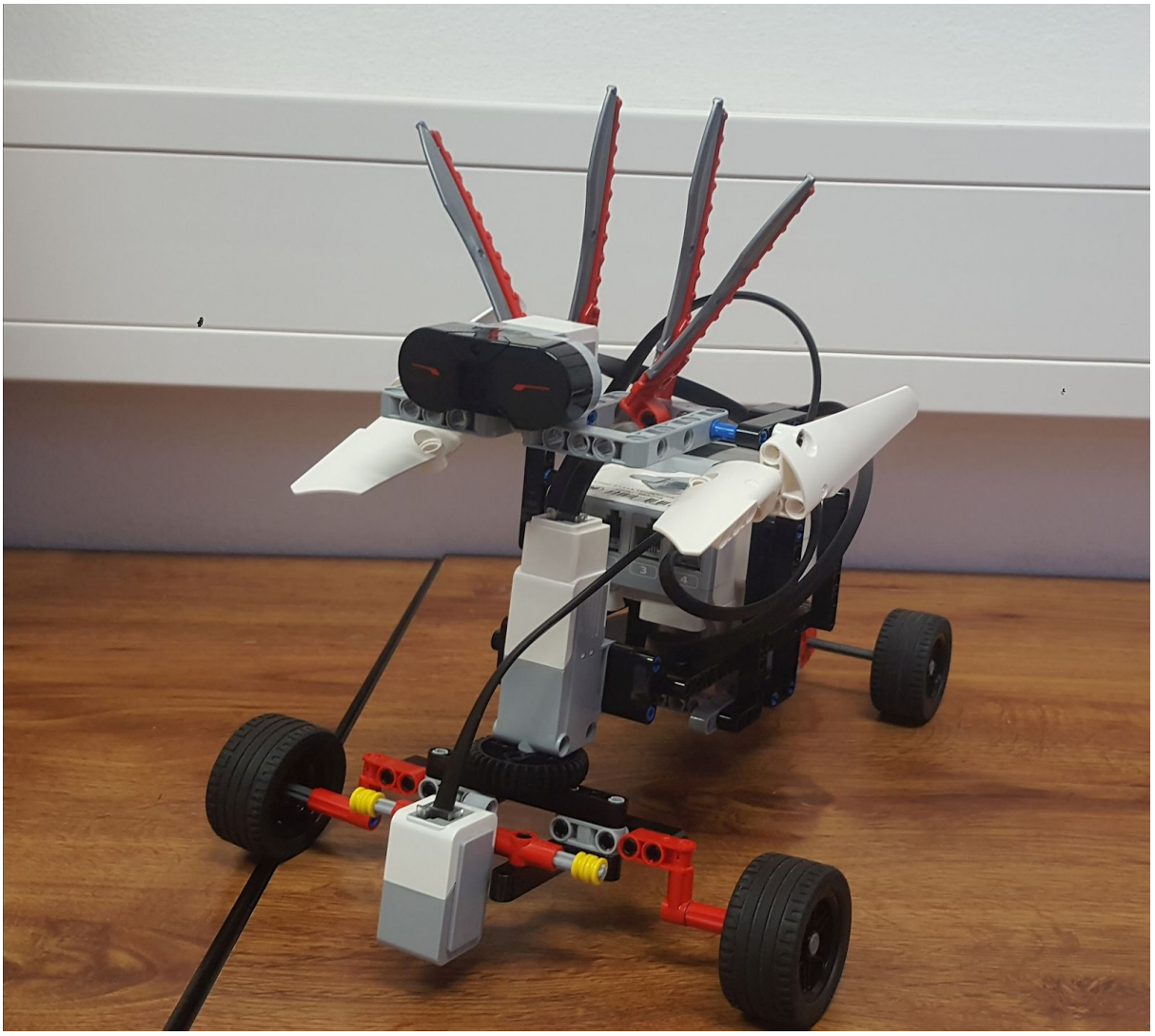
Project Real-time Systems

Alice Nohejlová, Pavel Fikar



Our task was to build and program a robot from Lego Mindstorms EV3. The robot rides, follows the line and reacts to obstacles.

