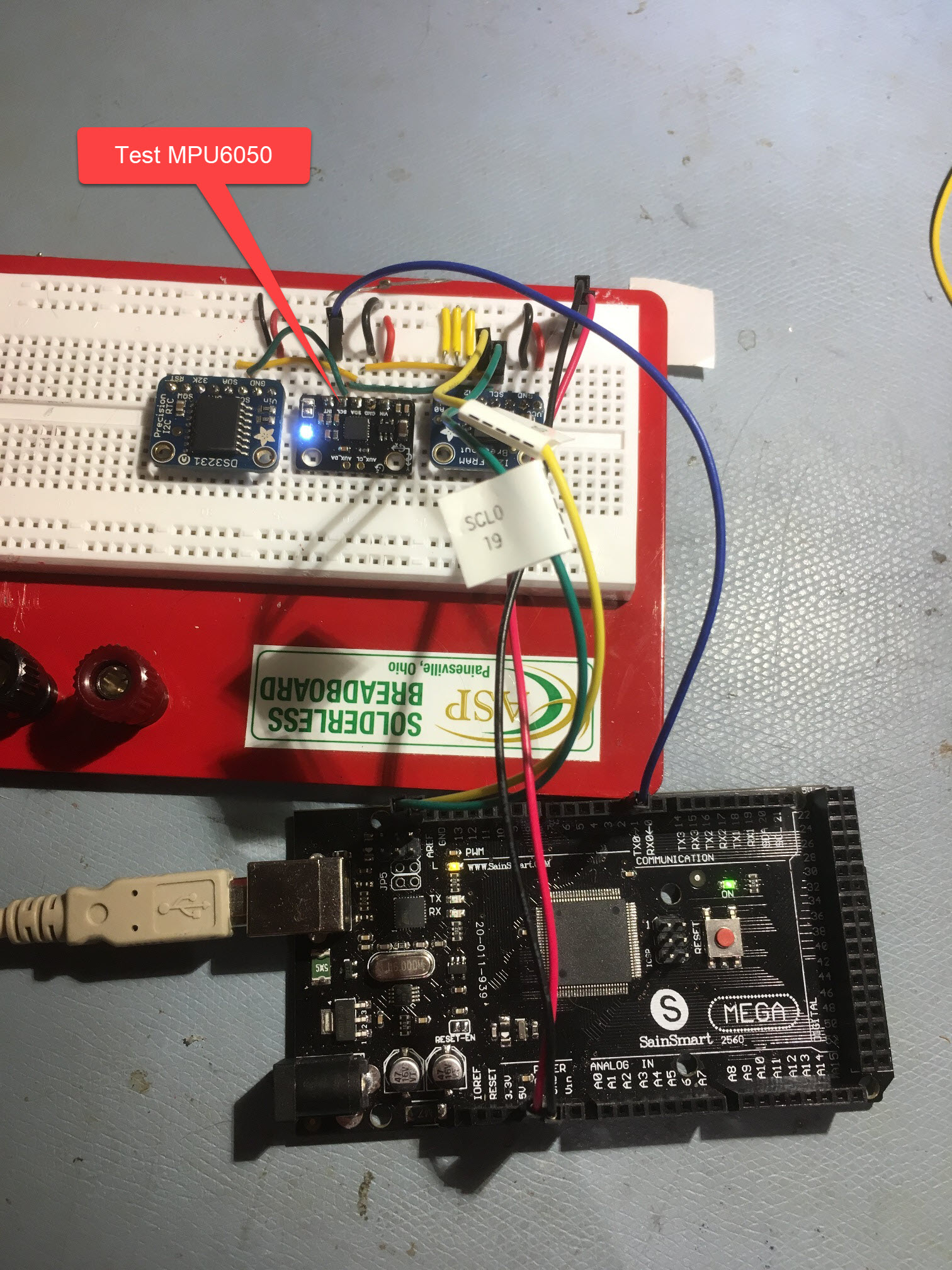
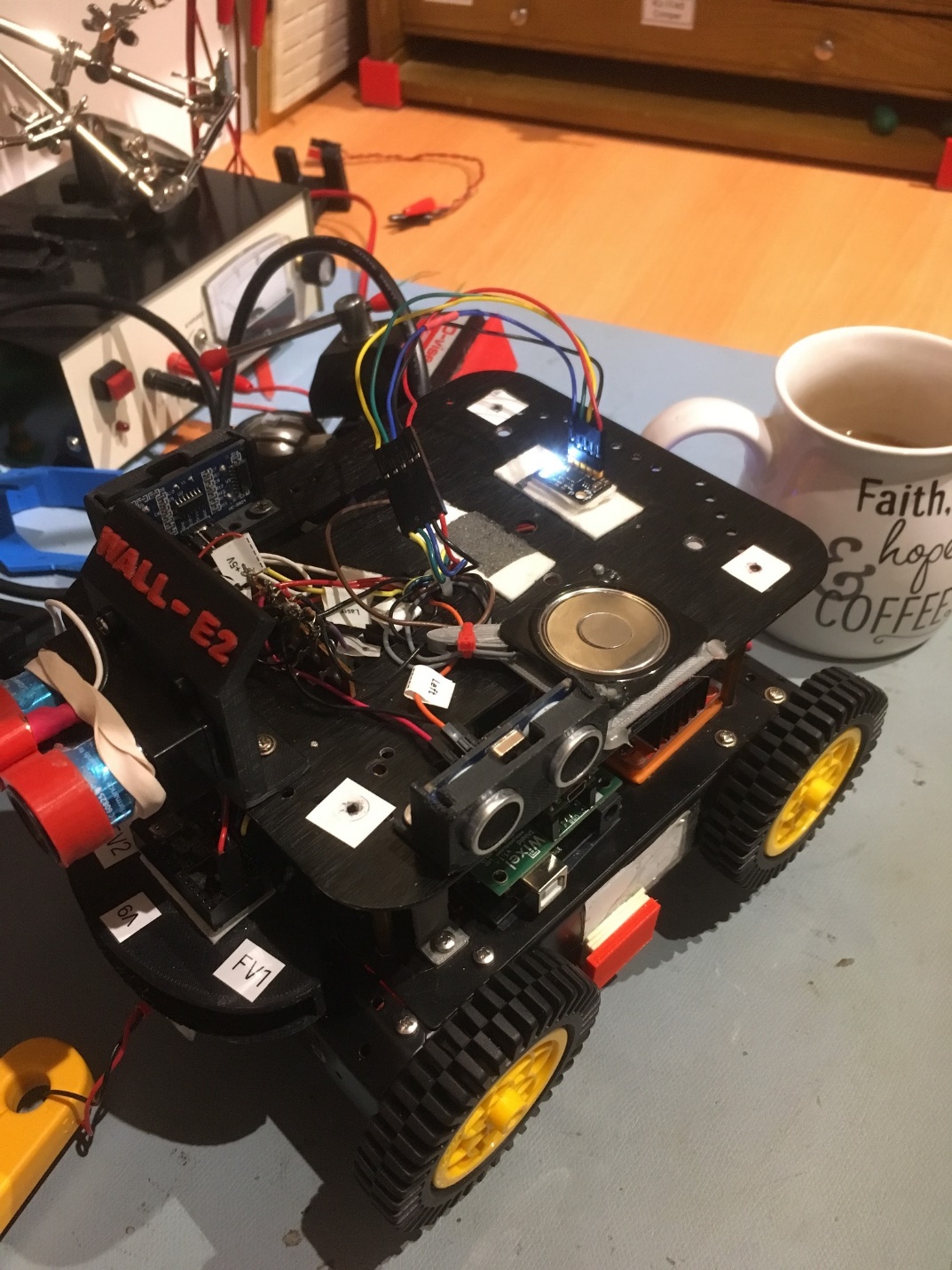
