Even a cheap inkjet printer knows when it's out of ink, but even relatively expensive 3D printers have no ability to detect filament feed issues. This is perhaps the most glaring design deficiency in the 3D printer world. Most add-on filament detectors are able to detect the presence of filament, but still cannot detect failure-to-feed situations. In our shop, we have wasted hundreds of dollars' worth of filament on large-format prints that failed due to a feed failure rather than a runout, so a good solution offers significant cost savings for our FDM operations.

In this application, the most straightforward way to reliably detect motion is to use magnets and a Hall-effect sensor. These are inexpensive and readily available, in configurations tailor-made for the Raspberry Pi architecture. We developed two sample designs; parts cost on either one is less than $20, and either one can be used in conjunction with, or instead of, a traditional switch-based filament sensor. The STL files can be downloaded at https://bit.ly/2AnIDVx.

Because this configuration produces a series of on/off pulses, it can't be monitored for a "true/false" status like a traditional switch-based filament detector. The monitoring must involve a "wait for first motion" sequence to avoid false triggering during the print initialization process (bed leveling, heating, etc.), followed by a timeout-based pulse detection loop. Each pulse should reset the timer, and if a pulse isn't detected within the timeout period, the print should be paused and some sort of notification should occur. An example shell script is included at the end of this document. If you're using the Enclosure Plugin (and if not, why not?!), you can configure a button to trigger the shell script, or you can just start it via init.d or in a terminal session. If you use an Enclosure Plugin button, you'll need to use a wrapper script (also included below) that starts the shell script and exits cleanly; otherwise, Enclosure Plugin will be hung in limbo waiting for the script to exit…which means that the OctoPrint web interface will not accept new connections as long as the script is running.

The optimal timeout value will depend on your specific setup. Obviously, for a given print, 3mm filament through a 0.5mm nozzle moves a lot slower than 1.75mm filament through a 0.8mm nozzle. The script keeps track of the longest pulse time, though, so you can set the timeout fairly long, maybe 30-45s for 1.75mm filament, 240-300s for 3mm filament, and then check the log file at the end to see the actual longest pulse time, which will help you figure out a good timeout value for your configuration. Another option would be to modify the script to:

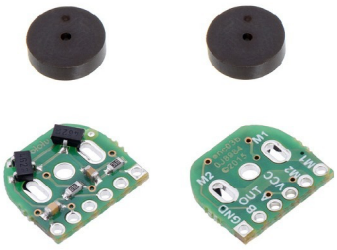
1. Calculate the length of filament that has gone through the sensor, based on the feed length per sensor pulse, which will depend on your setup. This would probably work best with Design 1 below; too many feed length variables with Design 2.
2. Query the OctoPrint API (/usr/bin/curl -s -H "X-Api-Key: YOUR_API_KEY" http://localhost/api/job) to get the "length" value (total filament needed for the job) and the "completion" value (% job complete), and use those to calculate the length of filament that should have been fed.
3. Compare the two numbers, and assume there's a feed issue if they don't match within some tolerance.

It would require some testing to determine whether this method would detect feed problems faster than a single-value timeout.

It would also be nice to convert the script to python, so it could be more easily incorporated into an OctoPrint plugin (either the Enclosure Plugin or a distinct project).

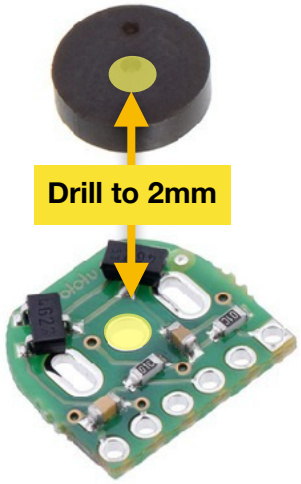
Design 1: Filament Motion Sensor

This design monitors the filament itself, similar to a traditional filament sensor. This requires less customization, so it may be easier to implement, it can probably detect failure-to-feed issues faster, and it definitely works better for slow-moving filament.



