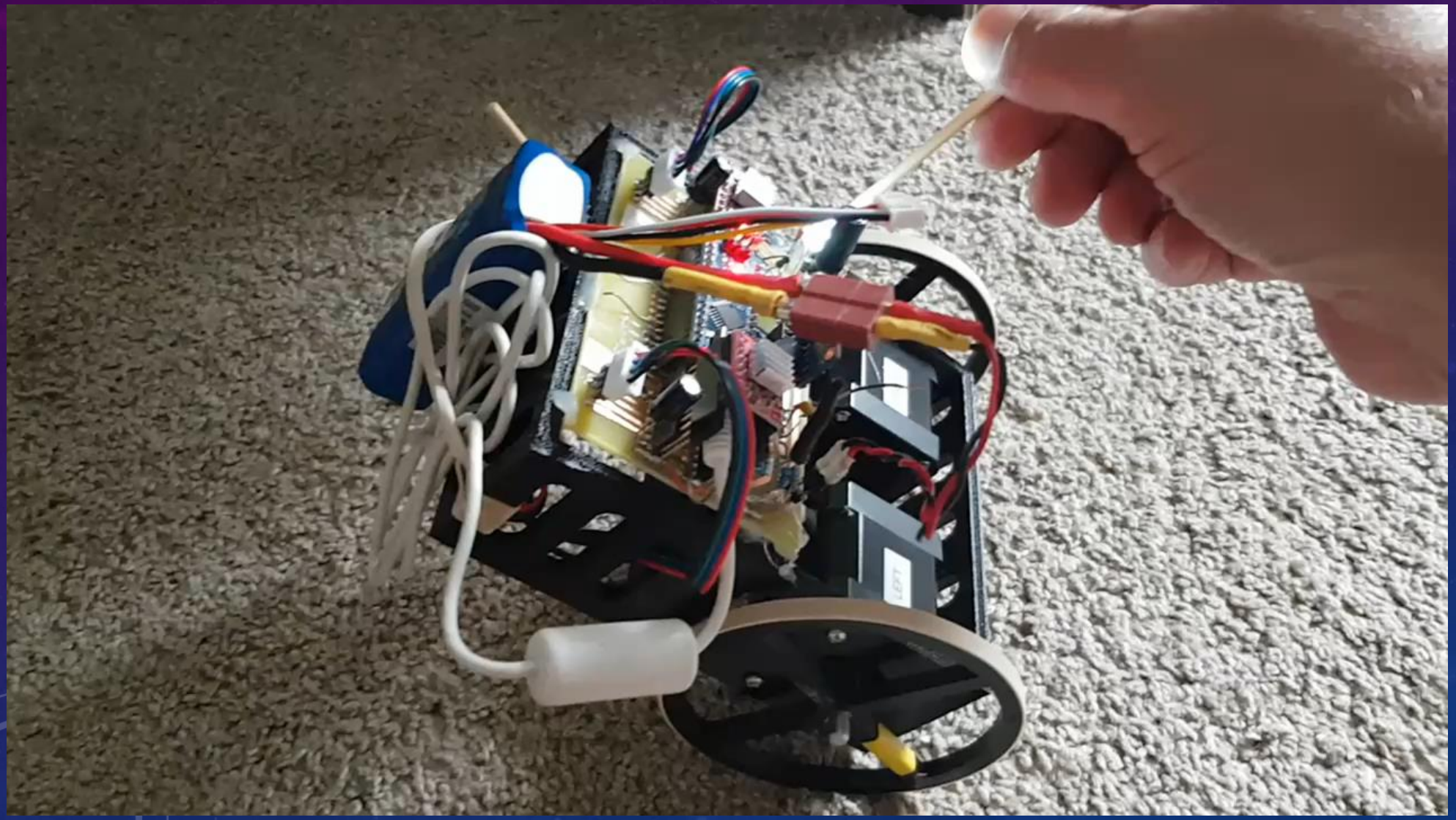
Developing my code

MOO HOO HA HA…ITS ALIVE!!