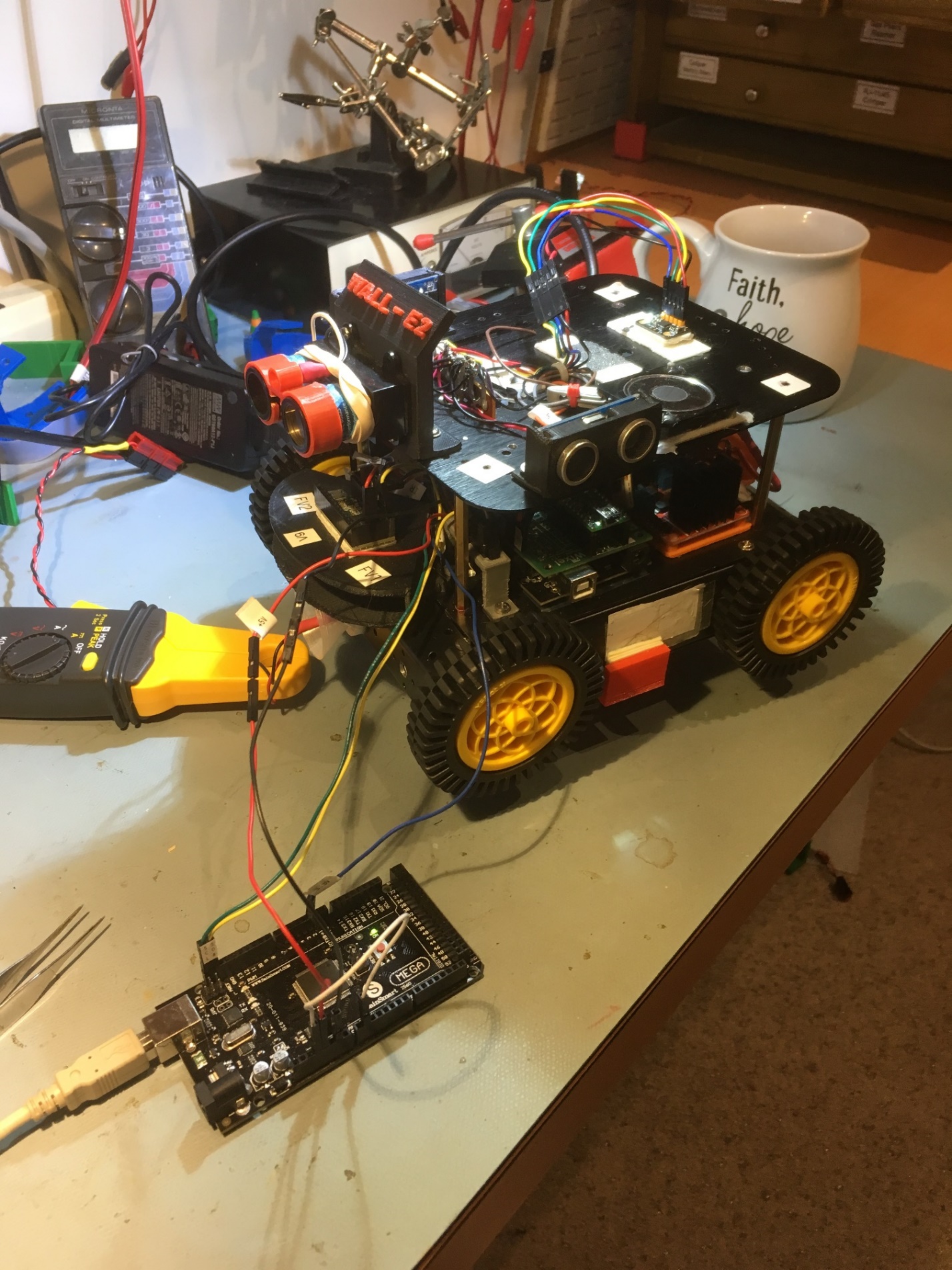
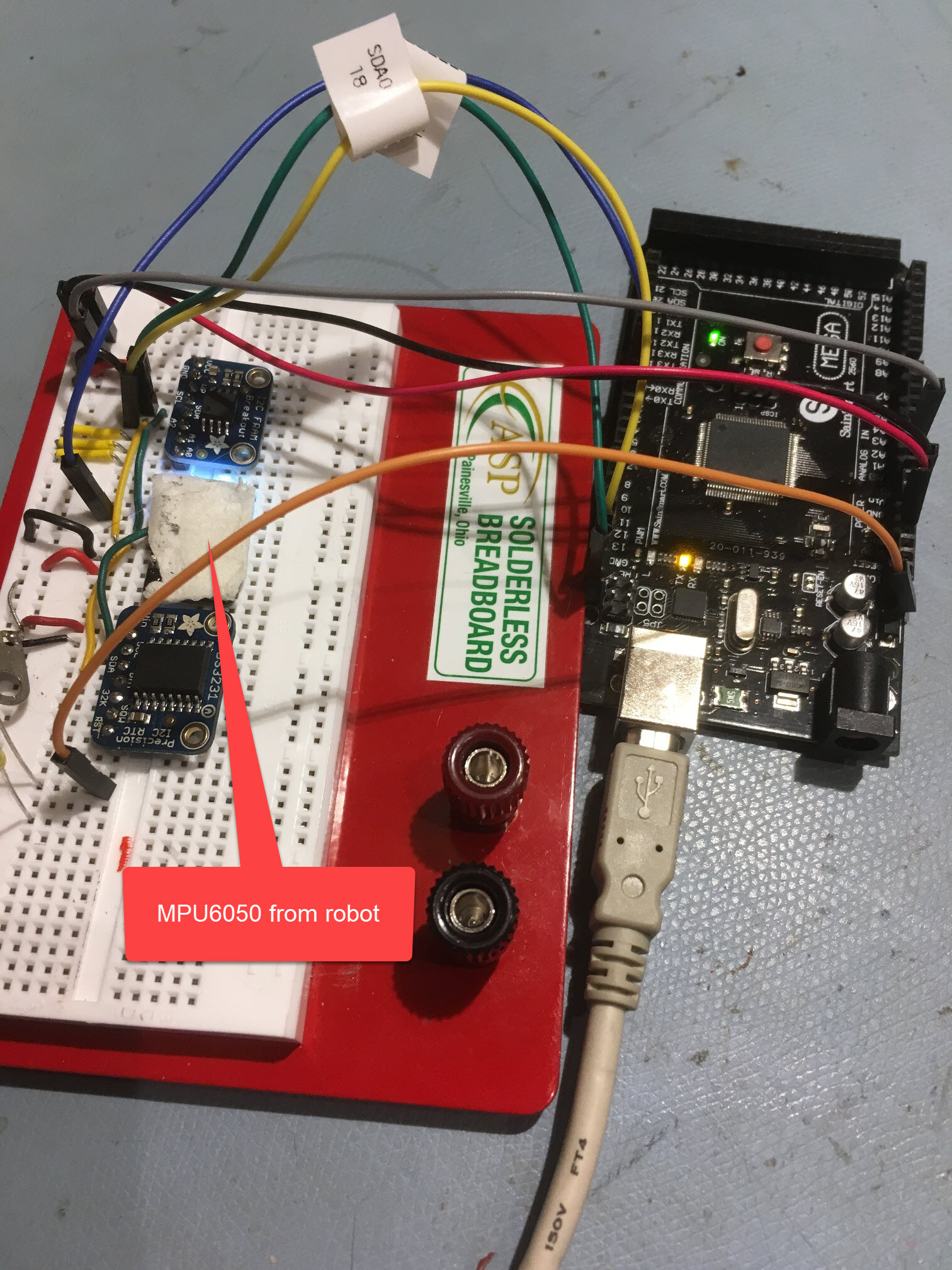
Wall-E2 Troubleshooting