Construction

The robot´s chassis is based on wheels rather than tracks. The robots uses one large motor, medium motor, IR sensor and color sensor. Large motor serves as main propulsion and propels the robot forward. Medium motor is responsible for steering by turning the front wheels. IR sensor measures distance and color sensor is used to measure the amount of light that reflect from the surface. Due to the physical limitations imposed by the physical construction of the robot, the maximum angle of the front wheels compared to the rear wheels that can be achieved is limited to 60 degrees to each side.

Software

As a platform for our program we used operating system leJOS, which is based on Java 7 Embedded for EV3. The installation on the brick was done through computer and required leJOS EV3 program and Java Runtime Environment in above specified version.

We created program EV3LineCar. The program was implemented in Java and developed in the environment of Eclipse IDE using EV3 leJOS plugin.

Tasks

1. IR Sensor measures a distance and based on the output modifies configuration of motor. Result is slowing down or accelerating.
2. Color sensor checks if it stands on white (or black) line and correct the heading (front wheel angle).
3. Scheduling of the tasks 1 and 2.

Distance task

Every job of distance task is checking whether there is an obstacle in front of the robot and and adjusts the speed of the robot accordingly or stops completely.
The execution time of distance job is approximately 15 ms and relative deadline is 150 ms.

Steering task

Steering task is responsible for keeping the robot on the line while as it moves. Steering is adjusted based on the result of measuring from the color sensor. The color sensor measures the amount of light that reflects from the ground.
The execution time of distance job is approximately 18 ms and relative deadline is 100 ms.

Scheduling

Each task has its own thread and scheduling of these tasks is done by the main thread. Jobs are released regularly and cannot be preempted. Scheduler always chooses the one with earliest deadline.

Implementation details

The entire program is design to use 3 threads. First thread is the main thread and this thread is responsible for creating the other threads and scheduling of jobs. Second thread is used for distance check task and it purpose is to execute distance check jobs and to release new one after previous one is executed. The second thread represents the instance of distance check task. Third thread is used for following the line respectively adjusting the steering to keep the robot following the line. This thread is responsible for executing steering jobs. The third thread is representing the instance of steering task.

The second and the third thread are very similar, but the main difference is in function that they call to execute their respective jobs. These threads are not created directly inside main class, but they are created when their respective tasks are started. When the task is created, pointers to the motors and sensor as well as queue for their jobs are passed to them via their constructor. Once they are created, they are started and they create their own thread. When thread is created, it starts to execute its run function. At the beginning the run function check if the pointers to motors and sensors where passed correctly. In case of the second thread, this check is done earlier in the constructor. After then both treads enter while(true) cycle, because these two tasks are supposed to run indefinitely and thread represents task and not only job. The first thing done in the cycle is creation on new job and its addition to the queue. Then value in suspender class (suspender class is described later) is set to false and thread set its own suspend variable to true and thread suspend itself with wait(). Until main thread calls appropriate function containing notify() and wakes up the thread. After notify the thread executes the appropriate job. Distance check for the second thread and steering for the third thread.

While executing the distance check task the distance is measured by IR sensor in range 0 to 100. Where 0 means that there is some obstacle right in front of the sensor, 100 means that there is no obstacle registered in front of the sensor. 100 is also maximum range of the sensor.
During every job, distance is measured, stored and compared to the last distance measured. If newly measured distance is less than 50 or the difference between old and new measurement is more than 10, then the speed of power motor should be adjusted.
The robot immediately stops, if it registers an obstacle in distance of 20 or less. If the new distance is smaller than the old distance, robot slows down with respect to the difference.
If the sensor measures greater distance than the last time, the robot accelerates or if it stopped before, starts moving again.