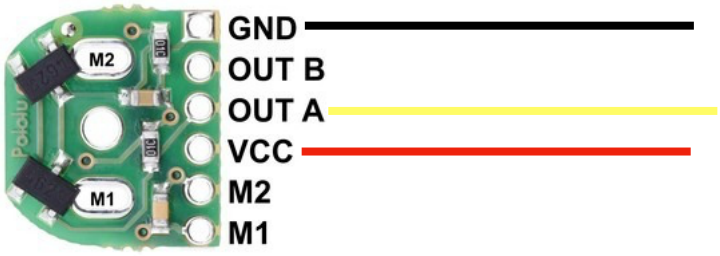
Procure a magnetic encoder pair kit (it includes two sets)…
https://www.pololu.com/product/3081

…and some #4x3/8” (M3x10mm) pan-head sheet metal screws, e.g. Lowe’s p/n 54840.

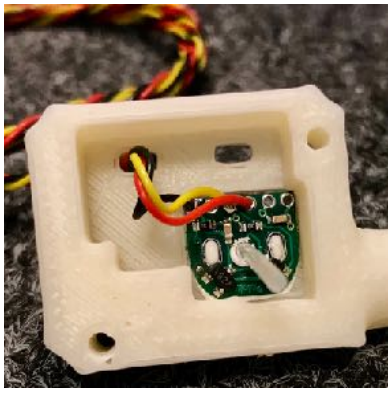


Drill out the center holes in the PCB and magnetic disk with a 5/64” (2mm) drill, so they slip over a piece of 1.75mm filament.

Solder three wires onto the PCB: GND, OUT A (or OUT B, your choice), and VCC.

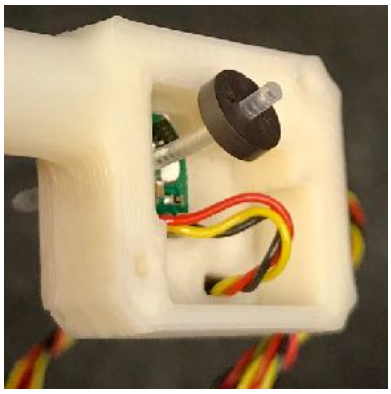


Print the sensor housing & cover. If it will be used inside a heated enclosure, use ABS, nGen, nylon, or some other heat-resistant filament (not PLA!). Nylon is preferred, to reduce filament feed friction. For 3mm filament, drill out the feed hole with a 9/64” (3.5mm) drill. The “nozzle” on the housing outlet is intended to create some distance between the sensor and the extruder, so the software has time to detect a filament runout and pause the print (hopefully) before the end of the filament reaches the extruder. If you don’t want/need it, cut it off.

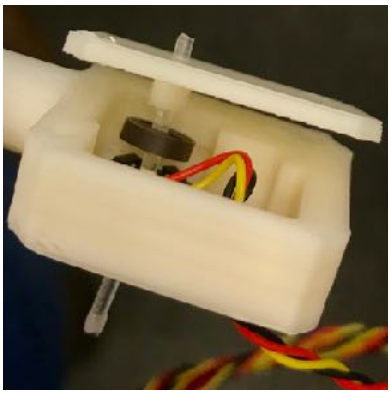


Use a dab of silicone or other non-conductive adhesive to glue the PCB into the square recess in the housing. The square end of the PCB (with the solder terminals) should cover the rectangular slot at one end of the recess. Use a piece of 1.75mm filament (nylon, if possible) to line up the PCB with the hole in the housing and serve as a shaft on which the disk will spin.

Slip the magnetic disk over the filament, and install the cover.



Secure the cover with two #4x3/8” screws. The slotted holes in the cover allow you to adjust the tension between the disk and the filament. You want just enough tension that the filament feeds reliably, and the disk rotates reliably.

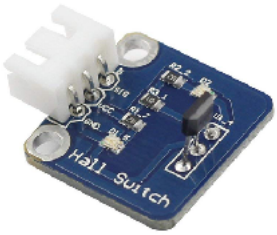


Connect the wires into your RPi. Red wire goes to 5V, black to ground, yellow to whatever pin you’re using for signal input.

