The MagPi

Issue 81 | May 2019 | magpi.cc

The official Raspberry Pi magazine

HOW TO BUILD A RASPBERRY PI PROJECT

- ✓ Get tips from makers
- ✓ Find the right project
- ✓ Do it yourself!

MAKE A GAMES CONSOLE

Step-by-step guide to playing the classics

Plus!

- Meet the new Jam HAT!
- Deploy AI to keep an eye on plants
- PiTalk smartphone reviewed

TOP 10 LAPTOP KITS

The best ways to make a portable Pi

£5.99

ELECTRONICS WITH GPIO ZERO

Use AI to build a plant monitor

Measure the height and width of objects with your Raspberry Pi using only the Pi Camera Module and OpenCV

MAKER

PJ Evans

PJ is a writer, developer, and Milton Keynes Jam wrangler. He has terrible taste in movies.

mrpjevans.com

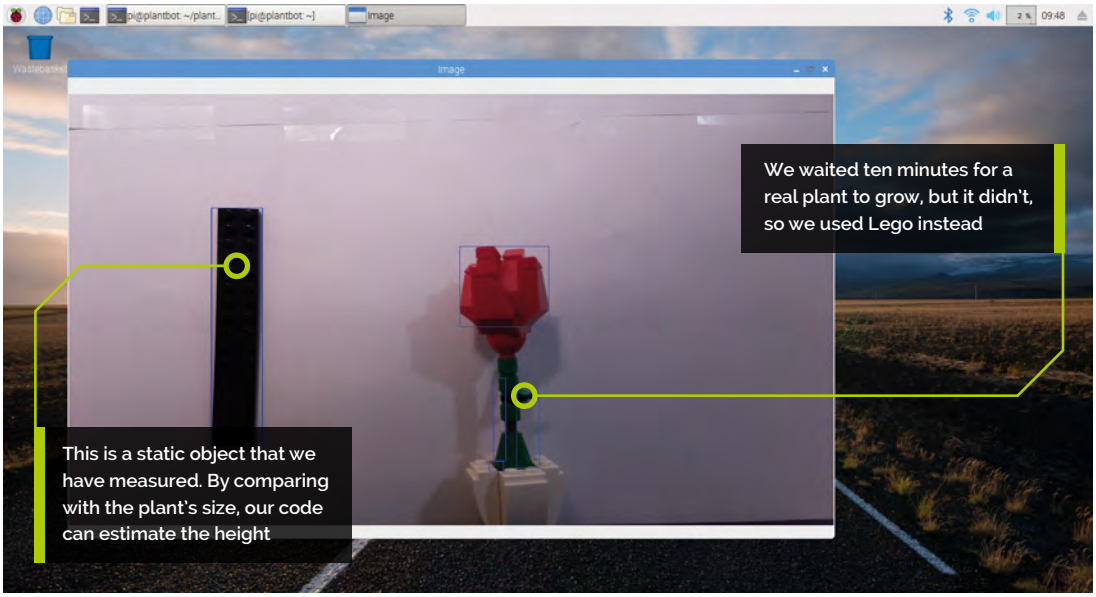
Do you know how tall your plant is? Do you wonder how high it is growing? Does using a ruler just sound like effort? Well, we have the tutorial for you. We're going to measure a plant just using images taken with a Pi Camera Module. In the process, we'll introduce you to OpenCV, a powerful tool for image analysis and object recognition. By comparing your plant to a static object, OpenCV can be used to estimate its current height, all without touching. By adding data from other sensors, such as temperature or humidity, you too can build a smart plant.

01 Pick a spot

We're going to be using OpenCV (version 2) to measure our plant. To make sure our measurement is as accurate as possible, several factors need to be considered. The background must be plain and contrast against the plant; black or white is ideal. The area you are using must be very well lit when the image is taken. Finally, the camera must be 90 degrees to the plant, facing it directly. Any angles on any axis will cause poor measurement. If you are mounting the camera at a significant distance, you may want to consider a telephoto lens.

02 Get a plant and object

So how do we pull off this measuring trick? We can estimate the height of an object by comparing it with another object that is a fixed and known width and height. The image taken by the camera needs to include both objects. Get a rectangular object of a comparable size to your



You'll Need

- Pi Camera Module **magpi.cc/camera**
- A houseplant
- Ruler or similar rectangular object
- Contrasting, plain backdrop

▲ Our testing rig. Note the camera is as close to 90 degrees to the plant as we could manage

plant; we used a long piece of Lego. A metal ruler would also work well. Mount your object in front of the camera so it is on the left of the image and the same distance from the camera as your plant.

03 Prepare your Pi

The software used to analyse the image is the powerful OpenCV library and its Python bindings. You can use any current Pi for this project, but the higher-end 3B/3B+ will be much quicker at processing the image. OpenCV requires an X Window system in place, so we need to start with Raspbian Stretch including the Raspberry Pi Desktop. Once ready to go and on the network, make sure everything is up-to-date by running:

```
sudo apt update && sudo apt -y upgrade
```

Finally, install the camera, if you haven't already, and enable it in Preferences > Raspberry Pi Configuration > Interfaces, then reboot.

