List of projects

1. **Birds**. Simulate a number of birds that exhibit a flock 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. Neighbors are detected by a short-range visual sensor sampling pixels in a semi-circular area (r, -PI, +PI).
   Interactive demo: https://eater.net/boids
   Background info1: https://www.red3d.com/cwr/boids/

2. **Pool game**. Simulate a pool game with 16 balls and six holes. The user must be able to select the direction and strength of the shot (by the mouse) and change a number of parameters, as the friction and dumping coefficients and the integration step. Pressing a key, the user can also enable/disable the visualization of the ball trails.

3. **Balls race**. Simulate N rolling balls, each falling down on a guide with a different shape: a tilted line, a piece of a parabola, a sector of a circle, and a piece of a brachistochrone, and a polynomial function. Each guide has the same start point (0, H) and the same end point (0, L). The program must allow the user to change the parameters of the guides, start the race, slowdown/speedup the simulation, count the elapsed time for each ball, determine the winner, reset the state, and visualize the speed vectors of the balls. See: https://www.geogebra.org/m/xHHUNEcR

4. **Marbles**. Simulate a number of colored marble balls that roll down on a surface with predefined fixed obstacles with different slope or moving obstacles (e.g., rotating sticks) positioned by the user with the mouse or read from a text file.

5. **Fountain**. Simulate a (2D) fountain sprinkling water from jets positioned in different directions. Water drops can be simulated by balls with a short wake periodically launched at different speeds. Implement a task to control jet speeds and directions following a predefined trajectory. Enable the user to change some parameters during simulation (e.g., lights, water speed, and jets orientation). See the following links for inspiration:
   Link 1: https://www.youtube.com/watch?v=Z171ZHAcGkw
   Link 2: https://www.youtube.com/watch?v=QFVgWjY6FMc

6. **Ufos**. Simulate a game where the user controls the position of a machine gun that can moves left or right and has to shoot alien spaceships (with different size) appearing on the top of the screen with different positions, directions and speed. Spaceships also shoot down to the user. The machine gun loses one unit of energy at every shot and 10 units every time it is hit, while it gains energy every time hits a spaceship (the gained energy is inversely proportional to its size).

7. **Fireflies**. Simulate N fireflies that slowly move and synchronize their blinking. Each firefly adjusts its phase by setting its value at the average phase of its neighbors. The program must enable the user to create new fireflies, vary their speed, change and visualize the detection radius, and change the task periods. Begin by setting a common period for all fireflies. Then, try to modify the algorithm to work with different initial phases and periods and synchronize both of them.

8. **Planets**. Simulate a star 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, enable the trail to visualize the past trajectory of each planet, and center the view on a desired planet. Look at the following link: https://en.wikipedia.org/wiki/Numerical_model_of_the_Solar_System.

9. **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.

10. **Soft body**. Simulate a soft body as N rigid balls connected by damped springs. Each ball is subject to gravity and elastic forces from other balls. The user should be allowed to move the body by the mouse, drop it down, change the gravity, the bumping coefficient, and the elastic/damping coefficients of the connections. See: https://www.youtube.com/watch?v=kyQP4t_wOGI&t=445s.

11. **Fluid in a box**. Simulate a fluid in a box using a set of interacting particles (taking collisions and forces into account). The box can rotate around its center and the user can impart commands to rotate it through the keyboard. The user should also be allowed to change the parameters of the simulation online. See the following web site for a demo and hints: https://google.github.io/liquidfun/.

12. **Sand/Fluid simulation**. Simulate the behavior of falling sand and fluid using cellular automata in 2D. The program has to allow the user to draw obstacles with the mouse and generate arbitrary amounts of the two materials on the screen. All parameters must be modifiable at runtime to see their effect on the simulation. See a tutorial on https://w-shadow.com/blog/2009/09/01/simple-fluid-simulation/ and a demo on https://www.youtube.com/watch?v=VLZjd_Y1gJ8&t=234s.

13. **Lake simulation**. Simulate a lake surface using a matrix of interacting elastic elements, whose parameters (mass, elasticity, dumping, and interaction) can be modified by the user. Then, simulate a set of falling stones generated by the user at different positions producing waves that propagate in two dimensions.

14. **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.

15. **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).

16. **Ball catching**. Simulate a system that launches balls at different speed and orientations that have to be caught by a moving basket controlled by the user.

17. **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).

18. **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.

19. **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.

20. **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.

21. **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.

22. **Visual tracking**. 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(s), camera, motors control, 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 position predicted by a Kalman filter. Gaussian noise should be added.

23. **Radar tracking**. Simulate a radar that scans a circular area and tracks a number of moving objects with different colored square windows. Each tracking window should follow the object with a period 10 times smaller than the rotation period of the radar using predicted positions estimated by a Kalman filter and updated at each scan. Each moving object and each tracking window is managed by a different task and Gaussian noise should be added to the acquired image.

24. **Patriots**. Simulate a defense system consisting of N Patriots that have to neutralize a number of enemy missiles, generated by the user. Patriots and enemy missiles are managed by different tasks. Each Patriot detects the presence of an enemy missile, estimates its position and speed and computes its trajectory in order to catch it. Catched missiles generate an explosion. Some crucial variables should be displayed on a state window.

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=nNWWLJZRxAU

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. **Pinball**. Simulate a pinball game with two moving paddles and some passive and active bumpers. It must be able to manage multiple balls at the same time, generated by the user.

28. **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.

29. **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).

30. **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 a cleaner 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.

31. **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 lanes and other vehicles. The highway can be displayed from top and the user must be able to zoom or track a specific vehicle.

32. **Car race**. Simulate a circuit where different cars are created by the user as interacting tasks. Each car is autonomous and reads a number of LiDAR sensors to detect obstacles at different directions. The circuit has to be upload from a file and the used can put additional obstacles with the mouse.