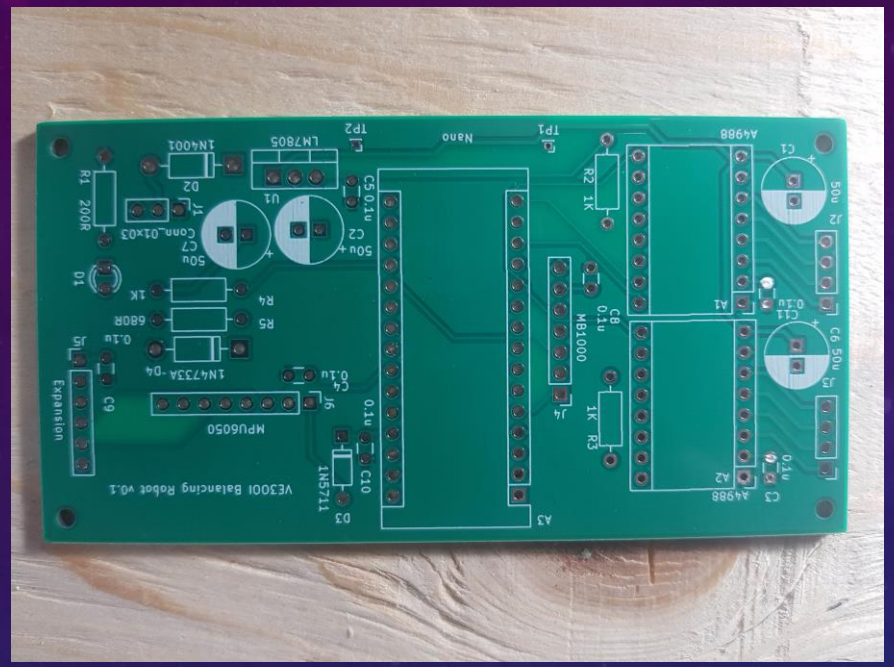
Using my code

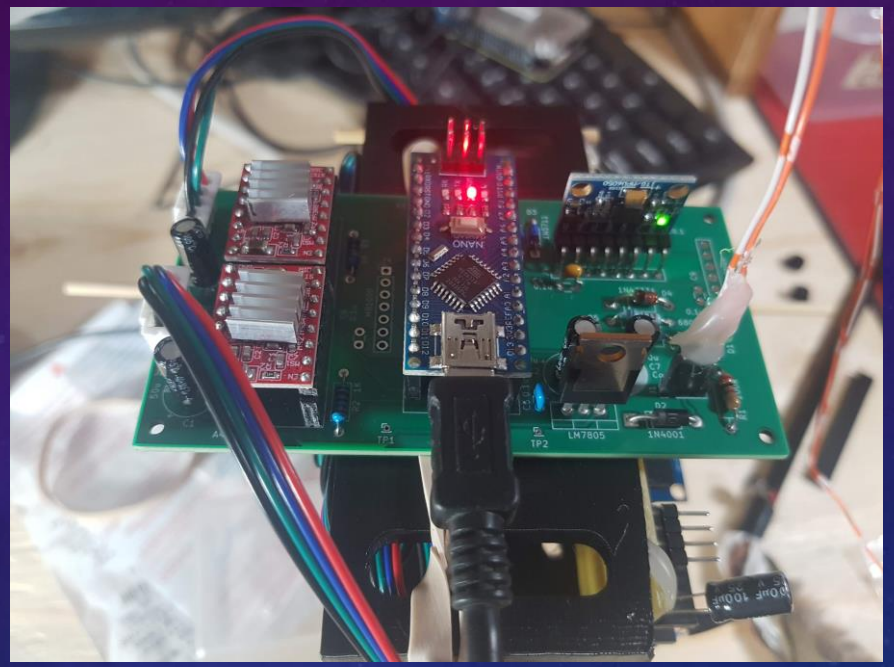
EVEN BETTER…

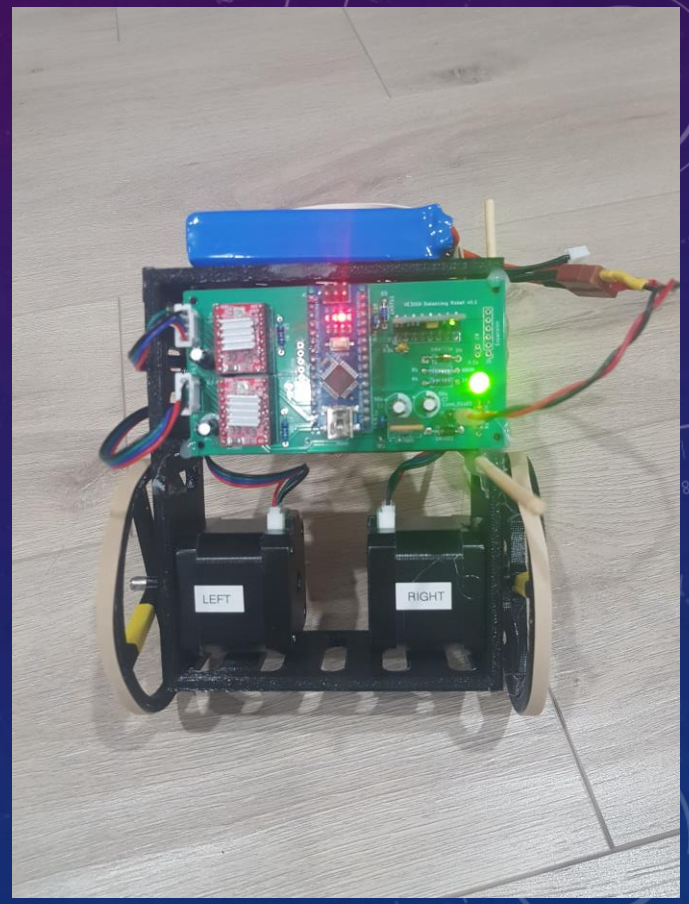


