Intelligent Robotics Curricular Unit

Luís Paulo Reis

lpreis@fe.up.pt

Director/Researcher LIACC

Associate Professor at FEUP/DEI

Armando Sousa

asousa@fe.up.pt

Researcher INESC-TEC

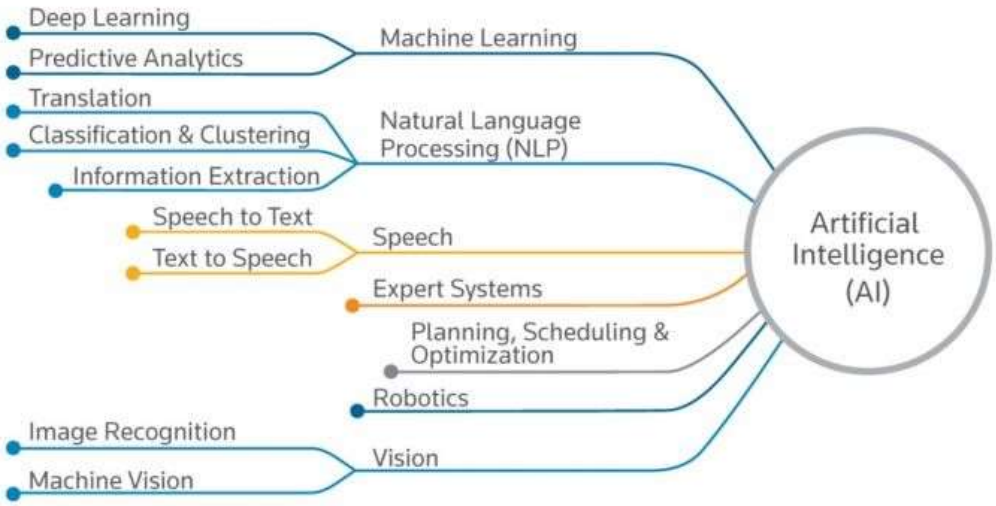
Assistant Professor at FEUP/DEEC

Obs: Language: English!

Artificial Intelligence (AI)

- **Intelligence**
  - "Capacity to **solve new problems** through the use of knowledge"

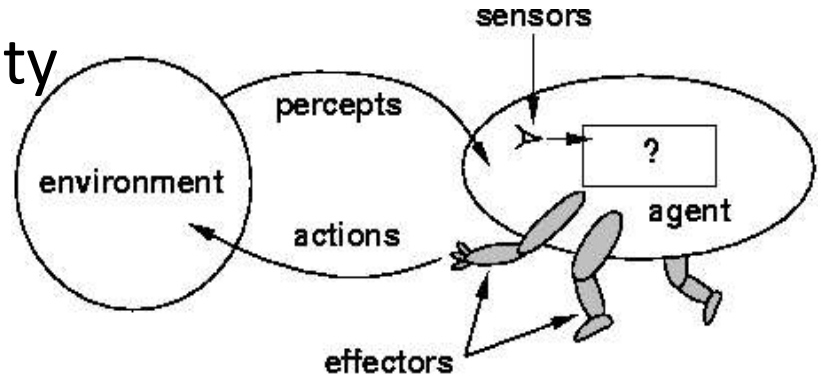
- **Artificial Intelligence**
  - "Science concerned with building **intelligent machines**, that is, machines that perform tasks that when performed by humans require intelligence"



Autonomous Agents and Multi-Agent Systems

Agent:

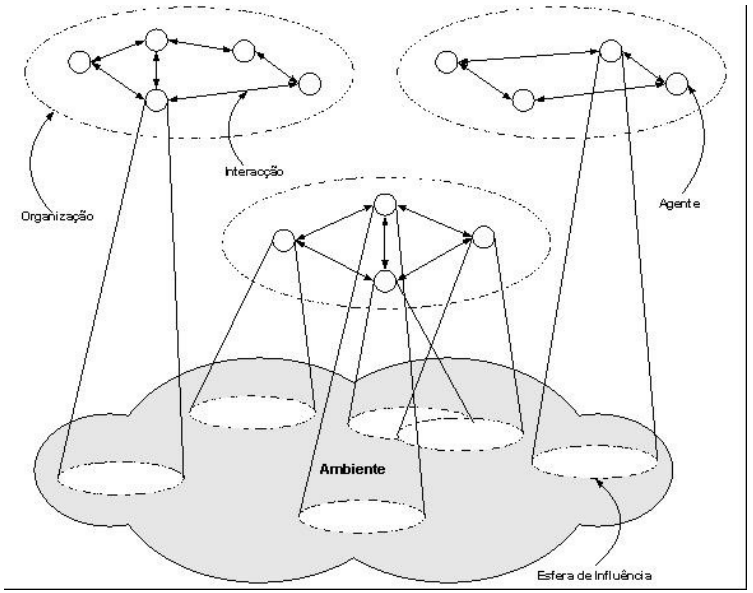
"Computational System, situated in a given **environment**, that has the ability to **perceive** that environment using **sensors** and **act**, in an **autonomous way**, in that environment using its **actuators** to fulfill a given **function**."



Russel and Norvig, "AI: A Modern Approach", 1995

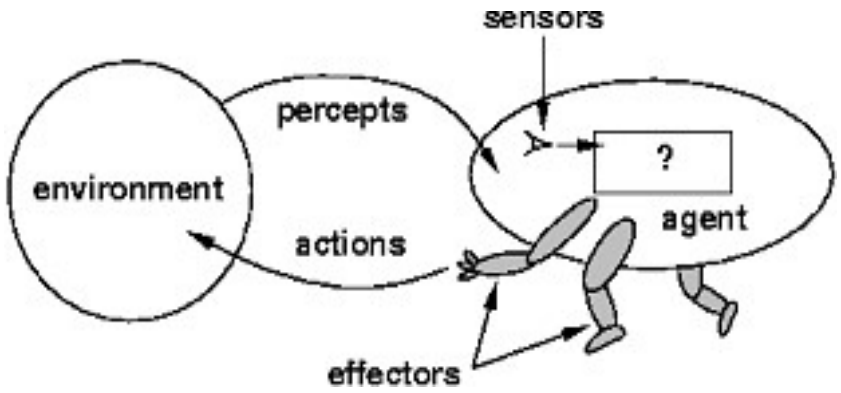
Multi-Agent System:

- Agents exhibit **autonomous behavior**
- **Interact** with other agents in the system



Robotic and Human Agents

- **Agent:**
  - Perceive its environment using sensors and executes actions using its actuators
  - Sensors:
    - Eyes, ears, nose, touch, …
  - Actuators:
    - Legs, Arms, hands, vocal cords, …



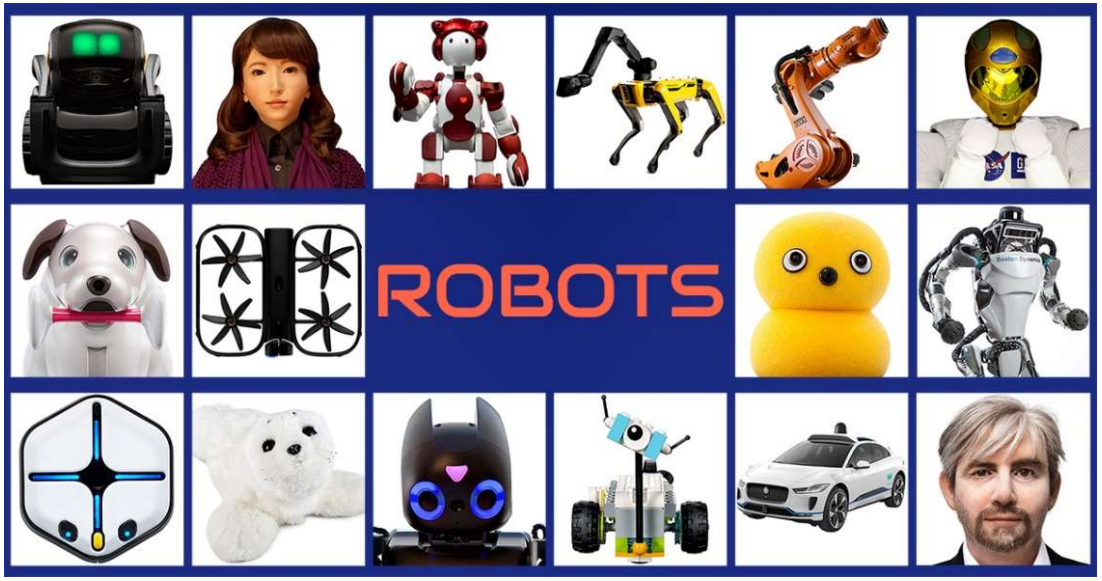
- **Robotic Agent:**
  - Sensors:
    - Cameras, sonar, infra-red, microphone
  - Actuators:
    - Motors, manipulators, speakers

Intelligent Robotics

- **Robotics**
  - Science and technology for **projecting, building, programming and using Robots**
  - Study of **Robotic Agents (with body)**
  - Increased Complexity:
    - **Environments:** Dynamic, Inaccessible, Continuous and Non Deterministic!
    - Perception: **Vision**, **Sensor Fusion**
    - Action: **Robot Control (Humanoids!)**
    - **Robot Architecture** (Physical / Control)
    - **Navigation** in unknown environments
    - **Interaction** with other robots/humans
    - **Multi-Robot Systems**

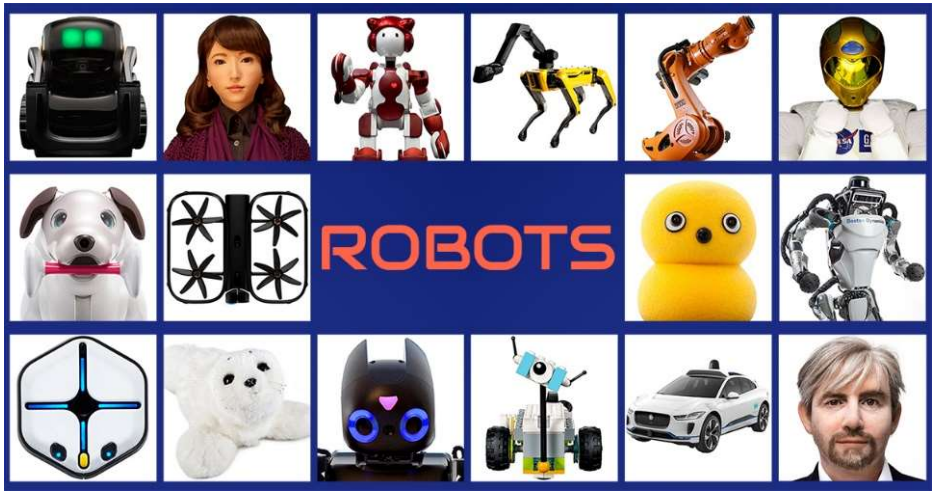
Definition of Robot

- **"Robot" --derives from Czech word *robota***
  - *"Robota"* : forced work or compulsory service
  - Term coined by Czech playright Karel Capek
- **Notion derives from 2 strands of thought:**
  - Humanoids -- human-like
  - Automata -- self-moving things
- **Current notion of robot:**
  - Programmable
  - Mechanically capable
  - Flexible
- **Best Definition of *robot:***
  - Physical agent that generates "intelligent" connection between perception and action



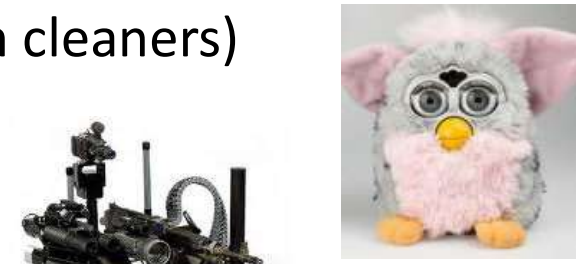
Definition of Robot

- **Robot "Robota" in Czech**
  - "*Robota*" : forced work or compulsory service
  - Karel Capek (1920)
- **General definitions:**
  - Simple: "Machine that is similar to humans in shape or function", "Machine that operates autonomously"…
  - "Physical Agent capable of establishing an (intelligent) connection between Perception and Action"
  - "Mechanical device capable of moving and that may perform physical tasks"
  - "Mechanical creature that may operate in an autonomous mode"
  - "Agent with Body!"