Option 2: Spool Motion Sensor

This design monitors the rotating filament spool. This reduces filament feed friction, and doesn't add additional wiring & hardware to the printer's flexing arm and extruder head, but it does require a spool support that uses standard 22mm-diameter sealed bearings. Search thingiverse.com for examples. This design works best for relatively fast-moving filament.

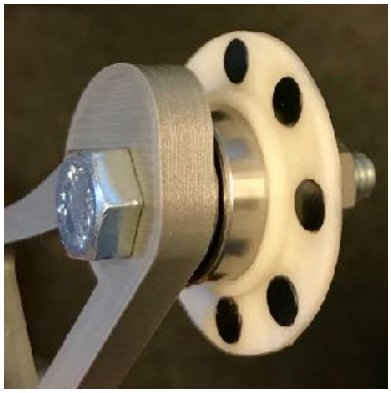
Procure some 6x2mm strong magnets (http://a.co/cG7rZ2z) and a Hall sensor module (http://a.co/iOIRnT7).

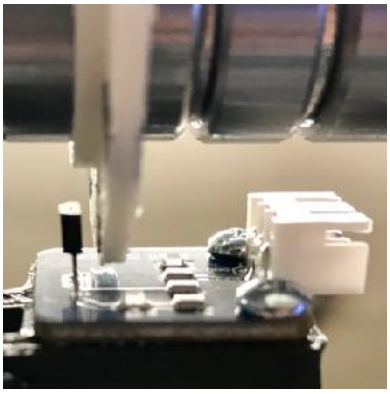


Print the magnet wheel.

Install the magnets into the wheel, making sure to orient all of the magnet poles in the same direction. An easy way to accomplish this is to let the magnets stack (so the poles are automatically lined up) and lay the wheel on a flat surface. Push the bottom magnet into a hole, then slide the stack sideways off of that magnet, to the next hole. Repeat until all the holes are filled, then coat the underside with a layer of your adhesive of choice to hold the magnets in place.

Slip the magnet wheel onto a spool support bearing that will spin reliably when the spool rotates. Make sure the wheel is aligned straight, so it spins without wobbling. Use a bit of glue to secure it to the bearing if necessary.

