program.cpp

```cpp
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>
#include <string>
#include <map>
#include <cstdlib> // For std::exit and EXIT_FAILURE
#include <pthread.h>
#include <sched.h>
#include <time.h>
#include <fstream>

#include <pthread.h>
#include <stdio.h>
#include <time.h>
#include <aio.h>
#include <math.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <sched.h>
#include <stddef.h>
#include <sys/sysinfo.h>
#include <semaphore.h>

#define MAX_FRAME 10
#define NUM_OF_THREAD 2
#define ESCAPE_KEY (27)
#define SYSTEM_ERROR (-1)
#define TARGET_CORE 0

#define SOFT_DEADLINE 0.135 // ms

sem_t transformation_semaphore, logging_semaphore;
static int total_deadline_miss = 0;
bool initial = true;

void *(*transformationType)(void *);
std::string resolution;
int schedPolicy = SCHED_FIFO; // Default policy
std::string transformationStr;
volatile int soft_exit = 0;
double sum_of_execution = 0;

std::string filePath = "./example.csv";

struct timespec update_interval;
struct timespec last_update_interval;
struct timespec start_interval;
struct timespec end_interval;

namespace canny
{
```

```
    int lowThreshold = 0;
    const int max_lowThreshold = 100;
    const int ratio = 3;
    const int kernel_size = 3;
    const char *window_name = "Edge Map";
}

typedef struct
{
    int threadId;
    int width;
    int height;
    cv::Mat frame;
    cv::Mat intermediate_src;
    cv::Mat output;
    std::string window_name;

} ThreadArgs_t;

typedef struct
{
    int period;
    int burst_time;
    struct sched_param priority_param;
    void *(*thread_handle)(void *);
    pthread_t thread;
    ThreadArgs_t thread_args;
    void *return_Value;
    pthread_attr_t attribute;
    int target_cpu;
} RmTask_t;

// Mat frame;
void print_scheduler(void)
{
    int schedType;
    schedType = sched_getscheduler(getpid());
    switch (schedType)
    {
    case SCHED_FIFO:
        printf("Pthread Policy is SCHED_FIFO\n");
        break;
    case SCHED_OTHER:
        printf("Pthread Policy is SCHED_OTHER\n");
        break;
    case SCHED_RR:
        printf("Pthread Policy is SCHED_OTHER\n");
        break;
    default:
        printf("Pthread Policy is UNKNOWN\n");
    }
}

void appendDataToCsv(int max_frame, float total_time)
{
    // Open the file in append mode
```

```
110     std::ofstream file(filePath, std::ios::app);
111 
112     if (!file.is_open())
113     {
114         std::cerr << "Failed to open the file for appending." << std::endl;
115         return;
116     }
117 
118     // Write the new data to the file
119     file << MAX_FRAME - max_frame << "," << total_time << "," << (1 / total_time) <<
    "," << schedPolicy << "," << resolution << "," << transformationStr << std::endl;
120 
121     // Close the file
122     file.close();
123 }
124 
125 void HoughLinePTransform(RmTask_t *data)
126 {
127 
128     std::vector<cv::Vec4i> linesP;
    // Will hold the results of the detection
129     cv::HoughLinesP(data->thread_args.intermediate_src, linesP, 1, CV_PI / 180, 50,
    50, 10); // Runs the actual detection
130 
131     // Draw the lines
132     for (size_t i = 0; i < linesP.size(); i++)
133     {
134         cv::Vec4i l = linesP[i];
135         cv::line(data->thread_args.output, cv::Point(l[0], l[1]), cv::Point(l[2],
    l[3]), cv::Scalar(0, 0, 255), 3, cv::LINE_AA);
136         cv::line(data->thread_args.frame, cv::Point(l[0], l[1]), cv::Point(l[2], l[3]
    ), cv::Scalar(0, 0, 255), 3, cv::LINE_AA);
137     }
138     cv::imshow(data->thread_args.window_name, data->thread_args.output);
139 }
140 
141 // Function to detect and draw circles in an image using the Hough Transform
142 void HoughCircleCam(RmTask_t *data)
143 {
144 
145     cvtColor(data->thread_args.frame, data->thread_args.intermediate_src,
    cv::COLOR_BGR2GRAY);
146     cv::medianBlur(data->thread_args.intermediate_src, data->
    thread_args.intermediate_src, 5);
147     std::vector<cv::Vec3f> circles;
148 
149     cv::HoughCircles(data->thread_args.intermediate_src, circles, cv::HOUGH_GRADIENT,