Current State of Robotics

- **Used to Perform:**
  - **Dangerous** or difficult **tasks** to be performed directly by humans
  - **Repetitive tasks** that may be performed more efficiently (or cheap) than when performed by humans
- **Robots moved from manufacturing, industrial applications to:**
  - **Domestic** Robots (Pets – AIBO, vacuum cleaners)
  - **Entertainment** robots (social robots)
  - Medical and **personal service** robots
  - **Military** and surveillance robots
  - **Educational** robots
  - Intelligent buildings
  - **Intelligent vehicles** (cars, submarines, airplanes)
  - New industrial applications (mining, fishing, agriculture)
  - Hazardous applications (space exploration, military apps, toxic cleanup, construction, underwater apps)
  - **Multi-Robot Applications and Human-Robot Teams!**

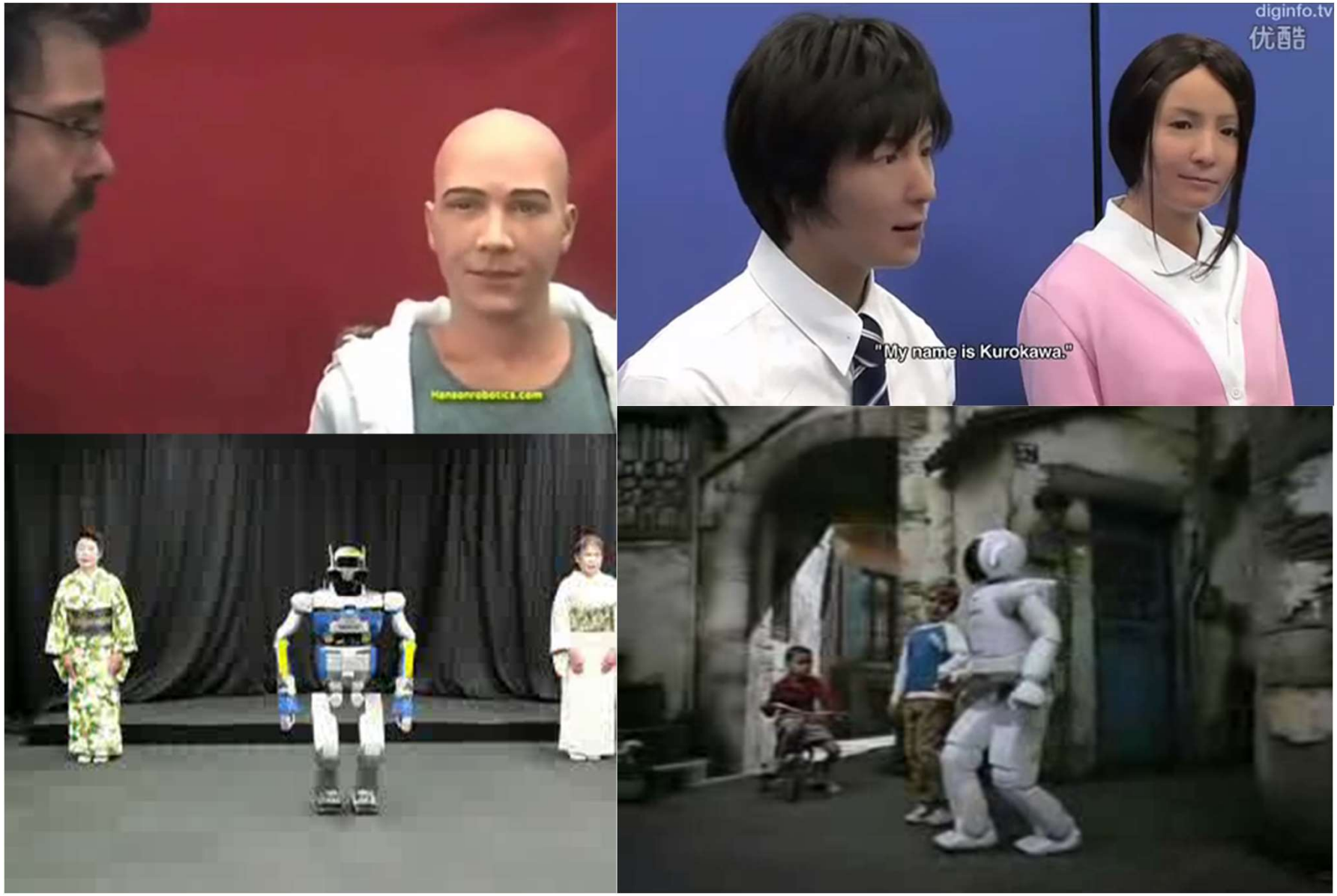




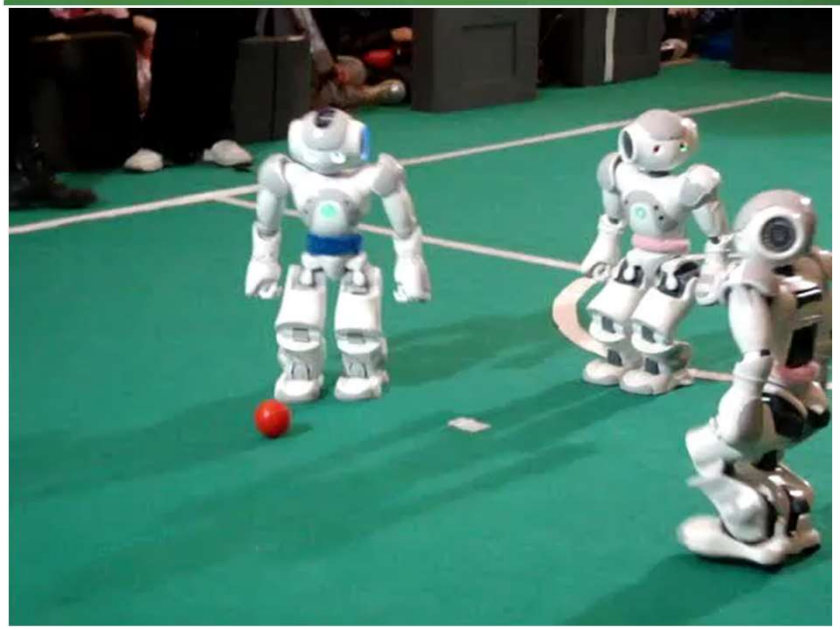

Robotics

- Autonomous driving car (Google)
- Service, mars explor., medical robotics (Motorman, Miimo, Roomba, Oz, Asimo, Nao)
- Exosekeleton (exoAtlete)
- Ambient Assisted Living
- Drones & Delivery (PT ConnectRobotics)
- Military, Assistive, Eldery, …
- Education, entertainment, …

AI - Humanoid Robotics



AI - Robotic Competitions - RoboCup



Control, Shape and Locomotion of Robots

- **Control:**
  - **Directly by a human** (space-shuttle robotic arm)
  - **Autonomous** based on its perceptions and decision methods (soccer playing robot in RoboCup)
- **Locomotion:**
  - **Wheels** (2, 4, etc.)
  - **Legs** (Bipeds, quadrupeds, hexapods)
  - **Snakes**
  - **Static** (Manipulators)
- Shapes:
  - **Humanoid** (shape and movement similar to humans)
  - **Mobile Robot** (autonomous vehicles)
  - **Industrial Manipulator** (shape depends on function)
  - **Reconfigurable** (change shape)

Robots: Hollywood vs. Real-World

- **Hollywood Robots:**
  - Human-like capabilities
  - "Sense all, know all"!
- **Real-World Robots:**
  - Insect or simple animal capabilities
  - "Sense little, know little"!

Visions: Dangers and Fears

- **Books:**
  - Frankenstein – 1818: Machine (monster) turns against its "creator"…
  - Work of Isaac Asimov about Robots and their interaction with society – IRobot (Asimov's laws of Robotics)
- **Old Movies:**
  - Metropolis (1926)
  - The Day the Earth Stood Still (1951)
  - Forbidden Planet (1956)

Visions: Dangers and Fears

- **Classical Movies:**
  - 2001 Space Odyssey (1968)
  - Star Wars (1977)
  - Blade Runner (1982)
  - Terminator (1984)
  - Matrix (1999)
  - Artificial Intelligence (2001)
  - IRobot (2004)
  - Ultron

Asimov's Robotic Laws

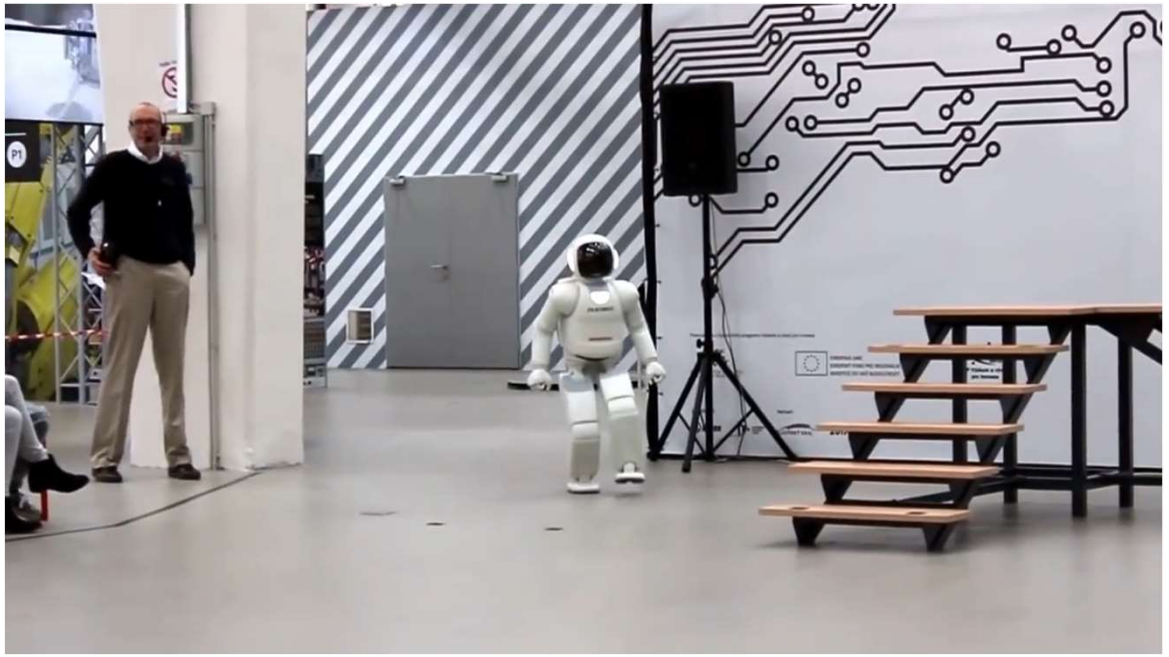
The **Three Laws of Robotics** are a set of three rules written by Isaac Asimov, which almost all Robots appearing in his fiction must obey. Introduced in his 1942 short story "Runaround", although foreshadowed in a few earlier stories:

Law 0) A robot **may not injure humanity** or, through inaction, allow it.

Law 1) A robot **may not injure a human being** or, through inaction, allow a human being to come to harm.

Law 2) A robot **must obey orders given to it by human beings**, except where such orders would conflict with the First Law.

Law 3) A robot **must protect its own existence** as long as such protection does not conflict with the First or Second Law.



Robotic Competitions

- **DARPA Grand-Challenge**
- **Intelligent Ground Vehicle Competition**
- **AAAI Grand Challenges**
- **RoboCup (Robotic Soccer World Championship)**
- **Robotic Soccer FIRA**
- **First Lego-League**
- **RoboOlympics**
- **Manitoba Robot Games**
- **Robotic Fight: BattleBots, RobotWars, Robot-Sumo**
- **Portuguese Competitions:**
  - Festival Nacional de Robótica – Portuguese Robotics Open (including autonomous driving)
  - Micro-Mouse / Ciber-Mouse (Micro-Rato / Ciber-Rato)
  - Firefighting Robots

Robot Composition

- **Sensors**
  - Used to perceive the world
- **Effectors and actuators**
  - Used for locomotion and manipulation
- **Controllers for the above systems**
  - Coordinating information from sensors
  - Commanding the robot's actuators
- **Robot:**
  - **Autonomous** system which exists in the **physical world**, can **sense** its environment and can **act** on it to achieve some goals

Challenges in Robotics

- **Perception**
  - Limited, noisy sensors
- **Actuation**
  - Limited capabilities of robot effectors
- **Thinking**
  - Time consuming in large state spaces
- **Environments**
  - Dynamic, fast reaction times needed
  - Inacessible, thing about sensing
  - Continous, huge state space
  - Non-Deterministic, no garanty of success

Uncertainty

- **Uncertainty** is a key property of existence in the physical world
- **Environment is stochastic and unpredictable**
- **Physical sensors** provide **limited, noisy, and inaccurate information**
- **Physical effectors** produce **limited, noisy, and inaccurate action**
- Models are simplified and inaccurate
- **Errors in perception, action and movement**

Uncertainty

- **A robot cannot accurately know the answers to the following questions:**
  - Where am I?
  - Where are my body parts, are they working, what are they doing?
  - What did I just do? Was my action successfull?
  - Am I capable to do X? What will happen if I do X?
  - Who/what/where are you? What are you doing?

