The goal of this project is the creation of a device with image processing capabilities and real-time constraints. The final project contains a raspberry pi and a camera with dedicated function the identification of the motion in the environment and the start of recording with ultimate goal the actual stitching of video footage that contains only the important information. This project is vital for two main reasons. 1) The significant reduction in memory usage. 2) The actual efficiency for the user that needs to investigate the footage.

The first goal was to find a platform that can work for this project. The setup should be able to perform the following tasks:

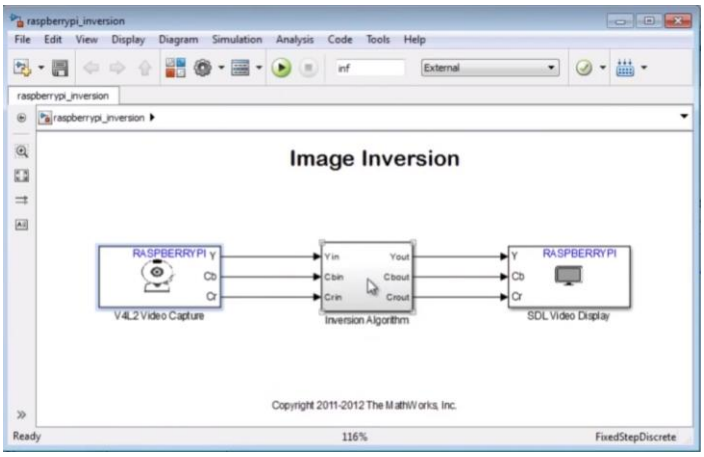
1. Read data from a camera (Raspberry Pi Camera Module V2 8MP,1080p) or a USB Web Camera (Logitech HD Pro Webcam C920).
2. Capable of executing Image processing functions and in general regular command lines from a computer or directly from Raspberry Pi.
3. The ability of saving separate image files and combining them to a final video.
4. The capability of working without a computer or a user in general.

With that being said, a good solution was the Hardware support package in MATLAB. That gives you the ability to develop software for

the Raspberry Pi directly through MATLAB – Windows PC. The setup is established with the connection of a computer to a router and the Raspberry Pi to the same router, through an Ethernet cable.

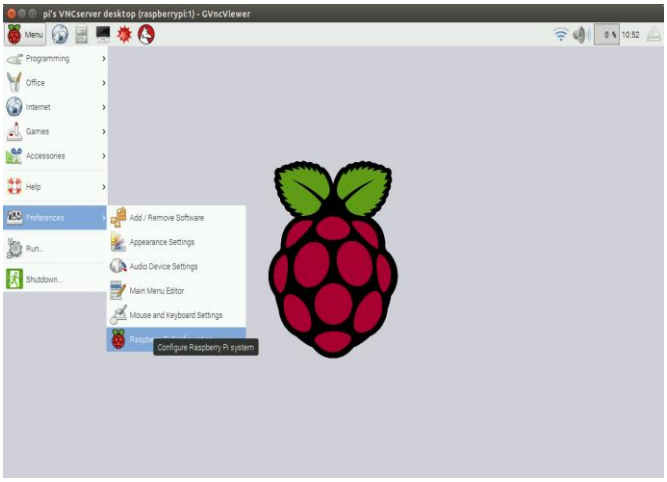
In this way, you are not directly connected to the device but you still have the ability to perform actions on it. Also, when you finalize the project you can upload it to the device once and then your program will run automatically.

First the process starts by connecting the device with the computer and making sure that everything works correctly. A major problem that was encountered is that the Raspberry Pi camera wasn't recognized.