Using my code

REAL PCB



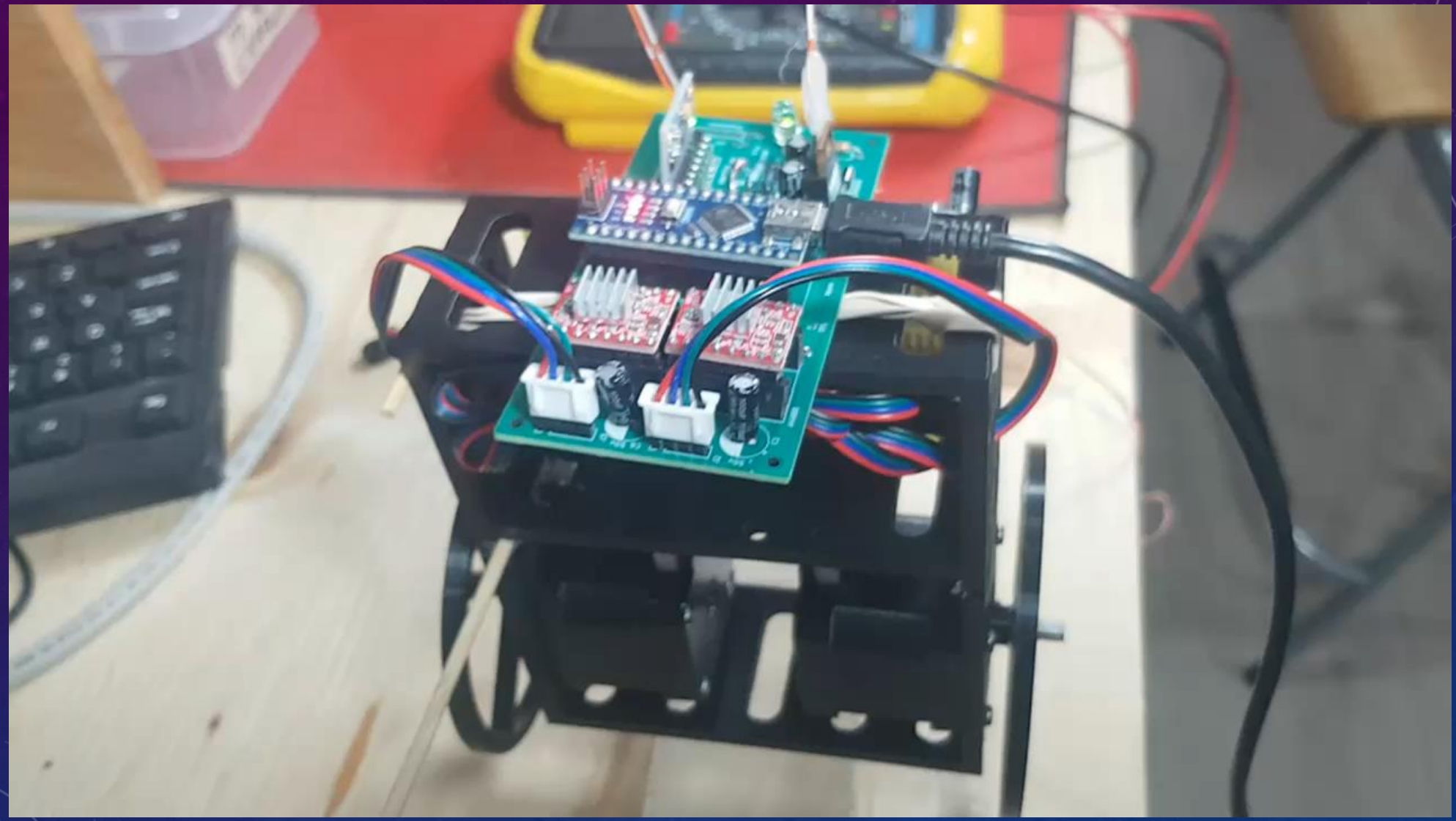




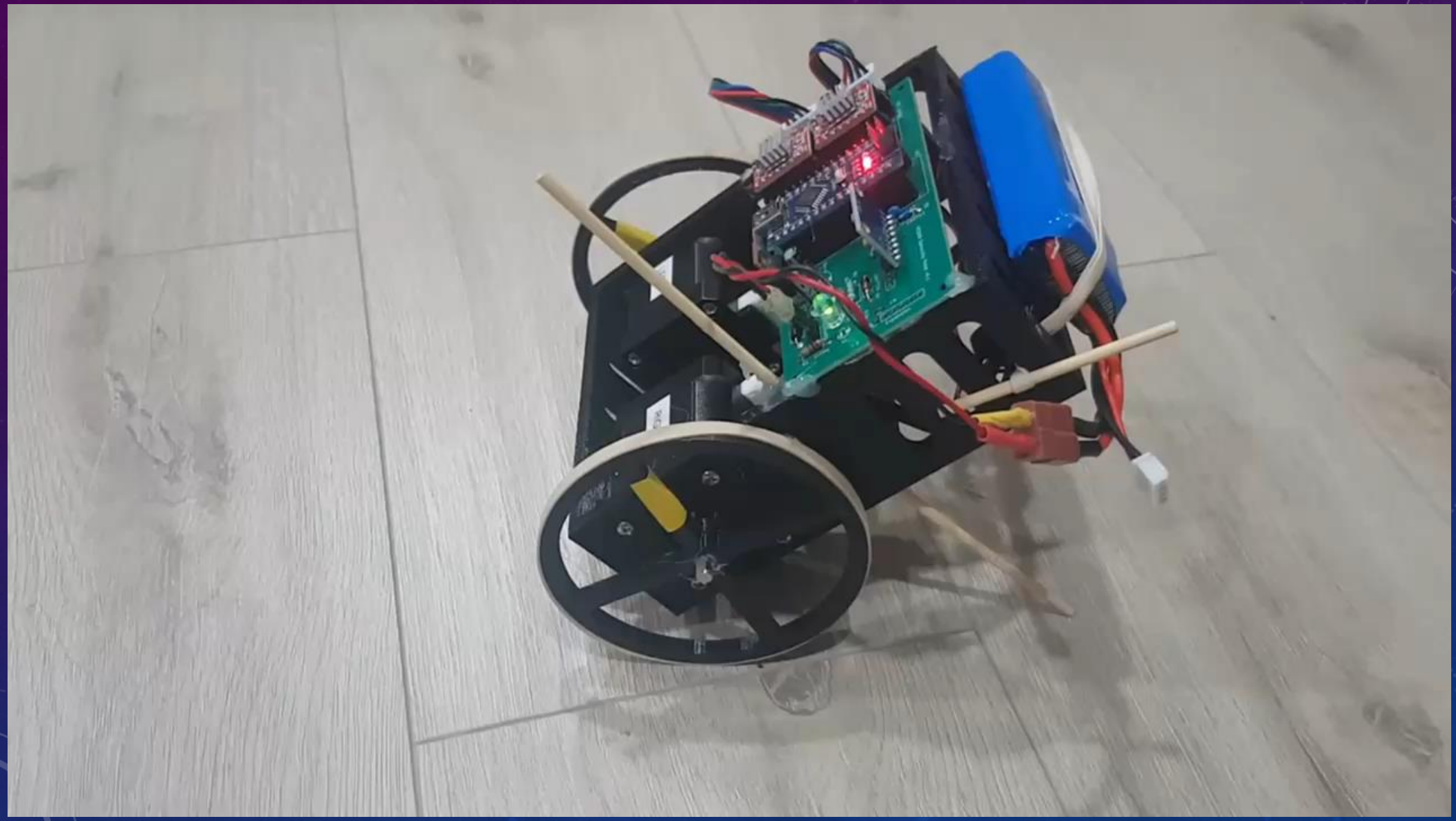
Wanted to Learn KiCad

- ✓ Created Schematic
- ✓ Created PCB with Autorouter (yes there is one for KiCad)
- ✓ Sent gerber to JLCPCB for 5 board for $4.87 US ($6.63 CDN)
- ✓ Boards worked PERFECTLY!

