MAVI : Preemption,Runtime Estimation and Co-execution of tasks

Ayush Verma
2013cs50280
Major Project Part 1(COD 892)

Preemption and Runtime estimation

1. Scope of Work

The scope of work was two folded , one was to implement estimation of runtime in detection applications for the MAVI project so that the Scheduler has an idea of the runtime taken by these applications and can decide on resource allocation accordingly , and another was to implement preemption in these application so that when they are terminated in between and are not able to do complete execution they can produce some or proportional output. The applications worked on were written previously :

- Texture Detection using GLCM
- Animal Detection using SVMLight
- Pothole Detection using opencvblobslib , eigen and clustering library
- Face Detection using Viola-Jones detector

2. Methodology

To approach Preemption there were two methods based on the assumptions about the underlying OS we can make either the termination occurs gracefully using a signal interrupt using SIGTERM or SIGINT like signals or it occurs abruptly something like SIGKILL interrupt. So the first approach can be to write a signal handler function which on getting a signal interrupt writes whatever output that is produced till now into file , another approach which tries to handle abrupt termination is to write the output whenever it is produced but too many writes causes overheads thus increasing the runtime , hence an optimal periodic writing strategy can be useful thereby trading off between losing some outputs and writing overhead.

Runtime estimation was approached by a statistical approach where the code was profiled to gauge percentage of total time taken by portions of code. Some points were identified where runtime values until these points were taken and the percentage of these values to total runtime was calculated for a large number of images. What was observed that for the first two applications these percentages did not vary much from the average and hence we could conclude that with some error these average percentages were a good indicator of the amount

of time passed out of the total runtime at these points. Hence we could calculate the expected runtime by dividing the current runtime by the average percentages. At any such point :

$$Expected\ Runtime = \frac{Current\ Runtime * 100}{Average\ Percentage}$$

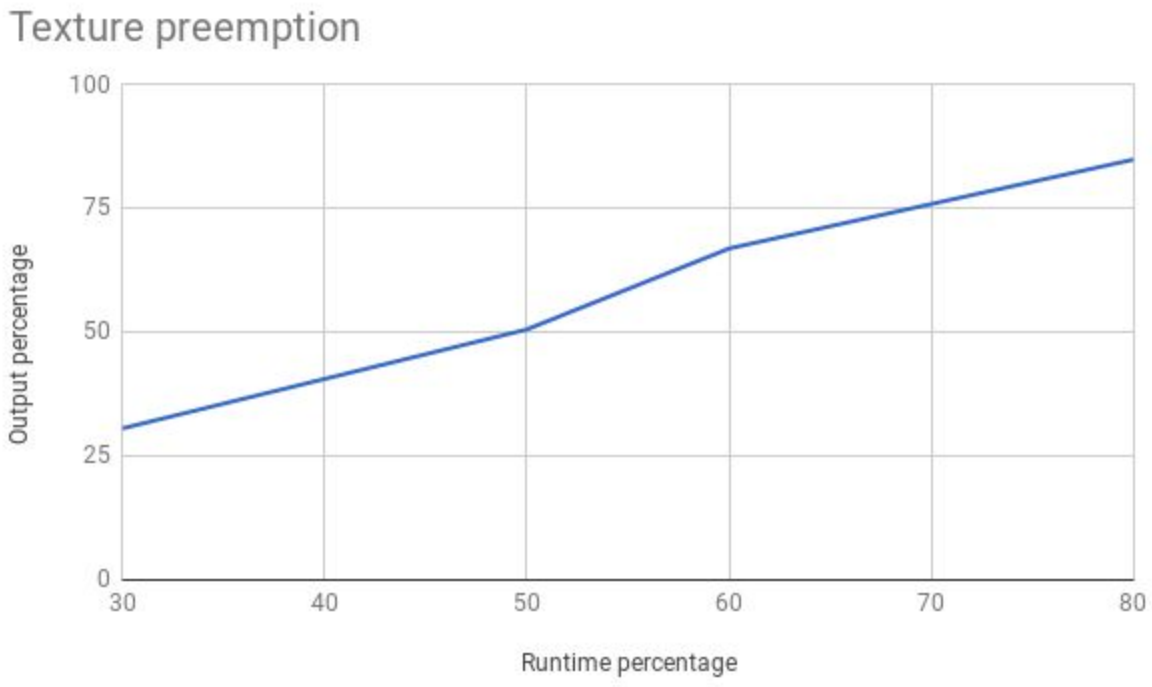
One problem that we encountered using this approach was that sometimes there was a call to the library function of opencv in the previous code. These functions were not pre-emptive and they were taking major portion of runtime and hence they needed to be profiled in order for a good estimation. So these specific classes and functions were inherited and edited respectively to make them pre-emptive and to profile their runtime.