Classical activity decomposition

- **Locomotion (moving around, going to places)**
  - factory delivery, AGVs, Mars Pathfinder, vacuum cleaners...
- **Manipulation (picking and handling objects)**
  - factory automation, robotic arms, production lines, automated surgery...
- **Division of robotics into two basic areas**
  - mobile robotics (move around)
  - manipulator robotics (static)
- **But these areas are together in domains like robot pets, robotic soccer and humanoid robots**

Intelligent Robotics

- **Intelligent Robotics Course Focus:**
  - "Mobile Robotics"! (not manipulator robotics)
  - *Intelligent Software! (not robotic hardware)*
  - *Cooperative Robotics*
  - Designing Algorithms that allow robots to perform cooperatively, complex tasks, autonomously, in unstructured, dynamic, partially observable, non-deterministic and uncertain environments

Software for Intelligent Robots

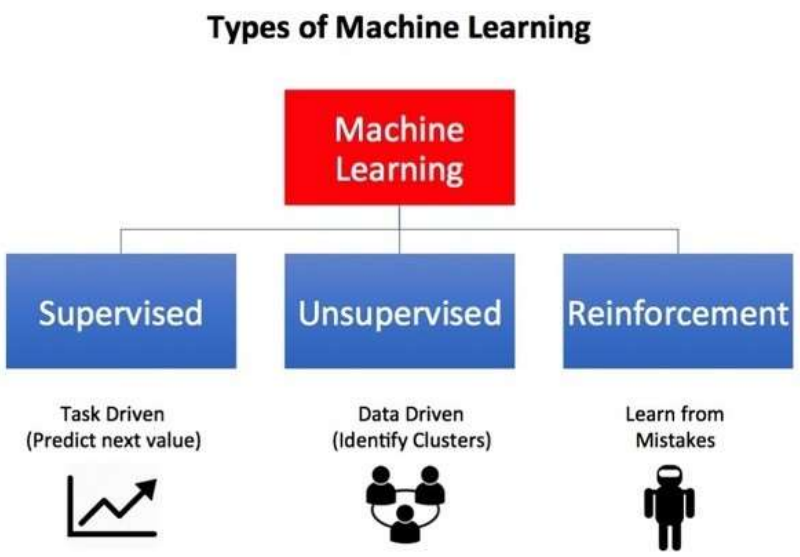
- **Software enabling autonomous mobile robots to perform, cooperatively, complex tasks, in unstructured, dynamic, partially observable, and uncertain environments:**
  - **Autonomous**: robot makes majority of decisions on its own; no human-in-the-loop control (as opposed to *teleoperated*)
  - **Mobile**: robot does not have fixed base (e.g., wheeled, as opposed to *manipulator arm*)
  - **Unstructured**: environment has not been specially designed to make robot's job easier
  - **Dynamic**: environment change while robot is "thinking"
  - **Partially observable**: robot cannot sense entire state of the world (i.e., "hidden" states)
  - **Uncertain**: sensor readings are noisy; effector output is noisy
  - **Complex Tasks**: Tasks are not easy (such as follow a straight line)
  - **Cooperatively**: Robot needs to cooperate with other robots/humans to be able to do the task

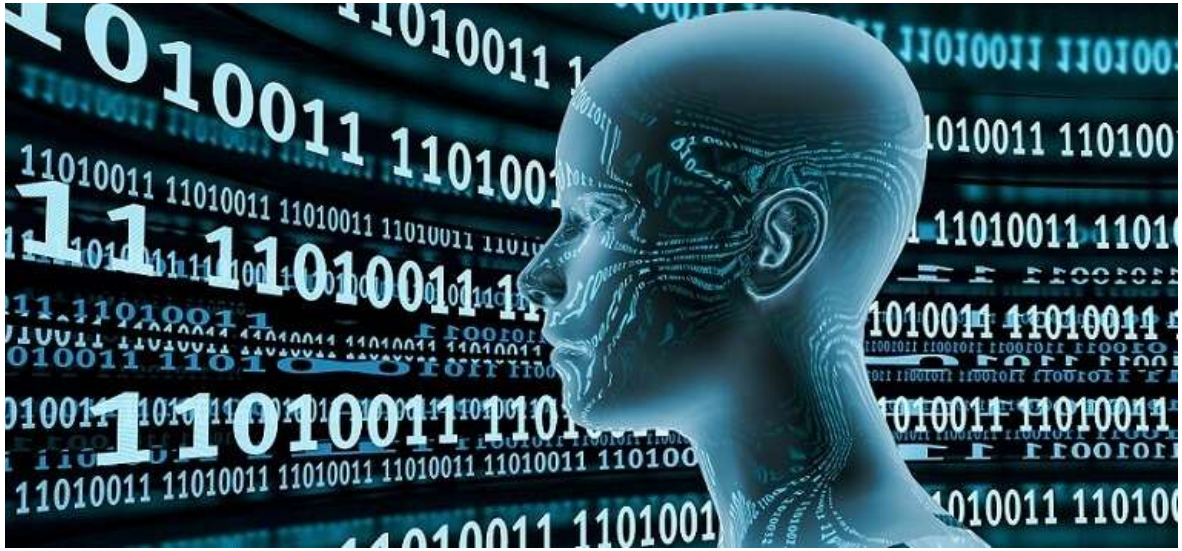
Not Covered in IR

- **Kinematics and dynamics:** covered in mechanical engineering
- **Teleoperated systems:** covered in mechanical / electrical engineering
- **Traditional robotic control theory:** covered in electrical engineering
- **Theory of mind, cognitive systems:** covered in psychology, cognitive science…
- ***Focus on computer science issues adapted to MEIC and PRODEI:***
  - *Algorithm development, Artificial Intelligence, Software architecture, etc.*

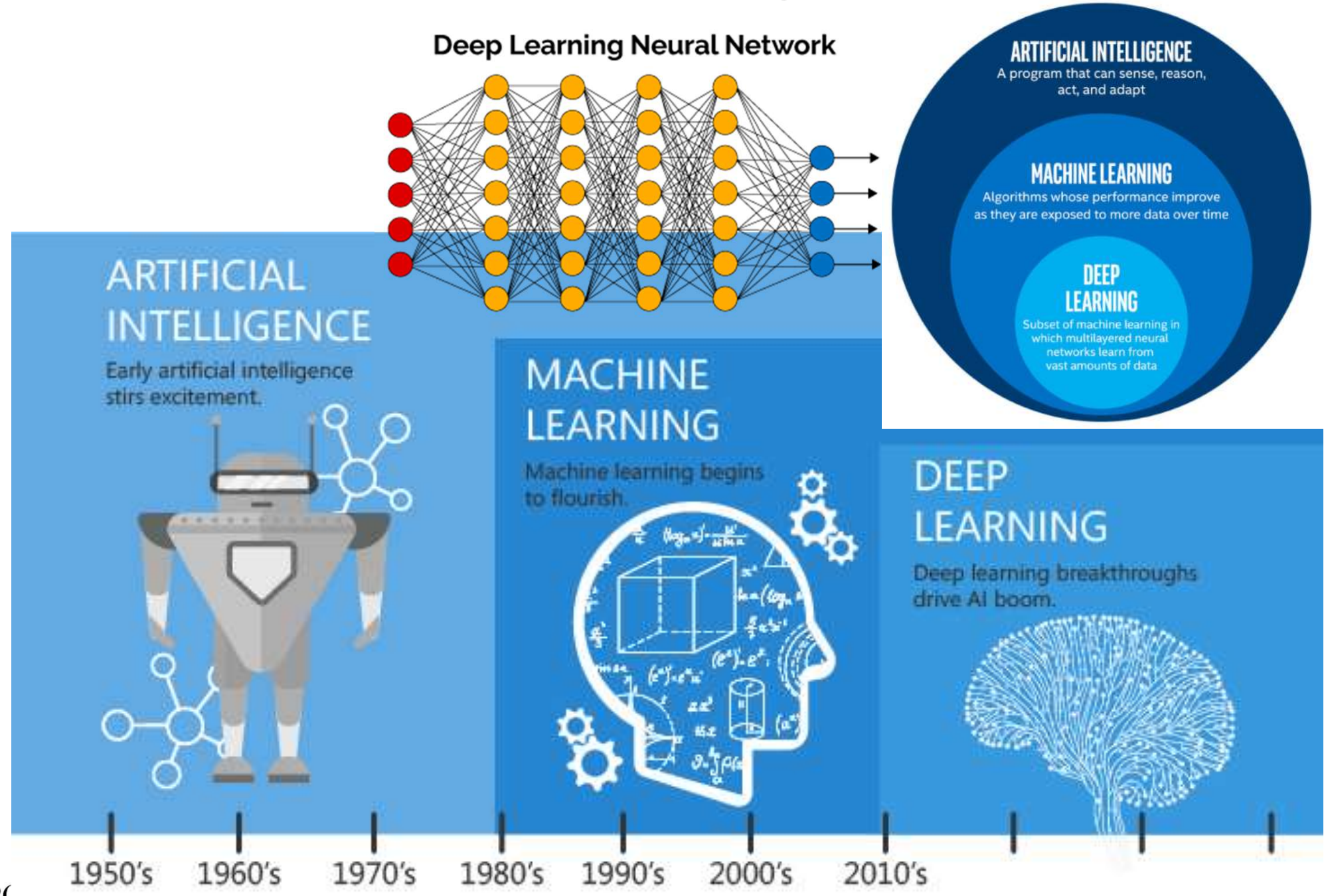
Machine Learning

Machine learning is a field of artificial intelligence that gives **computer systems** the **ability to "learn"** (e.g., progressively **improve performance** on a specific task) **from data/results of their actions**, without being explicitly programmed





Machine Learning - History



Machine Learning Motivation

Table-Tennis

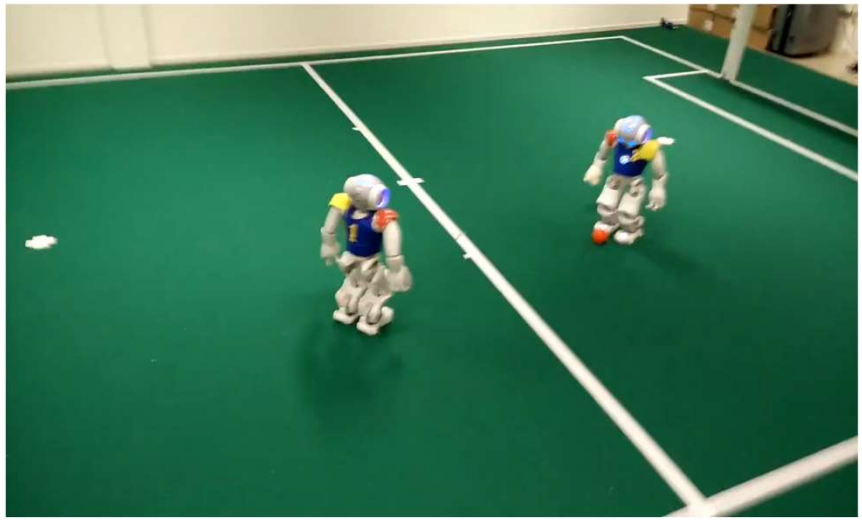
Robots

Mülling + Peters

Humans

Soccer Ball Passing

Robots

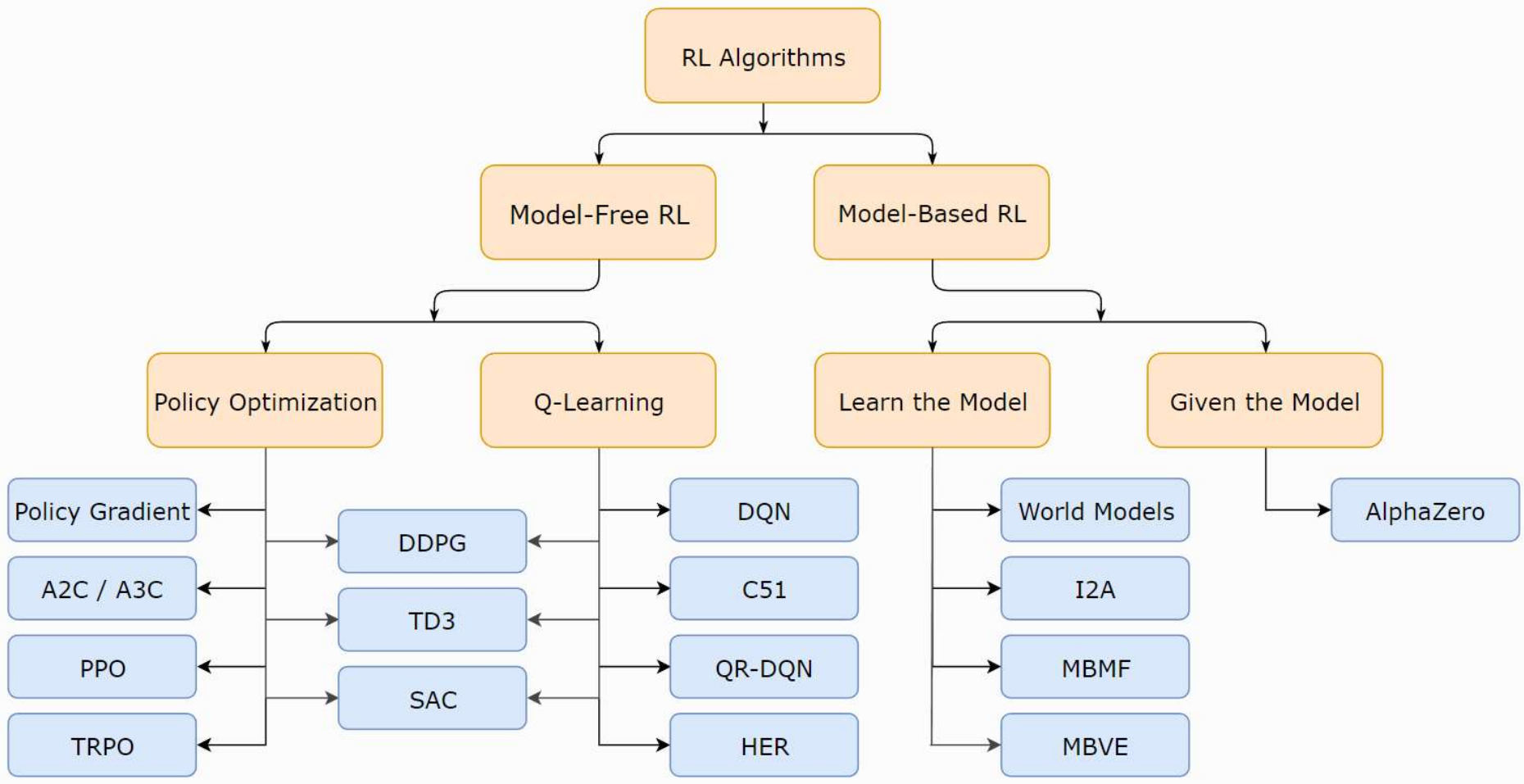


Erik Orjehag – LIU H1

Humans

We need **learning** and **adaptation** to improve robot skills!