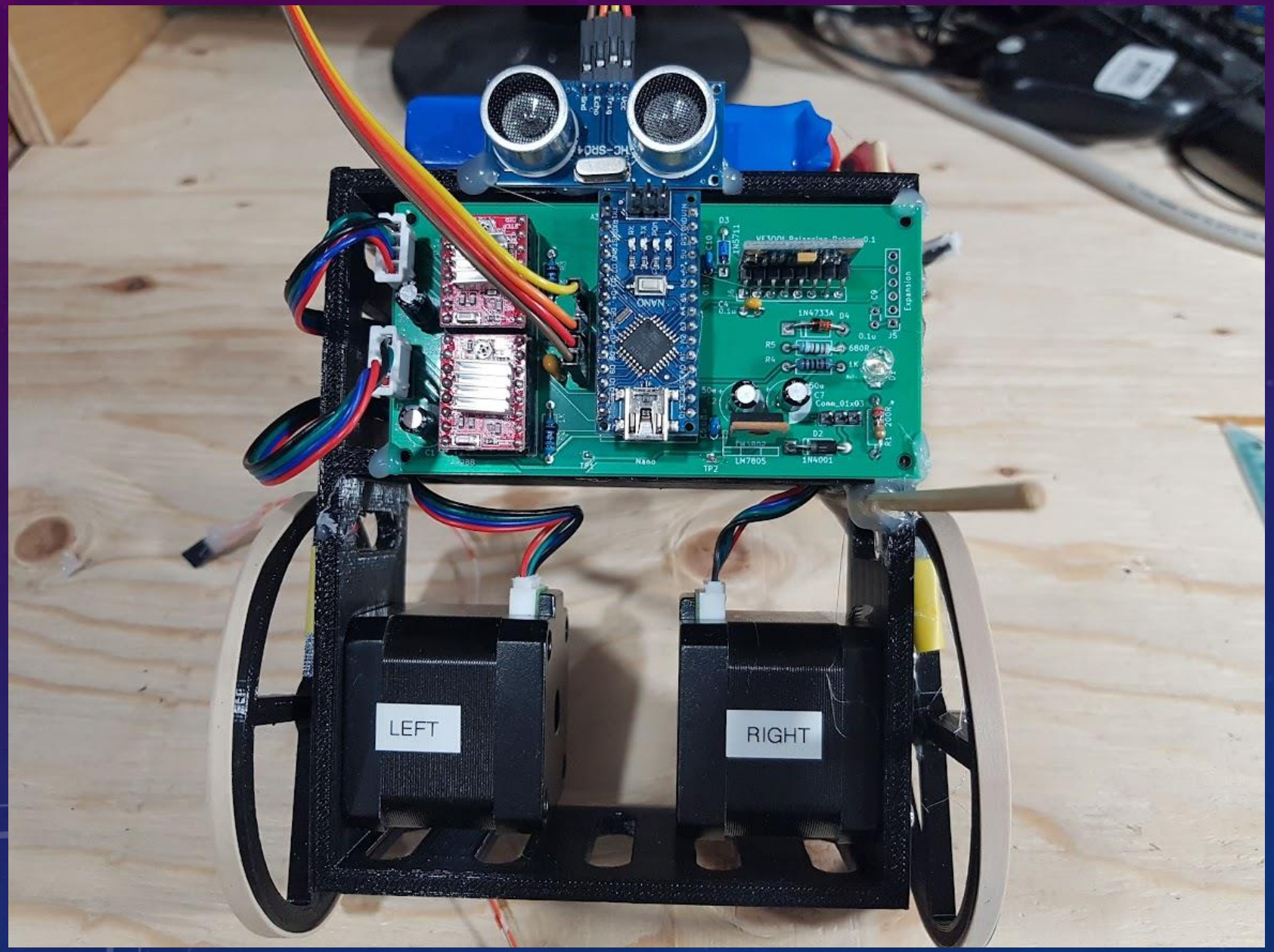
INITIAL POWER ON TEST

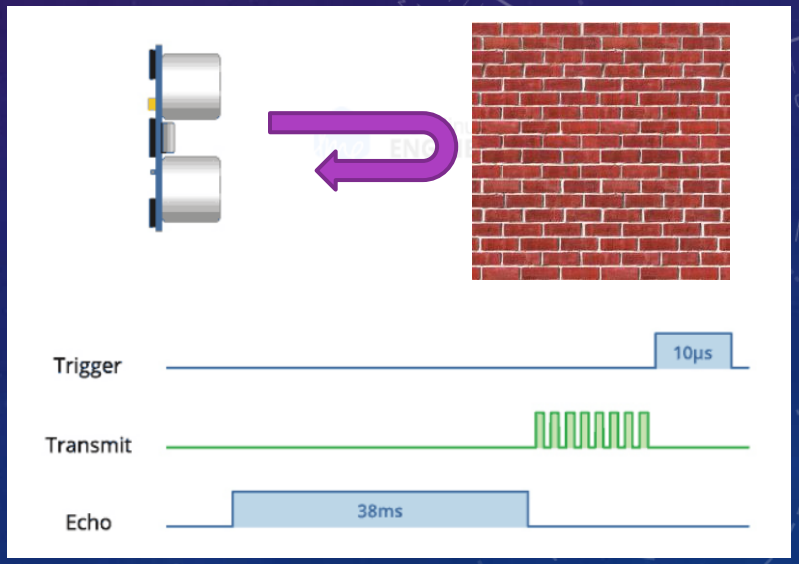


WORKED PERFECTLY!



WHAT'S NEXT... AUTONOMOUS ROBOT





CODE: RANGE FINDING

20us Timer ISR
(Toggles Trigger Pin)

```
ISR(TIMER1_COMPA_vect) {

// Stepper code is here...Not shown

  if (flags & MEASURE_DISTANCE) {
    tel.measureCounter++;
    if (tel.measureCounter <= 1) {
      PORTD &= 0b11111011;
    } else if (tel.measureCounter == 2) {
      PORTD |= 0b00000100;
    } else if (tel.measureCounter == 3) {
      PORTD &= 0b11111011;
      tel.measureCounter = 0;
      flags &= ~MEASURE_DISTANCE;
    }
  }
}
```

```
ISR(TIMER2_COMPA_vect) {  //
  if (!(flags & SAMPLE_MPU)) {
    flags |= SAMPLE_MPU;
  }

  if (flags & ENABLE_MSTIMER) {
    tel.msCounter++;
    if (tel.measureDistanceCounter++ >= 14) {
      tel.measureDistanceCounter = 0;
      tel.measureCounter = 0;
      flags |= MEASURE_DISTANCE;
    }
  }
}
```

4ms Timer ISR
(Trigger Measurement ever 60ms)