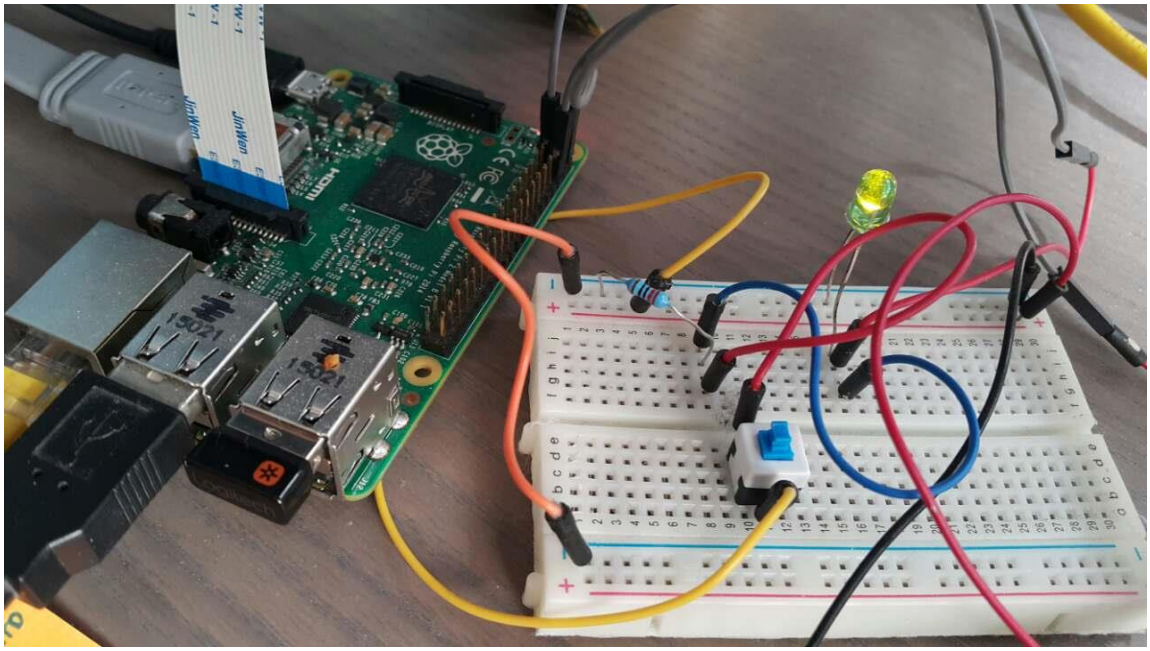
However, we were able to switch the process and use the USB web camera. For that reason, we were only able to use Simulink in MATLAB, which led to many problems with the creation of the three dimensional arrays and the manipulation of the information.

For all these reasons we decided to switch the whole project and use the raspbian operation system. The advantages are:



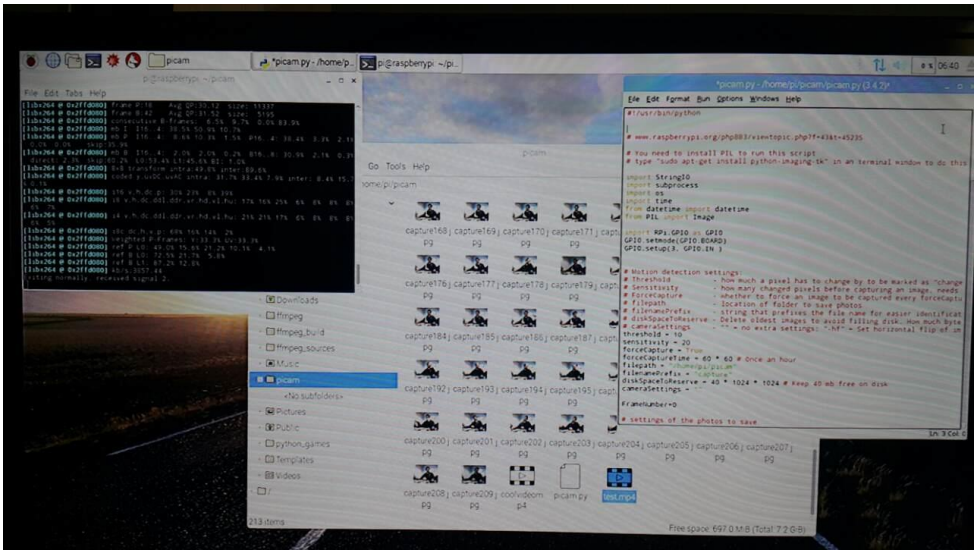
1) After downloading and installing packages we were able to identify and use the Raspberry Pi camera.
2) Easy development of the code via Python.
3) Again, after downloading many packages, we were able to use a command line that can stitch photos together and make them a time-lapse.

First of all, except the camera and the Raspberry Pi we decided to use a button. With that, the user can start and stop the program.



Let’s break down the system:

The goal is to develop an algorithm in python that identifies the push of the button. This gives the signal for initialization of the main program. Then the program should read from the Rasbpery Pi camera by capturing a low resolution image. This Data-image is used in an image processing algorithm that can identify if there is



motion between two frames. Finally, this function will return 1 if there is motion and 0 otherwise.