07/19/18:

* I can run a stock Arduiono Mega with an AdaFruit I2C DS3231 RTC and FRAM breakout boards, and a DFRobots I2C IMU6050 breakout board with my Arduino\_IMU6050\_Test3.ino sketch, and it will (at least seemingly) run forever (on the last run I stopped it at 4649 seconds or 78 minutes). However, if I run the same sketch on Wall-E2’s Mega 2560 with just the DFRobots IMU6050 breakout, it regularly hangs after a few tens of minutes.
* I connected the Mega’s +5V supply to one of the analog inputs so I could print it out each pass, and it appears correct and doesn’t appear to sag or be noisy on either the test setup or on the robot.
* 
* 
* Next, I disconnected the IMU6050’s 2nd-level wiring from the Robot’s Mega and reconnected them to the test Mega. Now the only change is from the robot’s Mega to the test Mega – all the wiring (with the exception of two short jumpers for the +5V & GND lines) is unchanged from the ‘on robot’ case.
* 
* I ran this case for quite a while, with the robot powered down. It finally hung up at 3342/60 = 55.7 minutes. I ran it again under the same conditions, and this time it hung up in 658/60 = 11 min. On a third run, it hung up at 638/60 = 10.4 min

07/20/18:

* After doing some more web research, I ran across a long-running post [here](https://github.com/jrowberg/i2cdevlib/issues/252) that discussed the problems with I2CDevLib and the MPU6050. Seems I’m not the only one having random hangups with this combination, and it seems likely this is due to a combination of a badly-written blocking while() loop in Wire.cpp and a bug in the 6050’s DMP (Digital Motion Processor) chip that can (and eventually does) cause the SDA line to latch low. This situation causes Wire.cpp to block, and the combination causes an infinite deadlock condition.
* I also found at least one post that suggested the MPU6050 problem was due to letting the AD0 line float high or low, rather than specifically hard-wiring it to VCC or GND. In the particular 6050 breakout board I have, there is a jumper to GND which, if removed, causes the ADO line to be pulled HIGH through a pullup resistor. I have removed the jumper as I need the HIGH address (0x69) vs the default 0x68. I’m not sure how much credence to put on this post, as the test Mega and the ASP plugboard combination appears to run forever with no problems – it’s just the robot’s physical setup (25-30 cm wire length) that seem to be the critical factor.
* With nothing else changed, I swapped the IMU6050 boards between the ASP board and the robot, and made some more runs (test Mega running ASP MP6050 using robot wiring harness). 1st one hung up at 23.1 min; 2nd at 6.3 min, 3rd at 1.3 min, 4th at 8.77 min.
* With nothing else changed, I made some runs with the test Mega, and the robot’s MPU6050 now on the ASP board. The only real change here is from the robot’s 25-30 cm wiring harness length to the ASP board’s 10 cm jumpers. I started it at about 10pm and it was still running fine the next morning, after 12 hours! Clearly the robot’s MPU6050 is functioning properly on the ASP board with short wires. Note also that the ASP board tests also include the RTC & FRAM units, so this is almost (the Teensy running the IR homing algorithm isn’t included) the worst case I2C configuration.
* 

07/21/18:

* So, my conclusion (so far) is that the long cable length on the robot is the cause of the hangup problems. Cable length issues are a well-known limitation for I2C use, and the usual remedy is to either reduce the cable length, reduce the value of the pullup resistors, reduce the I2C clock frequency, or some combination of all three
  + Reduce cable length: This would be hard to do on the robot without moving the sensor(s) from the 2nd deck to the first. While this is possible, it would be a real PITA and basically destroy the effectiveness of the 2nd deck structure
  + Reduce pullup resistor values. I’m not actually sure what the effective I2C bus pullup value is. I found the following information about the Arduino Mega:
    - **Hardware**  
      The hardware I2C bus on the Arduino Mega 2560 board is a 5V I2C bus, because the Arduino Mega 2560 has 10k pullup resistors to 5V for SDA and SCL.  
      The wires for I2C can only be short. Long wires will not work.  
      The worst thing is when SDA and SCL are in a flat ribbon cable next to each other. The crosstalk between SDA and SCL will make the I2C very unreliable.  
      Because it is a 5V I2C bus, you may not connect 3.3V sensors to that 5V I2C bus. In that case you need a level converter.  
      The total value of all pullup resistors in parallel should not be too high (more sensitive for electrical noise, longer wires are not possible) and not be too low. The I2C specificiation is maximum 3 mA pulling of down current to get the signal low.
    - I believe that the MPU6050, the RTC and the FRAM also have pullup resistors. From the datasheets, the Adafruit DS3231 RTC and FRAM units both have 10K pullups, but the DFRobots MPU6050 doesn’t appear to have any pullups installed. The Teensy 3.2 I’m using for IR homing doesn’t have any pullups as far as I can tell. I found [this post](http://www.dsscircuits.com/articles/47-effects-of-varying-i2c-pull-up-resistors) showing the effect of varying the pullup values and suggesting that values as low as 2-3K are reasonable, with 4.7K being a popular single-device value.
  + Reduce I2C clock speed. I found this information on the Adafruit forum:
    - I've run them at 50K, but I believe it will go down to at least 10KHz. You can try lower. It won't hurt anything.
    - They are both one and the same. One is easier to remember though (I'll let you work out which).
    - Wire.setClock(frequency) just (ultimately) does:
    - TWBR = ((F\_CPU / frequency) - 16) / 2;
    - You can set any frequency that ends up with that formula generating an integer (for TWBR) between 0 and 255.
    - So, for TWBR = 128, we get 128\*2+16 = F\_CPU/frequency, or frequency = F\_CPU/(128\*2+16) = 16000000/272 = 58,823.5294117 – a not-so-nice number. Trying frequency = 50,000, we get TWBR = ((1600000/50000)-16)/2 = 152, which sounds OK.

07/22/18:

* Yesterday I looked at the SLC & SDA lines with a scope, both on the ASP plugboard (short wires) and the robot (long wires), thinking I would see significant differences. However the scope traces for both the SCL & SDA lines were virtually identical for both cases. However, there must be **some** differences between the two, as the ASP (short wire) configuration will consistently run forever, and the robot (long wire) configuration will just as consistently hang up after a few 10s of minutes.