04 Install dependencies

OpenCV is a bit trickier to install than most packages. It has lots of dependencies (additional software it 'depends' on) that are not installed alongside it. We also need a few other things to get OpenCV talking to the camera, so open up a Terminal window and run the following commands:

```
sudo apt install python3-pip libatlas-base-dev libhdf5-100 libjasper1 libqtcore4 libqt4-test libqtgui4

pip3 install imutils opencv-contrib-python picamera[array]
```

plantbot.py

DOWNLOAD THE FULL CODE: magpi.cc/omacNA

> Language: **Python 3**

```
001. import argparse
002. import imutils.contours
003. import cv2
004. from picamera.array import PiRGBArray
005. from picamera import PiCamera
006. from time import sleep
007.
008. # Get our options
009. parser = argparse.ArgumentParser(description='Object
     height measurement')
010. parser.add_argument("-w", "--width", type=float,
     required=True,
011.                     help=
     "width of the left-most object in the image")
012. args = vars(parser.parse_args())
013.
014. # Take a photo
015. camera = PiCamera()
016. rawCapture = PiRGBArray(camera)
017. sleep(0.1)
018. camera.capture(rawCapture, format="bgr")
019. image = rawCapture.array
020.
021. # Cover to grayscale and blur
022. greyscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
023. greyscale = cv2.GaussianBlur(greyscale, (7, 7), 0)
024.
025. # Detect edges and close gaps
026. canny_output = cv2.Canny(greyscale, 50, 100)
027. canny_output = cv2.dilate(canny_output, None,
     iterations=1)
028. canny_output = cv2.erode(canny_output, None, iterations=1)
029.
030. # Get the contours of the shapes, sort l-to-r and create
     boxes
031. _, contours, _ = cv2.findContours(canny_output, cv2.RETR_
     EXTERNAL,
032.                                   cv2.CHAIN_APPROX_SIMPLE)
033. if len(contours) < 2:
034.     print("Couldn't detect two or more objects")
035.     exit(0)
036.
037. (contours, _) = imutils.contours.sort_contours(contours)
038. contours_poly = [None]*len(contours)
```

Top Tip

Location, location, location

Make sure you avoid your 1970s flock wallpaper. You need a well-lit plain background for this to work.

Although piwheels speeds things up considerably, you can expect the second command to take a little while to run, so now's a good time to get a cup of tea.

05 Calibration test

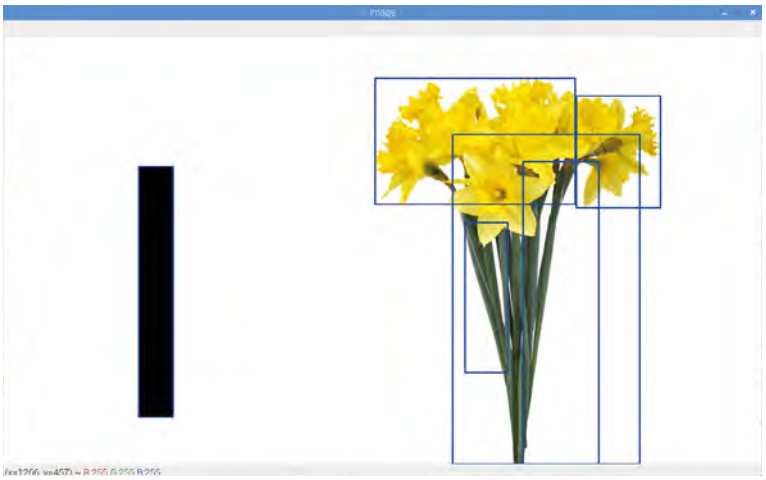
Before we start running code, make sure everything is lined up as you would like it. The easiest way to do this is to open a Terminal prompt on the Desktop and run the following command:

```
raspistill -t 0
```

This puts the camera in 'preview' mode (you'll need to be directly connected to see this – it won't work over VNC without extra configuration). Study the image carefully. Is the camera at an angle? Is it too high or low? Is there enough contrast between your object, plant, and background? How's the light? Press **CTRL+C** when you're finished.

06 Coding time

Enter the code from the **plantbot.py** listing in your editor of choice and save it. If you don't fancy typing, you can download the code and some sample images from **magpi.cc/akmZsW**. The code takes an image from the Pi Camera as a stream and sends it to OpenCV. The image is then placed through some cleaning filters and the shapes detected. We know the nearest shape to the 0,0 'origin' (the top-left of the image) will be our calibration object. So long as we know the width of that object, we can estimate the size of others.