3. Results

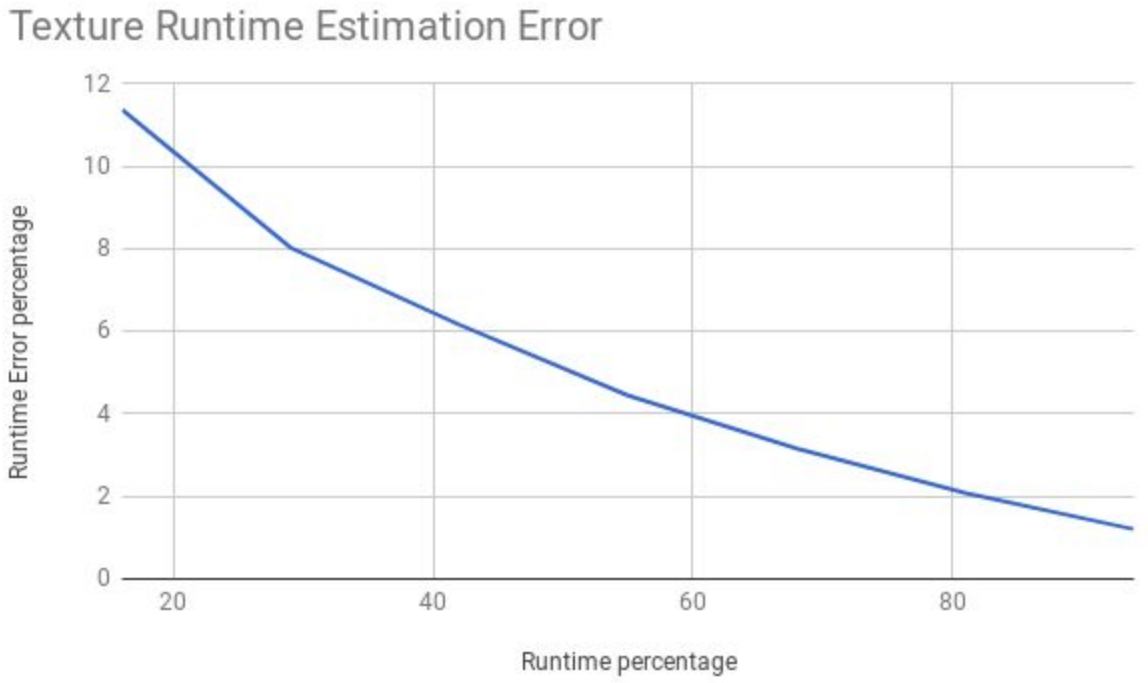
This section presents the results we got after implementing the above strategies to following applications.

- Texture Detection

The dataset for this was of 72 images , for preemption first the total runtime was calculated by complete execution of these images for every image , then the program was run for some % of the total runtime such as 30,50,60 etc. and number of objects output was calculated and the percentage of output in these cases vs total output was calculated. We get almost linear , at the end getting better than linear curve as shown in figure:

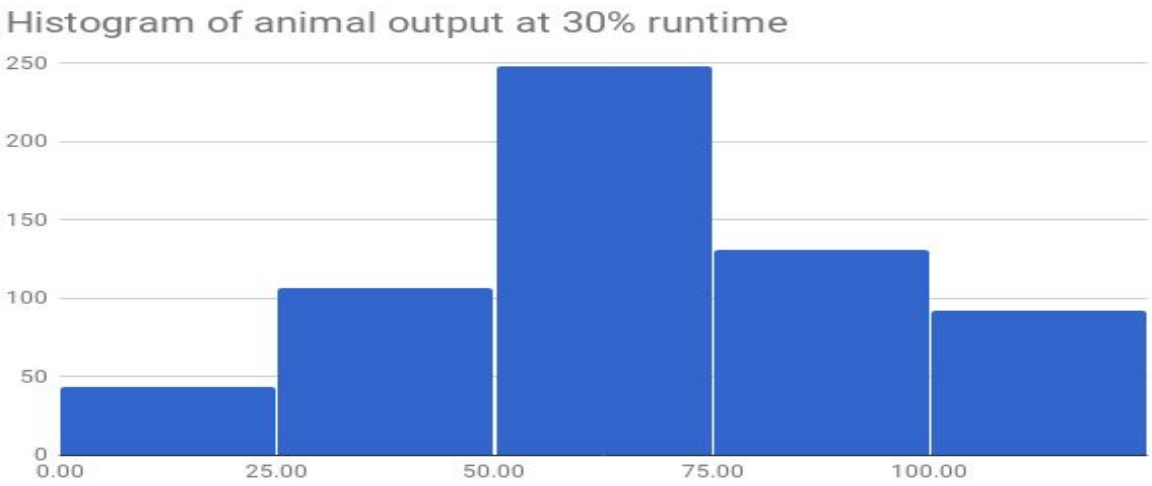


For runtime estimation , the values of estimated runtime at specific points were calculated and the error between actual runtime and estimated ones were calculated. The following figure illustrates the results , we observe a decrease in error % as we increase runtime for estimation and it converges to zero as runtime reaches 100 %.

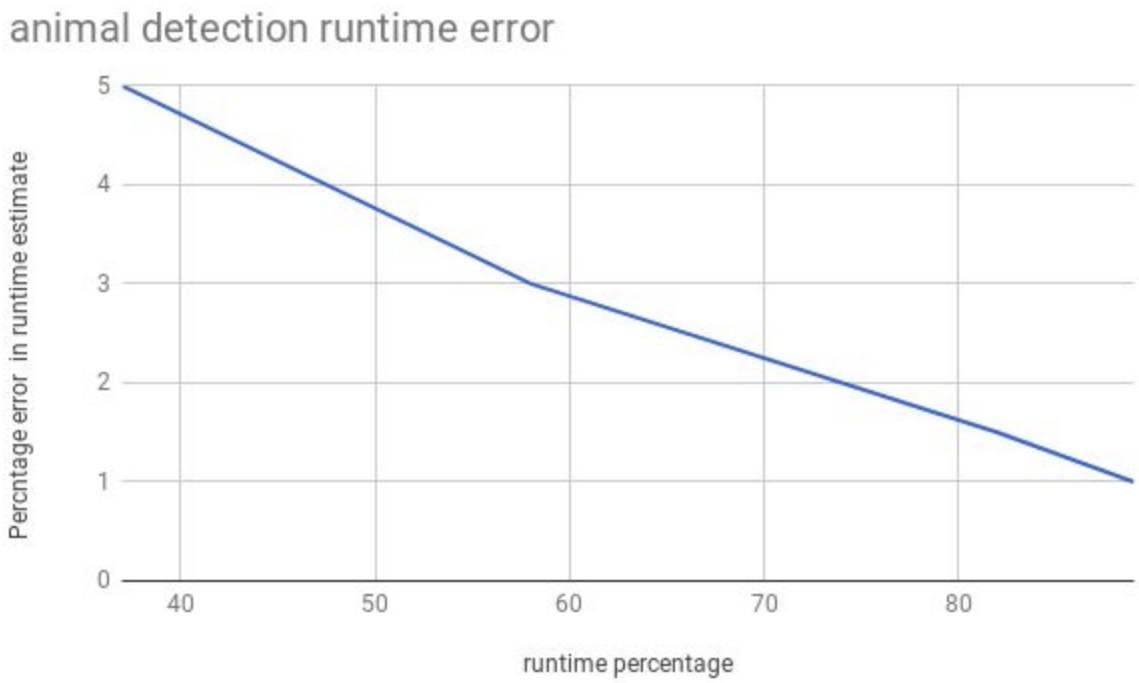


- Animal Detection

Around 600 images were tested for preemption by similar method as above and we found that on an average 62% of total output was recovered when program was stopped at 30% of complete runtime. Given below is a histogram of output percentage and images. As we see over 400 images recovered at least 50% of the total output.