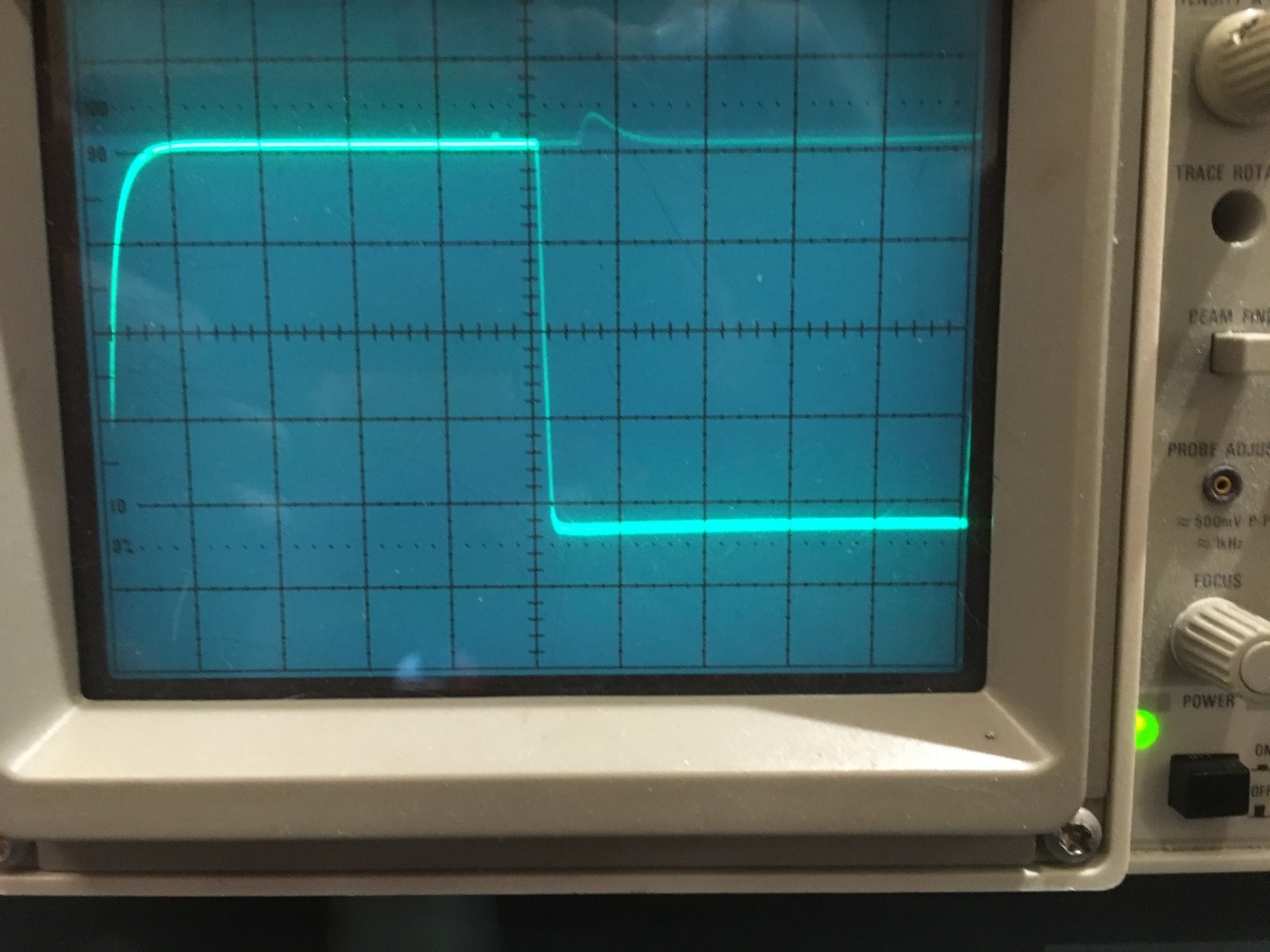


Figure 1: ASP Plugboard SCL line. 10uSec/cm, 1V/cm

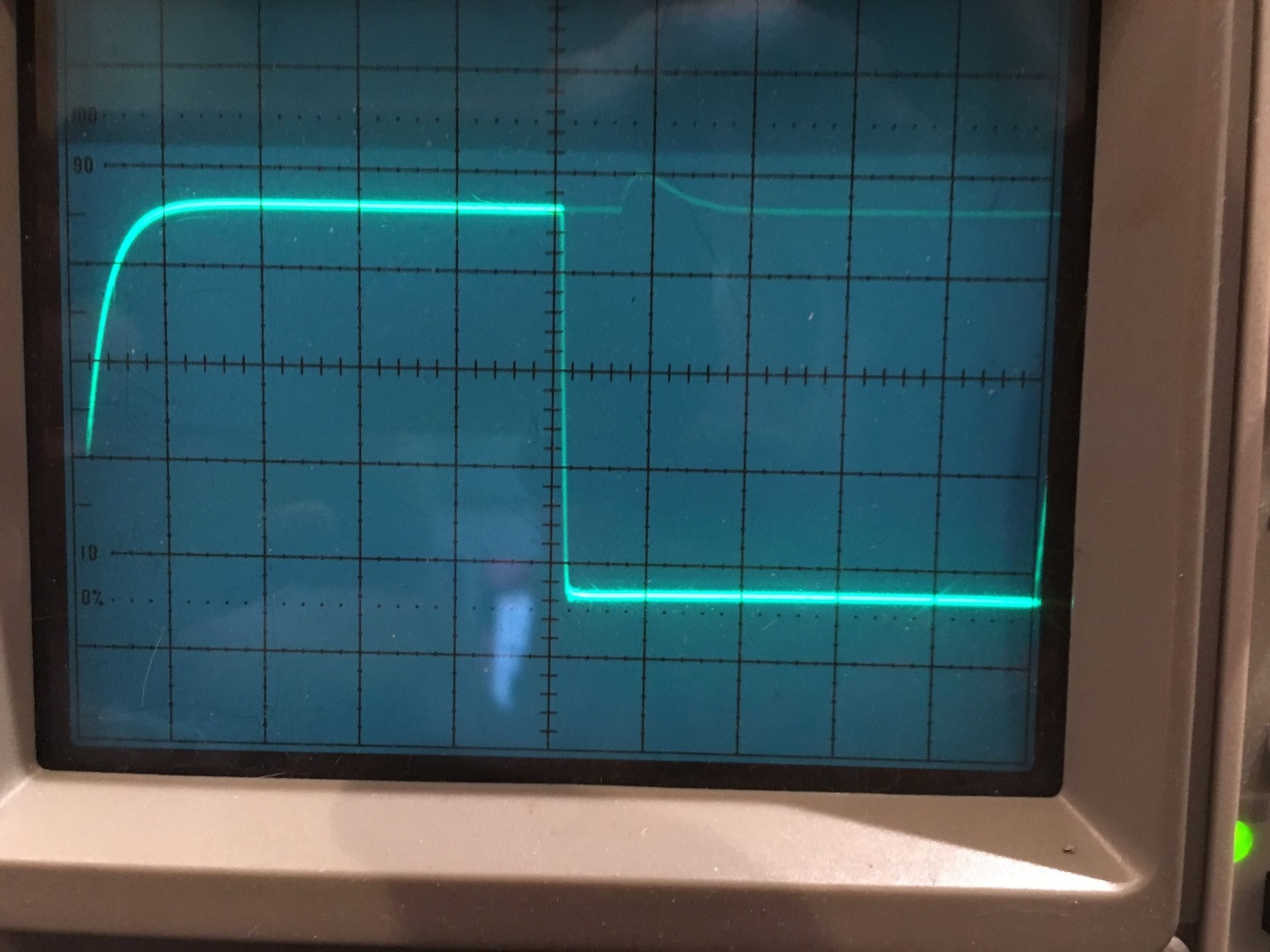


Figure 2: Robot Plugboard SCL line. 10uSec/cm, 1V/cm

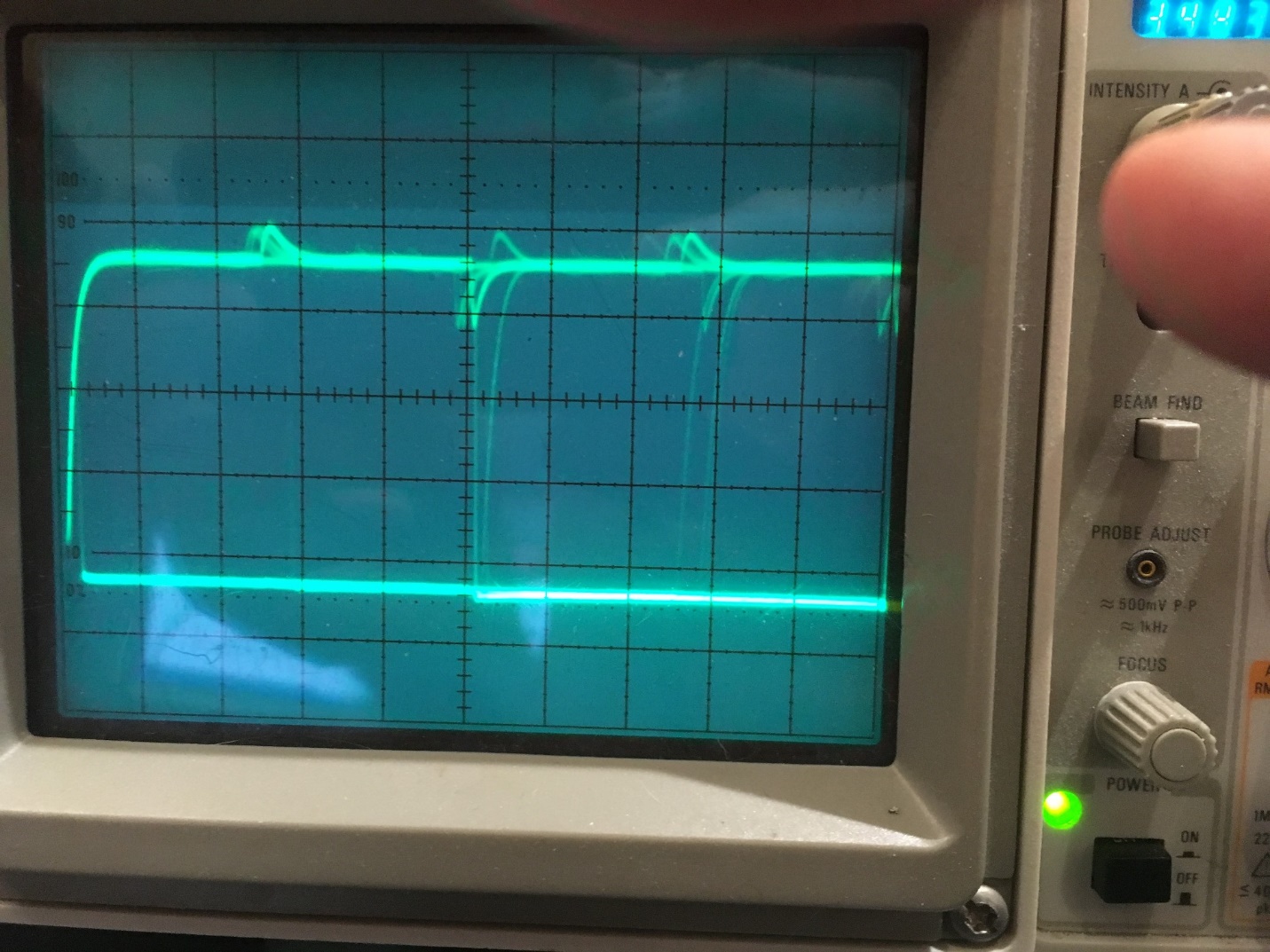


Figure 3: ASP Plugboard SDA line. 10uSec/cm, 1V/cm

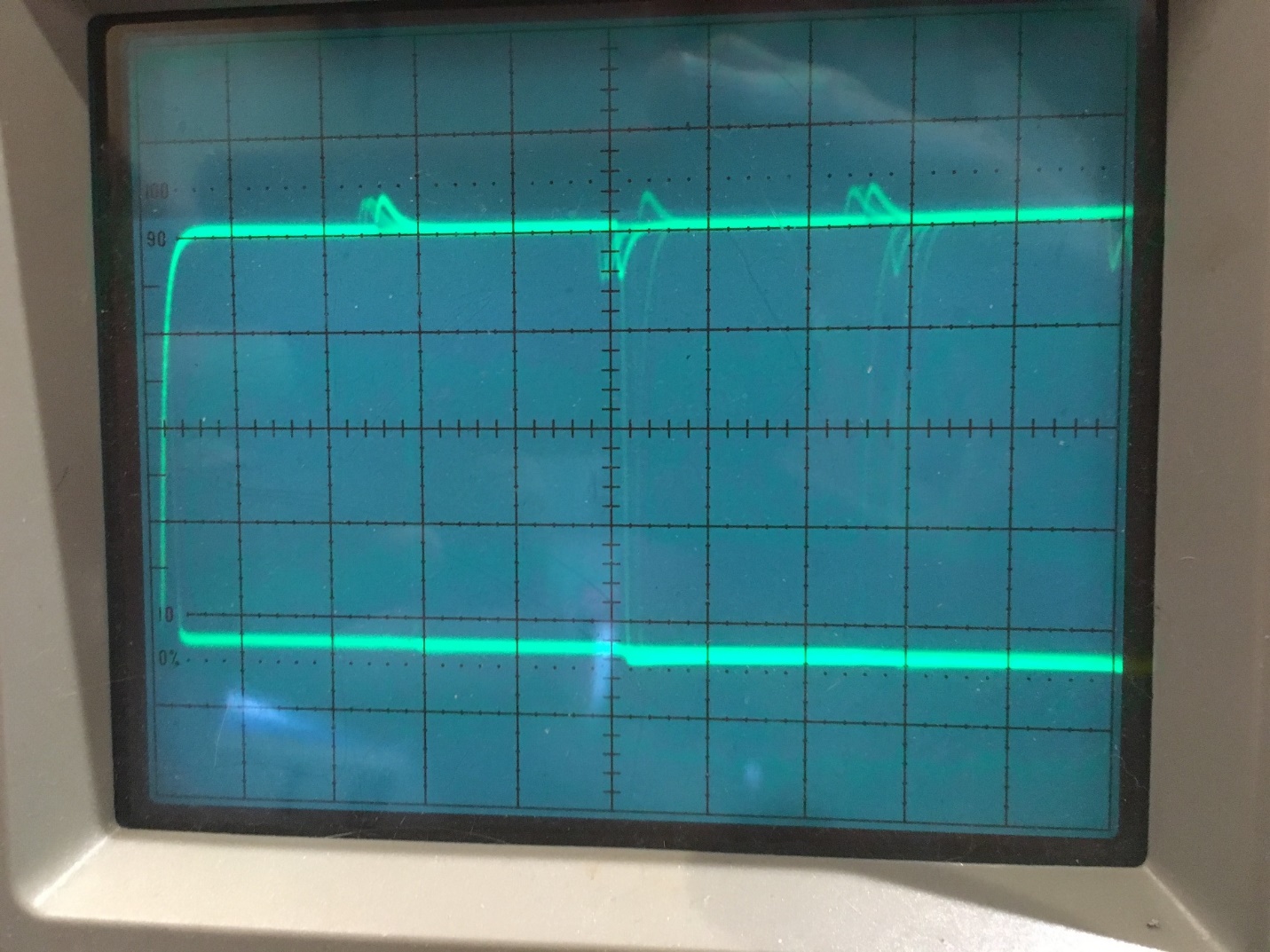


Figure 4: Robot Plugboard SDA line. 10uSec/cm, 1V/cm

* Next, I decided to try a lower I2C bus clock speed, to see if that affected the robot (long wire) configuration stability. Per my calculations above, it appears I can use a wire.SetClock(50000) to produce a TBWR value of 152. I added the call just after the Wire.begin() call in Arduino\_IMU6050\_Test3.ino. As the ‘before’ and ‘after’ scope photos show, the I2C clock was indeed slowed by a factor of 2 by the ‘Wire.setClock(50000)’ instruction. I let this run for a while to confirm stability for the ‘short wire’ configuration. I did note, however, that I am seeing a slow increment in the ‘reset count’ for this configuration – about 10 resets already after about 4 minutes. In my previous overnight run of the ‘short wire’ config at 100KHz clock freq, this value got up to 1000 (and then stopped incrementing for some reason).
* On a different, but related subject. The Teensy 3.2 I’m using for IR homing uses the I2C bus to pass left/right steering data to the main controller, so I’m concerned this function might be negatively impacted if I lower the I2C clock freq to fix the ‘long wire’ instability problem. Looking at the Teensy & robot code,
  + In IRHomeToChgStn(), The Mega gets new IR homing data from the Teensy with