This overlay shows how OpenCV has mapped the test image. It can be far more accurate, but this faster method meets our needs

07 Run the code

Let's take our first image. Carefully measure the width of your calibration object. Ours was 16 mm. To run the code, enter the following command in the same location as your **plantbot.py** file:

```
python3 plantbot.py -w 16
```

Replace the '16' with the width of your calibration object. After a few seconds, you'll see the image with, hopefully, a light blue rectangle around the leftmost object and one or more boxes around the plant. As the plant is an irregular object, OpenCV will find lots of 'bounding boxes.' The code takes the highest and lowest to calculate the height.

08 Big plant data

Once you've had fun measuring all the things, you can alter the code to write the results to a file. Add the following to the 'import' section at the start of the code:

```
import datetime
```

Now remove the final six lines (the first one starts 'print') and replace with:

```
print("\"" + str(datetime.datetime.now()) +
"\"," + str(plantHeight))
```

This version can be employed to create a CSV file of measurements that may be used to create graphs so you can track your plant's growth.

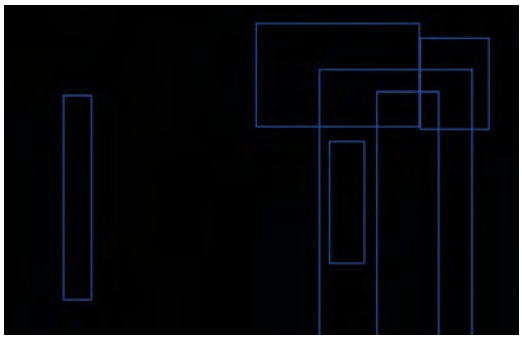
09 Keep on schedule

Let's say you wanted to take a measurement every day. You can do this by using Raspbian's cron system. Cron is an easy way to run scripts on a regular basis. To access the main cron file (known as a crontab), run the following:

```
sudo nano /etc/crontab
```

At the bottom of the file, add this line:

```
0 14    * * *   pi      python3 /home/
pi/plantbot/plantbot.py -w 16 >> /home/pi/
plantbot.csv
```



▲ An example of what OpenCV sees, with the calibration object on the left, and some daffodils on the right

This tells cron to run our script every day at 2pm (that's the '0 14' bit) and append its output to **plantbot.csv**, which can be read by any popular spreadsheet app.

10 Got water?

Now you're collecting data on your beloved plant's height, why not add other data too? A common Raspberry Pi project is to use soil moisture sensors. These are inexpensive and widely available (**magpi.cc/seyhTc**). See if you can change the script to get moisture data and include it in the CSV output. You could even send alerts when your plant needs some water. If you imported weather data from free public APIs such as **openweathermap.org/api** as well, you could see how changing conditions are affecting your plant!

11 More than plants

You can measure anything you want with this project, even growing youngsters, but why stop there? You've now had a taste of the power of OpenCV, which has many more capabilities than we've covered here. It is a powerhouse of computer vision that includes machine-learning capabilities that allow you to train your Pi to recognise objects, faces, and more. It's a popular choice for robot builders and plays a major part in the autonomous challenges of Pi Wars. We can especially recommend **pyimagesearch.com**, which provided inspiration for this tutorial.

Top Tip

No plant?

If you don't have a plant or suitable environment, we've provided some sample images here: **magpi.cc/omacNA**

> You can measure anything you want with this project, even growing youngsters, but why stop there?

plantbot.py (continued)

Language: **Python 3**

```
039. boundRect = [None]*len(contours)
040. for i, c in enumerate(contours):
041.     contours_poly[i] = cv2.approxPolyDP(c, 3, True)
042.     boundRect[i] = cv2.boundingRect(contours_poly[i])
043.
044. output_image = image.copy()
045. mmPerPixel = args["width"] / boundRect[0][2]
046. highestRect = 1000
047. lowestRect = 0
048.
049. for i in range(1, len(contours)):
050.
051.     # Too smol?
052.     if boundRect[i][2] < 50 or boundRect[i][3] < 50:
053.         continue
054.
055.     # The first rectangle is our control, so set the ratio
056.     if highestRect > boundRect[i][1]:
057.         highestRect = boundRect[i][1]
058.     if lowestRect < (boundRect[i][1] + boundRect[i][3]):
059.         lowestRect = (boundRect[i][1] + boundRect[i][3])
060.
061.     # Create a boundary box
062.     cv2.rectangle(output_image, (int(boundRect[i][0]),
     int(boundRect[i][1])),
063.                   (int(boundRect[i][0] + boundRect[i][2]),
064.                   int(boundRect[i][1] + boundRect[i][3])),
     (255, 0, 0), 2)
065.
066. # Calculate the size of our plant
067. plantHeight = (lowestRect - highestRect) * mmPerPixel
068. print("Plant height is {0:.0f}mm".format(plantHeight))
069.
070. # Resize and display the image (press key to exit)
071. resized_image = cv2.resize(output_image, (1280, 720))
072. cv2.imshow("Image", resized_image)
073. cv2.waitKey(0)
```