Reinforcement Learning

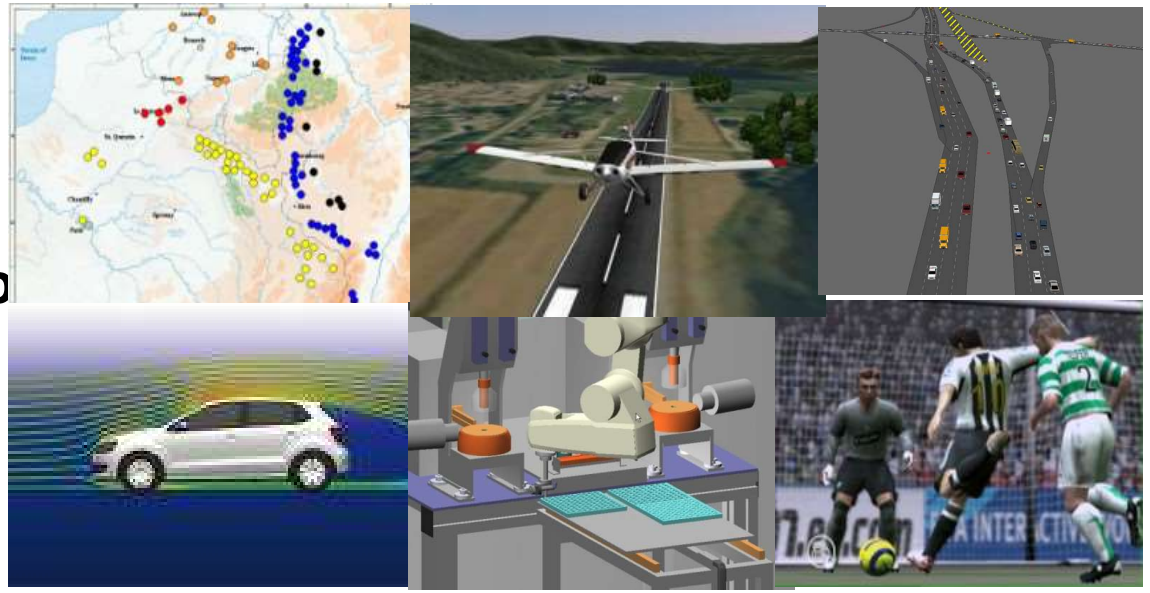


Agents and Multi-Agent Systems

- **To build individual autonomous intelligent agents is important**
- **However:**
  - Agents don't live alone…
  - Necessary to work in group…
  - Multi-Agent Applications!
  - Robotic Agents: Body, Complex Environment
  - Coordination in necessary:
    "To Work in Harmony in a Group"

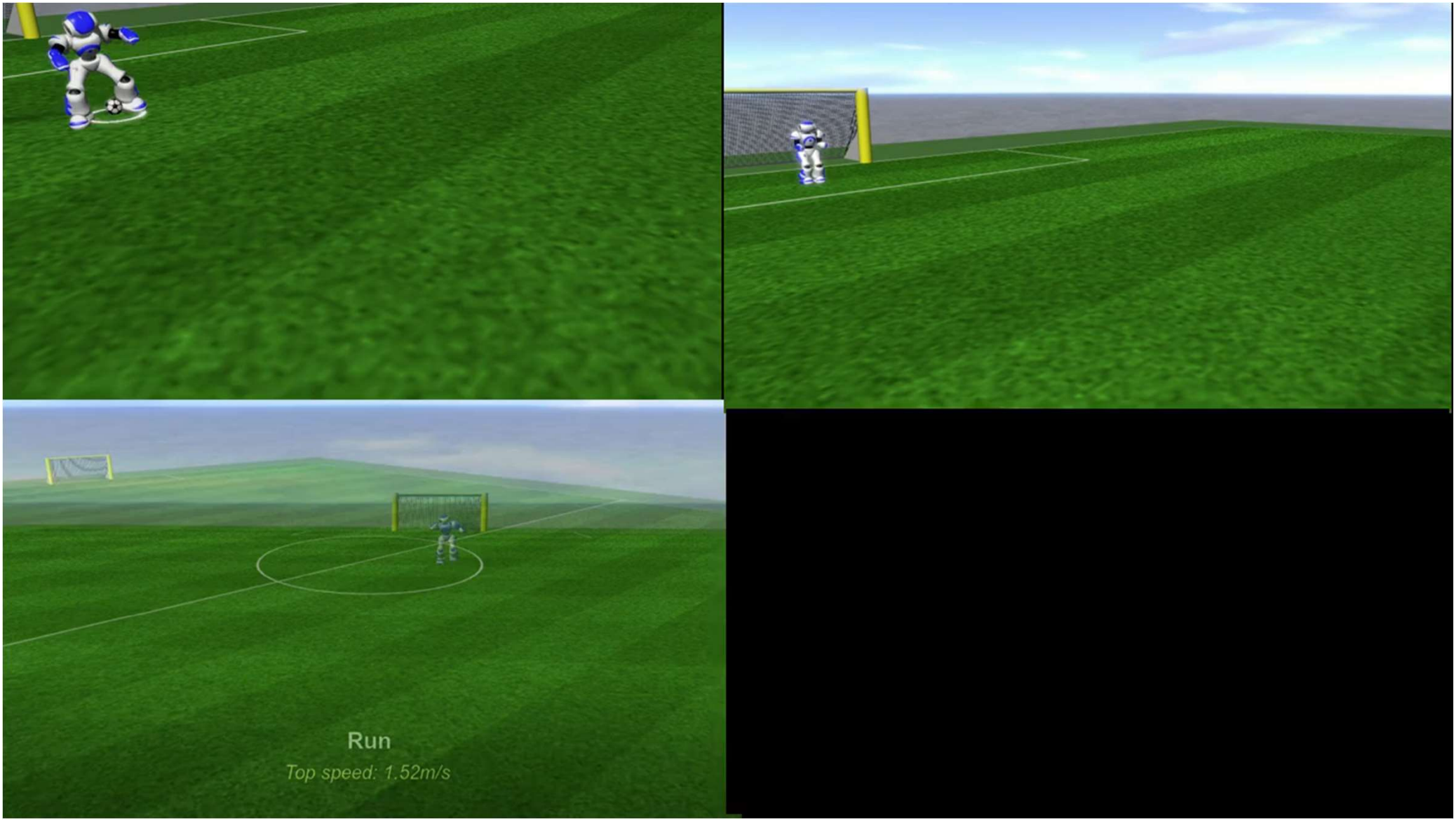
AI - Agent-Based Simulation

- **Simulation: Imitation of some real thing, state of affairs, or process, over time,** representing certain **key characteristics** or behaviours of the physical or abstract system
- Applications:
  - Understand system **functioning**
  - **Performance optimizatio**
  - Testing and validation
  - **Decision making**
  - Training and education
  - **Test future/expensive systems**

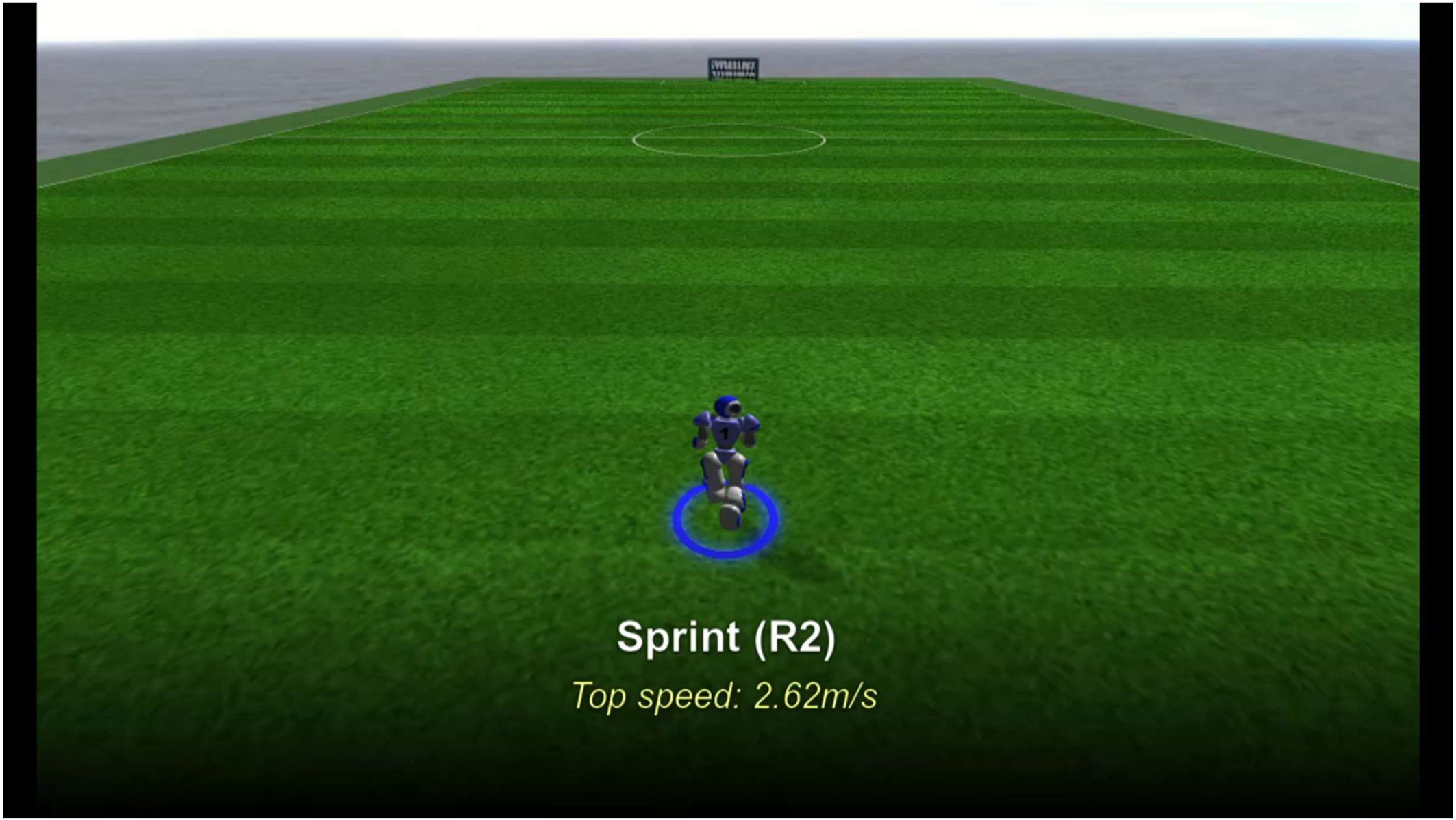


- For complex systems impossible to solve mathematically
- **Agent Based Modeling and Simulation**
- **Compress/Accelerate Time: Machine Learning**