Wire.requestFrom(slaveAddr, (size\_t)32);

I2C\_readAnything(FinalValue1);

I2C\_readAnything(FinalValue2);

I2C\_readAnything(tempfloat);

LRSteeringVal = (double)tempfloat;

So, this doesn’t appear to be too clock-dependent.

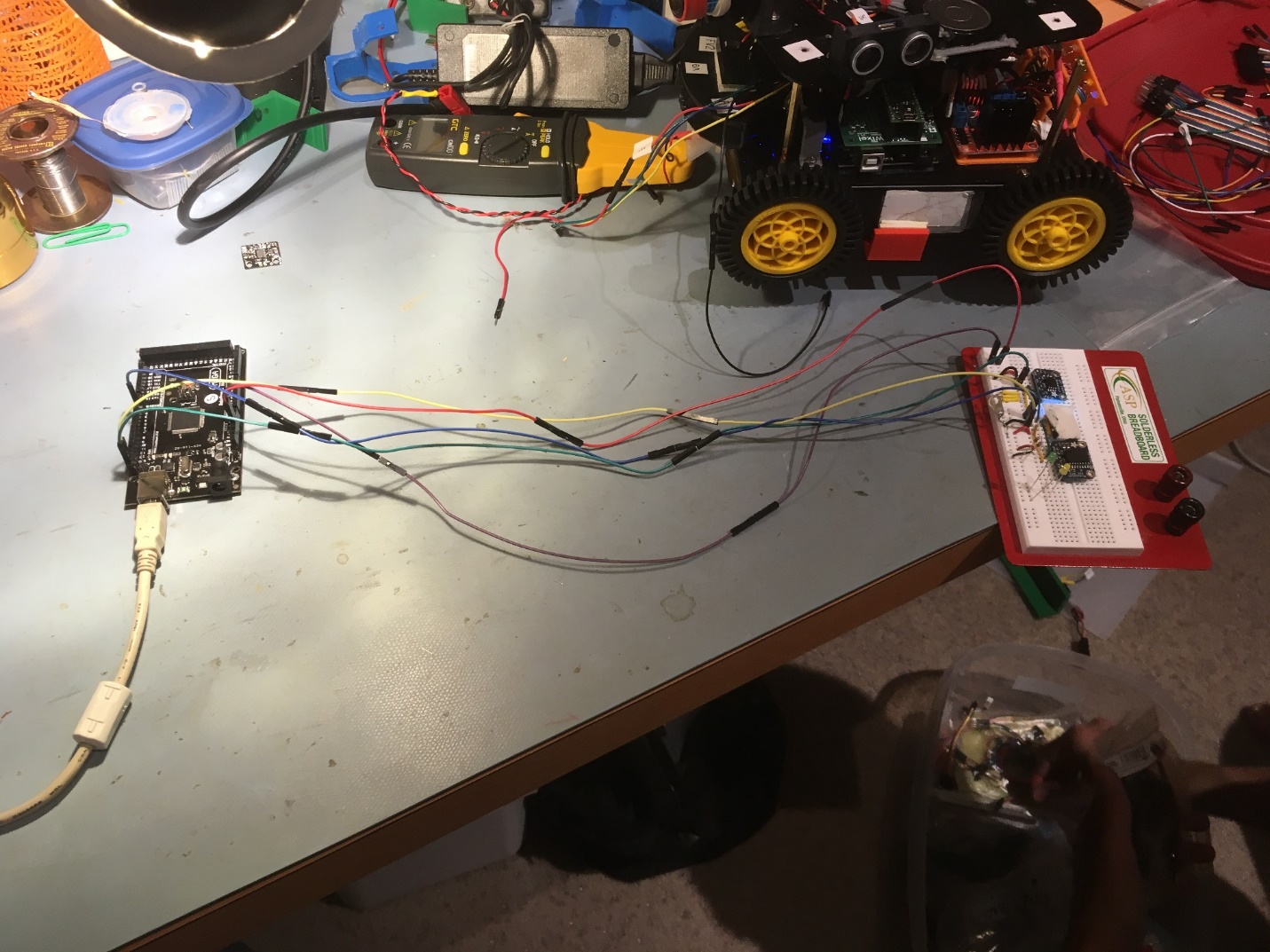
7/23/18:

* The ASP board, test Mega and robot sensor ‘short wire’ combination ran all night with no problems. The reset counter got up to 458. So, it is clear that the reduced clock rate doesn’t negatively impact the test (short wire) setup. Now to try the same trick in the ‘long wire’ robot configuration. Nope – it died at 13.25 min and 27.6 min.
* Just realized there is one other difference between the ‘short wire’ and ‘long wire’ configurations; the ‘short wire’ config also includes the Adafruit DS3231 RTC and FRAM breakout boards, both of which have 10K pullups on the I2C lines. So, switching from ‘short wire’ to ‘long wire’ configurations also switches from an effective 3.3K (short wire config) to 10K (long wire config). So, it **might** be that the entire hangup problem is just the difference in effective pullup resistor values. To test this, I can simply replace the sensor currently on the robot with the entire ASP plugboard, which will put the 3-sensor arrangement (with its included pullup resistors) on the far end of the long wire configuration, with the test Mega on the near end, as shown in the following photo.
* Unfortunately, this died after 23.7 min, so it **still** has something to do with the wire length. Added 5.1K pullups to the plugboard and started it again. This time it ran for just over 4 hours before hanging up. Changed the 5.1K pullups to 3.9K and tried again. Only 22 min this time. Tried again, and it hung up after 92 min.
* Changed back to the ‘short wire’ configuration, and started up again. This time it was still running fine the next morning, after 10 hours.