In runtime estimation similar to texture detection method was followed the results are shown in figure below , error % decreases as we move to larger percent of total runtime as expected.



Average estimate error at 35% runtime comes out to 5% and then decreases.

- Pothole Detection

Pothole detection Preemption was applied at around 40 images and execution was stopped at 30 & 50% of runtime on average 4% and 12.8% of objects were recovered respectively.
We concluded through code inspection and our data that pre-processing took much of the time and therefore not much of objects could be recovered as actual detection happened almost at the end of the code runtime.

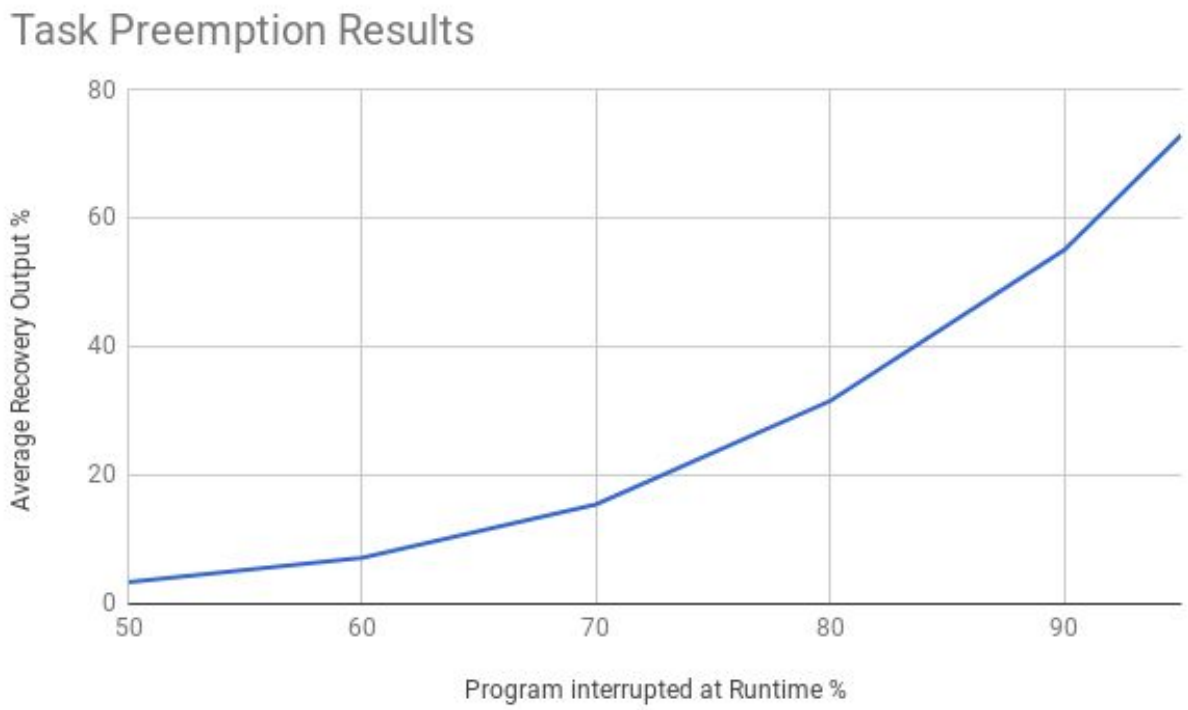
Runtime Estimation was applied to 60 images , when calculating percentages at specific points we found too much variation , hence we decided to see with this percentage the no. of objects output and we saw that yes the variation in how much percent a portion of code will take of the

total runtime depended on the number of objects detected and hence it became difficult to estimate that in advance and hence we got poor results with this strategy.