    1,
150                      data->thread_args.intermediate_src.rows / 16, // Change this
    value to detect circles with different distances to each other
151                      100, 30, 1, 30                                // Change the last
    two parameters (min_radius & max_radius) to detect larger circles
152     );
153 
154     for (size_t i = 0; i < circles.size(); i++)
155     {
156         cv::Vec3i c = circles[i];
157         cv::Point center = cv::Point(c[0], c[1]); // Circle center
```

```
158         circle(data->thread_args.frame, center, 1, cv::Scalar(0, 100, 100), 3, cv::LINE_AA);
159         int radius = c[2];
160         circle(data->thread_args.frame, center, radius, cv::Scalar(255, 0, 255), 3, cv::LINE_AA);
161     }
162
163     cv::imshow("detected circles", data->thread_args.frame);
164 }
165
166 void CannyThreshold(RmTask_t *data)
167 {
168     cvtColor(data->thread_args.frame, data->thread_args.intermediate_src, cv::COLOR_BGR2GRAY);
169
170     /// Reduce noise with a kernel 3x3
171     cv::blur(data->thread_args.intermediate_src, data->thread_args.output, cv::Size(3, 3));
172
173     /// Canny detector
174     cv::Canny(data->thread_args.output, data->thread_args.output, canny::lowThreshold, canny::lowThreshold * canny::ratio, canny::kernel_size);
175
176     /// Using Canny's output as a mask, we display our result
177     cv::Mat timg_grad;
178
179     // Initialize `timg_grad` to be the same size and type as the source frame, but filled with zeros
180     // Assuming `data->thread_args.frame` is a cv::Mat
181     timg_grad = cv::Mat::zeros(data->thread_args.frame.size(), data->thread_args.frame.type());
182
183     // Using Canny's output as a mask, copy the frame to `timg_grad` wherever the mask is non-zero
184     data->thread_args.frame.copyTo(timg_grad, data->thread_args.output);
185
186     // Display the result
187     cv::imshow(data->thread_args.window_name, timg_grad);
188 }
189
190 void *HoughLineTransform(void *args)
191 {
192     printf("Woring");
193     RmTask_t *data = static_cast<RmTask_t *>(args);
194     data->thread_args.window_name = "Houghline Transformation";
195     cv::VideoCapture cam0(0);
196     cv::namedWindow("video_display");
197     char winInput;
198
199     if (!cam0.isOpened())
200     {
201         exit(SYSTEM_ERROR);
202     }
203
204     cam0.set(cv::CAP_PROP_FRAME_WIDTH, data->thread_args.width);
205     cam0.set(cv::CAP_PROP_FRAME_HEIGHT, data->thread_args.height);
206
207     int max_frame = MAX_FRAME;
```

```
208     clock_gettime(CLOCK_REALTIME, &start_interval);
209 
210     while (max_frame)
211     {
212         sem_wait(&transformation_semaphore);
213         cam0.read(data->thread_args.frame);
214         cv::imshow("video_display", data->thread_args.frame);
215         cv::Canny(data->thread_args.frame, data->thread_args.intermediate_src, 80,
    240, 3);
216         cv::cvtColor(data->thread_args.intermediate_src, data->thread_args.output,
    cv::COLOR_GRAY2BGR);
217         cv::namedWindow(data->thread_args.window_name, cv::WINDOW_AUTOSIZE);
218         HoughLinePTransform(data);
219         if ((winInput = cv::waitKey(10)) == ESCAPE_KEY)
220         {
221             soft_exit = 1;
222             sem_post(&logging_semaphore);
223             cv::destroyAllWindows();
224             break;
225         }
226         max_frame--;
227         sem_post(&logging_semaphore);
228     }
229 
230     return nullptr;
231 }
232 
233 void *HoughCircleTransform(void *args)
234 {
235 
236     RmTask_t *data = static_cast<RmTask_t *>(args);
237     data->thread_args.window_name = "HoughCircle Transformation";
238     cv::VideoCapture cam0(0);
239     cv::namedWindow("video_display");
240     char winInput;
241     if (!cam0.isOpened())
242     {
243         exit(SYSTEM_ERROR);
244     }
245     cam0.set(cv::CAP_PROP_FRAME_WIDTH, data->thread_args.width);
246     cam0.set(cv::CAP_PROP_FRAME_HEIGHT, data->thread_args.height);