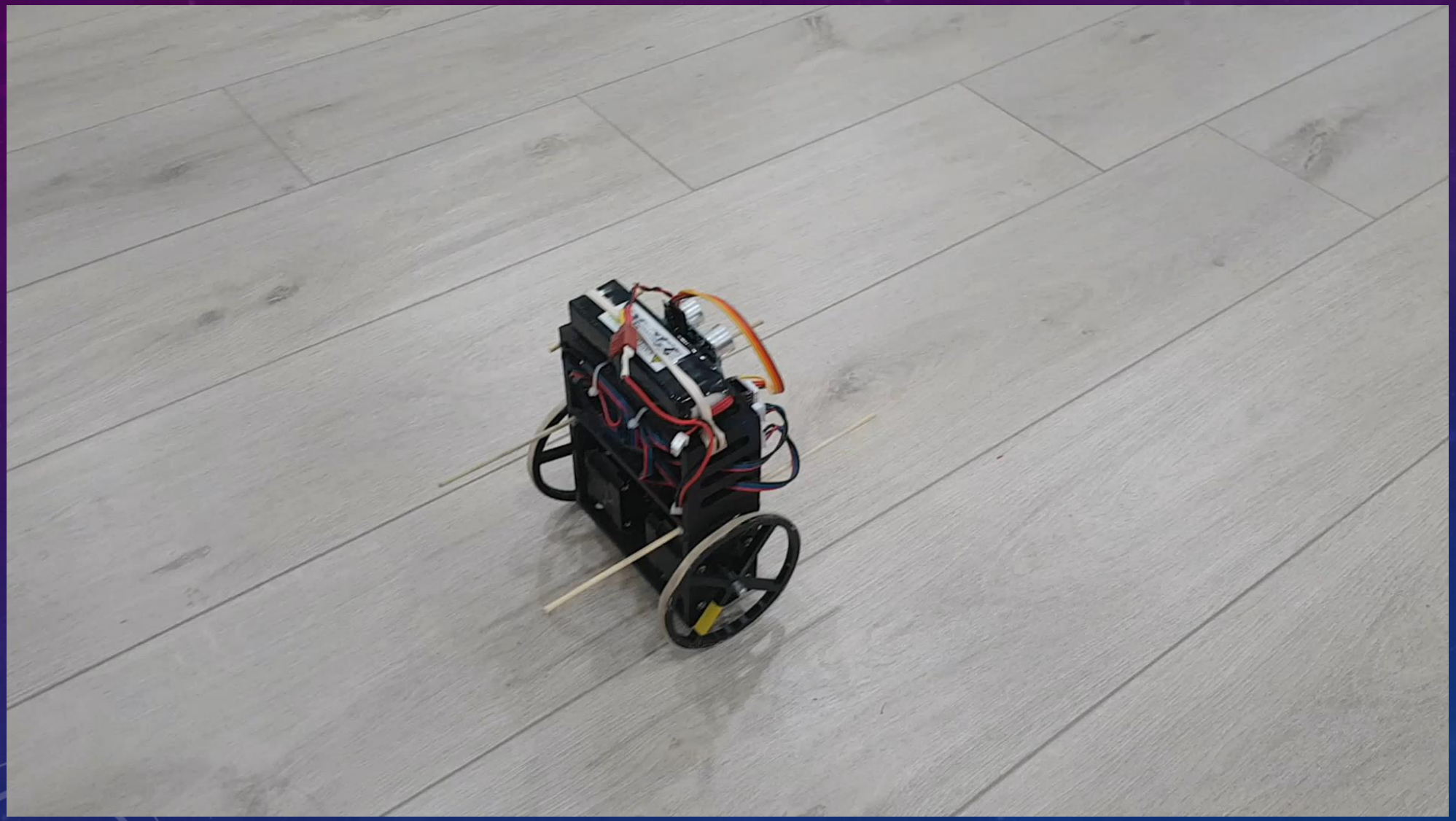
Ranging

```
bool checkDistance(void) {
  if (tel.enableDistanceCounter) {
    tel.distance = tel.distance + (tel.pulseWidth / 2) / 29.1;
    tel.distance /= 2;

    if (tel.distance <= 40) {
      if (tel.stopCounter++ >= 4) {
        tel.stopCounter = 0;
        tel.distance = 200;
        return false;
      }
    } else if (tel.stopCounter) {
      tel.stopCounter--;
    }
  }
  return true;
}
```

```
ISR(INT1_vect) {
  if (PIND & 0b00001000) {
    tel.pulseWidth = micros();
    tel.enableDistanceCounter = false;
  } else {
    tel.pulseWidth = micros() - tel.pulseWidth;
    tel.enableDistanceCounter = true;
  }
}
```

PIN Change ISR
(Measures pulse width)

ONE MONTH LATER.... #5 IS ALIVE



1. Drives forward
2. Checks distance
3. If < 40cm Stop and Turn ~90º
4. Pauses
5. Rinse and Repeat

Running Joop Brokking's PID code (http://brokking.net/)


Fin