Learning to Walk and Run

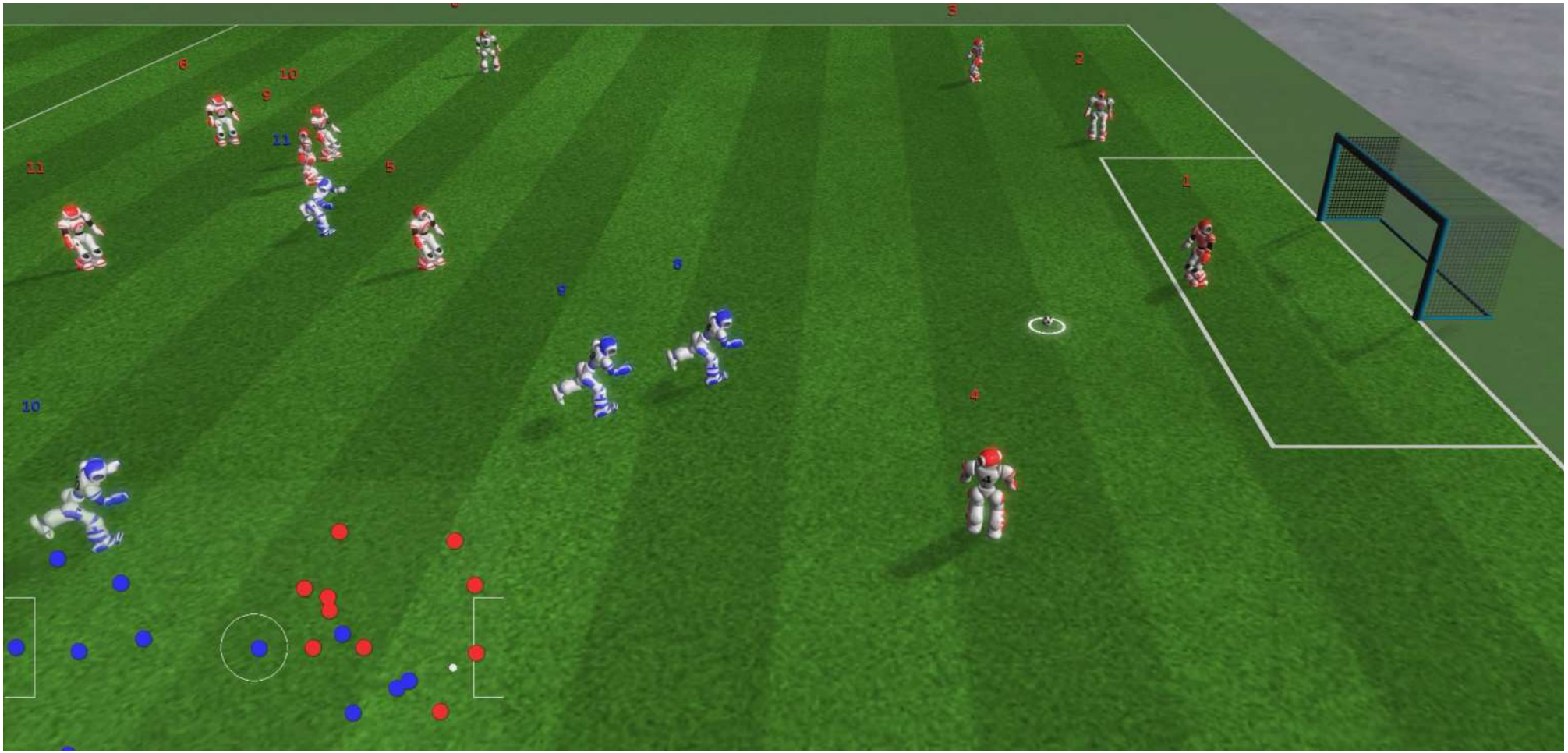


Learning to Sprint



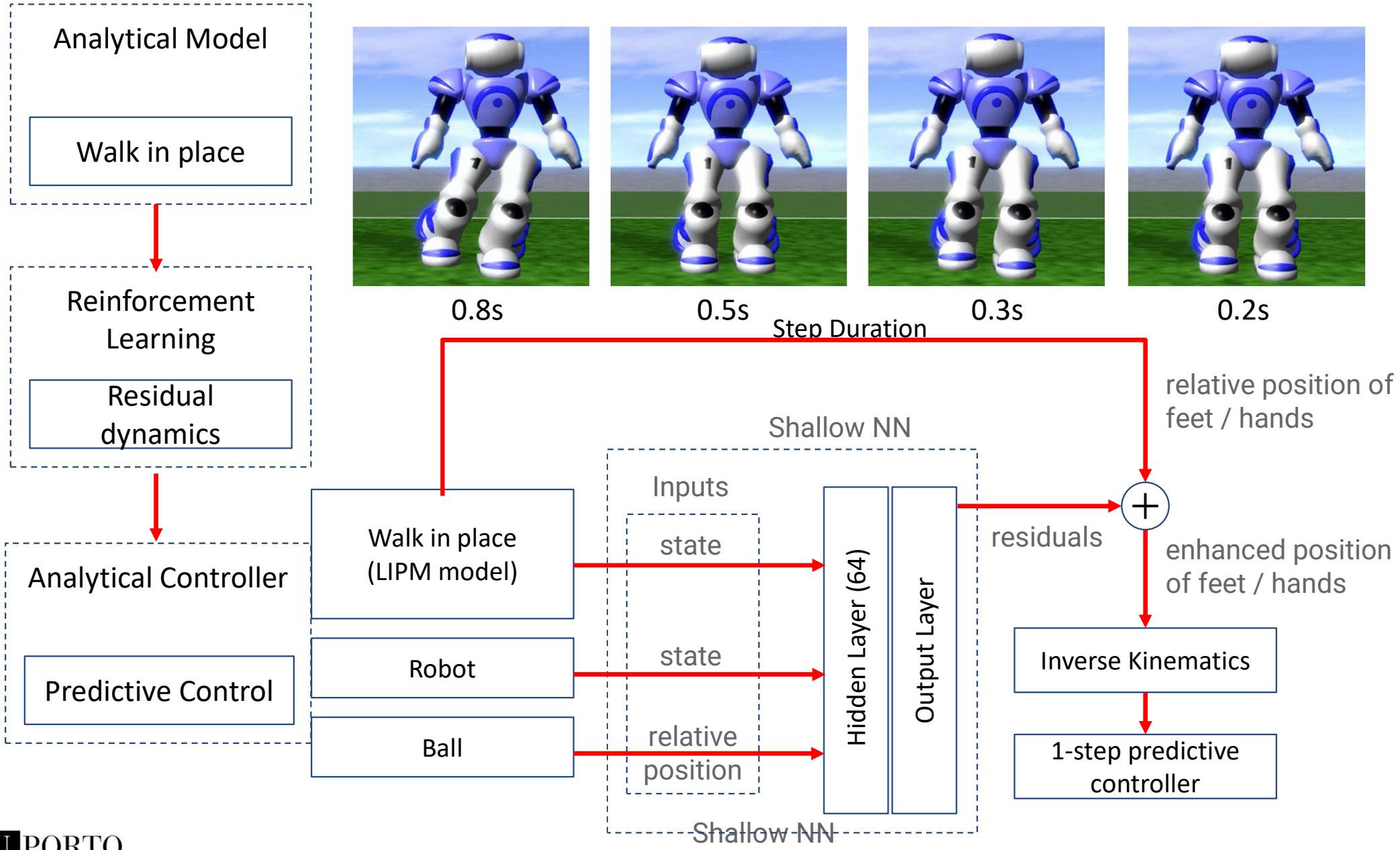
Learning to Sprint

Learning to Sprint



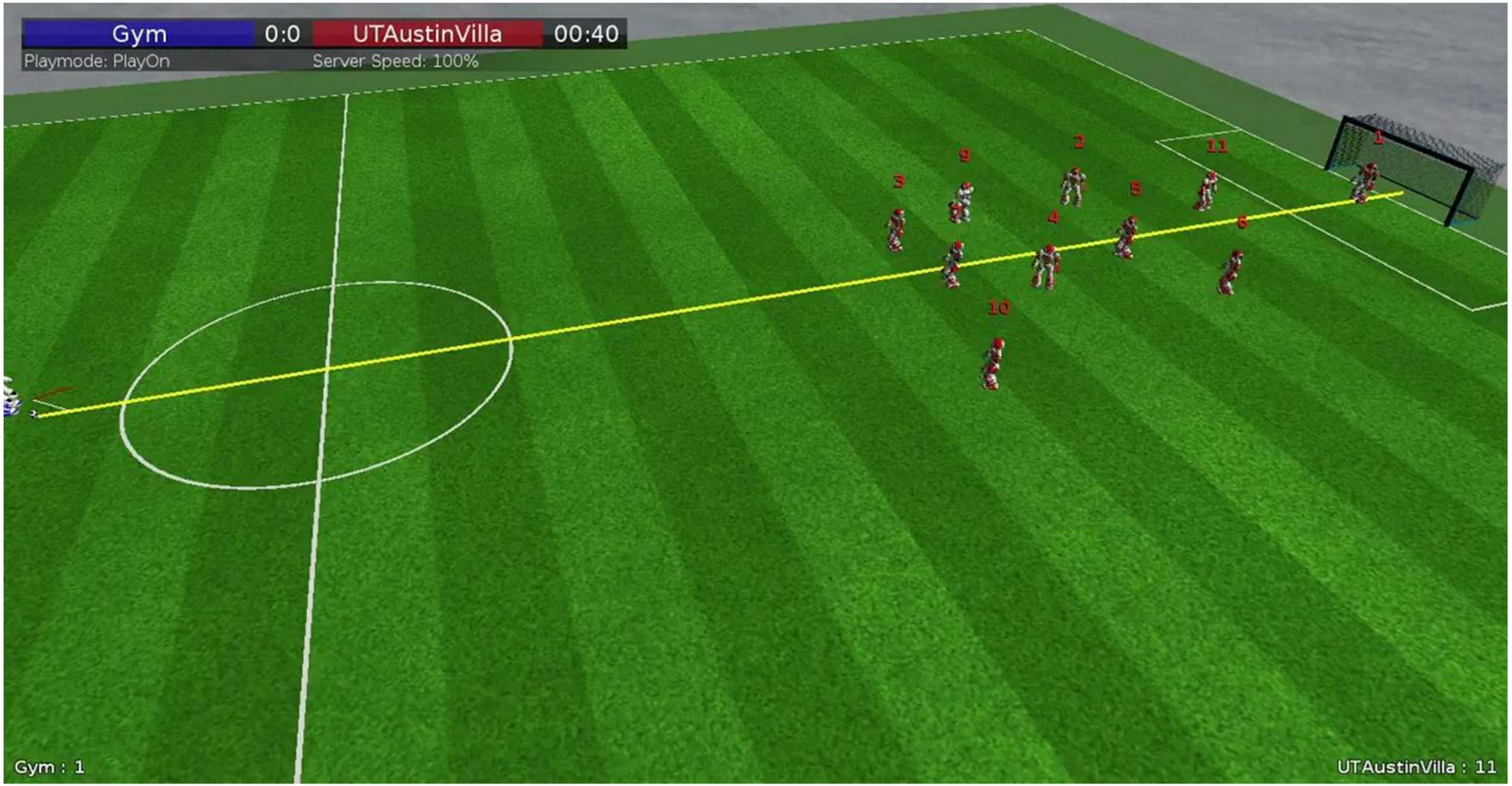
Video

Learning to Dribble



Learning to Dribble

Dribble against UT Austin Villa (2021)



Course Syllabus/Main Topics

3.1 Introduction to Intelligent Robotics

3.2 Robotics' Middleware, SDKs and ROS

3.3 Perception and Sensorial Interpretation

3.4 Locomotion and Action

3.5 Localization, Mapping and SLAM

3.6 Planning and Navigation

3.7 Robot Learning

3.8 Human-Robot Interaction

3.9 Cooperative Robotics and Human-Robot Teams

3.10 Robotics in the Future

Bibliography

[Siegwart et al., 2011] Roland Siegwart, Illah Reza Nourbakhsh, Davide Scaramuzza, Introduction to Autonomous Mobile Robots, 2nd Edition MIT Press, 2011,

[Murphy, 2019] Robin R. Murphy, An Introduction to AI Robotics, 2nd Edition, Bradford Book, MIT Press, Cambridge, Massachusetts, London England, 2019,

[Russel and Norvig, 2021] Stuart J. Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, 4th Edition, Pearson, 2021

[Sutton and Barto, 2018] Richard S. Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction. 2nd Edition, MIT Press, Cambridge, Massachusetts, London England, 2018,

[Thrun el al., 2005] Sebastian Thrun, Wolfram Burgard, Dieter Fox, Probabilistic Robotics, MIT Press, Cambridge, Massachusetts, London England, 2005. 62-3

[Choset et al., 2005] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, Sebastian Thrun, Principles of Robot Motion: Theory, Algorithms, and Implementations, Bradford Book, MIT Press, Cambridge, Massachusetts, London England, 2005.

[Peters, 2013] Jan Peters, Machine Learning for Robotics: Learning Methods for Robot Motor Skills, VDM Verlag Dr. Müller, 2013

[O'Kane, 2013] A Gentle Introduction to ROS - Jason M. O'Kane - https://www.cse.sc.edu/~jokane/agitr/

[Joseph, 2018] Robot Operating System (ROS) for Absolute Beginners: Robotics Programming Made Easy - Lentin Joseph – Associated Press, 2018

Evaluation System

C1) ROS Simple work: Implementation of a reactive robot for performing simple tasks using ROS

C2) Class Participation (including Kahoots and Moodle Activities)

C3) Practical Project concerning intelligent robotics:

- **C3a) Initial Contract/Intermediate presentation/oral defence**
- **C3b) Final demonstration, presentation/oral defence, and video**
- **C3c) Scientific paper describing the project**

Final Grade =

$$0.20*C1 + 0.20*C2 + 0.10*C3a + 0.25*C3b + 0.25*C3c$$

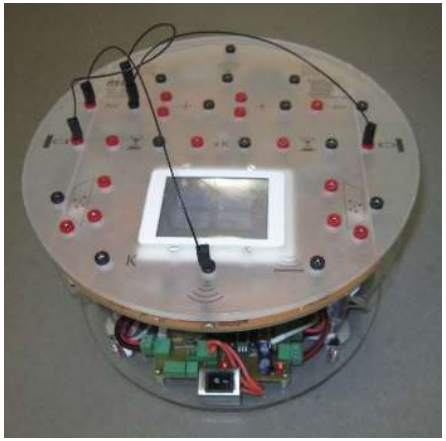
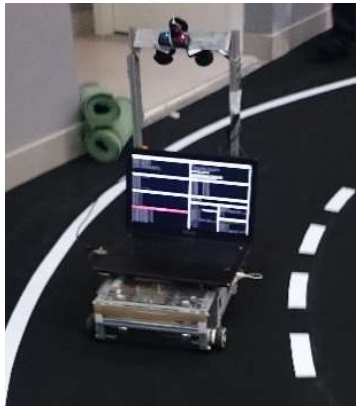
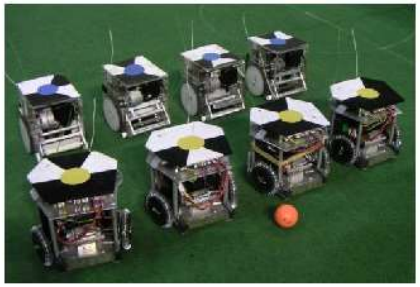
Platforms