247 
248     int max_frame = MAX_FRAME;
249     clock_gettime(CLOCK_REALTIME, &start_interval);
250     while (max_frame)
251     {
252         sem_wait(&transformation_semaphore);
253         cam0.read(data->thread_args.frame);
254         cv::imshow("video_display", data->thread_args.frame);
255         HoughCircleCam(data);
256         if ((winInput = cv::waitKey(10)) == ESCAPE_KEY)
257         {
258             soft_exit = 1;
259             sem_post(&logging_semaphore);
260             cv::destroyAllWindows();
261             break;
262         }
```

```
263         else if (winInput == 'n')
264         {
265             printf("input %c is ignored\n", winInput);
266         }
267         max_frame--;
268         sem_post(&logging_semaphore);
269     }
270 
271     return nullptr;
272 }
273 
274 void *CannyTransform(void *args)
275 {
276 
277     RmTask_t *data = static_cast<RmTask_t *>(args);
278     data->thread_args.window_name = "Canny Transformation";
279     cv::VideoCapture cam0(0);
280     cv::namedWindow("video_display");
281     char winInput;
282 
283     if (!cam0.isOpened())
284     {
285         exit(SYSTEM_ERROR);
286     }
287 
288     cam0.set(cv::CAP_PROP_FRAME_WIDTH, data->thread_args.width);
289     cam0.set(cv::CAP_PROP_FRAME_HEIGHT, data->thread_args.height);
290 
291     int max_frame = MAX_FRAME;
292 
293     clock_gettime(CLOCK_REALTIME, &start_interval);
294     while (max_frame)
295     {
296         sem_wait(&transformation_semaphore);
297 
298         cam0.read(data->thread_args.frame);
299 
300         cv::imshow("Canny Video", data->thread_args.frame);
301 
302         cv::namedWindow(data->thread_args.window_name, cv::WINDOW_AUTOSIZE);
303 
304         CannyThreshold(data);
305 
306         if ((winInput = cv::waitKey(10)) == ESCAPE_KEY)
307         {
308             soft_exit = 1;
309             sem_post(&logging_semaphore);
310             cv::destroyAllWindows();
311             break;
312         }
313 
314         max_frame--;
315         sem_post(&logging_semaphore);
316     }
317 
318     return nullptr;
```

```
319 }
320
321 void *LoggingThread(void *args)
322 {
323     int max_frame = MAX_FRAME;
324     clock_gettime(CLOCK_REALTIME, &last_update_interval);
325     while (max_frame)
326     {
327         printf("working log");
328
329         sem_wait(&logging_semaphore);
330
331         clock_gettime(CLOCK_REALTIME, &update_interval);
332         float total_sec = update_interval.tv_sec - last_update_interval.tv_sec;
333         float total_ns = ((float)(update_interval.tv_nsec -
    last_update_interval.tv_nsec) / 1000000000);
334         float total_time = total_sec + total_ns;
335
336         if (!initial)
337         {
338
339             if (total_time > SOFT_DEADLINE)
340             {
341                 total_deadline_miss++;
342             }
343             sum_of_execution += total_time;
344
345             printf("current fps/jitter: update_interval - last_update_interval : %f ,
    total_time taken for 1 frame: %f \n\r", (1 / total_time), total_time);
346
347             appendDataToCsv(max_frame, total_time);
348
349         }
350
351         if (soft_exit)
352         {
353             printf("Exiting thread 2\n\r");
354             break;
355         }
356         sem_post(&transformation_semaphore);
357         last_update_interval = update_interval;
358         max_frame--;
359         initial = false;
360     }
361     return nullptr;
362 }
363
364 int main(int argc, char **argv)
365 {
366
367     sem_init(&transformation_semaphore, false, 0);
368     sem_init(&logging_semaphore, false, 0);
369
370     sem_post(&transformation_semaphore);
371
372     const std::string expectedHeader = "Number,Execution_time,Frame_rate,
    Sched_Policy,Resolution,Transform";
```

```
    std::ifstream file(filePath);
    std::string currentHeader;

    if (file.is_open())
    {
        // Get the first line from the file
        std::getline(file, currentHeader);
