List of projects

Simulation projects

These projects simulate a physical system or the behavior of active agents interacting with each other and with the environment.

1. **Ball catcher**. Simulate a 2D Cartesian robot moving a basket on a x-y plane. The user launches balls with the mouse in different directions on the plane and the robot has to catch them. Balls are detected by a vision system that tracks their position and estimates their trajectory. You can use OpenGL 3D graphics or a simple self-made library. Joint are actuated by dc motors controlled by PID regulators. The user must be able to change the 3D view by mouse dragging.

2. **Highway**. Simulate a portion of a highway where different types of vehicles (motorbikes, cars, trucks) are created by the user as interacting tasks. Each type of vehicle is autonomous and has different driving rules (e.g., trucks cannot surpass other vehicles and have lower speed limits). Vehicles have different sensors to detect the road borders and the other vehicles. The highway can be displayed from top and the user must be able to zoom or track a specific vehicle.

3. **Car race**. Simulate a car race with N cars on a circuit that can be drawn using a mouse and saved in a file. N-1 cars must be autonomous (hence equipped by sensors that detect curves and other cars) and one is controlled by the player through the keyboard arrows.

4. **City**. Simulate an urban area with a number of crossing streets with traffic lights managed by a single traffic control task. Cars are randomly generated to enter and exit the area. Each car is autonomous and is controlled by a periodic task based on local sensors (no global information must be used).

5. **Image processing**. Simulate a fixed camera looking at a given portion of an animated screen and write 4 periodic tasks performing different computations on the same image and displaying the results on other images. For example, task1 displays the histogram, task2 displays the image after applying a threshold, task3 displays the difference between consecutive frames, task4 displays the result of a filter. The user must be able to vary some parameters of such tasks.

6. **ECG**. Display an ECG signal stored on a file (or synthetically generated) and write a task that performs a real-time signal processing that detects the relevant ECG parameters to identify and visualize some anomalies.

7. **Double Pendulums**. Simulate N double pendulums, where N and their initial conditions are read from a text file. Trajectory must be visualized as in the following examples:
   https://www.youtube.com/watch?v=V4hvENrtMeE
   https://www.youtube.com/watch?v=QXf95_EKS6E

8. **Telescopes**. Simulate an array of N telescopes (positioned at the bottom the screen) controlled to point at the centroid of the image of a moving planet (passing on the top of the screen). Each telescope introduces some random noise, hence the system integrates the various images to produce an average output image. All images must be shown on the screen. The program must allow the user to change the noise level and the motor control parameters of each telescope.

9. **Marble race**. Simulate a simplified version of the following environment:
   https://www.youtube.com/watch?v=I0FmKk74E_0

10. **Filters**. Simulate N filters of different types (e.g., low-pass, high-pass, and band-pass) selectable by the user. All filters receive the same input signal and produce different outputs. The input signal is generated by a periodic task and is also selectable from a given set (e.g., sinusoidal, square, sawtooth, step, square, and triangular).

11. **Fireflies**. Simulate a set of randomly flying fireflies that synchronize their flashing using one of the biologically inspired algorithms existing in the literature.

12. **Ball-Beam**. Simulate 2 ball-and-beam devices. For each device, a ball moves on a linear guide rotated on its center by a dc motor. Sensing the position of the ball, the controller must keep it in a desired position. The user must be able to push the ball to disturb a system and enable the controller to launch the ball to the other system and viceversa (using one ball for both systems).

13. **Segways**. Simulate a number of segways (implemented as concurrent tasks) with the possibility of changing the control parameters of a specific segway selected with the mouse on a control panel.

14. **Goalkeeper**. Simulate a robot goalkeeper consisting of a cart moving on a guide. Position and speed of the incoming ball must be read by a periodic task that samples the visual field at a given rate. Visual sensing, motor simulation, control, and display must be implemented as different tasks.

15. **Robot catching**. Simulate a robot arm with a basket as hand effector that has to catch balls that are thrown by a pan and tilt shooter. The ball is detected by vision and the shooter is controlled by the user (setting both angles and launching speed) Visual sensing, motor simulation, control, and display must be implemented as different tasks.