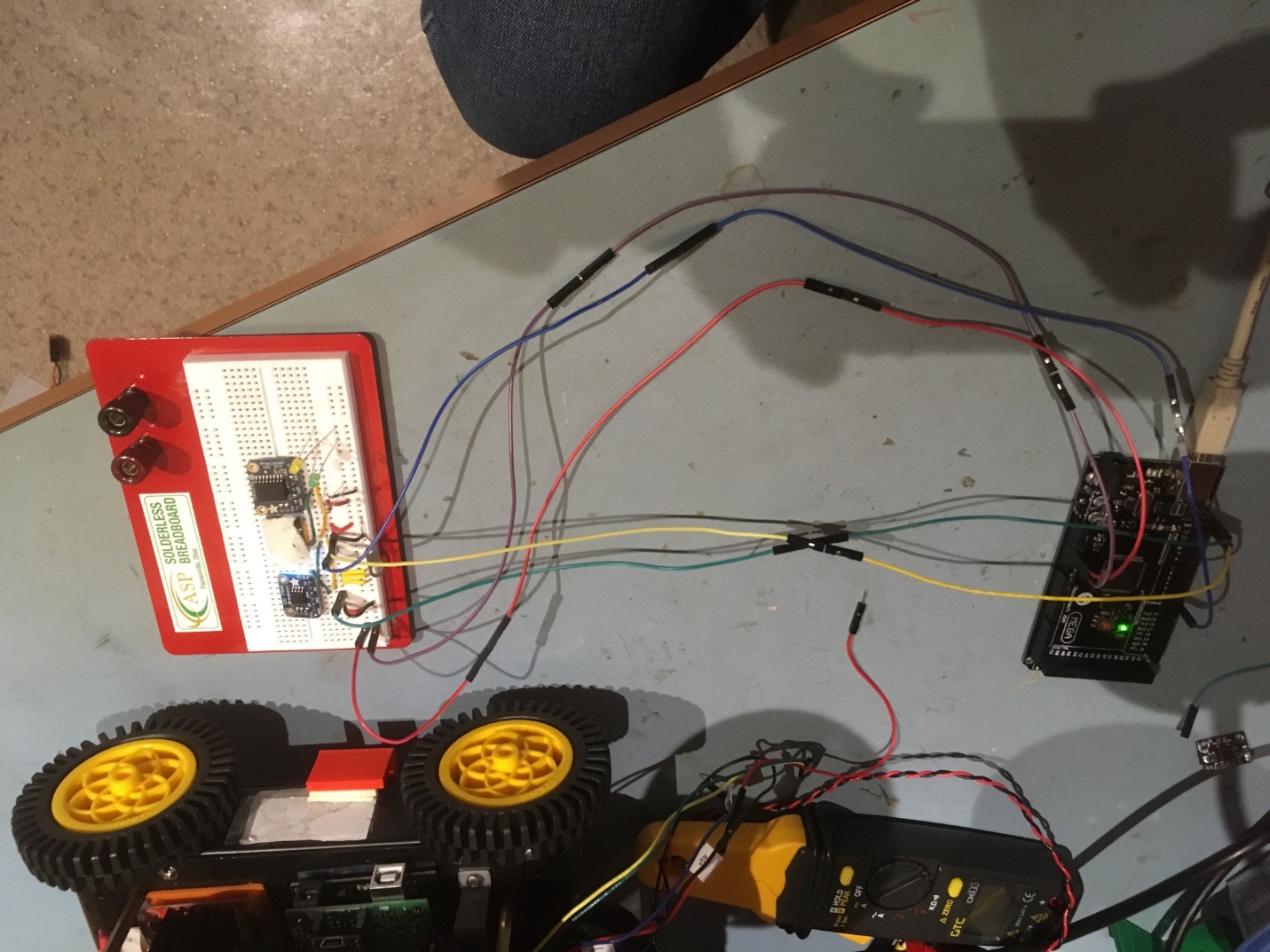
07/24/18:

* So far, I have done the following:
  + Reduced the clock speed by a factor of two. This had no effect on the problem; still hangs up on the robot, runs fine in the ‘short wire’ configuration
  + Reduced the pullup resistor value from approx 10K (just the Mega/MPU6050 combination on the robot) to 5.1K and then again to 3.9K. No effect
* I’m starting to wonder if the only difference between the ‘short wire’ and ‘long wire’ configuration is **just** the wire length. There actually are three other devices (the LIDAR and two Ping sensors) connected to power on the 2nd deck – maybe there is something going on with them that is causing intermittent power glitches?

07/25/18:

* Changed the experiment to incorporate a brand-new ‘long wire’ (~43cm) cable as shown below, completely independent of the robot. This ran for 1.9 hrs before I stopped it. It now seems probable that it’s not (just) the long cable length that is causing the problem – it is something to do with the robot, which may or may not have anything to do with the cable length.
* 
* Stopped the above experiment at 1.9 hrs and changed the I2C clock speed back to the default 100KHz and restarted. If, as expected, this experiment also runs for several hours without a hangup, then we’ll need to start looking at other potential factors besides just the cable length. Woops! This experiment hung up after about 45min! OK, so maybe we’re on to something.
* Changed back to 50KHz clock and re-ran, with no other changes. Still running after over two hours. Correction, it hung up at 2.15 hrs!
* Shortened the two I2C lines by about 10 cm and restarted. Still going at 1.75 hrs. It ran all night.

07/26/18:

* Changed the clock speed back to default speed (100KHz). Still running after 11.5 hours.
* 

So, it now appears that the cable length was, in fact, the most significant factor in the hangups experienced with my Arduino\_IMU6050\_Test3.ino test program. There is still a possibility that the pullup resistors are a contributing factor, but I doubt it and I’m not really sure I care – the pullups are good practice anyway.

* With 42 cm I2C lines, the program hangs at either 50 or 100KHz I2C bus speed, with or without 3.9K pullups
* With 30 cm I2C lines, the program runs indefinitely at either 50 or 100KHz I2C bus speed, with the 3.9K pullups (and may run without them – don’t know yet).
* The length issue is independent of anything to do with robot cabling peculiarities, excepting only the length. The hangup occurs with either the robot cabling, or a completely independent cable setup, as shown in the above photos.