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. **Birds.** Simulate a number of birds that exhibit a swarm behavior. If a bird has other birds in front, it follows those near him, otherwise it acts as a leader and moves according to a pseudo-random trajectory. If a leader approaches other birds, he becomes a follower. Birds in front are detected by a short-range visual sensor sampling pixels in a semi-circular area (r, -π, +π). When the user generates a loud noise, they enter in escape mode, increasing their speed and taking random directions, returning in the normal mode after a given interval of time.

2. **Fireflies.** Simulate a number of fireflies, each blinking at its own frequency (independent of the period). When the user presses a key, they start blinking as a function of the state of their neighbors and slowly synchronize. Pressing another key, they return in independent mode (hence they slowly desynchronize).

3. **Ants.** Simulate an ant colony where each ant is managed by a periodic task. The user generates food by using the mouse and the ants must find it, bring it to the nest and leave pheromone trails to communicate with the other ants.

4. **Evolution.** Simulate a number of beings that move on the environment to search for food. Beings are driven by simple rules, have a number of state variables (e.g., sex, age, speed, size, hunger, lifetime), but differ for type (e.g., sex, max age, max speed, hunger level). Beings of different sex can reproduce when they get in contact for some time, generating new beings. They also die for lack of food or high age. Keep track of the evolution by monitoring and displaying some variables.

5. **Planets.** Simulate a solar system with N planets, each managed by a periodic task. Parameters are read from a file and the program must allow the user to create a planet at a desired time, translate the view by mouse dragging, zoom, slowdown/speedup the simulation, and optionally visualize the past trajectory of each planet (whose length can also be varied by the user).

6. **Airport.** Simulate an airport with two runways managed by an air traffic control task. This task delivers instructions to the incoming aircrafts (holding or landing) and to the outgoing aircrafts (wait or take off) depending of the current traffic conditions. Incoming and outgoing aircrafts are generated randomly or by a key press.

7. **Trains.** Simulate a train station with 8 tracks that must be allocated to the incoming trains. Trains can have different priority and run according to a schedule stored in a file. They can arrive and leave from two tracks on the left and two on the right. The controller task has to manage track switches (also to be animated) and traffic lights on each track.

8. **Harbor.** Simulate a harbor with a number of ships that enter and exit the harbor and need be directed by a controller task in predefined locations. The position of the ships is tracked by a radar and represented on a display.

9. **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).

10. **Springs.** Simulate N mass-spring-damper systems subject to gravity. Each system can be activated at desired time and has a control panel with 3 cursors to change the 3 coefficients by the mouse. The program must allow the user to solicit each system by moving the mass with the mouse and change the gravity.
11. **Pendulums.** Simulate N pendulums, whose parameters (mass, length and position) are specified in a configuration file. The user must be able to push a desired pendulum and the program must be able to manage collisions between adjacent pendulums.

12. **Waves.** Simulate a square matrix \((x, y)\) of NxN elastic elements where each element oscillates along the z-axis influencing its neighbors with a user-defined parameter. By pushing a desired element through the keyboard and representing each element with a color related to its height, the system should generate waves as in a lake. All parameters must be modifiable at runtime.

13. **Scara.** Simulate a SCARA robot in a 3D space (you can use OpenGL or a simple self-made library). Joint are actuated by dc motors controlled in position by PID regulators. The interface must allow the user to change the 3D view by mouse dragging, move each joint by pressing buttons with the mouse and load a text file with a trajectory to be executed in world space.

14. **Crane.** Simulate a crane with at 3 degrees of freedoms (rotation, cart sliding, and gripper going up/down from cart). The crane must grasp objects from the ground and move them in another location at a different height. Control the crane to avoid load oscillations assuming a rigid cable behaving like a pendulum. Use 3D graphics (see notes in the Scara project).

15. **Ball catching.** Simulate a system that launches balls at different speed and orientations that have to be caught by a moving basket mounted as an end-effector of a Cartesian robot. Use 3D graphics (see notes in the Scara project).

16. **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).

17. **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.

18. **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.

19. **LEM.** Simulate a LEM that has to land on a planet from a mother spacecraft, 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. Develop both manual and autonomous control.

20. **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.

21. **Pan-tilt camera.** Simulate a pan-tilt mobile camera controlled to track moving objects on the screen. The target can be moved by the mouse or by a task, like a random fly or with a sinusoidal path. Target, camera, motors, graphics and user interface must be implemented by different periodic tasks. The target must be tracked in a moving windows of variable size, enlarged when the object is lost and moved in a predicted position.

22. **Multi-camera tracking.** Simulate a system with multiple fixed cameras, each looking at specific portion of the screen. The system has to reconstruct the trajectory of a number of different objects (characterized by different colors) moving in the environment by integrating the partial views of the cameras. The various trajectories are then represented on a monitor in another portion of the screen.

23. **Patriots.** Simulate a set of Patriot defense missiles that identify enemy targets, predict their trajectories and are launched to catch them. Each patriot manages a radar that scans a portion of the sky. Enemy missiles are randomly generated or launched by the user.
24. **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/](https://www.cs.utexas.edu/~teammco/misc/kalman_filter/)

25. **Quadrotors.** Simulate a set of a 2D quadrotors that follows a desired noisy GPS position estimated using Kalman filter. Make a user interface similar to the one implemented in the following demo: [https://www.youtube.com/watch?v=nWWLJZRxAU](https://www.youtube.com/watch?v=nWWLJZRxAU)

26. **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).

27. **Space battle.** Simulate a space battle with two spaceships, one of them autonomous and the other controlled by the keyboard.

28. **Pinball.** Simulate a pinball game of your choice.

29. **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.