16. **Robot juggling**. Simulate two robot arms with a ping pong bat as end effector juggling a ping pong ball between them. Ball is detected by a vision system shared by the robots.

17. **LEM**. Simulate a LEM (controlled by the keyboard) that has to leave its mother ship (rotating around a planet), land on a planet surface, take a rock sample, and leave the planet to meet the mother ship again. During its path, the LEM has to go through an asteroid belt rotating around the planet below the mother ship.

18. **Elevators**. Simulate N elevators in a building with M floors. People using the elevators are randomly generated. Elevators must allow clients to book the requests and stop to floors in the desired sequence. Elevator must move smoothly as controlled by motors.

19. **Patriots**. Simulate a set of Patriot defense missiles that identify enemy targets, predict their trajectories and are launched to catch them.

20. **Visual Tracking**. Simulate a pan-tilt camera that tracks moving objects on the screen. The target can be moved by mouse or by a task, using random or predefined paths. Target, camera, motors, graphics and user interface must be implemented by different periodic tasks. The tracking windows must be moved in a predicted position and enlarged when the object is lost.

21. **Kalman**. Simulate a Kalman filter to predict the mouse position on the screen. Add noise to position and show a fading path with variable length. Make a user interface similar to the one implemented in the following demo: https://www.cs.utexas.edu/~teammco/misc/kalman_filter/

22. **Pool**. Simulate the Pool game, where each ball is a periodic task. The user decides direction and intensity of its throw using the mouse. During the aiming phase, the system must optionally show the predicted trajectory of the ball hit by the stick up to the next ball.

23. **Cannon**. Simulate a cannon controlled by the user that must shoot a ball to catch a target that moves slowly on the ground on the other side of a wall. Position and height of the wall are randomly generated after each shot. Use 3D graphics (see notes in the Scara project).

24. **Levitron**. Simulate a set of N Levitrons, whose parameters are provided in a configuration file. The user must be able to reset and disturb a desired device (e.g., pushing the levitating mass).

25. **Fireworks**. Simulate a set of fireworks of different types selected, positioned, and triggered by the user at specific locations. Sounds have to be also generated.

26. **Skeet Shooting**. A skeet is lunched with random parameters from one side of the screen and must be hit by the player located at the bottom center of the screen. The skeet is launched with a random delay after a key press. The player changes the direction of the shotgun by the arrow keys and shoots by the SPACE key. He has two bullets for each skeet. Hit targets must explode in a (small) number of pieces animated by a periodic task activated on the event.

27. **Assembly chain**. Simulate an assembly chain where objects move on a conveyor belt and are mounted by a set of robot stations. Required stages are: visual recognition, selection (an object with anomaly is discarded), assembling (a piece from a stack is mounted on the object).

28. **Tanks**. Simulate N tanks that have to maintain the liquid at a desired level. Each tank has an output tap at the bottom to get the liquid and an input tap at the top to add new liquid. The liquid level is acquired by a proximity sensor located on the top. Each input tap is automatically controlled by a periodic task, while output taps are controlled by the user.

29. **Radar tracking**. Simulate a radar that scans an area with moving targets with different colors. Then, for each target, generate a tracking task that analyzes and predicts its trajectory, displaying the predicted location after K samples (K is selected at run time by the user).

30. **Lake**. Simulate a lake as an elastic 3D surface of oscillating elements. The program must allow the user to modify the element parameters, set predefined parameter configurations, and inject variable disturbances at specific locations with the mouse.

Kernel projects

These projects require the implementation of a given operating system mechanism for real-time systems or require the development of a real-time application on top a specific operating system.

31. **Schedule**. Visualize the execution of a task set (including the main) with an adjustable time scale. Visualize activation times, deadlines, critical sections with different colors. Build a task set in which a priority inversion occurs. Then run the set using PIP and PCP to show that priority inversion disappears. Also visualize the instantaneous workload as a function of time.

32. **Sporadic Server**. Implement a Sporadic Server and build a test case to graphically visualize its behavior on a set of aperiodic tasks activated by pressing a key on the keyboard.