        file.close();

        // Check if the current header matches the expected header
        if (currentHeader != expectedHeader)
        {
            // The headers do not match, so we need to write the correct header

            // Store the rest of the file content
            std::string restOfFileContent;
            std::string line;
            while (std::getline(file, line))
            {
                restOfFileContent += line + "\n";
            }

            // Write the correct header and the rest of the file
            std::ofstream outFile(filePath);
            outFile << expectedHeader << "\n"
                    << restOfFileContent;
            outFile.close();

            std::cout << "Header was corrected in the CSV file." << std::endl;
        }
        else
        {

            std::cout << "CSV header is correct." << std::endl;
        }
    }
    else
    {
        std::cerr << "Could not open the file." << std::endl;
        return 1;
    }

    void *(*transformationType)(void *) = CannyTransform;
    std::map<std::string, int> schedPolicyMap = {
        {"SCHED_FIFO", SCHED_FIFO},
        {"SCHED_RR", SCHED_RR},
        {"SCHED_OTHER", SCHED_OTHER}};

    std::map<std::string, void *(*)(void *)> TransformationPolicyMap = {
        {"Canny", CannyTransform},
        {"HoughLine", HoughLineTransform},
        {"HoughCircle", HoughCircleTransform}};

    for (int i = 1; i < argc; i++)
    {
        std::string arg = argv[i];
```

```
429         if (arg.find("--transformation=") == 0)
430         {
431             transformationStr = arg.substr(std::string("--transformation=").length())
;
432             if (TransformationPolicyMap.find(transformationStr) !=
TransformationPolicyMap.end())
433             {
434                 transformationType = TransformationPolicyMap[transformationStr];
435             }
436             else
437             {
438                 std::cerr << "Unknown transformation type: " << transformationStr <<
std::endl;
439                 std::exit(EXIT_FAILURE);
440             }
441         }
442         else if (arg.find("--resolution=") == 0)
443         {
444             resolution = arg.substr(std::string("--resolution=").length());
445         }
446         else if (arg.find("--sched_policy=") == 0)
447         {
448             std::string policyStr = arg.substr(std::string("--sched_policy=")
.length());
449             if (schedPolicyMap.find(policyStr) != schedPolicyMap.end())
450             {
451                 schedPolicy = schedPolicyMap[policyStr];
452             }
453             else
454             {
455                 std::cerr << "Unknown scheduler policy: " << policyStr << std::endl;
456                 std::exit(EXIT_FAILURE);
457             }
458         }
459     }
460
461     std::cout << "Resolution: " << resolution << std::endl;
462     std::cout << "Scheduler Policy: " << schedPolicy << std::endl;
463
464     pthread_t threads[NUM_OF_THREAD];
465     cpu_set_t threadcpu;
466
467     CPU_SET(TARGET_CORE, &threadcpu);
468
469     RmTask_t tasks[NUM_OF_THREAD] = {
470         {.period = 20,     // ms
471          .burst_time = 10, // ms
472          .priority_param = {0},
473          .thread_handle = transformationType,
474          .thread = threads[0],
475          .thread_args = {0, 0, 0},
476          .return_Value = NULL,
477          .attribute = {0, 0},
478          .target_cpu = TARGET_CORE},
479
480         {.period = 20,     // ms
481          .burst_time = 10, // ms
```

```
482             .priority_param = {0},
483             .thread_handle = LoggingThread,
484             .thread = threads[0],
485             .thread_args = {0, 0, 0},
486             .return_Value = NULL,
487             .attribute = {0, 0},
488             .target_cpu = TARGET_CORE},
489     };
490 
491     pthread_attr_t attribute_flags_for_main; // for schedular type, priority
492     struct sched_param main_priority_param;
493 