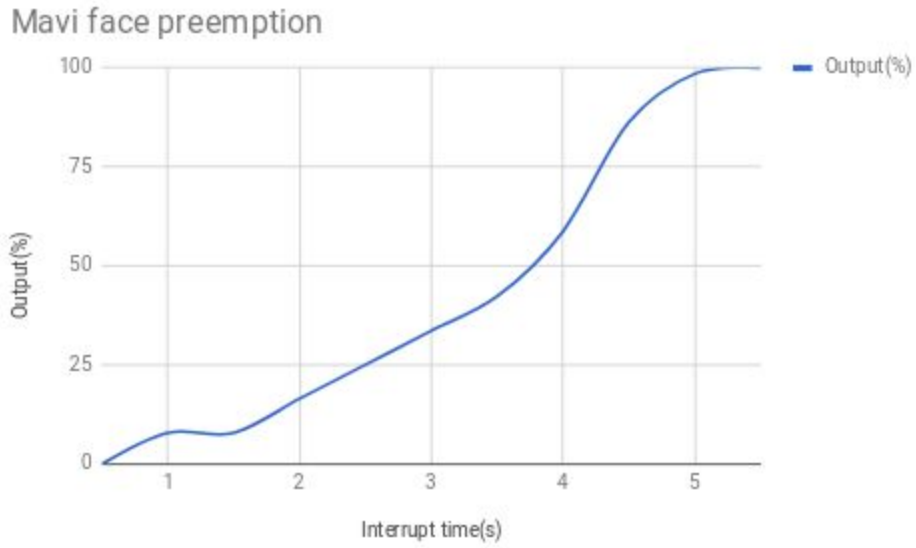
We saw that average error % was around 25-30% at 30,50% runtime points. And unlike the two applications we saw above where error was converging the error in this application failed to converge until 80% runtime point.

- Face Recognition

Face detection and recognition using openface was tested for Preemption for both single face and group images. Execution was stopped from 50-90% of actual runtime,
In single face images face is not detected until the end of loop so interruption before 90% runtime leads to no output most of the time.The application was run on 130+ group face images and could recover only 10% of the output for 100+ images when interrupted at 60% runtime. The distribution of output recovery percent versus Percent of Actual Runtime when program is interrupted is given below:



Another experiment was run on MAVI face dataset run on 300 images interrupting the program at times 0-6s so as to get an absolute sense of time rather than relative in previous case.



Runtime estimation was also done on Mavi face dataset and the following results were observed :

Runtime %	Average Error(s)	Std Deviation(s)
2	2.32	4.19
32	0.31	0.19
54	0.19	0.09
68	0.09	0.05

That is we can get a reasonable estimate of the runtime at around 30% of execution time.

Co-execution of Tasks

1. Scope of Work

We are interested in the analysis of degradation in runtime when tasks are run in parallel with each other , we had 3 tasks : animal detection(MobilenetSSD) , signboard detection and face recognition(openface) . The task was run these applications to run in different combinations on Raspberry Pi 3B and analyze the degradation in runtime. Data had to be obtained for 100 images for each combination.

2. Methodology

As the animal detection and face detection code was in python , I wrote a python control program to spawn off processes of different tasks , manage I/O and measure runtimes. First a Pi3B was setup with all the tasks independently working and then experiments were done with single task execution, 2 at a time and 3 at a time execution.

3. Results

- Face Recognition

With face recognition 100 images from the Mavi dataset were used and readings of average runtime was taken over 10 images. The following graph shows the degradation in runtime when face recognition is executed with animal detection:

Average degradation in this case is 2.4s , std deviation is 0.7s.

With Signboard it so happens that as signboard takes very little time to finish , until preprocessing is done for face recognition, signboard thread is finished and hence has no real effect on image runtimes , hence the threads effectively end up being sequential.

- Animal Detection

Animal detection was tested on 100 images from the Mavi Cow dataset and as above reading of average of 10 images were taken.

Here we see the degradation when animal and face are run together. The average degradation is 0.72s with a standard deviation of 0.12s.

As in the case of face recognition , signboard and animal run sequentially due to the low runtime of signboard so that when actual animal detection starts , signboard is finished till then.

4. Conclusion

We have successfully analyzed the preemption and estimation of runtime for 4 different applications also have analyzed the co-execution of tasks for animal and face detection. Although signboard analysis is irrelevant due to its low runtime, but for a better data we need to design the signboard experiment so as to make it co-execute with other applications which will be easier once the control system is setup. For future work this analysis will be used in making an efficient scheduler for MAVI applications taking into account the runtime estimations and also the degradation of runtimes when applications are run concurrently.