33. **Elastic scheduling**. Implement a test case to graphically visualize the behavior of the Elastic Task Manager.

34. **RT-Arduino**. Develop a concurrent application on the Arduino platform using a real-time interface provided by the RETIS Lab for periodic task management.

35. **RT-Arduino internals**. Develop new mechanisms in the Arduino framework to support concurrent applications on the Arduino platform.

36. **VxWorks-App**. Implement a real-time application on VxWorks, on an embedded platform. Both the kernel and the platform will be provided by the RETIS Lab.

37. **VxWorks-CAB**. Implement the CAB mechanism into VxWorks and a multithread application to test its behavior.

38. **VxWorks-Ptask**. Implement the Ptask library on VxWorks and a multithread application to test its behavior.

39. **VxWorks-multicore**. Implement a real-time application on VxWorks using a multicore platform (provided by the RETIS Lab) to compare response times under global scheduling (default in VxWorks) e partitioned scheduling (achievable by setting core affinities). Details are to be discussed in person.

Drone projects

These projects require the interaction with a 3D graphic simulation engine (provided by the RETIS Lab) for the display of a virtual environment were virtual drones operate. Drone modeling and control have to be implemented in Linux as real-time tasks.

40. **Visual landing**. Develop a program to control a virtual drone to land on a target landmark using visual feedback. A virtual camera must be implemented to get images from the virtual world.

41. **Follower**. Develop a program to control a virtual drone that follows an object that moves into the virtual world using a predefined trajectory that can be randomly generated or specified by the user through the mouse or by a file.

42. **Obstacle Avoidance**. Develop a program to control a virtual drone to reach a given position in space avoiding moving obstacles in a virtual world. Obstacles are detected using a laser sensor mounted on the drone.

43. **Balancing drone**. Develop a program to control a virtual drone that balances a pole, whose orientation is detected by a virtual sensor. See: https://www.youtube.com/watch?v=15DIidigArA

44. **Catching drone**. Develop a program to control a virtual drone that catches a ball thrown by the user. See: https://www.youtube.com/watch?v=2kIk96FyTpU

45. **Juggling drone**. Develop a program to control a virtual drone that juggle a ball thrown by the user. See: https://www.youtube.com/watch?v=3CR5y8qZf0Y

Multimedia projects

These projects require the acquisition or the generation of audio/video signals. Input signals can be acquired from the PC audio port or alternatively from a file.

46. **FFT**. Develop a program that acquires an audio signal (either from file or from microphone) and display both the amplitude and the frequency spectrum (computed by FFT) as a function of time. The spectrum should be optionally visualized as a waveform or a 2D (time-freq.) color map. The program must provide an graphical interface for a band equalizer.

47. **Mini Moog**. Simulate an additive synthesizer that sums three waveforms (sine, triangle and square). The frequency of the waveforms is decided by the user by pressing a key corresponding to a specific note. Each waveform passes through a dedicated band-pass filter (with adjustable parameters), the outputs of which are summed with an adjustable scale factor (volume).

48. **Image2sound**. Create a program that reads a picture from a file and converts it into sound. Process different features of the image by concurrent tasks each playing a different instrument.

49. **Sound2image**. Create a program that reads an audio file and converts it into a dynamic image. Consider to process different aspects of the audio stream by concurrent tasks, each drawing a different part (or layer) of the image.

50. **Tempo**. Acquire the microphone signal and process the sound to extract the tempo generated by stick hitting the table. Then use the extracted information to vary the playing speed of a song in order to follow the tempo given by the stick.

51. **Table sound**. Attach a microphone to a table and process the signal in order to generate sound as a function of the produced signal. The sound can be generated via MIDI notes or by manipulating a wave file.

52. **Visual piano**. Develop a program that acquires MIDI messages from a piano keyboard (provided by the RETIS Lab) and displays a virtual keyboard that visualizes the notes with names and colors (see for example https://www.youtube.com/watch?v=stsBqdo9Svc) and also displays the notes using a “piano roll” notation (with a vertical time axis).