Execution is steering job is rather simple, because the robot is following transition between white and black lines. The reason for this is described in Difficulties section. At the start of the steering task the baseline value is measured. Baseline represents the value that the robot tries to follow. When executing the steering job, new measurement is done and the result is compared do the baseline value. If the new value is higher than baseline, then the robot is more on the white line then black and vise versa. The value of the difference is then used as an input for steering correction. If the difference is positive then robot turns the front wheels to the right and if the difference is negative then turns to the left. The change is done

in degrees and it's 1/8 of the difference (e.g. if difference is 16 then the front wheels are turned 2 degrees to the right). 1/8 is used to make steering smoother. The change is done relatively compared to the current steering angle.

Each job is represented by its own instance of the job class that contains the release time and deadline for the particular job.

Due to problems with thread synchronization the special suspender class is created. Main thread can check the content of the class and decide whether to continue waiting or to run. Value false means it can continue running and true means that the thread should sleep.

The main class (main thread) is responsible for starting everything and scheduling jobs. The execution of main function starts by allocating all motors and sensors, creating instance of suspender class as well as instances of distance check and steering and starting the distance check and steering tasks. After that the scheduling cycle is started. At the beginning of the cycle new jobs are polled from the queues if the previous ones were finished and new are available (e.g. previous iteration executed steering job so new steering job is polled), after that the scheduling begins. If jobs from both tasks are ready, then their deadlines are compared and the job with lower deadline is selected for execution. If only one job is ready, then this job is executed. The job execution starts by setting value in suspender class to true to prevent main thread from running out of order when other thread should be running. After this, the thread corresponding to the job being executed is woken up and main thread sleeps for 25ms. This time frame gives the other thread time to run without interruption. In uncommon case that task thread does not finish execution of a job within 25ms the main thread goes into the wait cycle. While in this cycle the main thread check the value in the suspender class and it sleeps for 5 ms if it is true or end the wait cycle and continues running if false. Only thread associated with tasks can set suspender value to false.

Concurrency and timing

Concurrency is used in form of two threads representing two tasks that both want to run. So the scheduler has to decide who will run next after each job execution. Timing is critical. If steering task didn't execute any job in half a second then the robot could drive of the line and never return. In case of check distance the robot could crash into obstacle in front of him. So these tasks are hard-realtime and enough jobs from both tasks need to be executed in correct order (e.g. not 10x distance check and then 10x steering). The correct timing in this case is provided be correctly set deadline of each job. The scheduler chooses job with the earliest deadline when the previous job is done.

Difficulties

- There was a very complicated access to required software – Java Embedded for LeJOS operating system, because download of Java is restricted. In order to download any Java other than Java 8, we needed to have an Oracle account.
- Bluetooth connection was unstable and setup was not intuitive.
- Problem with scheduling synchronization - At first we wanted to implement EDF scheduling algorithm, but there was a problem with the main thread which gained control when other tasks should be executed. So we decided to do the clock-driven scheduling with fixed frames. Then we found the way to synchronize the main thread by Suspender class and now the EDF algorithm is applied.
- It is difficult to measure execution time of a job without affecting it.
- We had to lower the maximum speed in order to notice all the color changes to 270 degrees per second.
- Original idea was to follow the line, but this proved very difficult. After multiple hours of work the code was still not working. Main problem was to recognize to which side of the line is robot driving (left or right of the line). In the end we decided to not follow the line but rather follow transition between two different color lines instead. Used colors are black and white to make it easier. With this approach recognizing left and right is easy.

Distribution of work

- idea for tasks - P. F.
- design and construction of robot - both
- design of tasks - both
- implementation of distance task - A. N.
- implementation of steering task - P. F.
- design and implementation of multithreading environment - both
- writing report - both

Conclusion

We created hard-realtime operating robot which fulfills the described tasks. It operates on multiple threads and is scheduled by EDF algorithm. Despite some difficulties we enjoyed working on the project and finished it successfully.

The most interesting part of the project, where we learned the most, was dealing with multiple threads and scheduling. And the funniest part was construction of the robot.