In the case of 1, the algorithm should capture a full resolution image and save it in a file for further use. Finally, when the button is pushed the program will stop taking more images. Then it will take the images that have already been captured and stitch them together on a video time-lapse.

```
# Capture a small test image (for motion detection)
def captureTestImage(settings, width, height):
    command = "raspistill %s -w %s -h %s -t 200 -e bmp -n -o -" % (settings, width, height)
    imageData = StringIO.StringIO()
    imageData.write(subprocess.check_output(command, shell=True))
    imageData.seek(0)
    im = Image.open(imageData)
    buffer = im.load()
    imageData.close()
    return im, buffer
```

More specifically, for this particular project we use 1296x972 image size and 100x75 test image size. The algorithm consists of 4 functions:

1. Capture a small test image (for motion detection).
2. Save a full size image to disk.
3. Keep Disk Space Free.
4. Get Free Space.

```
# Save a full size image to disk
def saveImage(settings, width, height, quality, diskSpaceToReserve):
    keepDiskSpaceFree(diskSpaceToReserve)
    time = datetime.now()

    global FrameNumber

    #filename =FrameNumber #filepath + "/" + filenamePrefix + "-%04d%02d%02d-%02d%02d%02d.jpg"
    #filename =filepath + "/" + filenamePrefix + "%02d.jpg" % (time.second)
    filename =filepath + "/" + filenamePrefix + "%01d.jpg" % (FrameNumber)
    #"debug.bmp saved, %s changed pixel" % changedPixels
    #"debug.bmp saved, %s changed pixel" % FrameNumberchangedPixels
    #threshold=threshold*1
    #'Image' +
    FrameNumber=FrameNumber+1
    subprocess.call("raspistill %s -w %s -h %s -t 200 -e jpg -q %s -n -o %s" % (settings, width
    print "Captured %s" % filename
```

The goal of the main algorithm is to compare nearby images. For this reason, we use three loops: one for the range of images that we will search, the second for the x axes and the third for the y axes.

Then in every one of these loops, we use a very simple algorithm for comparison. If the pixels values in the image that are changed are more than the sensitivity level, we need to take a full resolution image and save it.

```
for z in xrange(0, testAreaCount): # = xrange(0,1) with default-values = z will only have the value of 0 = only one scan-area = whole picture
    for x in xrange(testBorders[z][0][0]-1, testBorders[z][0][1]): # = xrange(0,100) with default-values
        for y in xrange(testBorders[z][1][0]-1, testBorders[z][1][1]):   # = xrange(0,75) with default-values; testBorders are NOT zero-based, buffe
            if (debugMode):
                debugim[x,y] = buffer2[x,y]
                if ((x == testBorders[z][0][0]-1) or (x == testBorders[z][0][1]-1) or (y == testBorders[z][1][0]-1) or (y == testBorders[z][1][1]-1)
                    # print "Border %s %s" % (x,y)
                    debugim[x,y] = (0, 0, 255) # in debug mode, mark all border pixel to blue
            # Just check green channel as it's the highest quality channel
            pixdiff = abs(buffer1[x,y][1] - buffer2[x,y][1])
            if pixdiff > threshold:
                changedPixels += 1
                if (debugMode):
                    debugim[x,y] = (0, 255, 0) # in debug mode, mark all changed pixel to green
            # Save an image if pixels changed
            if (changedPixels > sensitivity):
                takePicture = True # will shoot the photo later
            if ((debugMode == False) and (changedPixels > sensitivity)):
                break  # break the y loop
        if ((debugMode == False) and (changedPixels > sensitivity)):
            break  # break the x loop
    if ((debugMode == False) and (changedPixels > sensitivity)):
        break  # break the z loop

if (debugMode):
    debugimage.save(filepath + "/debug.bmp") # save debug image as bmp
    print "debug.bmp saved, %s changed pixel" % changedPixels
```

Finally when the user pushes the button, the algorithm will stop and start the process of creating the video from the already saved images.

```
if (flagCreateVideo==1):
    print('aaaaaaaaaa')
    subprocess.call("ffmpeg -r 60 -f image2 -s 1296x972 -i capture%01d.jpg -vcodec libx264 -crf 25 -pix_fmt yuv420p test.mp4", shell=True)
    subprocess.call("omxplayer test.mp4", shell=True)
    #ffmpeg -r 60 -f image2 -s 1296x972 -i capture%01d.jpg -vcodec libx264 -crf 25 -pix_fmt yuv420p test.mp4
    #omxplayer test.mp4
```