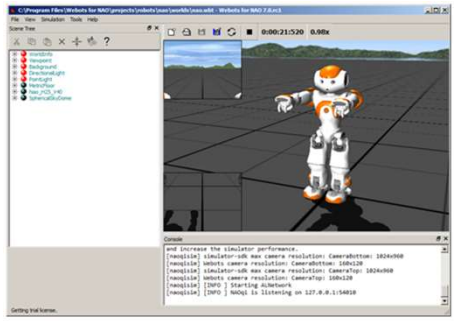
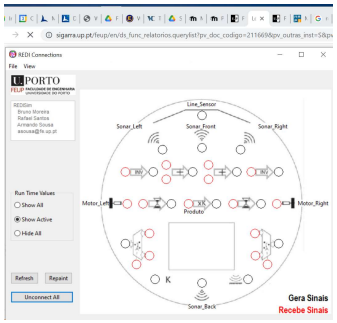
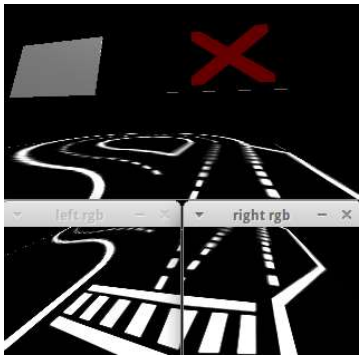


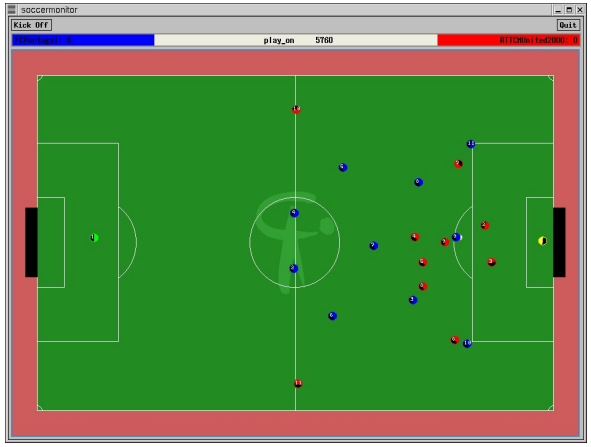


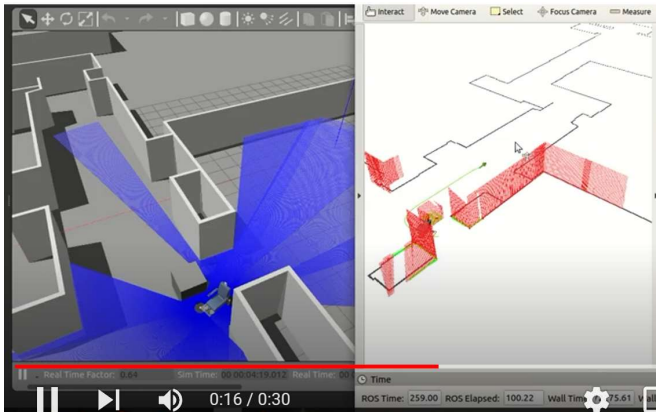


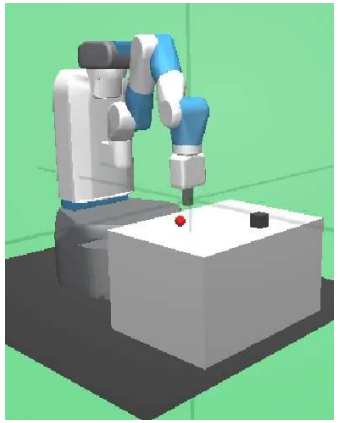





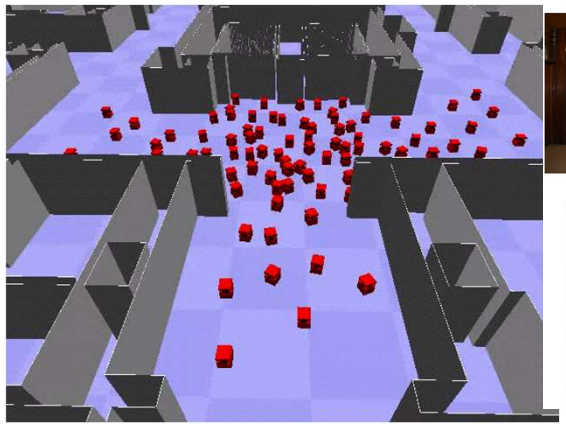

Simulators

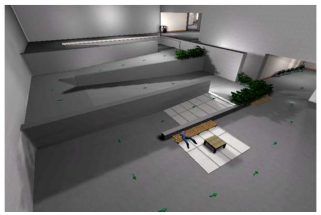
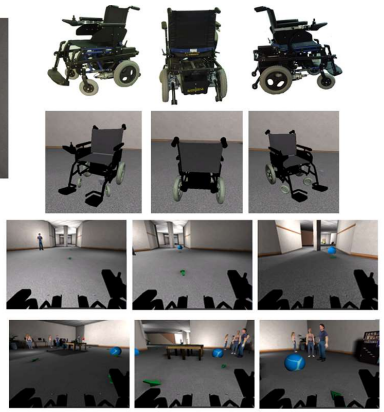











Base Codes

- FCP2D – FCPortugal Simulation 2D RoboCup Agent and Team,
- CiberFEUP 2.0 – Simple Agent Base Code for Ciber-Mouse
- FCP3D Agent – FCPortugal Humanoid Simulation 3D RoboCup Agent and Team, Base Code, C++
- SPlanner Software for Creation, Execution and Test of Setplays
- FCP3D Goalie – Simulation 3D RoboCup Goalie Agent
- EuRoc/TIMAIRIS Sim. Base Code for Robotic Manipulator Tasks
- **Conde Simulator and ROS based Autonomous Driving Agent Base Code**
- **IW Navigation Simulator - ROS based Simulator, SLAM and Navigation Base Code**
- **PyFCP3D – FCPortugal Humanoid Simulation 3D RoboCup Agent and Team, Base Code, developed in Python**

Competitions

- Ciber-Mouse (Ciber-Rato) and Micro Mouse (Micro-Rato)
- **Festival Nacional de Robótica – Portuguese Robotics Open (several competitions)**
- **RoboCup – World Championships and European Championship (several competitions)**
- Robot@Factory at the Portuguese Robotics Open
- **Autonomous Driving ("Condução Autónoma") at the Portuguese Robotics Open**
- Soccer Simulation2D League at RoboCup, EuroRoboCup and FNR
- **Soccer Simulation3D League at RoboCup, EuroRoboCup and FNR**

Students Projects/Works

Projects for the Robotics course at M.EIC 2021/2022:

- **Autonomous Driving Conde Simulator,**
- **DuckieTown Simulator,**
- **IntellWheels SLAM, IntellWheels Navigation**
- **IntellWheels Wheelchair Dating/Following with RL**
- **RoboCup3D Humanoid Walking/Kicking/GetUp Learning**
- **RoboCup3D Goalie Challenge Learning**
- **RoboCup3D Path Planning/Dribbling Learning**
- **RoboCup3D Multi-Robot Learning**
- **Multi-Robot Collaborative Coverage in ROS**
- **Real Intelligent Wheelchair SLAM in ROS (Zed/RealSense L515)**
- **TurtleBot - LIDAR SLAM**
- **DuckieTown - Robot Assembly and AI City Driving**
- **DuckieTown - Robot Assembly and City SLAM**
- **DuckieTown - Robot Assembly and MadDuckieAvoidance**

Students Projects/Works

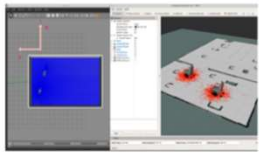


Fig. 3. On the left the gazebo simulated environment and the consequent visualisation in rviz.

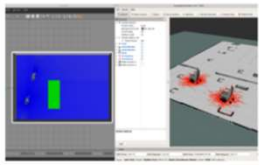


Fig. 4. The green zone represents the random locations of the goal during the training.

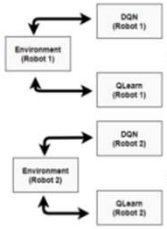


Fig. 5. Logical independence between the environment and the agent.

the environment and feed the input neural network, and Q-Learning algorithms.

The sampling process starts by subscribing the ROS topic where the messages of the laser scan are redirected and find the ray with minimal distance to an obstacle. Then, according with a predefined threshold (10 rays) is collected the neighbour rays of that minimal hit point (Figure 6).

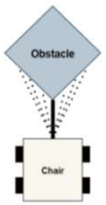


Fig. 6. Sampling lidar data according to minimal hit distance.

C. Relation between the agents and environment

Each chair is considered an independent agent acting independently to the environment constraints (Figure 5). Also, two different versions of each agent were designed using different reinforcement learning algorithms: Deep Q-Network (DQN) and Q-Learning.

The learning processes started by defining an algorithm where the leader chair tried to find a goal placed in random positions. The DQN model was trained by running 350 episodes with 1000 steps each with a learning rate of 0.00025. Then, the same neural network model was reused and applied to the follower chair, which is inserted in a different environment, instead of following a goal, follows the leader chair.

The Q-Learning algorithm [25] was trained with 10000 with 5000 steps by episode, with tuned parameters of: alpha 0.1, epsilon 0.3259 and epsilon discount of 0.99999. The same approach of DQN was followed, and the same parameters were assigned to the follower chair.

Fig. 1: Robots playing in the magmaOffenburg Keepaway challenge

1: Initialize $Q(s,a) = 0$, for all $(s,a) \in \mathcal{S} \times A$
2: **for each** episode **do**
3: $s \leftarrow$ state of the environment
4: **while** s is not terminal **do**
5: $a \leftarrow \Pi(s)$
6: Execute action a
7: $r \leftarrow$ reward obtained from the action
8: $s' \leftarrow$ state of the environment
9: $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s',a'))$
10: $s \leftarrow s'$
11: **end while**
12: **end for**

large time cost. The agents also have the ability to get back up in case they fall.

B. Reinforcement Learning

Reinforcement learning is a machine learning paradigm that allows agents to learn behaviors through trial-and-error with a dynamic environment. With this type of learning, an agent receives as input some indication about the state of the environment and then chooses an action which will affect this state. The agent will also receive a reinforcement signal, a scalar value which can either be positive (reward) or negative (punishment). The ultimate goal for the agents is to learn a policy which indicates which actions they should take according to the circumstances and which maximizes the sum of the values of the reinforcement signals in the long-run [5].

One of the most commonly used algorithms for reinforcement learning is Q-learning, first introduced by Watkins in 1989 [6]. In this algorithm, agents use a two-dimensional look-up table where each cell represents a state-action pair (s,a) and contains a value which represents the potential reward in the long-run of choosing action a when they sense the environment is on state s.

The advantages of Q-learning include being simple to implement and only requiring mild assumptions of its environment. In particular, it does not require an explicit model of the environment. However, this algorithm also presents some drawbacks as it can only handle discrete representations of the state of the environment and actions, and it has trouble scaling with increases in the state-action space [7].

The α parameter in algorithm 1, with $0 < \alpha \leqslant 1$, represents the learning rate, which reflects how much the newly acquired information affects the existing Q-value.

The γ parameter, with $0 \leqslant \gamma \leqslant 1$, is the discount factor, which affects the impact of future rewards. If γ is set to 0, then the agent will only consider the immediate rewards of their actions.

$\Pi(s)$ represents the *action selector*, which tells the agent which action to follow given what it sensed from the environment. One possible *action selector* would be to choose the action a with that maximizes $Q(s,a)$, with s being the current state. However, this strategy (that maximizes *exploitation* and disregards *exploration*) does not enable the agents to explore different alternatives, which could lead to greater long-term rewards. On the other hand, too much exploration does not allow the agent to benefit from the shorter-term rewards as actions with lower Q-values may lead to negative rewards. Thus, a good action selection policy must be able to balance *exploitation* and *exploration*.

A widely used policy $\Pi(s)$ is Softmax, which uses action-selection probabilities that are computed using formula 1.

$$Pr(a|s) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{a' \in A} e^{\frac{Q(s,a')}{\tau}}} \quad (1)$$

Where A is the set of possible actions and τ is a positive parameter known as the *temperature*. Higher temperature values make it so all actions have approximately the same chances of being chosen, whereas lower temperatures give more weight to actions with higher Q-values [8].

A variant of the Q-learning algorithm is SARSA, whose only difference is on the formula used to update the Q-values. As seen in algorithm 2, when updating the Q-value, SARSA includes the discounted value of the action in the successor state selected by the policy, rather than the action with maximal value. Thus, SARSA is said to be an *on-policy* method, whereas Q-learning is *off-policy* [8].

C. Fuzzy control

Fuzzy controllers use linguistic variables, which consist of a set of natural language terms. One example of a variable is *height*, which could have as terms *short*, *medium* and *tall*. The main issue is that the concepts expressed by these variables are subjective. For instance, there is no universally accepted height threshold above which a person is considered to be *tall*. Thus, rather than using traditional sets, fuzzy sets are used to model these linguistic terms. The difference between a classical set and a fuzzy set is that the membership function $\mu(x)$ that indicates whether an object x belongs to the set

Fig. 5. Sequential overlapped frames of the learned skills. Sprint experiment the running skill (a) and Dribbling skill (b).

using (a) and dribbling (b). The former was obtained from specific experiment under the running scenario, which will covered in section IV-A3.

For each experiment, three policies were optimized with critical initial conditions. Unless otherwise stated, the results re always averaged over those policies for 1000 episodes (h, totaling 3000 samples.