33. **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).

34. **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.

35. **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.

36. **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. For instance, a moving paintbrush can be controlled by different spectral features of the audio stream (color, thickness, trajectory changes, speed, etc.).

37. **ECG**. Display an ECG signal stored on a file and write a task that performs a real-time signal processing that detects the relevant ECG parameters to identify and visualize the anomalies.

38. **Fireworks**. Simulate a number of fireworks with different colors and effects. The user launches fireworks by pressing keys. Each firework start as a task that launches the fire with a given speed. Then, at a certain altitude, it explodes generating N other fires (i.e., tasks) launched in all directions. The children fires generated by the fire have the same color, which must fade away after a certain time. All fires leave a trail in the sky that must fade away.

39. **Air Hockey**. Simulate an air hockey game where a disk (moving the field without friction) has to be put into the opponent gate through a cylindrical paddle moved by the mouse. The adversary player has to be controlled by the computer.

40. **Archery**. Simulate a game where the user has to shoot arrows with a bow to hit various targets. Use the mouse to control the speed and tilt angle, and left/right buttons to control the pan angle. Simulate the presence of wind, whose speed is also visualized on the screen. The graphic has to show the frontal and the side views of the targets and the side view the arrow trajectory.

41. **Bugs life**. Simulate a number of simple organisms (bugs) that can move, sense the presence of food, eat, age, die, and reproduce when two adults (not old) of opposite sex stay close longer than a given interval. A bug dies if it does not eat for long time, eat too much, or becomes too old. Sensors provide data in a limited visual field (e.g., a circular sector described by r, $-\theta$, $+\theta$) and can distinguish obstacles, food and other bugs. Each bug is managed by a task whose code determines its behavior.

42. **Synth**. Simulate a sound synthesizer that generates samples by mixing a number of basic wave functions (e.g., sinusoidal, squared wave, saw, triangular). The user selects at runtime the waves to be mixed and presses the note to be played on a virtual keyboard. The user can also write a chord symbol (e.g. Cm7b5) and the corresponding notes (C, Eb, Gb, Bb) are automatically played by the program.

43. **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.

44. **Sound from gestures**. Acquire images from a camera of a person moving in front of it, analyze them and convert them into sound according to a given mapping algorithm.

45. **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.

46. **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.

47. **Inverted pendulums**. Implement an N inverted pendulums, each controlled by a different task using the same code. Develop a graphic interface to allow the user to change parameters, activate/terminate tasks, and visualize the angle of each pole as a function of time.

48. **Lunar lander**. Simulate a lunar lander where the user can control four propellers to accelerate in different directions. Visualize fuel consumption and all state variables.

Projects on Neural Networks

49. **Neural characters recognition**. Implement a neural network (one hidden layer is enough) that recognizes handwritten characters drawn by the mouse (with given thickness) on a small window with white background. Image acquisition, neural inference, command interpreter and graphical outputs are managed by different tasks.

50. **Neural plate recognition**. Implement a neural network (one hidden layer is enough) that recognizes the characters of a license plate provided as an image file. Image acquisition, neural inference, command interpreter and graphical outputs are managed by different tasks.

51. **Neural tracking**. Simulate a pan-tilt camera controlled by Q-learning that must learn to track moving targets. Each action consists of a small increment (left, right, up, down), or no action. Rewards are generated as a function of the position of the target centroid with respect to the center of the image. Image acquisition, inference, command interpreter and graphics are managed by different tasks.

52. **Robot crawling by RL**. Simulate a robot that uses Reinforcement Learning to control its two links to move (see https://www.youtube.com/watch?v=6afhNot8dIo). Image acquisition, neural inference, command interpreter and graphical outputs are managed by different tasks.

53. **Amoeba by RL**. Use Q-learning to train an agent to move in the environment, eat good food, and discard bad food. Food is randomly generated. The agent can sense the presence of food and its color. It can perform the following actions: no action, move forward, move left, move right, eat food below it. A positive reward is given if eating good food and a negative reward if eating bad food. Also, a negative reward is given if the agent does not move or does not eat any food for long time.

54. **Autonomous driving by RL**. Simulate a car moving on a track controlled by Reinforcement Learning. Track state is sensed by 3 or more lidar beams. Sensor acquisition, neural inference, command interpreter and graphical outputs are managed by different tasks.

55. **Inverted pendulum by RL**. Use Q-learning to train an agent to control an inverted pendulum. A negative reward is generated when the pole exceeds a given angle (e.g. +/-12 degrees) or the cart exceeds a given position on the x-axis. Neural inference, command interpreter and graphics are managed by different tasks.

56. **Ball & Beam by RL**. Use Q-learning to train an agent to control a ball-and-beam device. A positive reward is generated when the ball stays around the center of the beam for a minimum number of steps. A negative reward is generated when the ball hits one of the beam limits. Neural inference, command interpreter and graphics are managed by different tasks.

57. **Lunar lander by RL**. Use Q-learning to train an agent to drive the engines of a lunar module to land on the Moon. A positive reward is generated when the Lander touch the ground within a given area and with a speed lower than a maximum value. A small negative reward is given when lighting any rocket. A big negative reward is given when the Lander lands outside the given area or lands with a speed higher than the limit. Neural inference, command interpreter and graphics are managed by different tasks.

58. **Patriot by RL**. Use Q-learning to train an agent to control a defense system to shoot a moving target (e.g., an enemy missile) moving horizontally on top of the screen (left to right or vice versa). A positive reward is generated every time the target is hit, a small negative reward is given for each shot, and a big negative reward is given when the target is missed. Neural inference, command interpreter and graphics are managed by different tasks.