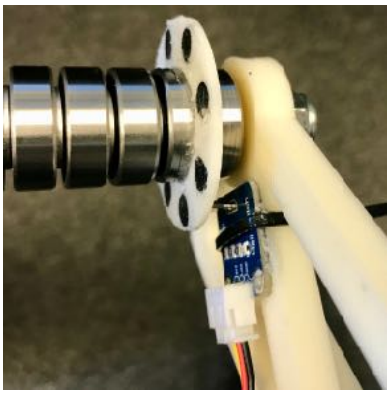


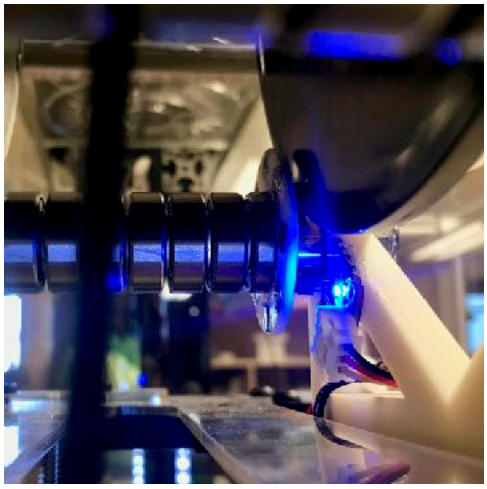


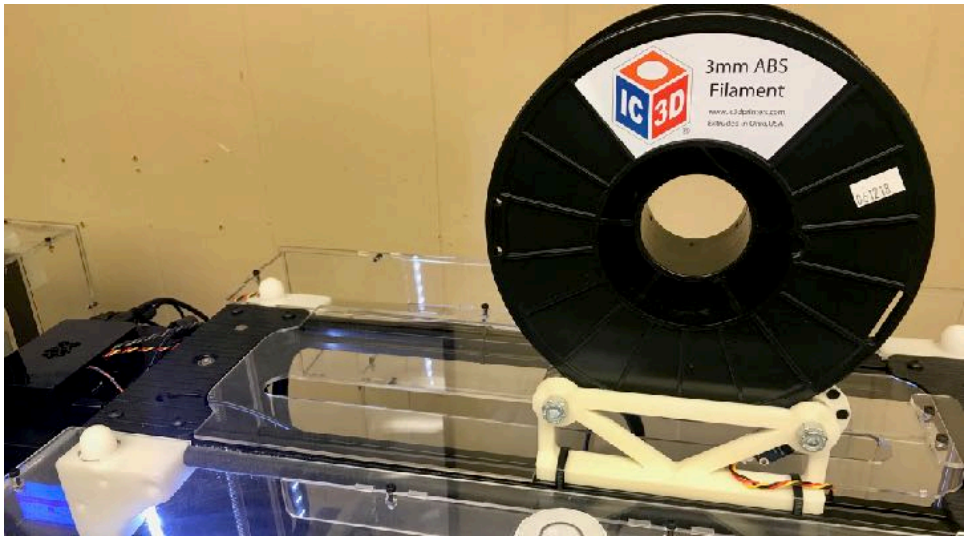
Mount the Hall sensor board in such a way that the magnets pass near the sensor as the wheel rotates. The sensor is more sensitive to one magnetic pole than the other, so you can adjust the characteristics of the output pulse by changing the orientation & distance between the sensor and the wheel.

Connect the wires from the Hall sensor into your RPi. Red wire goes to 5V, black to ground, yellow to whatever pin you want to use for signal input.

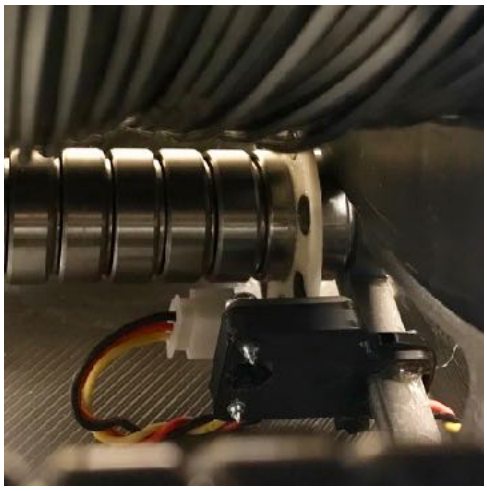
These photos show an example configuration of the wheel and sensor, on a spool support rig for our Lulzbot TAZ6. The sensor should really be mounted with one of its faces parallel with the magnets (so the PCB is perpendicular to the wheel), but this arrangement works too—it's just more sensitive to the placement of the sensor. In this setup, shifting the sensor even 1/16" in either direction causes it to stop detecting.







Another example setup on top of a gMax 1.5 XT+. This one has the "correct" sensor orientation.