Running Skill

In this scenario, the main goals are speed and equilibrium, th greater focus on the former concept. With this n mind, ee experiments were realized:

1) Turn & Go – The robot must run as fast as possible in a predefined direction, but the initial position and orientation is set randomly;
2) Run Challenge – Similarly to the RoboCup's Gazebo running challenge [13], [14], the robot starts in the correct direction and must run linearly as fast as possible;
3) Sprint – equal to the previous experiment but the episodes are shorter.

1) Turn & Go: Table III shows the parameters that were osen for this experiment. The robot is initialized as in Fig.

TABLE II
AT PPO HYPERPARAMETERS FOUND BY OPENAI FOR THE MUJOCO 3D HUMANOID ENVIRONMENT [17]

Parameter	Value
Maximum time steps	10e6
Time steps per actor batch	2048
Clipping	0.2
Entropy Coefficient	0.0
Optimization epochs	10
Optimization learning rate	3e-4
Optimization batch size	64
Gamma	0.99
Lambda	0.95
Adam epsilon	1e-5
Annealing Schedule	linear

2. The episode has a maximum length of 585 steps (11.7s), but it may end earlier if the agent falls or goes beyond a certain threshold (see Table I). The reward compensates a positive speed on the desired direction and penalizes any deviation.

Using a direct control approach (i.e., the server receives actuator speed commands directly from the PPO's MLP output layer), the robot runs for an average of 6.2m at a speed of 0.83m/s. With an indirect control approach, the MLP output values represent absolute joint angular positions. The specific target angle for joint i is given by:

$$\theta_i^{target} = (x_i + 1) * \frac{b_i - a_i}{2} + a_i, \quad (2)$$

where x_i is the MLP output value for joint i, clipped to the range $[-1, 1]$; and a_i and b_i are the lower and upper bounds, respectively, of the joint's angular range. The unclipped angular velocity of actuator i at time step t is then computed by the following equation:

$$\omega_i^t = (\theta_i^{target} - \theta_i^{t-1})/k, \quad (3)$$

where θ_i^{t-1} is the angular position of joint i in the previous time step; and k is a constant that adjusts the speed at which the target angle is reached.

For the indirect control approach, multiple experiments were realized to evaluate the optimization performance for k different k values. The results displayed in Fig. 4a show the average distance traveled by the robot along the x-axis. Note that the actual trajectory length is greater, but it was omitted since it is not part of this scenario's objective. The maximum average distance value was 9.53m with a standard deviation (SD) of 4.18m, obtained for $k = 18$.

Analogously, Fig. 4b shows the average absolute drift on the x-axis. This linear speed is calculated for each episode, by dividing the displacement along the x-axis by the total number of time steps. This includes the initial stage where the robot tries to turn in the correct direction. The maximum average speed was 1.15m/s with a SD of 0.33m/s, and corresponds to $k = 4$, for which the average distance was 2.84m.

2) Run Challenge: On this task, the indirect control approach was employed, with $k = 7$, and without automatic head movements. The chosen parameters are displayed in Table IV. Since in RoboCup 2017, the Gazebo running challenge had a length of 20 seconds [15], the maximum number of steps per episode was increased to 1100 (22s). The extra time was given to mitigate the cases where the robot falls at the end of the episode (further details in section V).

TABLE III
SIMULATION PARAMETERS FOR THE TURN & GO EXPERIMENT[1]

Parameter	Value
Max. steps	300
Reward	$x_t - \|x_t\| * 0.2$
Initial coordinates	$x \in [-1.5, -1]$, $y \in [-0.5, 0.5]$
Initial angle	$\theta \in [-90°, 90°]$

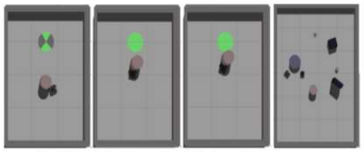


Fig. 3: The robot learning to push a cylinder in two different environments.

***Abstract*—Robot pushing is a task with immense relevance in the robotics research community. The full achievement of pushing could enable robots to navigate and operate in complex, unstructured environments much like a human person does. Its impact in manufacturing robots could be extensive and lead to an increase in global production. Taking into consideration the importance of this task and the wide, extensive use of q-learning in the reinforcement learning community, in this paper, we focus on a simple pushing task and perform an assessment of the q-learning technique. Characteristics such as state precision and exploration/exploitation techniques are some of the pillars of the performed assessment. Through our study, we conclude that, whilst q-learning has had large adaptation, its classical version leads to unsatisfactory results whilst still having concerning computational efficiency that easily overpowers most average machines on the task of robot pushing.**

***Index Terms*—robot pushing, q-learning, ROS, Gazebo**

I. Introduction

Intelligent robotics has been a fertile field of both academic and industrial based investigation. The pursuit of automated machines has been very much tied to the augmented shared necessity to increase productivity and reduce labour intensive jobs, with several past human-based tasks being successfully carried out by robots in the contemporary world [1]–[3].

As the complexity of the environments and tasks that robots are asked to perform increase drastically, research in the area of intelligent robotics must move towards increasingly accurate, versatile and efficient algorithms.

One of the tasks that might prove to be a central pillar of the deployment of robots into society is pushing [4], [5]. Pushing consists of having the robot push target objects to desired locations. Whilst robot pushing may be heavily impactful in industrial robots, especially those that handle objects, its applications extend even further. Hoping to deploy independent robots in any type of uncontrolled, unstructured environments, would require them to push obstacles away from their path or towards specific locations [6]. As such, pushing might become one of the base modules for robotics in a wide range of applications.

Considering optimisation techniques that are widely applied to reinforcement learning pipelines, q-learning [7]–[9] stands out due to its large adaptation to several tasks and simplicity of application.

Q-learning is a model-free reinforcement learning technique that can be described as a method of asynchronous dynamic programming. The objective of q-learning is to enable agents to learn an optimal policy to act in Markovian domains without having to build a map of the domain itself. Since the Markovian assumption is used, the environments in q-learning are assumed to be stochastic yet with independent events. This optimisation technique has been widely applied in robotics research due to its usefulness in converging towards optimal policies in a model-free form [8], [10], [11].

Building upon this, we consider both the primal task of robot pushing and the widely adopted q-learning technique as the object of our research. Through this work, we hope to answer the following questions: *how can classical q-learning be applied to robot pushing tasks?* and *to what extent is the trade-off between accuracy and computational resources worth its cost when applied to robot pushing?*

One of the tasks that might prove to be a central pillar of the deployment of robots into society is pushing [4], [5]. Pushing consists of having the robot push target objects to desired locations. Whilst robot pushing may be heavily impactful in industrial robots, especially those that handle

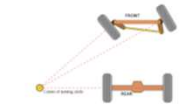


Fig. 3. Ackermann Steering representation.

One important thing to take into account is that each tire path has different radius which means that each tire is at a different rate. The tires of the outside travels a further distance than the inside ones in the same time, that means it's rotating more quickly. In descending order angular velocity of each tire goes from front right tire to rear right tire to front left tire to rear left tire.

So the front right tire is rotating faster and the rear left tire is rotating slower. This happens because the rear left tire has the smallest distance to travel. Which means that our right side is moving faster than our left side when going on the left-hand turn and vice-versa if going on the right-hand turn.

If we are in four-wheel drive on a pavement or something like that, we have enough tracking, enough grip. So we're gonna need a differential so that we can split the speeds between the front tires and the rear tires. Otherwise, we're going to have some buckling, it's not going to work and we're going to wear the tires and we're going to be slipping, we'll have under-steer.

Which means that if we put in a center differential on that point then it would allow our tires to rotate at a different rate.

A. ROS Ackermann Control System

For our ackermann control system we used Ackermann controller that ROS offers. It assumes Ackermann front-wheel steering. The left and right front wheels are generally at different angles. To simplify, the commanded angle corresponds to the yaw of a virtual wheel located at the center of the front axle, like on a tricycle. Positive yaw is to the left. (This is *not* the angle of the steering wheel inside the passenger compartment.)

Zero steering angle velocity means change the steering angle as quickly as possible. Positive velocity indicates a desired absolute rate of change either left or right. The controller tries not to exceed this limit in either direction, but sometimes it might.

```
# virtual angle (radians)
float32 steering_angle

# rate of change (radians/s)
float32 steering_angle_velocity
```

Drive at requested speed, acceleration and jerk (the 1st, 2nd and 3rd derivatives of position). All are measured at the

Fig. 1: AlphaBot2 front view.

Fig. 2: Adaptation of Conde's track.

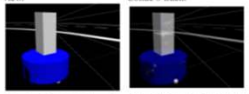


Fig. 3: Model of the AlphaBot2 in the Gazebo simulation.

Fig. 4: Visualisation of the model joints in the Gazebo simulation.

III. Simulations and Experiments

In order to verify the correct implementation of the model and system architecture, nodes and topics, the robot was launched is an empty Gazebo's word (fig. 1), and messages were published to the topics /alphabot2/vertical and /alphabot2/horizontal through the *rostopic command-line tool*.

Using the command: *rostopic pub /alphabot2/vertical std_msgs/Float64 "data: -45"* the camera starts to face the ground, represented in fig. 8 where it is possible to see the new position of the Tilt and its vertical axis as the dashed red line which is tilted -45 degrees. To control the Pan a std_msgs/Float64 is published to the /alphabot2/horizontal topic using the command: *rostopic pub /alphabot2/horizontal std_msgs/Float64 "data: 45"* (Fig 10). It is possible to see that the camera is positioned to the right of its initial position when a negative value is submitted to the horizontal topic and it is positioned in the left if a positive value is submitted.

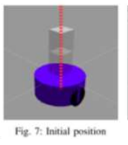


Fig. 7: Initial position

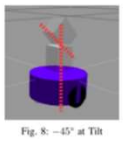


Fig. 8: −45° at Tilt

1) Ros Topics:

- "/alphabot2/control": Publish a message of type *geometry_msgs/Twist* setting *linear velocity x* and *angular velocity z* to control the movement of the AlphaBot2;
- "/alphabot2/vertical": Publish a message of type *std_msgs/Float64* setting a value between -90 and 90 to control the tilt angle of the camera;
- "/alphabot2/horizonal": Publish a message of type *std_msgs/Float64* setting a value between -90 and 90 to control the pan angle of the camera;
- "/alphabot2/camera/image_raw": Subscribe to this topic to read a message of type *sensor_msgs/Image*.

2) System requirements: For the reuse of the system, on the side of the simulator the minimum requirements are the Ubuntu 16.04, the ROS Kinetic and the Gazebo7. As to the real robot, the used system was Raspbian GNU/LINUX 9 (stretch).

B. System architecture

The system architecture is based in the ROS meta-operating system. Through the implementation of nodes which publish and subscribe to specific topics message-passing between processes is accomplished. Two levels of packages (pkg) are responsible for the control of the robots behaviour in the Gazebo simulation and in the real robot. Hence, on the side of the Gazebo simulation two nodes are implemented for the readout of the camera sensor, which publishes a raw image, and a pkg for the control of the joints present in the camera support (motions of pan and tilt), this subscribes to the angle positions for the two motors. The connections between nodes using ROS for the real the robot are presented in fig. 5, and in fig. 6 is shown the simulator architecture. Using these packages, students and developers can program and test their own applications subscribing or publishing to the topics that will be described next.

is analyzed, allowing for faster processing and better error calculation. The way it was implemented, the user is allowed to select the relative distance of the top of the ROI to the top of the image. This is a floating point number between 0.0 and 0.8, where 0.0 represents the very top of the image and 0.8 the bottom, because the ROI height is fixed at 20% of the image height.

Once the ROI is selected, a mask that detects only the desired color is computed. Then, the inverse of this mask is replaced by white, so that the only non white pixels are those belonging to the line. This image is then converted to gray scale, blurred and finally a threshold is computed, so that the line pixels become white and everything else is black.

In the final image processing step, the contours of the image are computed. In most scenarios a single contour will be computed, or none, if there is no line of the chosen color in the ROI. But in order to account for those cases where two or more lines of the same color or two parts of the same line are visible in the ROI, the most central contour is selected. In fact, for each contour, its moments [9] are computed and the centroid is then calculated as

$$\left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right).$$

The minimum distance of the centroids to the center of the ROI determines which contour is selected. The error corresponding to that contour is then given by

$$1 - 2 * \frac{m_{10}/m_{00}}{d},$$

where d represents the width of the ROI. This yields a floating point number between -1 and 1, where -1 means the line is in the far right of the ROI, 1 means it is in the far left and 0 means it is in the middle.

Figure 5. ROI with line, contours and centroid.

Figure 5 shows the ROI after threshold, with the line contours in green and the computed centroid in red. Since the line is slightly to the right of the center of the ROI, the corresponding error will be negative.

2.4.2. Initial random movement. Until the first time a line is detected in the camera image from the phone, the robot will move randomly. The implementation of this random movement is very straightforward, a simple movement command is sent to the robot, setting the speed of each wheel to a random integer between 1 and 5. When the next camera image is received if the processing still fails to reveal a line to follow, a new random movement command is sent to the robot.

2.4.3. Follow line movement. After a line first is detected the robot will attempt move along following that line. To do this in a stable manner, a PID controller was implemented. The result is that at each image processed, the angular velocity of the robot is given by

$$\omega = K_p * P + K_i * I + K_d * D,$$

where K_p, K_i, and K_d are the proportional, integral and derivative constants defined by the user, and P, I and D are the proportional, integral and derivative terms, respectively. In this case, $P = e$, $I = I + e$ and $D = e - d$, where e is the current error obtained from the image processing and d is the error obtained in the previous image. When the robot is initialized, e, d and I are all set to 0.