30. **Ping pong robot.** Simulate a Cartesian robot playing ping pong (in a 2D environment). The robot sees the ball through a camera. Motors have to be simulated with a proper transfer function and controlled by two PID regulators. The ball must be pushed with the paddle against the adversary to provide the required energy. The adversary can be the user or another robot.

31. **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).

32. **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 and 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 and control parameters of each telescope.

33. **Fishing.** Simulate a game in which you can fish in a lake with different types of fishes that eat different type of food floating in the water. Each fish has a short range vision and can recognize food based on its color.

34. **Skeet Shooting.** A skeet is launched 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.

35. **Mix.** Divide the screen in 9 (3x3) windows and simulate a set of physical systems controlled by periodic or aperiodic tasks (activated by pressing a key). Each system is displayed in a different window. Examples of animations can be a bouncing ball, a pendulum, a fly, a rotating Earth, a spring, a clock, a tank filling by drops generated by pressing a key (this is an aperiodic task), etc.

36. **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).

37. **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 range sensors that detect curves and other cars) and one is optionally controlled by the player through the keyboard arrows.

38. **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.
39. **Assembly chain.** Simulate an assembly chain where different objects go through different stages on a conveyor belt. Required stages are: visual recognition, selection (an object is discarded if the vision system detects an anomaly), assembling (a robot pick a piece depending on the specific object and mount it on the object).

40. **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 through the mouse.

---

**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.

41. **Image2sound.** Create a program that reads a picture from a file and converts it into sound. Consider to process the image by multiple concurrent tasks, each controlling a small moving window and using a different function to map image features to sound for a different instrument.

42. **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.

43. **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.

44. **Table sound.** Attach a microphone to a table (or any object) and process the produced signal to generate sound. The sound can be generated using MIDI notes. Consider to process the signal by multiple concurrent tasks, each analyzing different features and producing sound for a different instrument.

45. **Sound from gestures.** Acquire images from a camera, analyzed them and convert them into sound according to a give mapping algorithm.

---

**Kernel projects**

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

46. **Scheduling monitor.** Develop a set of functions to display the execution of tasks (including main) with an adjustable time scale. Visualize activations, 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.

47. **Aperiodic Server.** Implement a Deferrable Server or a Sporadic Server and build a test case to graphically visualize its behavior on a set of aperiodic tasks activated by pressing a key.

48. **Elastic scheduling.** Implement an Elastic Task Manager that varies task periods to cope with overload conditions. Develop a graphic interface to allow the user to change task parameters, activate/terminate tasks, and visualize the output of the elastic compression algorithm.

49. **CAB.** Implement the CAB mechanism and a test case to graphically visualize their behavior. Implement a test application to compare the communication delays experienced by CABs and classical semaphores.

50. **Communication ports.** Implement a communication port with different policies to enable tasks to exchange data via message passing. Then develop a graphic application to visualize the port behavior under different situations.
Hardware projects and neural networks

These projects consist in the development of a concurrent application on some hardware platform (e.g., the Arduino Due platform using the ARTE real-time interface provided by the RETIS Lab (see the ARTE website: [http://arte.retis.santannapisa.it/](http://arte.retis.santannapisa.it/))) or the Raspberry Pi.

51. **Neural characters recognition.** Implement a neural network on a Raspberry Pi that recognizes handwritten characters acquired by a camera. Image acquisition, neural inference, command interpreter and graphical outputs are managed by different tasks.

52. **Neural plate recognition.** Implement a neural network on a Raspberry Pi that recognizes a car plate acquired by a camera. Image acquisition, neural inference, command interpreter and graphical outputs are managed by different tasks.

53. **Artificial Nose.** Implement a smell recognition system using Arduino to acquire a set of smell sensors (provided by the RETIS Lab) and a Raspberry Pi (or similar) to process the collected data by a neural network.

54. **Neural audio recognition.** Implement a recurrent neural network on a Raspberry Pi that recognizes a set of sounds.

55. **Real-Time Deep Learning.** Implement an application (downloading a trained neural network) on the Jetson TX2 platform by NVIDIA (provided by the RETIS Lab) that acquires images from a camera and performs visual tracking on a set of objects.

56. **Robot crawling.** Develop a real-time application on a Raspberry Pi (or Arduino) that uses Reinforcement Learning to control a 2-link robot to move (see [https://www.youtube.com/watch?v=6afhNot8dlo](https://www.youtube.com/watch?v=6afhNot8dlo)). The robot can be fully simulated on Linux and visualized by Allegro. The two systems can communicate by serial line or Ethernet.

57. **Neural balance.** Develop a real-time application on a Raspberry Pi (or Arduino) that uses Reinforcement Learning to balance a ball on a rotating beam. The position of the ball is detected by a camera (see [https://www.youtube.com/watch?v=YYZXVt4dxAE](https://www.youtube.com/watch?v=YYZXVt4dxAE)). The system can be fully simulated and displayed under Linux. The two systems can communicate by serial line or Ethernet.

58. **Neural pendulum.** Develop a real-time application on a Raspberry Pi (or Arduino) that uses Reinforcement Learning to control an inverted pendulum simulated and displayed under Linux. The two systems can communicate by serial line or Ethernet.

59. **Neural tracking.** Develop a real-time application on a Raspberry Pi (or Arduino) that uses Reinforcement Learning to control a virtual camera simulated and displayed under Linux to track moving objects. The two systems can communicate by serial line or Ethernet.

60. **Autonomous driving.** Develop a real-time application on a Raspberry Pi (or Arduino) that uses Reinforcement Learning to control a virtual car simulated and displayed under Linux that has to move on a given track. The car detects the track borders through a set of range sensors. The two systems can communicate by serial line or Ethernet.