Sample Shell Script

```
#!/bin/bash
#
# FilamentMonitor.sh -- Jon L. Gardner, jon@brazoslink.net
#
# Monitors a Raspberry Pi GPIO input for pulses from a filament motion sensor.
# Triggers a print pause via OctoPrint API, and optionally notifies via IFTTT.

# set timeout in seconds
timeout=$1

if [[ ! $timeout ]]; then
 /bin/echo "Warning: No timeout specified, defaulting to 30s."
 timeout=30
else
 /bin/echo "Timeout set to ${timeout} seconds."
fi

# select input pin to monitor for motion sensing
GPI=5

# OctoPrint API key
apikey=YOUR_OCTOPRINT_API_KEY

# IFTTT API key and event name (from Enclosure Plugin settings)
ifttt_key=YOUR_IFTTT_API_KEY
ifttt_event=YOUR_IFTTT_EVENT_NAME

# prepare the pins
IO=/sys/class/gpio
if [[ $GPI ]]; then
 /bin/echo "Configuring input on BCM ${GPI}."
 if [ ! -d $IO/gpio${GPI} ]; then
  /bin/echo "${GPI}" > $IO/export
 fi
 /bin/echo "in" > $IO/gpio${GPI}/direction
else
 /bin/echo "Error: No GPIO pin specified!"
 exit 1
fi

function trap_ctrlc ()
{
  /usr/bin/printf "\nLongest pulse time was $maxtime seconds.\n"
  exit 2
}

function monitor_gpio ()
{
   ok=0
   exp=0
   val1=$(/bin/cat $IO/gpio${GPI}/value)
   while [[ $ok = 0 ]]; do
    val2=$(/bin/cat $IO/gpio${GPI}/value)
    if [[ $val1 != $val2 ]]; then
     val1=$val2
     ok=1
     exp=0
    fi
    if [ $init == 0 ]; then
     ctime=`/bin/date +%s`
     etime=`/usr/bin/expr $ctime - $stime`
     if [ $etime -gt $timeout ]; then
      ok=1
      exp=1
     fi
    fi
    sleep 0.2
   done
}
```

```
function check_print_status ()
{
  # query OctoPrint for the main tool setpoint
  done=0
  setpoint=`/usr/bin/curl -s -H "X-Api-Key: "$apikey"" http://localhost/api/printer | /bin/grep -A3
tool0 | /bin/grep target | /usr/bin/cut -f2 -d:`
  # if 0.0, assume the print is done
  if [ $setpoint = 0.0 ]; then
   done=1
  fi
}

trap "trap_ctrlc" 2
init=1
maxtime=0

while :
do

 if [ $init == 1 ]; then
  # wait for the first change before starting the timer
  /bin/echo "Monitoring BCM ${GPI} for initial filament movement..."
  monitor_gpio
  /bin/echo "Initial pulse detected."
  /bin/echo "Monitoring with timeout of $timeout s..."
  init=0
 fi

 # once things are moving, detect if things stop moving for longer than the timeout
 stime=`/bin/date +%s`
 monitor_gpio

 if [ $exp == 1 ]; then

  /bin/echo "Filament has stopped moving."

  check_print_status

  if [ $done == 0 ]; then
   # pause the print via OctoPrint API (assumes localhost)
   /bin/echo '{"command": "pause" , "action": "pause"}' | /usr/bin/curl -s -X POST -d @- -H "X-Api-
Key: "$apikey"" -H "Content-Type: application/json" http://localhost/api/job > /dev/null

   # notify via IFTTT
   if [[ $ifttt_key ]]; then
    /bin/echo '{"value1": "Printer paused by filament motion sensor." , "value2": "Longest pulse
delay '$maxtime' s."}' | /usr/bin/curl -s -X POST -d @- -H "Content-Type: application/json"
https://maker.ifttt.com/trigger/$ifttt_event/with/key/$ifttt_key > /dev/null
   fi

   # wait for the timeout period, then start over
   /bin/echo "Pausing $timeout s for reset..."
   /bin/sleep $timeout
   init=1
  else
   # assume the print is finished, and exit
   echo "Print appears to be finished."
   trap_ctrlc
  fi
 else
  #  maintain record of longest pulse time
  ftime=`/bin/date +%s`
  ttime=`/usr/bin/expr $ftime - $stime`
  /usr/bin/printf "."
  if [ $ttime -gt $maxtime ]; then
   maxtime=$ttime
  fi

 fi

done
```

Wrapper Script (for starting via Enclosure Plugin)

```
#!/bin/bash
#
# startFM - wrapper script for FilamentMonitor.sh
#
# kill any running FilamentMonitor processes
while :
do
pid=`ps ax|grep -v grep|grep FilamentMonitor|cut -c1-5`
if [[ $pid ]]; then
 sudo kill $pid
else
 break
fi
done
# set timeout from command line
timeout=$1
if [[ ! $timeout ]]; then
 timeout=30
fi
cd /home/pi
# to monitor progress, ssh to the OctoPi and "tail -f fm.log"
nohup ./FilamentMonitor.sh $timeout > fm.log &
echo "FilamentMonitor started (PID $!) with ${timeout}s timeout.”
```

Sample Enclosure Plugin Output Configuration