Since the velocity for each wheel must be specified in any move command, the final line following robot movement is given by setting the left wheel speed to $v - \omega$ and the right wheel speed to $v + \omega$, where v is the linear velocity defined by the user.

If at some point during its movement the robot loses sight of the line, that is, the ROI of the current image has no line of the desired color, then a new artificial error is computed as $e = d/|d|$, which means that the error is -1 if the previous error was negative and 1 if it was positive. Given that the robot just lost sight of the line, which we assume is continuous, the previous error should not be 0, but in the unlikely chance it is, then the current error is also set to 0. After this artificial error is computed, movement proceeds as usual.

3. Results

Since there is no simulator to test Robobo and the line following program, non virtual tests had to be conducted. For these tests the maps in Figures 2 and 3 where printed in large white paper and the robot was set up those maps, as shown in Figure 6.

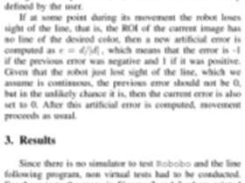


Figure 6. The colored lines map with Robobo.

Because the random movement is not very relevant and is so simple, there is no real need to test it. Hence, for most tests, the robot was placed directly on top of the line

a linear system and obtained an image with linear distances from the robot's edge "figure 7 right".

Fig. 7. Transforming spherical capture into linear system.

We then detected an object by analysing the color differences between the background and then used the same equations 1, 2, 3, 4 and 5 to relate the light bulb radius, the height of the mirror surface, the height of the camera lens and the field of view (FOV) with the distance the detected object is.

IV. Experiments and Results

After resolving the equations 1, 2, 3, 4 and 5 we concluded that we needed a height of 116.6mm and a vertical FOV of 33.66°or a height of 77.9mm and a horizontal FOV of 43.85°to view the whole light bulb.

However, after setting that height, we analyzed that viewing the whole light bulb was to wasteful. We needed higher resolution, which slowed down the computing time and a lot of the reflected space was reflected towards infinity. So, we decided to put the mirror surface at a height of 100mm, with a FOV of 29.56°, which is close to the 33.66°. Please notice that since we crop the image to a square proportion, only the smaller of the FOVs is important.

To test the system, we placed the AlphaBot in a location where the background had a uniform grayish color, surrounded by objects with very distinct colors.

Fig. 8. AlphaBot surrounded by obstacles with distinct colors.

V. Conclusions and Future Work

Implementing a 360°obstacle detection system on a low-budget system 3D Printers helped a lot in iterating between support structures

References

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. pages 404–417. Springer, Berlin, Heidelberg, 2006.

[2] Peng Chen, Yuanjie Dang, Ronghua Liang, Wei Zhu, and Xiaofei He. Real-Time Object Tracking on a Drone with Multi-Inertial Sensing Data. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):131–139, jan 2018.

[3] I. Colomina and P. Molina. Unmanned aerial systems for photogrammetry and remote sensing: A review, jan 2014.

[4] S. Darms, J L. Buessler, G. Hermann, J P. Urban, and B. Kusenopatro. Visual servoing quadrotor control in autonomous target search. In *Proceedings - 2011 IEEE 3rd International Conference on System Engineering and Technology, ICSET 2013*, pages 319–324. IEEE, aug 2013.

[5] Halit Ergezer and Kemal Leblebicioglu. Path planning for UAVs for maximum information collection/pptsp. *IEEE Transactions on Aerospace and Electronic Systems*, 49(1):502–520, jan 2013.

[6] Daniele Fontanelli, David Macii, and Tizar Rizano. A fast and low cost vision based line tracking measurement system for robotic vehicles. *ACTA IMEKO*, 4(2):90–99, jun 2015.

[7] C. Gomez, A. C. Hernandez, J. Crespo, and R. Barber. Integration of Multiple Events in a Topological Autonomous Navigation System. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 41–46. IEEE, may 2016.

[8] Satoshi Hoshino and Kyohei Niimura. Robot Vision System for Real-Time Human Detection and Action Recognition. pages 507–519. 2018.

[9] Yue Hee Kho, Aminath Elaf Abdulla, and Jessica Chan Zhai Yan. A vision-based autonomous vehicle tracking robot platform. In *2014 IEEE Symposium on Industrial Electronics & Applications (ISIEA)*, pages 173–178. IEEE, sep 2014.

[10] Hyon Lim and Sudipta N. Sinha. Monocular Localization of a moving person onboard a Quadrotor MAV. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2015-June, pages 2182–2189. IEEE, may 2015.

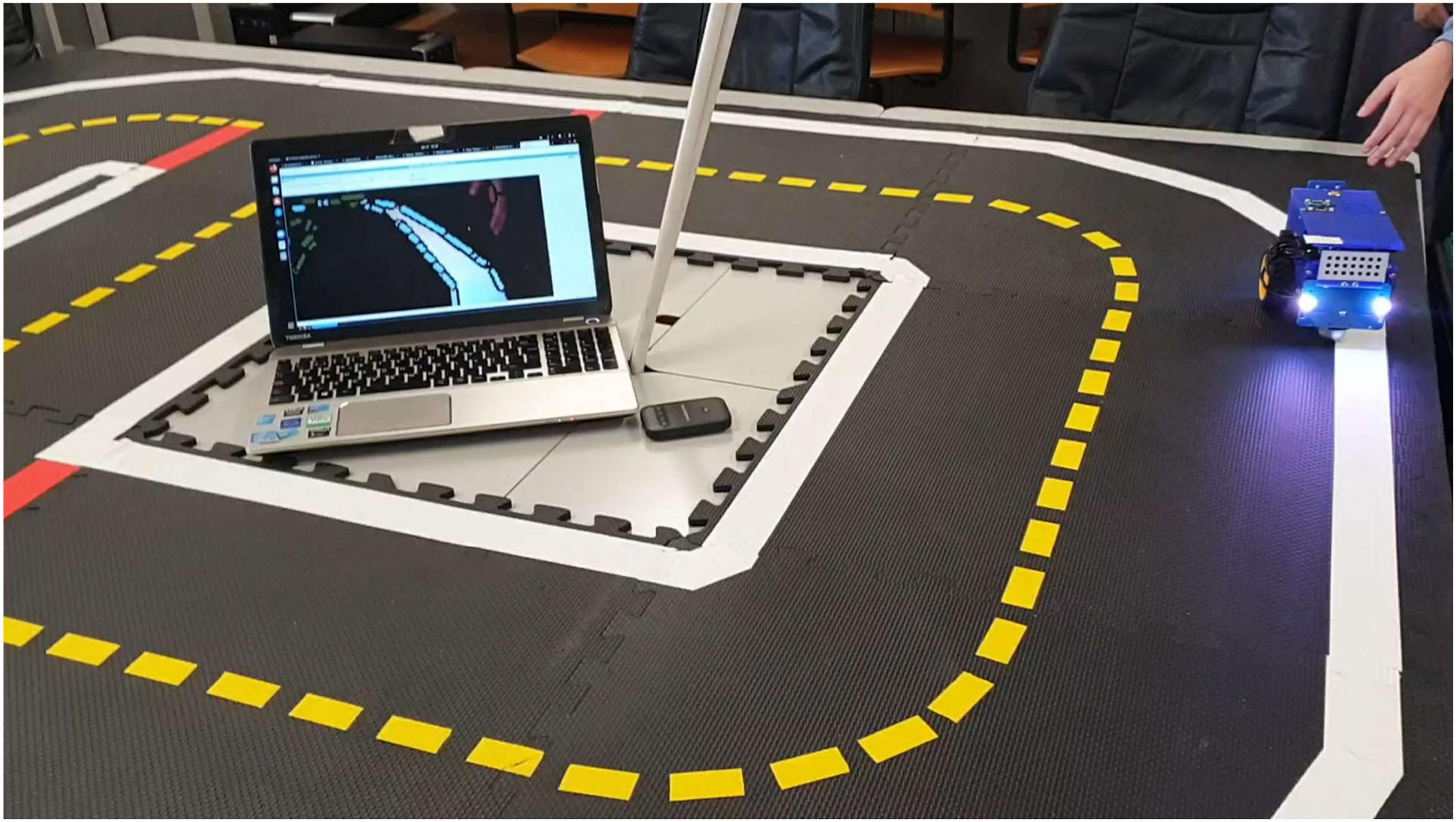
[11] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, nov 2004.

[12] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. The GRASP Multiple Micro-UAV Testbed. *IEEE Robotics & Automation Magazine*, 17(3):56–65, sep 2010.

Sample Results of the Students' Works

1. João Ferreira, Filipe Coelho, Armando Sousa, Luís Paulo Reis, *BulbRobot - Inexpensive Open Hardware and Software Robot Featuring Catadioptric Vision and Virtual Sonars.* In: Robot 2019: Fourth Iberian Robotics Conference. ROBOT 2019. Advances in Intelligent Systems and Computing, Vol. 1092, pp. 553-564, Springer, Cham. https://doi.org/10.1007/978-3-030-35990-4_45. (Springer, SCOPUS)
2. Ana Beatriz Cruz, Armando Sousa, Luìs Paulo Reis, *Controller for Real and Simulated Wheelchair with a Multimodal Interface Using Gazebo and ROS,* 2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), pp. 164-169, Ponta Delgada, Portugal, Online Event, art nº 19631905, DOI: https://doi.org10.1109/ICARSC49921.2020.9096195, (IEEE, SCOPUS)
3. José Aleixo Cruz, Henrique Lopes Cardoso, Luís Paulo Reis, Armando Sousa, Reinforcement Learning in Navigation and Cooperative Mapping, 2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), pp. 200-205, Ponta Delgada, Portugal, Online Event, art nº 19631919, DOI: https://doi.org/10.1109/ICARSC49921.2020.9096136, (IEEE, SCOPUS)
4. Liliana Antão, Armando Sousa, Luís Paulo Reis, Gil Gonçalves, *Learning to Play Precision Ball Sports from scratch: a Deep Reinforcement Learning Approach,* 2020 International Joint Conference on Neural Networks (IJCNN), Art Nº 20006737, Glasgow UK, July 2020, DOI: https://doi.org/10.1109/IJCNN48605.2020.9207518, (IEEE, CORE A/A, SCOPUS)
5. Gonçalo Leão, C. M. Costa, Armando Sousa, Luís Paulo Reis, Germano Veiga, Using Simulation to Evaluate a Tube Perception Algorithm for Bin Picking. 2022, ROBOTICS Journal, 11(2 46), 46 (14), DOI: https://doi.org/10.3390/robotics11020046 (SCImago: Q1/Q2)

Student Projects



Learning Outcomes

- Acquire knowledge of current state and trends in Robotics
- Demonstrate understanding of the problems of intelligent robotics, particularly by selecting appropriate techniques to model and solve them
- Have a broad critical understanding of how Artificial Intelligence and Machine Learning may be applied generally to Intelligent Robotics
- Appreciate the problems associated with designing and programming intelligent robots and multi-robot systems for different problems
- Develop research work, demonstrate the origins of the ideas by referencing sources used in the context of intelligent robotics, being aware of the best projects/research works in this area around the world

Teaching Methods

- **Challenging students to Higher Level Learning as appropriate in a PhD/MSc program.** Of course low level learning, i.e., comprehending and remembering basic information and concepts is important. However emphasis will be on problem solving, decision making and creative thinking/design
- **Use Active Learning.** Exposition will be made mostly with interaction in theoretical classes. Use of appropriate materials/ simulators/ platforms/ problems
- **Use simple but structured sequence of different learning activities** (lectures, demos, exercises, reading, analysis, writing, oral pres., design, experiments)
- **Opening classes with basic principles** to lay the foundation for complex and high level learning tasks in later, complex classes and assignments
- **Detailed feedback given to students** about the quality of their research work and learning process. High level, active learning require to know whether they are "doing it correctly!"
- **High-level teaching** method enable to increase skills in research in all other areas related to informatics and computer science

Summary

- **Method:**
  - Theo + Practical + Active Learning + Problem Based Learning
- **Programme:**
  - **Robotics, Intelligent Robotics and Simulation, ROS**
  - **Perception/Decision/Action**
  - **Mapping, Localization, Navigation, Planning**
  - **Learning, Interaction, Cooperative Robotics**
- **Emphasis on Programming Intelligent Machines**
- **Practical Knowledge Application with:**
  - **Simulators / Robotic Platforms**
- **Not needed:**
  - Electronics + Digital Systems + Electricity + Control
- **Tools: Your choice (ROS, Simulators / Robotic Platforms)**
- **Problem: Your choice (Autonomous Driving, Soccer Robot, Game Playing Robot, …)**
- **Scientific Content**
  - Collaboration in R&D Projects
  - Write and Analyze Scientific Papers

Conclusions

- Intelligent Robotics; ROS – Robot Operating System; Perception and Sensorial Interpretation; Locomotion and Action; Localization, Mapping and SLAM; Planning and Navigation; **Robot Learning**; Human-Robot Interaction; Cooperative Robotics; Robotics in the Future
- **AI for robotics** with emphasis on **ML and RL, HRI and Multi-Robot Coordination and Learning**
- **Platforms, Simulators, Base Codes, Projects and Competitions**
- **Higher level learning, active learning, Kahoots and structured sequence of different learning activities**
- **Detailed feedback to students about their research work and learning process**

Intelligent Robotics Curricular Unit

Luís Paulo Reis

lpreis@fe.up.pt

Director/Researcher LIACC

Associate Professor at FEUP/DEI

Armando Sousa

asousa@fe.up.pt

Researcher INESC-TEC

Assistant Professor at FEUP/DEEC