494     printf("This system has %d processors configured and %d processors available.\n",
    get_nprocs_conf(), get_nprocs());
495 
496     printf("Before adjustments to scheduling policy:\n");
497     print_scheduler();
498 
499     int rt_max_prio = sched_get_priority_max(schedPolicy);
500     int rt_min_prio = sched_get_priority_min(schedPolicy);
501 
502     main_priority_param.sched_priority = rt_max_prio;
503     for (int i = 0; i < NUM_OF_THREAD; i++)
504     {
505         tasks[i].priority_param.sched_priority = rt_max_prio;
506 
507         // initialize attributes
508         pthread_attr_init(&tasks[i].attribute);
509 
510         pthread_attr_setinheritsched(&tasks[i].attribute, PTHREAD_EXPLICIT_SCHED);
511         pthread_attr_setschedpolicy(&tasks[i].attribute, schedPolicy);
512         pthread_attr_setschedparam(&tasks[i].attribute, &tasks[i].priority_param);
513         pthread_attr_setaffinity_np(&tasks[i].attribute, sizeof(cpu_set_t), &
    threadcpu);
514     }
515 
516     pthread_attr_init(&attribute_flags_for_main);
517 
518     // pthread_attr_setinheritsched(&attribute_flags_for_main,
    PTHREAD_EXPLICIT_SCHED);
519     pthread_attr_setschedpolicy(&attribute_flags_for_main, schedPolicy);
520     pthread_attr_setaffinity_np(&attribute_flags_for_main, sizeof(cpu_set_t), &
    threadcpu);
521 
522     // Main thread is already created we have to modify the priority and scheduling
    scheme
523     int status_setting_schedular = sched_setscheduler(getpid(), schedPolicy, &
    main_priority_param);
524     if (status_setting_schedular)
525     {
526         printf("ERROR; sched_setscheduler rc is %d\n", status_setting_schedular);
527         perror(NULL);
528         exit(-1);
529     }
530 
531     printf("After adjustments to scheduling policy:\n");
532     print_scheduler();
```

```
533
534     int width = 0, height = 0;
535     size_t xPosition = resolution.find('x');
536     if (xPosition != std::string::npos)
537     {
538         width = std::stoi(resolution.substr(0, xPosition));
539         height = std::stoi(resolution.substr(xPosition + 1));
540     }
541
542     for (int i = 0; i < NUM_OF_THREAD; i++)
543     {
544         // Create a thread
545         // First paramter is thread which we want to create
546         // Second parameter is the flags that we want to give it to
547         // third parameter is the routine we want to give
548         // Fourth parameter is the value
549         printf("Setting thread %d to core %d\n", i, TARGET_CORE);
550
551         if (pthread_create(&tasks[i].thread, &tasks[i].attribute, tasks[i]
    .thread_handle, &tasks[i]) != 0)
552         {
553             perror("Create_Fail");
554         }
555     }
556     printf("Threads created \n\r");
557     for (int i = 0; i < NUM_OF_THREAD; i++)
558     {
559         printf("Joining thread %d\n", i);
560         int join_result = pthread_join(tasks[i].thread, &tasks[i].return_Value);
561         if (join_result != 0)
562         {
563             printf("pthread_join failed for thread %d: %s\n", i,
    strerror(join_result));
564         }
565         else
566         {
567             printf("Thread %d joined successfully.\n", i);
568         }
569     }
570
571     if (pthread_attr_destroy(&tasks[0].attribute) != 0)
572         perror("attr destroy");
573     if (pthread_attr_destroy(&tasks[1].attribute) != 0)
574         perror("attr destroy");
575
576     clock_gettime(CLOCK_REALTIME, &end_interval);
577
578     float total_sec = end_interval.tv_sec - start_interval.tv_sec;
579     float total_ns = ((float)(end_interval.tv_nsec - start_interval.tv_nsec) /
    1000000000.0);
580     float total_time = (total_sec + total_ns);
581
582     printf("Average FPS %f, Average time %f \n\r", (MAX_FRAME / sum_of_execution),
    sum_of_execution/MAX_FRAME);
583     printf("Total deadline miss : %d\n\r", total_deadline_miss);
584     sem_destroy(&transformation_semaphore);
585     sem_destroy(&logging_semaphore);
```

```
586
587     return 0;
588 }
```